第 13 回
情報科学ワークショップ

The 13rd Workshop on Theoretical Computer Science, Ikoma, Nara, September 2017 (WTCS2017)

増澤　利光
角川　裕次
大下　福仁
首藤　裕一

主担当：大阪大学、NAIST

本ワークショップは、JST SICORP 「災害や攻撃に対してデータ依存公共ユーティリティの生存性と継続的操作を効率よく実現する手法」の平成２９年度ワークショップとして開催します。

序言

本論文集は 2017 年 9 月 4 日〜6 日，かんぽの宿 大和平群（奈良県生駒郡）にて開催された第 13 回情報科学ワークショップでの発表原稿をまとめたものである． 本ワークショップは，日本の並列／分散計算の研究者が研究室以上の議論と研究会未満のフォーマルさを目指し，合宿形式でお互いを徹底的に切りまくる「チャンバラ大会」を目的とし，広島大学の中野浩嗣先生の呼びかけで，大阪大学，九州大学，九州工業大学，京都工芸繊維大学，名古屋工業大学，奈良先端科学技術大学院大学の研究者有志によって 2005 年に始まったものである．2005 年 9 月に第 1 回を出雲で開催して以来，第 2 回瀬戸，第 3 回門司，第 4 回長浜，第 5 回広島，第 6 回桑名，第 7 回福岡，第 8 回神戸，第 9 回唐津，第 10 回福山，第 11 回北名古屋，第 12 回富士吉田に続き，今回で 13 回目の開催となる．今回は九州大学，九州工業大学，広島大学，大阪府立大学，京都大学，名古屋大学、名古屋工業大学，豊橋技術科学大学，東京工業大学，法政大学，奈良先端科学技術大学院大学，大阪大学の 12 大学から 50 人の参加者があり，32 件の発表が行われた．また，9 月 5 日の自由討論の時間を利用して，泉泰介先生（名古屋工業大学）の特別講演「ネットワーク上の分散グラフアルゴリズムと最適化」を開催し，分散グラフアルゴリズムの最新の研究動向と成果を紹介いただき，参加者の好評を得た．今回の開催場所は，奈良県生駒郡のかんぽの宿 大和平群（やまとへぐり）で，世界遺産である法隆寺（現存する世界最古の木造建築物）と古都奈良（世界最大の木造歴史建築物である東大寺など）へのアクセスも至便な場所であった．悠久の歴史を感じさせる緑豊かな大和の地で，並列／分散計算の最先端研究について夜遅くまで活発な討論が行われ，有意義な時間を共有することができた．2015 年度から，学生をはじめとする若手研究者たちのモチベーションの向上を念頭に，参加した大学教員らの審査により，優れた研究に対して優秀研究賞を，さらに素晴らしいプレゼンテーションを行った学生を対象に優秀プレゼンテー ション賞を授与している．今回は，1 件の論文に優秀研究賞を，5 名の学生に優秀プレゼンテーション賞を授与したが，本予稿集にその受賞一覧を掲載している．受賞者らには今後の益々の活躍を，そして未受賞者諸君には次の受賞者を目指して更なる研鑽を積んでいただきたいと思う．なお 2018 年度は九州工業大学・九州大学を中心とした九州チームが主担当の予定である． 最後に，今回の開催にあたりご協力をいただいた皆様，ワークショップ運営にご協力をいただいた参加者の皆様に厚く御礼申し上げる．

2017 年 9 月

増澤　利光
角川　裕次
大下　福仁
首藤　裕一

目次

セッション 3A

セッション 3B

受賞者一覧

優秀研究賞（1 件）

・ 山中寿登，小野廣隆（名古屋大学）
単位円グラフの L(2,1)-ラベリングのための近似アルゴリズム

優秀プレゼンテーション賞（5 件）

・ 安見嘉人（奈良高専）
“大域公平性を仮定した個体群プロトコルモデルにおける k 分割アルゴリズム”
・ 木谷裕紀（九州大学）
“2 人単貧民の必勝判定アルゴリズム”
・ 迫田賢宣（名古屋大学）
“コード配色の変更を認めるマスターマインドの最適な推測回数”
・ 木村友彦（広島大学）
“WebRTC を用いたツインビュー型 P2P ビデオ会議システム”
・ 奥村圭佑（東京工業大学）
“未知環境における動的なアンカーを使った移動ロボット群の協調探索”

極小支配集合問題の一般化とその自己安定アルゴリズムについて

大阪大学 大学院情報科学研究科
小林永樹 首藤裕一 角川裕次 増澤利光

Abstract

A self-stabilizing system is guaranteed to eventually reach and stay at a legitimate configuration regardless of the initial configuration. In this paper, we propose a generalization of the classical k-redundant dominating set problem, and propose a self-stabilizing algorithm for finding a minimal generalized dominating set in an arbitrary network under the synchronous daemon. The classical k-redundant dominating set in a distributed system is a set of nodes such that each node is contained in the set or has k neighbors in the set. On the other hand, in our generalized dominating set, each node i is given its domination wish set $C_i = \{W_1^i, W_2^i, \cdots : W_x^i \subseteq N_i\}$, where N_i is a set of neighbors of node i, and each node i not in the dominating set has some $W_x^i \in C_i$ such that each member in W_x^i is in the dominating set. We show that the worst case stabilization time of the proposed algorithm is $O(n)$ rounds, where n is the number of nodes.

1 Introduction

A *self-stabilizing* system is guaranteed to eventually reach and stay at a legitimate configuration regardless of the initial configuration [1]. This enables a distributed system to be adaptive to transient faults and topology changes in a network. Two important requirements to a self-stabilizing algorithm are *closure* and *convergence* properties. The closure property ensures that once a system reaches a legitimate configuration, it stays at legitimate configurations forever unless new transient faults or topology changes occur. The convergence property ensures that regardless of the initial configuration, the system reaches a legitimate configuration in a finite time.

A *dominating set* in a distributed system is a set of nodes such that each node is contained in the set or has at least one neighbor in the set. A *k-redundant dominating set* [4] is a set of nodes such that each node is contained in the set or has at least k neighbors in the set. We call members of a dominating set *dominators* and the remainder *dominatees*. A *minimal dominating set* is useful for clustering and routing in an ad hoc wireless network. A dominating set (resp. k-redundant dominating set) is minimal if and only

if no proper subset of the set are a dominating set (resp. k-redundant dominating set). In these definitions, each node uniformly requires the same amount of domination, that is, each dominatee has at least one or k dominators in the neighbor respectively. So, these definition cannot give each node a different amount of domination. In this paper, as a further generalization of these problems, we propose the *generalized dominating set problem* in which domination requirements may not be uniform by nodes. For example, domination requirements can be decided by the network performance, the node performance, the network topology, the degree of a node and so on. In addition, a generalized dominating set can also express a weighted version of the k-redundant dominating set: given a network where each node is assigned a positive weight, each node is required to be a dominator or to be dominated by neighboring dominators whose total weight is k or more.

Contribution of this paper: The contribution of this paper is twofold. First, we introduce the generalized dominating set problem. Second, we propose a self-stabilizing algorithm for finding a minimal generalized dominating set in an arbitrary network under the synchronous daemon. In this paper, we assume the execution model where all nodes execute actions simultaneously in a lock-step fashion in each round (the synchronous daemon), and the communication model where each node can directly read local variables of neighbors (the state-reading model). These models are commonly used in literature of self-stabilization. Our algorithm repeats a sequence of four phases, and all nodes must execute an identical phase at each round. To realize the synchronization of the four phases, the self-stabilizing phase-clock synchronization algorithm [11] is utilized. The convergence time of our algorithm is $O(n)$ rounds, where n is the number of nodes.

Related works: N. Guellati and H. Kheddouci [5] surveyed self-stabilizing algorithms for finding a minimal dominating set and a minimal k-redundant dominating set under various kinds of daemons in various network topologies. The *minimum* dominating set problem is NP-hard, so several self-stabilizing algorithms [8, 15, 12, 13, 14] for the *minimal* dominating set (MDS) problem have been proposed. The first research of self-stabilizing algorithms for the minimal k-reduntant dominating set (MKDS) problem has been developed by Kamei and Kakugawa [6] which assumes a tree network under the central and the distributed daemons, and the convergence times of their algorithms are both $O(n^2)$ steps. Huang et al. [9, 10] presented two self-stabilizing algorithms for the minimal 2-redundant dominating set (M2DS) problem in an arbitrary network. Recently, Wang et al. [2, 3] proposed self-stabilizing algorithms for the MKDS problem, assuming the central and the distributed daemons, both of which stabilize in $O(n^2)$ steps. The results are summarized in Table 1. Note that MGDS in the table de-

notes a minimal generalized dominating set proposed in this paper.

Table 1: Self-stabilizing algorithms for various dominating set problems

Reference	Problem	Topology	Daemon	Convergence time
Hedetniemi et al. [8]	MDS	Arbitrary	Central	$(2n+1)n$ steps
Xu et al. [15]	MDS	Arbitrary	Synchronous	$4n$ rounds
Turau [12]	MDS	Arbitrary	Distributed	$9n$ steps
Goddard et al. [13]	MDS	Arbitrary	Distributed	$5n$ steps
Chiu et al. [14]	MDS	Arbitrary	Distributed	$4n$ steps
Huang et al. [9]	M2DS	Arbitrary	Central	$O(n)$ steps
Huang et al. [10]	M2DS	Arbitrary	Distributed	not mentioned
Kamei and Kakugawa [6]	MKDS	Tree	Central	$O(n^2)$ steps
Kamei and Kakugawa [6]	MKDS	Tree	Distributed	$O(n^2)$ steps
Kamei and Kakugawa [7]	MKDS	Arbitrary	Synchronous	$O(n)$ steps
Wang et al. [2]	MKDS	Arbitrary	Central	$O(n^2)$ steps
Wang et al. [3]	MKDS	Arbitrary	Distributed	$O(n^2)$ steps
This paper	MGDS	Arbitrary	Synchronous	$O(n)$ rounds

Organization of this paper: The rest of the paper is organized as follows. Section 2 presents formal definitions of the system model and the generalized dominating set (MGDS) problem. Section 3 presents our algorithm for the MGDS problem under the synchronous daemon in an arbitrary network topology. Section 4 gives correctness proof and performance evaluation. Section 5 gives concluding remarks.

2 Preliminaries

2.1 System model

A *distributed system* is modeled by an undirected graph $G = (V, E)$, where $V = \{0, 1, 2, \cdots, n-1\}$ is a set of n nodes and E is a set of m bidirectional communication links. Each node $i \in V$ has a unique identifier denoted by ID_i which is a nonnegative integer value. With abuse of notation, we use i to denote ID_i when it is clear from context. N_i denotes a set of nodes to which node i is adjacent, called *neighbors*. As a communication model, we assume each node can read local states (or variables) of neighbors without delay. This model is called the *state-reading model*. Each node can update its own local state only, but each node can read local states of neighbors. A *configuration* of a distributed system G is specified by an n-tuple $\gamma = (s_0, s_1, \cdots, s_{n-1})$, where s_i stands for the state of node i $(0 \leq i \leq n-1)$. Let Γ be a set of all possible configurations. An atomic step of each node i consists of the following two steps: (1) read local states of all neighbors and (2) update its local state depending on its current state and the states read from its neighbors. In this paper, we assume the *synchronous daemon* for node execution such that all nodes execute atomic

steps simultaneously in a lock-step fashion, and computations progress in rounds: in each round, every node executes an atomic step. Notice that each node reads the neighbors' states that are the ones at the beginning of the current round and updates its own state.

2.2 Self-Stabilization

When the configuration changes from γ to γ' ($\neq \Gamma$), the transition is denoted by $\gamma \mapsto \gamma'$. For any configuration γ_0, an *execution* Π starting from γ_0 is a maximal (possibly infinite) sequence of configurations $\Pi = \gamma_0, \gamma_1, \cdots$ satisfying $\gamma_t \mapsto \gamma_{t+1}$ for each $t \geq 0$.

definition 1. *Let Γ be the set of all possible configurations. A distributed system is* self-stabilizing *with respect to $\Lambda \subseteq \Gamma$ if and only if the following two conditions are satisfied.*

- Convergence: *Starting from an arbitrary configuration, a configuration eventually becomes one in Λ*
- Closure: *For any configuration $\gamma \in \Lambda$, any configuration γ' such that $\gamma \mapsto \gamma'$ is also in Λ.*

□

Each $\gamma \in \Lambda$ is called a *legitimate* configuration.

2.3 The Generalized Dominating Set Problem

A classical dominating set is formally defined as follows. Let $G = (V, E)$ be a distributed system.

definition 2. *A* dominating set *D of G is a subset of V such that for each node $i \in V$, i is in D or $|N_i \cap D| \geq 1$ (or there exists j such that $j \in N_i \cap D$).* □

definition 3. *[4] A* k-redundant dominating set *D of G is a subset of V such that for each node $i \in V$, i is in D or $|N_i \cap D| \geq k$.* □

definition 4. *A dominating set (resp. a k-redundant dominating set) D of G is* minimal *if no proper subset of D is a dominating set (resp. a k-redundant dominating set) of G.* □

The 1-redundant dominating set problem is equivalent to the dominating set problem. Hence, the k-redundant dominating set problem is a generalization of the dominating set problem. The generalized dominating set introduced in this paper is defined as follows.

definition 5. *Let* $C_i = \{W_1^i, W_2^i, \cdots, W_{c(i)}^i\}$ *for each node* i $(0 \leq i \leq n-1)$ *where* $W_x^i \subseteq N_i$ $(1 \leq x \leq c(i))$*, and let* $C = (C_0, C_1, \cdots, C_{n-1})$*. A* generalized dominating set *D of G with respect to C is a subset of V such that for each node i, i is in D or there exists $W_x^i \in C_i$ such that $W_x^i \subseteq D$. We call C_i a domination wish set of node i, and C a domination wish list.* □

definition 6. *A generalized dominating set D of G is* minimal *if no proper subset of D is a generalized dominating set of G.* □

A generalized dominating set is a further generalization of a k-redundant dominating set. Besides, we define the minimal generalized dominating set problem in a distributed system.

definition 7. *Let $G = (V, E)$ be an undirected graph modeling a distributed system. The* distributed minimal generalized dominating set problem *is defined as follows.*

Input of node i **:** *A domination wish set C_i.*

Output of node i **:** *A status $d_i = true$ or $d_i = false$.*

Condition : *A node set $\{i \in V : d_i = true\}$ is a minimal generalized dominating set of G with respect to $C = \{C_0, C_1, \cdots, C_{n-1}\}$.*

□

3 The Proposed Algorithm

3.1 Variables

In this section, we propose a self-stabilizing algorithm for the distributed minimal generalized dominating set problem. In our algorithm, each node i uses two constants, one external variable (controlled by external activity), two macro symbols and four shared variables. The constants are described as follows.

- **set of nodes** $N_i \subseteq V$: A set of neighbors of node i.
- **domination wish set** $C_i = \{W_1^i, W_2^i, \cdots : W_x^i \subseteq N_i\}$

The external variable is described as follows.

- **int** $PhaseClock_i \in \{1, 2, 3, 4\}$: We assume that external activity makes this variable increase by one (in the circular order) at each round as 1,2,3,4,1,2,3,4,1,2,$\cdots$ and take the same value in all nodes at each round. Our algorithm implicitly executes the self-stabilizing algorithm [11] for a phase clock synchronization simultaneously to maintain $PhaseClock_i$. For simplicity, in our algorithm, we omit the description of the phase clock synchronization algorithm.

Besides, we use macro symbols as follows.

- **set of nodes** $D_i = \{j \in N_i : d_j = true\}$: A set of neighboring dominators of node i. Consequently, a set $N_i - D_i$ means a set of neighboring dominatees of node i.
- $C'_i = \{W^i_x \in C_i : W^i_x \subseteq D_i\}$: A subset of C_i such that for each $W^i_x \in C'_i$, each node in W^i_x is a dominator. A dominatee i is *dominated* when $C'_i \neq \emptyset$.

The shared variables are described as follows.

- **boolean** d_i : This variable is *true* (resp. *false*) if node i is a *dominator* (resp. *dominatee*). We call this variable *status*. Note that the meanings of the *status* and *state* are different in this paper; the state means the set of the variables of node i.
- **boolean** $Permission^i_j$: This variable is used by node i to give a neighboring dominator $j(\in D_i)$ permission to become a dominatee. $Permission^i_j = true$ means that a dominatee i is dominated by the other set of dominators $(\in C'_i)$ even if $j \in D_i$ turns to be a dominatee. In other case, $Permission^i_j$ is *false*.
- **boolean** $ChangeFlag_i$: Node i sets this variable *true* if node i intends to change its *status* from a dominator to a dominatee or from a dominatee to a dominator.
- **node name** $Pointer_i$: This variable is assigned one node $j \in N_i \cup \{i\}$ to approve j's status change. Node j can change its status if $Pointer_l$ points to j for each node $l \in N_j \cup \{j\}$.

3.2 Algorithm Outline

Let us explain the idea of the proposed algorithm. The main feature of our algorithm is that once a dominator i turns to be a dominatee, node i never changes its status afterwards, that is, node i is dominated by at least one set of dominators in C'_i afterwards. Intuition of the status change rules of each node i is described as follows.

- **Rule 1 dominatee → dominator :** A dominatee i ($i.e., d_i = false$) turns to be a dominator ($i.e., d_i = true$) if it is not dominated, that is, $C'_i = \emptyset$.
- **Rule 2 dominator → dominatee :** A dominator i turns to be a dominatee if it is dominated and each neighboring dominatee $j(\in N_i - D_i)$ is also dominated even if node i turns to be a dominatee.

This idea for the algorithm seems intuitively correct; Rule 1 makes a set dominating, and Rule 2 makes a set minimal. However, its straightforward implementation does not work correctly under the synchronous daemon which is assumed in this paper. Let us observe three nodes, say i, j and k in the network such that nodes j and k are neighbors of node i, but nodes j and k are not neighbors each other, that is, node i is in the middle of nodes j and k. Suppose that node i is a dominatee with $C_i = \{\{j\}, \{k\}\}$, and nodes j and k are dominators. By Rule 2, nodes j and k simultaneously become dominatees if each of them has at least one set of dominators in C_j and C_k respectively. Then, node i has no set of dominators in C_i, and node i is still a dominatee; node i is not dominated.

To avoid such a scenario, we disallow the simultaneous status changes of nodes j and k in the above setting. Generally speaking, we avoid violation of domination by disallowing simultaneous status changes of two nodes within distance two (*e.g*, nodes j and k in the above example). The idea for such a control is described below.

- Each node i reads the status from each of its neighbors. Node i can now detect whether the condition of Rule 1 is satisfied. Concerning Rule 2 at each neighboring dominator j, node i can detect whether it is still dominated even if node j turns to be a dominatee. If so, node i notifies node j of permission to become a dominatee.

- According to the permissions, each dominator can know whether or not the condition of Rule 2 is satisfied. When node i satisfies the condition of Rule 1 or Rule 2, it notifies its neighbors that it intends to change its status by setting $\mathit{ChangeFlag}_i := \mathit{true}$.

- To disallow nodes within distance two to simultaneously change their statuses, we use the pointer $\mathit{Pointer}_i$; node i sets $\mathit{Pointer}_i := j$ where $j \in N_i \cup \{i\}$ is the node with the smallest ID among $\{h \in N_i \cup \{i\} : \mathit{ChangeFrag}_h = \mathit{true}\}$. Node i sets $\mathit{Pointer}_i := \mathit{null}$ if no node h in $N_i \cup \{i\}$ satisfies $\mathit{ChangeFrag}_h = \mathit{true}$.

- After the pointer assignment, node i changes its status if node i is pointed by all the neighbors and itself.

By this, we prevent the simultaneous status changes by nodes within distance two. In **Algorithm 1**, we show the detailed actions of each node i at each phase. The algorithm repeats the sequence of the four phases. We explain the action of node i in each phase as follows.

- **Phase 1:** Each node i updates $\mathit{Permission}_j^i$ for each neighbor j. Node i sets $\mathit{Permission}_j^i := \mathit{true}$ if node i is a dominatee and $\{s \in C_i' : j \notin s\} \neq \emptyset$ holds. In other case, $\mathit{Permission}_j^i = \mathit{false}$. This variable is used in Phase 2.

- **Phase 2:** Each node i updates $ChangeFlag_i$. Node i sets $ChangeFlag_i = true$ if the condition of Rule 1 or Rule 2 (mentioned above) is satisfied, and $ChangeFlag_i = false$ otherwise. This variable is used in Phase 3 and 4.

- **Phase 3:** Each node i updates its $Pointer_i$. Node i sets $Pointer_i$ to one node $j \in N_i \cup \{i\}$ with the smallest ID among $\{h \in N_i \cup \{i\} : ChangeFlag_h = true\}$. Node i sets $Pointer_i := null$ if there exists no neighbor j such that $ChangeFlag_j$ is $true$. This variable is used in Phase 4.

- **Phase 4:** Each node i changes its status ($d_i := \neg d_i$) if the following two conditions are satisfied.

  1. Node i intends to change its status, that is, $ChangeFlag_i = true$.
  2. Node i is pointed by each neighbor j and itself, that is, $\forall j \in N_i \cup \{i\} : Pointer_j = i$.

3.3 Example

Fig. 1 shows a possible execution of our algorithm. The distributed system consists of six nodes 0,1,$\cdots$,5. In the initial configuration (Fig. 1(a)), black nodes 2,3 and 5 represent dominators and white nodes 0,1 and 4 represent dominatees. Fig. 1(b) shows an execution of Phase 1. Since $C'_4 = \{\{3\}, \{5\}\}$, node 4 is dominated even if node 3 turns to be a dominator, and node 4 sets $Permission_3^4 = true$ (this is represented by the small black arrow). Similarly, node 4 sets $Permission_5^4 = true$. Since $C'_1 = \{\{2\}\}$, node 1 is not dominated if node 2 turns to be a dominatee, and node 1 sets $Permission_2^1 = false$. Similarly, node 0 sets $Permission_5^0 = false$. In all the other cases, $Permission_j^i = false$ (this is not used). Fig. 1(c) shows an execution of Phase 2. Since $C'_0 = \emptyset$, node 0 sets $ChangeFlag_0 = true$ (this is represented by the black flag). Since $C'_3 \neq \emptyset$ and node 3 receives permission from each neighboring dominatee (in this case, from node 4 only), node 3 also sets $ChangeFlag_3 = true$. Fig. 1(d) shows an execution of Phase 3. Since $ChangeFlag_0 = true$ and there exists no neighbor i of node 0 such that $ChangeFlag_i = true$, node 0 is pointed by nodes 1,5 and itself (this is represented by the big black arrows). Similarly, node 3 is pointed by nodes 2,4 and itself. Fig. 1(e) shows an execution of Phase 4, and nodes 0 and 3 change their statuses since they are pointed by each neighbor of them respectively.

Algorithm 1 Actions of node i in each phase.

Constants:

N_i : A set of neighbors

$C_i = \{W_1^i, W_2^i, \cdots : W_x^i \subseteq N_i\}$: A domination wish set

Macros:

$D_i = \{j \in N_i : d_j = \mathit{true}\}$: A set of neighboring dominators

$C_i' = \{W_x^i \in C_i : W_x^i \subseteq D_i\}$: A subset of C_i such that each element of W_x^i is in D_i

Shared Variables:

d_i : Node i is a dominator (resp. dominatee) if $d_i = \mathit{true}$ (resp. *false*).

$\mathit{Permission}_j^i$: This value is *true* if node i gives its neighboring dominator j ($\in D_i$) permission to become a dominatee.

$\mathit{ChangeFlag}_i$: This value is *true* if node i intends to change its status.

$\mathit{Pointer}_i$: This variable is assigned one node $j \in N_i \cup \{i\}$ such that $\mathit{ChangeFlag}_j$ is *true*. Then, node i approves j's status change.

Phase 1:

for each neighbor j of node i :
if $(d_i = \mathit{false}) \wedge (\{s \in C_i' : j \notin s\} \neq \emptyset)$ **then**
    $\mathit{Permission}_j^i := \mathit{true}$
else
    $\mathit{Permission}_j^i := \mathit{false}$
end if

Phase 2:

if $(d_i = \mathit{false}) \wedge (C_i' = \emptyset)$ **then**
    $\mathit{ChangeFlag}_i := \mathit{true}$
else if $(d_i = \mathit{true}) \wedge (C_i' \neq \emptyset) \wedge (\forall j \in N_i - D_i : \mathit{Permission}_i^j = \mathit{true})$ **then**
    $\mathit{ChangeFlag}_i := \mathit{true}$
else
    $\mathit{ChangeFlag}_i := \mathit{false}$
end if

Phase 3:

if $\{j \in N_i \cup \{i\} : \mathit{ChangeFlag}_j = \mathit{true}\} \neq \emptyset$ **then**
    $\mathit{Pointer}_i := \min(\{j \in N_i \cup \{i\} : \mathit{ChangeFlag}_j = \mathit{true}\})$
else
    $\mathit{Pointer}_i := \mathit{null}$
end if

Phase 4:

if $(\mathit{ChangeFlag}_i = \mathit{true}) \wedge (\forall j \in N_i \cup \{i\} : \mathit{Pointer}_j = i)$ **then**
    $d_i := \neg d_i$
end if

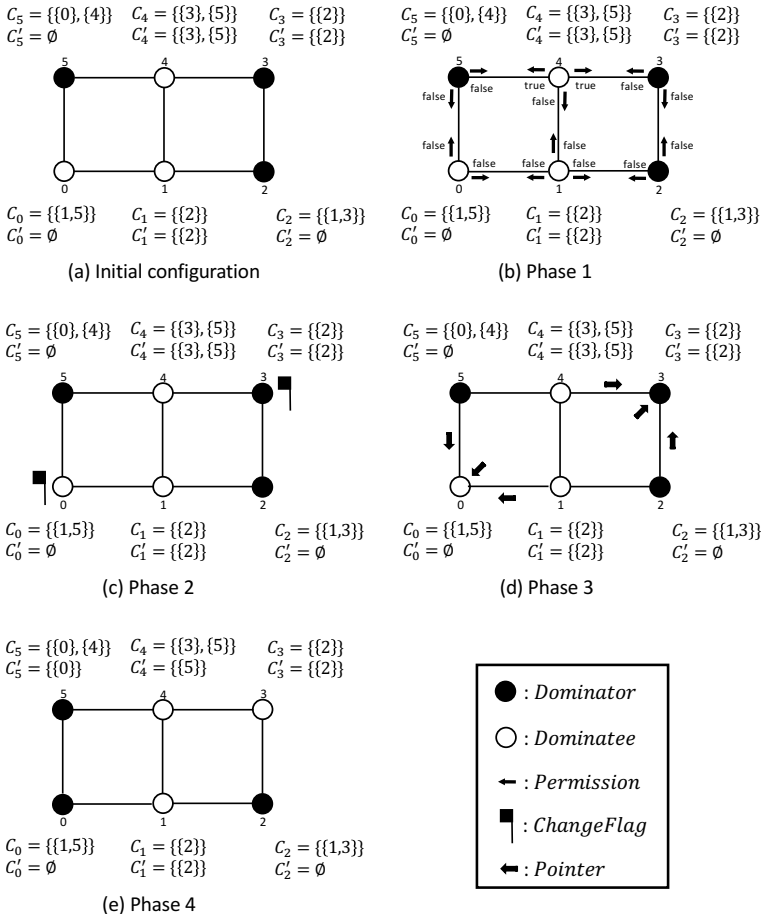


Figure 1: A possible execution of our algorithm

4 Correctness Proof and Convergence Time Evaluation

In this section, we show the proof of correctness of the proposed algorithm. Let Γ be a set of all possible configurations. First, we define a set of legitimate configurations $\Lambda \subseteq \Gamma$ of our algorithm as follows.

definition 8. $\Lambda = \{\gamma \in \Gamma : \{i \in V : d_i = true\}$ *is a minimal generalized dominating set of G=(V,E) in* γ $\}$ □

In the following correctness proof, we consider only suffix of any execution that starts from Phase 1 after the self-stabilizing phase clock algorithm converges to a legitimate configuration. This implies that all nodes executes the identical phase at each round and the value of variable $Permission_j^i$ used in Phases 2 is the one set in Phase 1. The similar property holds for all the other variables. We define the two predicates A_i and B_i for each node i that are used in Phase 2 as follows.

$A_i \equiv (d_i = false) \wedge (C_i' = \emptyset)$

$B_i \equiv (d_i = true) \wedge (C_i' \neq \emptyset) \wedge (\forall j \in N_i - D_i : Permission_i^j = true)$

Then, we show the proof of the closure property of our algorithm.

lemma 1. *If both of the predicates* A_i *and* B_i *are false at every node* i *in* γ*, both of them remain so in any* γ*' such that* $\gamma \mapsto \gamma$*'.*

proof. *We prove the lemma by contradiction. Suppose there exists some node* i *such that either of its predicates* A_i *and* B_i *turns to be true in* γ'*.*

First, we assume that the predicate A_i *turns to be true in* γ'*, that is,* $(d_i = false) \wedge (C_i' = \emptyset)$ *is satisfied in* γ*'. Because both of the predicates* A_i *and* B_i *of all nodes are false in* γ*,* $C_i' \neq \emptyset$ *holds in* γ*. So, at least one neighboring dominator* j *must turn to be a dominatee so that* $C_i' = \emptyset$ *is satisfied in* γ*', by executing Phase 4. This implies that* $ChangeFlag_j$ *is true in* γ*, that is, the predicate* B_j *of node* j *is true in* γ*. This contradicts with the assumption.*

The similar proof is applied to the predicate B_i*.* □

lemma 2. *Both of the predicates* A_i *and* B_i *are false at every node* i *in* γ *if and only if* $\gamma \in \Lambda$*.*

proof. *First, we show that if the both of predicates* A_i *and* B_i *of each node* i *are false in* γ*, then* $\gamma \in \Lambda$*. Let* i *be any dominatee in* γ*. Since the predicate* A_i *is false, node* i *is dominated. Thus,* γ *is a configuration with a generalized dominating set. Let* i *be any dominator in* γ*. Since the predicate* B_i *is false, either or both of (1)* $C_i' = \emptyset$ *and (2)* $\exists j \in N_i - D_i : Permission_i^j =$

false are satisfied. Suppose that we change the value of d_i to false. Then, node i or at least one neighboring dominatee $j(\in N_i - D_i)$ comes not to be dominated in the resulting configuration. This configuration has no longer a generalized dominating set. Thus, γ is a configuration with a minimal generalized dominating set.

It is clear by definition of Λ that if $\gamma \in \Lambda$, then both of the predicates A_i and B_i of each node i are false in γ. □

The following Theorem 1 is obtained from Lemmas 1 and 2.

theorem 1. *For any configuration $\gamma \in \Lambda$, any configuration γ' that follows γ is also in Λ* (closure property).

Finally, we show the proof of the convergence property of our algorithm.

lemma 3. *For any node i, $C'_i \neq \emptyset$ always holds after node i changes its status from a dominator to a dominatee (or after d_i turns to be zero from one).*

proof. *We prove the lemma by contradiction. Suppose that for a dominator i, C'_i is empty after node i turns to be a dominatee. Since dominator i sets $ChangeFlag_i = true$ when $C'_i \neq \emptyset$ (Phase 2), $C'_i \neq \emptyset$ holds when it becomes a dominatee (Phase 4). Also $Pointer_i = i$ holds when it becomes a dominatee, which disallows any neighbor of node i to change its status at the same round. Thus, without loss of generality, we can assume that C'_i becomes empty after node i becomes a dominatee due to the status change of a neighboring dominator j from a dominator to a dominatee. This implies that $Permission^i_j$ is true when node j changes its status, and this means node i is still dominated even if node j turns to be a dominator. Another condition $Pointer_i = j$ for allowing node j to change its status guarantees that any neighboring dominator other than node j cannot change its status at the same time, that is, C'_i does not become empty. This contradicts with the assumption.* □

The following Lemma 4 is obtained from Lemma 3.

lemma 4. *For any node i, both of the predicates A_i and B_i are always false after node i changes its status from a dominator to a dominatee (or after d_i turns to be zero from one).*

lemma 5. *Let γ be an illegitimate configuration (i.e., $\gamma \notin \Lambda$) at the beginning of Phase 1. Then, at least one node change its status at the next Phase 4.*

proof. *We prove the lemma by contradiction. Suppose that no node changes its status at the Phase 4. It is clear that the node with the smallest ID in $\{j \in V : ChageFlag_j = true\}$ changes its status at Phase 4 if exists. Thus,*

by assumption, for each node i, $ChangeFlag_i$ is false at Phase 2 following the Phase 1 starting at γ, and this implies both of the predicates A_i and B_i are false. By Lemma 2, $\gamma \in \Lambda$ holds. This contradicts with the assumption. □

lemma 6. *For any node i, both of the two predicates A_i and B_i turn to be false in $O(n)$ rounds.*

proof. *By Lemma 4, if any node i changes its status at most twice, both of the predicates A_i and B_i turn to be false. By Lemma 5, at least one node changes its status every four phases before reaching the legitimate configuration. So, both of the predicates A_i and B_i of all nodes turn to be false within $8n$ rounds.* □

From Lemmas 2 and 6, the following theorem holds.

theorem 2. *The algorithm converges to a legitimate configuration in $O(n)$ rounds once the self-stabilizing phase-clock synchronization algorithm converges to its legitimate configuration (convergence property).*

By the Theorems 1 and 2, we obtain Theorem 3.

theorem 3. *The algorithm is a self-stabilizing algorithm for the minimal generalized dominating set problem with $O(n)$ convergence time under the synchronous daemon.*

5 Conclusion

In this paper, we presented the new generalization of a dominating set which generalizes the classical dominating set and the classical k-redundant dominating set. The generalized dominating set is a proper generalization since more general domination can be realized than the classical dominating sets. For example, each node can designate the nodes it wants to be dominated by them, which is impossible for the classical dominating set and the k-redundant dominating set. In addition, we proposed a self-stabilizing algorithm for finding a minimal generalized dominating set under the synchronous daemon. The convergence time of our algorithm is $O(n)$, where n is the number of nodes. The algorithm works even under the distributed daemon when we apply the self-stabilizing techniques for simulating the synchronous daemon under the distributed daemon.

References

[1] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM 17(11), pp.643–644*, 1974.

[2] G. Wang, H. Wang, X. Tao, J. Zhang. A self-stabilizing algorithm for finding a minimal k-dominating set in general networks. *In Proceedings of the International Conference on Data and Knowledge Engineering, pp.74–85*, 2012.

[3] G. Wang, H. Wang, X. Tao, J. Zhang and J. Zhang. Minimising k-dominating set in arbitrary network graphs. *In Proceedings of the 9th International Conference on Advanced Data Mining and Applications, pp.120–132*, 2013.

[4] J. F. Fink and M. S. Jacobson. N-domination in graphs. *John Wiley and Sons*, 1985.

[5] N. Guellati and H. Kheddouci. A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. *Journal of Parallel and Distributed Computing 70(4), pp.406–415*, 2010.

[6] S. Kamei and H. Kakugawa. A self-stabilizing algorithm for the distributed minimal k-redundant dominating set problem in tree network. *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies(PDCAT), pp.720–724*, 2003.

[7] S. Kamei and H. Kakugawa. A self-stabilizing approximation algorithm for the distributed minimum k-domination. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E88-A (5), pp.1109–1116*, 2005.

[8] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs and P. K. Srimani. Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. *Computers & Mathematics with Applications, pp.805–811*, 2003.

[9] T. C. Huang, C. Y. Chen and C. P. Wang. A linear-time self-stabilizing algorithm for the minimal 2-dominating set problem in general networks. *Journal of Information Science and Engineering 24(1), pp.175–187*, 2008.

[10] T. C. Huang, J. C. Lin, C. Y. Chen and C. P. Wang. A self-stabilizing algorithm for finding a minimal 2-dominating set assuming the distributed demon model. *Computers and Mathematics with Applications 54(3), pp.350–356*, 2007.

[11] T. Herman and S. Ghosh. Stabilizing phase-clocks. *Information Proceeding Letter 5(6), pp.259–265*, 1995.

[12] V. Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Information Processing Letters 103(3), pp.88–93*, 2007.

[13] W. Goddard, S.T. Hedetniemi, D.P. Jacobs, P.K. Srimani and Z. Xu. Self-stabilizing graph protocols. *Parallel Processing Letters 18(1), pp.189–199*, 2008.

[14] W. Y. Chiu, C. Chen and S. Y. Tsai. A $4n$-move self-stabilizing algorithm for the minimal dominating set problem using an unfair distributed daemon. *Information Proceeding Letter 114(5), pp.515–518*, 2014.

[15] Z. Xu, S. T. Hedetniemi, W. Goddard and P. K. Srimani. A synchronous selfstabilizing minimal domination protocol in an arbitrary network graph. *Proceedings of the Fifth International Workshop on Distributed Computing, pp.26–32*, 2003.

極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズムについて

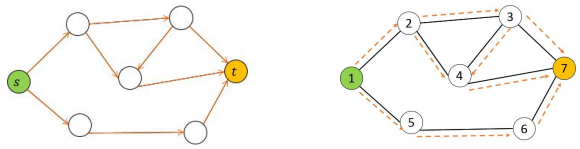
A Study on a Self-Stabilizing Algorithm for Constructing a weak-connected (2,2)-Directed Acyclic Mixed Graph

青野宏紀 [1]　金鎔煥 [1]　片山喜章 [1]
Hiroki Aono　Yonghwan Kim　Yoshiaki Katayama

名古屋工業大学大学院 [1]
Graduate School of Engineering, Nagoya Institute of Technology

1　まえがき

本研究では，任意の無向グラフ上に極大弱連結 (2,2)-DAMG を構成する自己安定アルゴリズムを提案する．自己安定アルゴリズムとは，任意の初期ネットワーク状況から実行を開始しても，目的のシステム状況に到達できる分散アルゴリズムであり，一時故障に対する耐故障性を持ち，トポロジの変化に対応できるという特徴がある．DAG(Directed Acyclic Graph) とは，閉路の無い有向グラフのことで，内向辺を持たないノードをソース，外向辺を持たないノードをシンクという．極大 $(\mathcal{S},\mathcal{T})$-DAMG(Directed Acyclic Mixed Graph) とは，与えられた任意の連結無向グラフにおいて，あらかじめ決められた $\mathcal{S}$ 個のノードと $\mathcal{T}$ 個のノードをそれぞれソースとシンクになるように構成された有向辺と無向辺が混在するグラフで，無向辺を一つでも方向付けると，指定されていないノードがソースやシンクとなるか，有向閉路が生じる．極大弱連結 $(\mathcal{S},\mathcal{T})$-DAMG とは，各ソースが少なくとも一つのシンクに有向経路を持ち，かつ各シンクが少なくとも一つのソースからの有向経路を持つものである．関連したものとして，transport net がある．



(i)transport net　(ii)st-ordering

図 1　transport net と st-ordering

transport net とは，指定された単一のソース，シンク以外のノードがソースやシンクにならないように構成された DAG である (図 1(i))．transport net は任意の 2-連結グラフにおける極大 (1,1)-DAMG と同じである．

また，transport net と関連した問題に st-ordering(st-numbering) がある．st-ordering 問題とは，各ノードに st-順序 (st-order) を割り当てる問題であり，ソースに 1，シンクに n(n は総ノード数)，それ以外のノードには自分よりも小さい st-order を持つノードと大きい st-order を持つノードが少なくとも 1 つずつ隣接するように 2〜n-1 の st-order を割り当てる．与えた st-order に基づき辺を方向付けることで transport net を得られる (図 1(ii))．つまり，st-ordering 問題を解くことで，同時に transport net を構成することができる．文献 [3, 4, 5] では，st-ordering 問題を解くアルゴリズムを提案しており，これらのアルゴリズムから transport net を構成することができる．

Ebert[3] は st-ordering 問題を解く逐次アルゴリズムを提案した．ソースを根とする深さ優先探索木を構築して，木辺による s から t への経路や木辺以外の辺 (後退辺) を用いた経路からなるリストを作成し，そのリストに基づき st-ordering を行う．Aranha ら [4] は，文献 [3] の逐次アルゴリズムのアプローチを基に，st-ordering 問題を解く分散アルゴリズムを提案した．Chaudhuri ら [5] は，文献 [3] のアプローチに基づいて，st-ordering 問題を解く自己安定アルゴリズムを提案した．ソースを根とする深さ優先探索木を構築し，木辺を用いたソースからシンクへの初期のリストを作成した後，各ノードがその初期経路リストに隣接ノードの情報を基に新しい経路の情報を追加することで作成した各ノードごとの部分経路リストや親子関係，子孫ノード数などの情報に基づいて st-ordering を行う．Karaata ら [6] は，ソース，シンクそれぞれを根とする 2 つの幅優先探索木を構築し，各ノードのソースおよびシンクからのそれぞれの距離を用いて，transport net を構成する自己安定アルゴリズムを提案した．ソースからシンクへ向かうような方向付けを行った後，指定されたシンク以外に経路の終点となるノードに接続する内向辺のうち，反転させても閉路を形成しない有向辺を 1 つ反転させることで transport net を構成する．transport net は 2-連結グラフを仮定しており，これらのアルゴリズムで transport net が構成可能なのは 2-連結グラフである．また，大野ら [7] は，st-ordering 問題を解く自己安定アルゴリズムを利用して，任意の連結グラフ上に transport net と同様に指定されたソース，シンク以外がソースやシンクにならない DAG である極大 (1,1)-DAMG を構成する自己安定アルゴリズム，さらにシンクを 1 つ増やした極大 (1,2)-DAMG を構成する自己安定アルゴリズムを提案した．本研究では，任意の連結グラフ上に極大弱連結 (2,2)-DAMG を構成する自己安定アルゴリズムを提案する．

2　モデルと定義

2.1　無向グラフ

無向グラフ G は 2 項組 $G=(V, E)$ で定義される．V は空でないノードの集合であり，E はノード間に存在する無向辺の集合である．2 つのノード u，$v\in V$ 間に無向辺が存在するとき，u と v は隣接するといい，$(u, v)\in E$ と表す．

Gの相異なるノードからなる系列$< v_0, v_1, \cdots v_m >$が各 $i(0 \leq i \leq m-1)$ に対して $(v_i, v_{i+1}) \in E$ を満たすとき，この系列を経路 $v_0 - v_m$ という．$< v_0, v_1, \cdots v_m >$と辺 (v_m, v_0) が存在するとき，系列$< v_0, \cdots v_m, v_0 >$を閉路という．Gの任意の2頂点 u，vに，経路 $u-v$ が存在するとき，Gは連結であるといい，あるノード u，vに対して経路 $u-v$ が存在しないとき，Gは非連結であるという．

定義 1. (**関節点**)　連結なグラフ $G=(V, E)$ において，取り除くと Gが非連結となるようなノード $v \in V$を関節点という．

閉路を持たないかつ連結であるグラフを木という．グラフ Gを木としたとき，次数1のノードを Gの葉という．

2.2　深さ優先探索木

グラフを可能な限り深く探索することを深さ優先探索という．深さ優先探索では，探索の始点 vに隣接するノードのうち，未訪問であるノード uに移動し uを訪問済みとして，uからまた未訪問である uの隣接ノードへの移動を繰り返す．訪問した先で，隣接に未訪問のノードが存在しなければ，未訪問の隣接ノードが存在するノード wまで探索してきたルートを引き返し，wの隣接ノードのうち未訪問であるノードへの移動を繰り返す．連結グラフを深さ優先探索することによって生成される全域木を深さ優先探索木という．

2.3　有向グラフと DAG

有向グラフ $\vec{G}$ は2項組 $\vec{G}=(V, \vec{E})$ で定義される．Vは空でないノードの集合であり，$\vec{E}$ はノード間に存在する有向辺の集合である．ノード v_i からノード v_j への有向辺を $\overrightarrow{(v_i, v_j)} \in \vec{E}$ で表し，$\overrightarrow{(v_i, v_j)} \in \vec{E}$ が存在するとき，v_i は v_j への外向辺を持ち，v_j は v_i からの内向辺を持つという．内向辺のみを持つノードをシンク，外向辺のみを持つノードをソースと呼ぶ．$\vec{G}$ の相異なるノードからなる系列$< v_0, v_1, \cdots v_m >$が各 $i(0 \leq i \leq m-1)$ に対して $\overrightarrow{(v_i, v_j)} \in \vec{E}$ を満たすとき，この系列を有向経路 $v_0 \rightarrow v_m$ という．長さが2以上の有向経路 $v_0 \rightarrow v_m$ と辺 $\overrightarrow{(v_m, v_0)}$ が存在するとき，系列$< v_0, \cdots v_m, v_0 >$を有向閉路という．

有向閉路のない有向グラフを DAG(Directed Acyclic Graph) といい，内向辺を持たないノードをソース，外向辺を持たないノードをシンクと呼ぶ．

2.4　transport net 構成問題と st-ordering 問題

相異なる2つのノード s，tを持つ連結度が2以上の無向グラフ $G=(V, E)$ において，sのみがソース，tのみがシンクとなる DAG を構成するように全ての辺を方向付ける問題を transport net 構成問題という ([3, 5])．また，相異なる2つのノード s，tを持つ連結度が2以上の無向グラフ $G=(V, E)$ において，sに 1，tに n(n はノード数)，s，t以外のノードに $u \in V$に $2 \sim n-1$ の整数 (st-order) を割り当てる問題を st-ordering 問題という．ただし，uに割り当てる整数は uより大きい整数を持つノード，uより小さい整数を持つノードが，それぞれ少なくとも1つ隣接するように割り当てなければいけない ([3]，[5])．st-order を割り当てたグラフにおいて，小さい st-order を持つノードから大きい st-order を持つノードへ向かうように辺を方向付けると，transport net が形成される．すなわち，st-ordering 問題を解くことで，同時に transport net 構成問題も解くことができる．

2.5　ネットワークとプロセス

本論文では，n個のプロセスが通信リンクで接続された任意の形状の連結ネットワーク Nを扱う．ネットワークは非同期であり，ネットワーク中に特別なプロセスを4つ持つ．これらをそれぞれソース，シンクと呼ぶ．

Nの n個のプロセスの集合を $P = \{p_1, p_2, \cdots p_n\}$ とする．N中の通信リンクの集合を $\mathcal{L}$ とする．このとき，ネットワーク Nは2項組 $N=(P, \mathcal{L})$ で定義される．

プロセス p_1, p_2 をソース，プロセス p_{n-1}, p_n をシンクとし，それぞれ相異なる識別子を持つ．また，ソース s_1, s_2，シンクを t_1, t_2 と表す．s_1, s_2，t_1, t_2 以外の各プロセス $\{p_3, \cdots, p_{n-2}\}$ は全て匿名であり，添字 $3, \cdots n-2$ は表記の容易化のためだけに用いる．$(p_i, p_j) \in \mathcal{L}(1 \subseteq i, j \subseteq n)$ のとき，p_i，p_j 間に全二重リンクが存在する．任意のプロセス p_i では任意の辺 (p_i, p_j) に対してポート番号が全順序関係を持って割り当てられており，自身につながる各ポートは区別できるとする．

プロセス間の通信は，レジスタ通信モデルを仮定する．レジスタ通信モデルでは，任意の隣接プロセス p_i，p_j の間に存在する2つの共有レジスタ R_{ij}，R_{ji} を用いて通信を行う．R_{ij} は p_i が書き込み p_j が読み込むレジスタ，R_{ji} は p_j が書き込み p_i が読み込むレジスタである．また，p_i と各隣接プロセスの間に存在するこれらのレジスタを p_i の隣接レジスタという．読み込み，書き込みはそれぞれ関数 *read* と関数 *write* によって行う．R_{ji} から p_i へ読み込む関数 *read* と，p_i から R_{ij} へ書き込む関数 *write* は次のように定義する．

定義 2. (**関数** *read*, *write*)

- $read(R_{ji})$：レジスタ R_{ji} から読み込んだデータを返す関数
- $write(R_{ij}, x)$：データ x をレジスタ R_{ij} へ書き込む手続き

2.6　スケジュール

各プロセス p_i の状態を q_i とし，N上のアルゴリズムにおけるネットワーク状況を $c=(q_0, q_1, \cdots q_{n-1})$ と表す．ただし，状態 q_i はプロセス p_i が書き込む共有レジスタすべての内容を含むものとする．また Nの取り得るすべてのネットワーク状況の集合を Cと表す．つまり，p_i の取り得るプロセス状態の集合を Q_i とすると，$C = Q_0 \times Q_1 \times \cdots \times Q_{n-1}$ である．

S を P の任意の部分集合とし，$\mathcal{A}$ を任意のアルゴリズムとする．任意のネットワーク状況を $\gamma \in C$ とし，S に属するすべてのプロセスが $\mathcal{A}$ に従って 1 原子動作を行うことによりネットワーク状況が γ から γ' になるとき $\gamma \mapsto (S, \mathcal{A})\gamma'$ と表す．

定義 3. (スケジュールと実行) 空でないプロセス集合の無限系列 $T{=}S_0, S_1, S_2\cdots$ をスケジュールという．ネットワーク状況の無限系列 $E{=}(\gamma_0,\gamma_1,\gamma_2\cdots)$ において，各 $i(\geq 0)$ に対して，γ_{i+1} が存在し，$\gamma \mapsto (S, \mathcal{A})\gamma_{i+1}(i \geq 0)$ を満たすとき，E を「初期状況 γ_0，スケジュール T に対するアルゴリズム $\mathcal{A}$ の実行」と呼ぶ．

定義 4. (公平なスケジュール) スケジュール T にすべてのプロセス $p_i \in P$ が無限回現れるとき，スケジュール T は公平であるという．

本論文では，公平なスケジュールのみを対象とし，以降は公平なスケジュールを単にスケジュールと呼ぶ．スケジュールによって選ばれたプロセスは，1 原子動作のみを行うことができる．スケジュール T が選び出すプロセス数と 1 原子動作の違いにより，いくつかのモデルが考えられる．本論文では以下に示す D デーモンを扱う．

- プロセス数 : 任意の $i(\geq 0)$ について，$|S(i)| \geq 1$
- 1 原子動作 : 全隣接レジスタから情報を読み込み，自身の内部状態を変化させ，さらに全隣接レジスタへの書き込みを行う

2.7 自己安定アルゴリズム

アルゴリズム $\mathcal{A}$ がある状況の集合 $C_{\mathcal{LE}} \subset C$ に関して，次の 2 つの条件を満たすとき「アルゴリズム $\mathcal{A}$ は $C_{\mathcal{LE}}$ に関して自己安定である」といい，$SS(\mathcal{A}, C_{\mathcal{LE}})$ と表す．また，$SS(\mathcal{A}, C_{\mathcal{LE}})$ が成立するとき，$C_{\mathcal{LE}}$ を「アルゴリズム $C_{\mathcal{A}}$ に関して正当な状況」という．ただし，$\mathcal{A}$ が明らかな場合，単に正当な状況という．

1. 到達可能性
   任意のネットワーク状況 $c{\in}C$ と任意のスケジュール T に対し，c から始まる，スケジュール T によるアルゴリズム $\mathcal{A}$ の実行により，$c \overset{*}{\mapsto} c'$ かつ $c'{\in}\ C_{\mathcal{LE}}$ となる状況 c' が存在する．
2. 閉包性
   任意の状況 $c \in C_{\mathcal{LE}}$ に対し，$c \overset{*}{\mapsto} c'$ ならば c' は $C_{\mathcal{LE}}$ に属する．

すなわち，任意の初期状態 $c \in C$ から開始して，スケジュールによるアルゴリズムの実行により有限時間内に正当な状況 $c' \in C_{\mathcal{LE}}$ に到達し，一度 c' に到達した後はずっと正当な状況であり続ける (正当な状況で安定する) とき，システムは自己安定であるとする．

2.8 公平な合成

複数の自己安定アルゴリズムを合成することを公平な合成といい，自己安定アルゴリズム $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k$ が正当な状況で安定したときの出力を $\mathcal{A}_{k+1}$ の入力として用いることができる [9]．$\mathcal{A}_{k+1}$ は $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k$ が正当な状況に到達したかどうかに関わらず実行される．全ての $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k$ が正当な状況で安定したとき，$\mathcal{A}_{k+1}$ は正当なふるまいへ収束し始める．公平な合成により，2 つの自己安定アルゴリズム $\mathcal{A}_1, \mathcal{A}_2$ を合成すると，次の条件を満たす．

- $\mathcal{A}_1$ が書き込む変数を $\mathcal{A}_2$ は読み込むが書き込まない
- $\mathcal{A}_2$ が書き込む変数を $\mathcal{A}_1$ は書き込みも読み込みもしない

プロセスは 1 原子動作で，$\mathcal{A}_1$，$\mathcal{A}_2$ それぞれの原子動作をこの順で実行する．$\mathcal{A}_1$ と $\mathcal{A}_2$ の組み合わせを合成という．公平な合成では，$\mathcal{A}_1$ が正当な状況で到達した後，$\mathcal{A}_2$ も有限時間内に正当な状況に到達する．これは 3 つ以上のアルゴリズムの公平な合成においても明らかである．

2.9 極大 ($\mathcal{S}$,$\mathcal{T}$)-DAMG 構成問題

定義 5. (極大 ($\mathcal{S}$,$\mathcal{T}$)−**DAMG**) 任意の無向グラフを $G = (V, E)$，ソースノードの集合を $S = \{s_1, s_2, s_3, \cdots\} \subset V$，シンクノードの集合を $T = \{t_1, t_2, t_3, \cdots\} \subset V$ とする．ただし，$S \cap T \neq \phi$ とし，$|S| = \mathcal{S}$，$|T| = \mathcal{T}$ とする．ただし，ソースは内向辺を持たず，*1* つ以上の外向辺を持ち，無向辺を持ってもよく，シンクは外向辺を持たず，*1* つ以上の内向辺を持ち，無向辺を持ってもよい．G において，次の条件を満たすように構成された有向辺と無向辺が混在するグラフを極大 *($\mathcal{S}$,$\mathcal{T}$)*-DAMG という．

1. 全ての $s \in S$ をソース，全ての $t \in T$ をシンクとする
2. 全ての $v \in V - S - T$ は，ソースにもシンクにもならない
3. 有向閉路を持たない
4. 方向付けられていない辺を 1 つでも方向付けると，上記の条件を満たさない

本論文では極大 ($\mathcal{S}$,$\mathcal{T}$)-DAMG に次のように条件を加え極大強連結 ($\mathcal{S}$,$\mathcal{T}$)-DAMG，極大弱連結 ($\mathcal{S}$,$\mathcal{T}$)-DAMG を定義する．

極大強連結 ($\mathcal{S}$,$\mathcal{T}$)-DAMG 全てのソースは全てのシンクへの有向経路を持つ

極大弱連結 ($\mathcal{S}$,$\mathcal{T}$)-DAMG 各ソースは 1 つ以上のシンクへの有向経路を持ち，かつ各シンクは 1 つ以上のソースからの有向経路を持つ

極大 ($\mathcal{S}$,$\mathcal{T}$)-DAMG が構成されたグラフにおいて方向付けられていない辺を 1 つでも方向付けると条件を満たさなくなるため，極大 ($\mathcal{S}$,$\mathcal{T}$)-DAMG は，有向辺の数が極大であるといえる．ただし，指定したノードすべてをソース，シンクにできない場合があることに注意する．例えば連続した 3 つのノードが S に含まれる場合，真ん中のノードに接続する辺にソースとなるように辺を方向付けると，両端のノードへの内向辺となり，両端のノードはソースとならない．T の場合も同様である．任意の無向グラフ上に極大 ($\mathcal{S}$,$\mathcal{T}$)-DAMG を構成する問

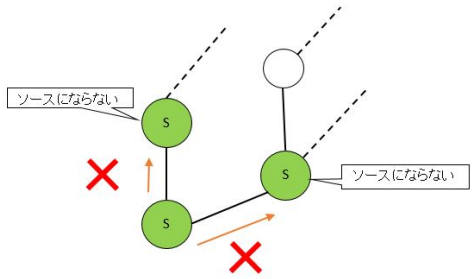


図 2　ノード全てをソースにできない場合（例）

題を，極大 ($\mathcal{S}$,$\mathcal{T}$)-DAMG 構成問題という．また，極大 ($\{s_1, s_2\}$,$\{t_1, t_2\}$)-DAMG のようにソースノード集合とシンクノード集合で表現することもある．

本論文では，2 つのソース，2 つのシンクを持ち，各ソースは 1 つ以上のシンクへの有向経路を持ち，かつ各シンクは 1 つ以上のソースからの有向経路を持つ極大弱連結 (2,2)-DAMG 構成問題を解く自己安定アルゴリズムを提案する．ただし，与えられるグラフはソース 1 つとシンク 1 つでカットセットにならないとする．

3　極大 (1,1)-DAMG 構成自己安定アルゴリズム $MDAG_{(1,1)}$

本章では，特別な 2 つのプロセス $s(\in \mathcal{S})$，$t(\in \mathcal{T})$ を有する任意の連結無向グラフ $G = (V, E)$ で極大 (1,1)-DAMG を構成する自己安定アルゴリズム $MDAG_{(1,1)}$ について述べる．

G における極大な 2-連結部分グラフを 2-連結ブロックと呼び，G 上の s から t への任意の単純経路を $s-t$，s を根とする深さ優先木上の s から t への経路を $s \overset{*}{-} t$ とする．極大 (1,1)-DAMG が構成された G を図 3 に示す．

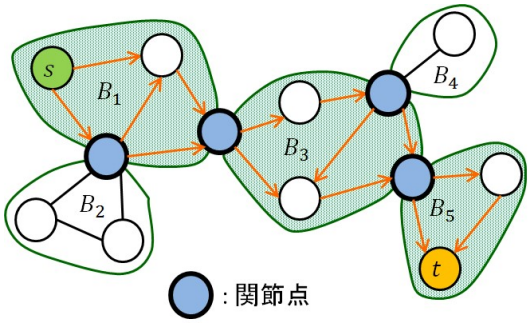


図 3　極大 (1,1)-DAMG

G について, 次のことが確認できる．まず G は, 関節点で分離すると 2-連結ブロックに分けられる．これらのうち，s を根とする深さ優先木の経路 $s \overset{*}{-} t$ の辺を含むブロックを構成する辺集合のみが有向辺となり，それ以外の辺は無向辺となる．経路 $s \overset{*}{-} t$ の辺を含まない辺集合を方向付けると，s, t 以外がソース，シンクとなったり，有向閉路が生じたりする．また，経路 $s \overset{*}{-} t$ 上に現れる辺を含む 2-連結ブロックでは，$s \overset{*}{-} t$ 上に現れる 2 つの関節点をそれぞれソース，シンクとする transport net が構成されている．これらのことから，任意の連結無向グラフ上に極大 (1,1)-DAMG を構成するための手順を以下に示す．

1. s を根とする深さ優先探索木を構築する．
2. 関節点を求め，2-連結ブロックに分割する．
3. 経路 $s \overset{*}{-} t$ を発見する．
4. 経路 $s \overset{*}{-} t$ 上の辺を含む 2-連結ブロックと，含まない 2-連結ブロックに分類する．
5. 経路 $s \overset{*}{-} t$ 上の辺を含む 2-連結ブロック内で st-ordering を行う．
6. 求めた st-order をもとに，2-連結ブロック内で transport net を構成する．

これらの手順を実現するために，複数の自己安定アルゴリズムを公平な合成によって合成する．$MDAG_{(1,1)}$ の全体の構造は図 4 のようになる．

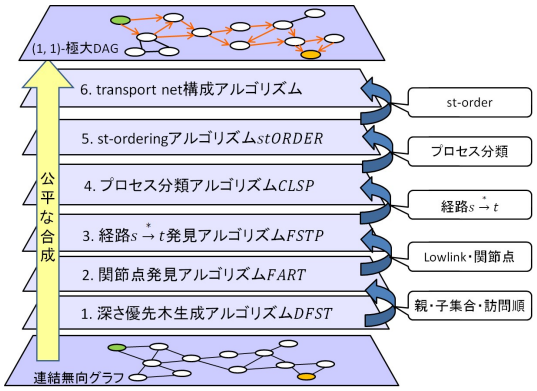


図 4　$MDAG_{(1,1)}$ における公平な合成

4　極大 (1,2)-DAMG 構成自己安定アルゴリズム

本章では，他のプロセスと区別できる特別な 3 つのプロセス $s(\in \mathcal{S})$，t_1，$t_2(\in \mathcal{T})$ を有する任意の連結無向グラフ $G = (V, E)$ で，極大 (1,2)-DAMG を構成する自己安定アルゴリズムについて述べる [7]．ただし，s, t_1, t_2 は関節点でないとする．

このアルゴリズムでは G を 2 連結成分に分割し，各 2 連結成分で極大 (1,1)-DAMG を構成するか，極大 (1,2)-DAMG を構成するか，何も構成しないかを分類し，連結ネットワーク上に極大 (1,2)-DAMG を構成している．G 上で s を根とする深さ優先木を構成する．G 上の s から t_1 への任意の単純経路を $s - t_1$, s から t_2 への任意の単純経路を $s - t_2$ とし，s を根とする深さ優先木上の s から t_1 への経路を $s \overset{*}{-} t_1$，s から t_2 への経路を $s \overset{*}{-} t_2$ とする．

図 5 に G に極大 (1,2)-DAMG が構成された例を示す．経路 $s \overset{*}{-} t_1$，経路 $s \overset{*}{-} t_2$ の辺を含む各ブロックでは，極大 (1,1)-DAMG，または極大 (1,2)-DAMG が構成されており，それ以外のブロックに含まれる辺は無向辺となる．ただし，極大 (1,2)-DAMG が構成された 2-連結ブロックにおける全ての辺が有向辺にならないことに注意する．

ここで極大 (1,1)-DAMG，極大 (1,2)-DAMG が構成されるブロックに注目する．ブロック内だけを見たときに

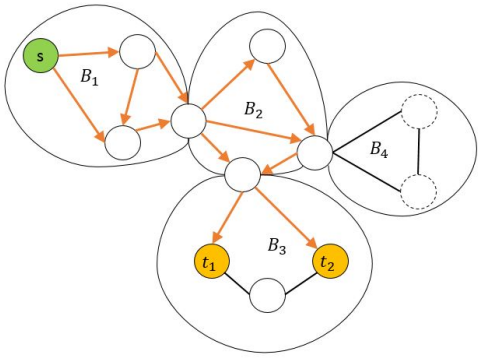


図 5　極大 (1,2)-DAMG

s，t_1，t_2 を除いてソース，シンクとなるのは極大 DAMG が構成される 2 つのブロックに接する関節点である．この関節点が各ブロックにおけるソース，シンクの機能を持つとき，その関節点を仮想ソース，または仮想シンクと呼び，s を実ソース，t_1，t_2 を実シンクと呼ぶことにする．極大 DAMG が構成される 2-連結ブロックは実または仮想ソースを 1 つ，実または仮想シンクを 1 つないし 2 つ持ち，そうでないブロックは実・仮想ソース．実・仮想いずれのシンクも持たない．実シンク，仮想シンクそれぞれの個数によってブロックを場合分けする (図 6)．

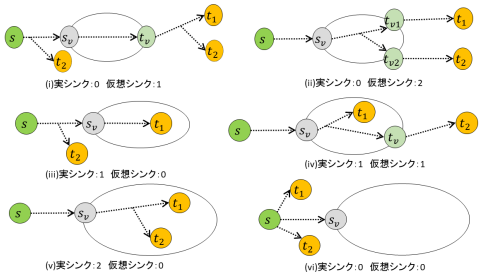


図 6　実・仮想シンクの個数による 2-連結ブロックの場合分け

実・仮想シンクを持たないブロックには有向辺が現れない．実・仮想シンクの合計が 1 であるブロックはそれをシンクとする極大 (1,1)-DAMG が構成される．また，実シンクを 2 つ持つブロックはそれらをシンクとする極大 (1,2)-DAMG が構成される．仮想シンクを 2 つ持つブロック，および実・仮想シンクを 1 つずつ持つブロックではシンクを 2 つ持つことになるが，シンクの役割を持つプロセスが他に存在する場合，仮想シンクは必ずしもシンクの役割を果たす必要はない．そのため，実・仮想シンクを 1 つずつ持つブロックでは実シンクを，仮想シンクを 2 つ持つブロックではどちらか 1 つの仮想シンクをシンクとする極大 (1,1)-DAMG を構成することができる．これらのことから任意の形状のネットワークに極大 (1,2)-DAMG を構成するための手順を以下に示す．

1. s を根とする深さ優先木を構築する．
2. 関節点を求め，2-連結ブロックに分割する．
3. 各 2-連結ブロックごとの実・仮想シンクそれぞれの個数を数え，実仮想シンクを持たないブロック，極大 (1,2)-DAMG を構成するブロック (実シンクを 2 つ持つブロック)，極大 (1,1)-DAMG を構成するブロック (それ以外のブロック) に分割する．
4. 3 の分割に基づき，各ブロックで極大 (1,2)-DAMG，極大 (1,1)-DAMG を構成する．

これらの手順を実現するために，複数の自己安定アルゴリズムを公平な合成によって合成する．手順 1 はアルゴリズム DFST[10]，手順 2 はアルゴリズム FART[8]，手順 3 はアルゴリズム FRVT[7]，手順 4 は極大 (1,1)-DAMG 構成自己安定アルゴリズム，2-連結ブロックの極大 (1,2)-DAMG 構成自己安定アルゴリズムにより実現される．アルゴリズム全体の構造は図 7 のようになる．

図 7　極大 (1,2)-DAMG 構成自己安定アルゴリズムにおける公平な合成

4.1　各ブロックの実・仮想シンク数え上げアルゴリズム $FRVT$

s を根とする深さ優先木が構成された $G = (V, E)$ を仮定し，すべての関節点が求まり，各 2-連結ブロックに分解されているとする．各ブロックの実シンク，仮想シンクの個数の組み合わせによってブロックに極大 (1,1)-DAMG，極大 (1,2)-DAMG が構成される，もしくは辺の方向付けを行わないかが決まる．つまり，実，仮想シンクそれぞれの個数がわかれば，各ブロックの分類を行うことができる．本節では各ブロックの実シンク，仮想シンクそれぞれを数え上げるアルゴリズム $FRVT$ を説明する．

アルゴリズム $FRVT$ では，深さ優先木の葉から根に向かって探索を行い，実シンク，または仮想シンクを見つければ実シンク数，または仮想シンク数を数え上げていく．また，実，仮想シンクを見つけたら，そのプロセスの訪問順も個数と一緒に根に向かって伝達していく．これを繰り返すと，ブロック内で最も訪問順が小さいプロセス P_a に，そのブロックに存在するすべての実シンク，仮想シンクの情報が集約することになる．P_a は 2 つの 2-連結ブロックに接する関節点であり，経路 $s \overset{*}{-} t_1$ または経路 $s \overset{*}{-} t_2$ 上に現れる辺を含むブロックにおいて，P_a は実・仮想ソースとなる．また，仮想シンクであるプロセスは必ず関節点であり，その子孫に実シンクが存在する．ある 2-連結ブロックにおける関節点が仮想シンクであるかどうかは，その関節点が接する別のブ

ロックに存在する子からそのブロックの実・仮想シンクの有無を調べることで分かる．関節点 P_a があるブロックの仮想ソースならば，P_a が接する別のブロックにおいて P_a は必ず仮想シンクとなり，仮想ソース，仮想シンクの両方を兼ねるため，2 つのブロックの実・仮想シンクの個数を持つことになる．アルゴリズム $FRVT$ は公平な合成により，各プロセスは親プロセス集合，子プロセス集合，訪問順，Lowlink を定数として参照するものとし，関節点の情報も持っているとする．アルゴリズム $FRVT$ の手順を以下に示す．

1. 関節点 P_a 以外のプロセス：子プロセスが持つ実・仮想シンクそれぞれの個数の合計を計算し，また子プロセスが持つ実・仮想シンクの訪問順の集合の和集合を求める．
2. もし自分が t_1 もしくは t_2 であるなら，実シンクの個数を 1 加算し，実シンクの訪問順の集合にプロセスの訪問順を加える．
3. 関節点 P_a：自身の訪問順よりも小さい LowLink をもつ P_a の子プロセスが持つ実・仮想シンクそれぞれの個数を計算し，また子プロセスが持つ実・仮想シンクの訪問順の集合の和集合を求める．
4. もし自分が t_1 もしくは t_2 であるなら，実シンクの個数を 1 つカウントし，実シンクの訪問順の集合に自分の訪問順を加える．
5. 訪問順と一致する Lowlink をもつ P_a の子プロセスがもつ実・仮想シンクの合計が 1 以上であるなら，P_a は仮想シンク，かつ P_a は仮想ソースである．
6. P_a が仮想シンクであるならば，仮想シンクの個数を 1 加算し，仮想シンクの訪問順の集合に自分の訪問順を加える．

アルゴリズム $FRVT$ を実行することによって，各関節点および s は自分が属するブロックの実・仮想シンクの情報を得ることができる．実・仮想シンクの個数の組み合わせによって，ブロックの分類が次のように定まる．

1. 実・仮想シンクを持たないブロック：極大 DAMG が構成されないブロック
2. 実シンクを 2 つもつブロック：極大 (1,2)-DAMG が構成されるブロック
3. それ以外のブロック：極大 (1,1)-DAMG が構成されるブロック

4.2 2-連結ブロックの極大 (1,2)-DAMG 構成アルゴリズム $BMDAG_{(1,2)}$

t_1, t_2 はシンクであり，外向辺を持たないため構成されたグラフにおいて有向経路 $t_1 \to t_2$ と $t_2 \to t_1$ が形成されることはない．そのため 2-連結ブロックにおける極大 (1,2)-DAMG は有向辺と無向辺が混在するグラフになる場合がある．以下に 2-連結ブロックで極大 (1,2)-DAMG を構成する手順を示す．

1. t_1 を取り除いたグラフ上で s をソース，t_2 をシンクとする極大 (1,1)-DAMG を構成する
2. t_2 を取り除いたグラフ上で s をソース，t_1 をシンクとする極大 (1,1)-DAMG を構成する
3. 手順 1，2 で構成したグラフを統合する (ただし，辺がどちらのグラフでも方向付けられている場合，手順 1 で構成されたグラフにおける方向づけを選択する)

5 極大 (2,2)-DAMG 構成自己安定アルゴリズム

本章では，特別な他のプロセスと区別できる 4 つのプロセス $s_1, s_2(\in S), t_1, t_2(\in T)$ を有する任意の連結無向グラフ $G = (V, E)$ で極大弱連結 (2,2)-DAMG を構成する自己安定アルゴリズムについて述べる．ただし，ソース 1 つとシンク 1 つの集合が G のカットセットにならないとする．すなわち，S と T のプロセスをそれぞれ 1 つずつ除去した場合でもネットワークは必ず連結である．G から s_2 とそれに接続する辺を取り除いたグラフを G_1，s_1 とそれに接続する辺を取り除いたグラフを G_2 とする．

文献 [7] で述べられている方法を用いて G_1 上で極大 (1,2)-DAMG を構成，同様に G_2 上で極大 (1,2)-DAMG を構成する．構成された 2 つの極大 (1,2)-DAMG の方向付けを利用して極大弱連結 (2,2)-DAMG を構成する．

任意形状の連結ネットワークに極大弱連結 (2,2)-DAMG を構成するための手順を以下に示す．

1. G から s_1 とそのプロセスを端点とする全ての辺を取り除いたグラフに対して，s_1 をソースとする極大 (1,2)-DAMG 構成問題を解く．
2. 同様に，G から s_2 とそのプロセスを端点とする全ての辺を取り除いた辺に対して，s_2 をソースとする極大 (1,2)-DAMG を構成する．
3. 1,2 で構成された 2 つの極大 (1,2)-DAMG の方向付けを利用して極大弱連結 (2,2)-DAMG を構成する．

これらの手順を実現するために複数の自己安定アルゴリズムを公平な合成によって合成する．手順 1,2 は極大 (1,2)-DAMG 構成自己安定アルゴリズム [7] によって実現される．手順 3 の各辺の方向付けに関しては，本章で自己安定アルゴリズムを提案する．提案アルゴリズムの全体の構造は図 8 のようになる．

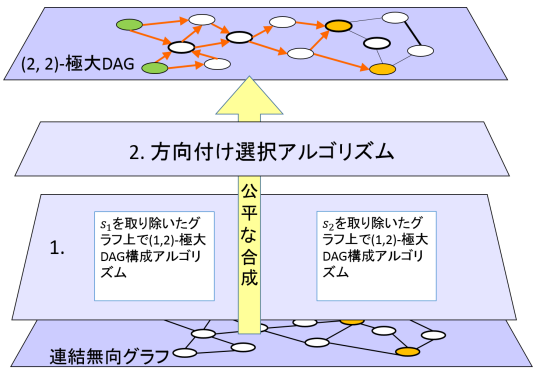


図 8 極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズムにおける公平な合成

5.1 方向付け選択アルゴリズム ASSD

s_2 とそのプロセスを端点とする全ての辺が取り除かれた G_1 上で s_1 をソース，t_1, t_2 をシンクとする極大 (1,2)-DAMG が構成される方向付けの結果，プロセス p_i が持つ方向付けを $D_i^1 = \{d_{ix}^1 | \forall p_x \in N_i\}$ とする．d_{ix}^1 は内向辺であるとき IN，外向辺であるとき OUT，無向辺であるとき NULL とする．s_1 とそのプロセスを端点とする全ての辺が取り除かれた G_2 上で s_2 をソース，t_1 , t_2 をシンクとする極大 (1,2)-DAMG が構成される方向付けの結果，プロセス p_i が持つ方向付けを D_i^2 とする．D_i^2 も D_i^1 と同様に定義する．

本節では，これらの 2 つの極大 (1,2)-DAMG の方向付けから方向付けを選択することで極大弱連結 (2,2)-DAMG を構成するアルゴリズムを提案する．

5.1.1 方向付け選択アルゴリズム ASSD の概略

方向付け選択アルゴリズム ASSD の手順を次に示す．

1. 以下のルールにしたがって各辺の方向付けを行う．
   (a) d_{ix}^1 と d_{ix}^2 が一致する場合はその方向付けを採用する．
   (b) d_{ix}^1 と d_{ix}^2 のいずれかしか方向付けられなかった場合は，方向付けられた方向を採用する．
   (c) d_{ix}^1 と d_{ix}^1 が逆向きに方向付けられた場合は，d_{ix}^1 を採用する．

5.2 極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズム

方向付け選択アルゴリズムにおいて，任意のプロセス p_i が扱う定数，変数を示す．

- p_i が書きこむ変数
  - $d_{ix} \in \{IN, OUT, NULL\}$: 極大弱連結 (2,2)-DAMG が構成される辺 (p_i,p_x) の方向付け
- p_i の定数
  - N_i : p_i の隣接プロセスの集合
  - $d_{ix}^1 \in \{IN, OUT, NULL\}$: s_1 をソース，t_1, t_2 をシンクとする極大 (1,2)-DAMG で方向付けられる辺 (p_i,p_x) の方向付け
  - $d_{ix}^2 \in \{IN, OUT, NULL\}$: s_2 をソース，t_1, t_2 をシンクとする極大 (1,2)-DAMG で方向付けられる辺 (p_i,p_x) の方向付け

d_{ix}^1，x_{ix}^2 は文献 [7] のアルゴリズムで用いた変数であり，本節のアルゴリズムではそれらの変数を参照する．これらの定数や変数を用いて，方向付けの選択アルゴリズムを実現する．Algorithm1 にアルゴリズムを示す．

以下に動作例を示す．

s_2 が取り除かれたグラフ上で s_1 をソースとする極大 (1,2)-DAMG(図 9) と s_1 が取り除かれたグラフ上で s_2 をソースとする極大 (1,2)-DAMG（図 10) が構成されている．これらによる各辺の方向付けがそれぞれ方向付け選択アルゴリズムの入力となる D_i^1 と D_i^2 となる． 方向付け選択アルゴリズムにしたがって方向付けたものが図 11 である．方向付け選択アルゴリズムによると，各辺について s_1 をソースとする極大 (1,2)-DAMG で方向付けられておらず，s_2 をソースとする極大 (1,2)-DAMG で方向付けられている時のみ s_2 をソースとする極大 (1,2)-DAMG で方向付けられたそれに従って方向付けが行われるので今回の動作例では s_2 につながる辺のみが s_2 をソースとする極大 (1,2)-DAMG で方向付けられた方向となり，それ以外の辺は s_1 をソースとする極大 (1,2)-DAMG で方向付けられた方向付けとなっている．

Algorithm 1 方向付け選択アルゴリズム ASSD

```
for each x ∈ N_i do
  if d^1_ix = NULL then
    d_ix = d^2_ix
  else
    d_ix = d^1_ix
  end if
end for
```

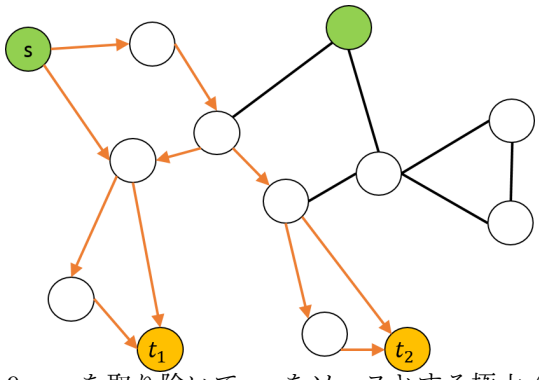


図 9 s_2 を取り除いて s_1 をソースとする極大 (1,2)-DAMG

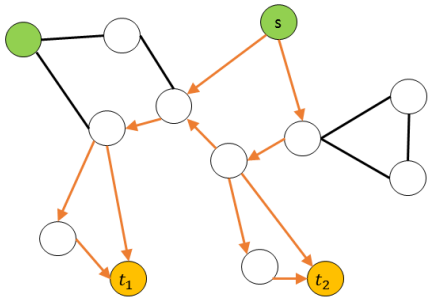


図 10 s_1 を取り除いて s_2 をソースとする極大 (1,2)-DAMG

5.3 正当性の証明

本章では，極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズムの正当性の証明を行う．証明を行う前に正当な状況を定義し，証明は正当な状況に到達することと正当な状況に一度到達したら正当な状況であり続けることを示す．

極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズム

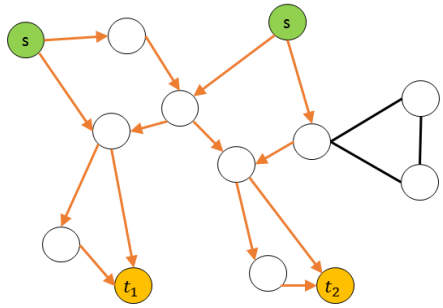


図 11　アルゴリズムを適用して構成した極大弱連結 (2,2)-DAMG

を実行したときの正当な状況を以下のように定義する．

定義 6. (極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズムにおける正当な状況)

1. 各ソースは 1 つ以上のシンクへの有向経路を持ち，かつ各シンクは 1 つ以上のソースからの有向経路を持つ
2. 極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズムによって構成されるグラフは有向閉路を持たない
3. ソースは 1 つ以上の外向辺を持ち内向辺を持たない，かつシンクは 1 つ以上の内向辺を持ち外向辺を持たない
4. s_1, s_2, t_1, t_2 以外は内向辺と外向辺を両方持つか，または無向辺のみを持つ
5. 方向付けられていない辺を 1 つでも方向付けると条件 2,3,4 のうち 1 つ以上を満たさなくなる

これらについて満たされていることを 1 つずつ証明する．

補題 1. 各ソースは 1 つ以上のシンクへの有向経路を持ち，かつ各シンクは 1 つ以上のソースからの有向経路を持つ

証明． ソース 1 つとシンク 1 つでカットセットにならないことより，ソースを 1 つ取り除いてもシンクは関節点にならないので，s_2を取り除いて s_1をソース，t_1, t_2 をシンク とする極大 (1,2)-DAMG を構成することができる．アルゴリズム ASSD では s_1 をソースとする極大 (1,2)-DAMG で方向付けられるものはそのまま方向付けられるので s_1, t_1, t_2 はソースは 1 つ以上のシンクへの有向経路を持ち，シンクは 1 つ以上のソースからの有向経路を持つ．よって s_2 が 1 つ以上のシンクへの有向経路を持つことが言えればよい．

s_1 を取り除き，s_2をソースとする 極大 (1,2)-DAMG を構成する．このときの有向経路 $s_2 - t_1$ は複数あり，そのうちのどれか 1 つについて考える．$s_2 - t_1$ 上のプロセスにおいて，s_1 をソースとする極大 (1,2)-DAMG を構成するプロセスのうち最も早く出現するプロセスを p_i とすると s_2 から p_i までの経路には s_1 をソースとする極大 (1,2)-DAMG は方向付けしない．$s_2 - t_1$ を $s_2 \to p_j \to \cdots \to t_1$ とすると，s_2 を取り除いて構成された極大 (1,2)-DAMG の有向経路 $s_1 - t_1, s_1 - t_2$ のうちどれか 1 つだけでも p_j が含まれていれば，s_1 をソースとする極大 (1,2)-DAMG による方向付けで p_jから シンクまで到達していて，アルゴリズム ASSD によって d^1 が選ばれるので，s_2 からシンクへの全ての辺が方向付けられ到達可能である．$s_1 - t_1, s_1 - t_2$ に p_j が含まれていなければ $s_2 - t_1$ で次のプロセスへと繰り返すことにより，s_1 をソースとする極大 (1,2)-DAMG で方向付けられていないプロセスのみを辿って t_1 へと到達する．このときその経路上の全ての辺に s_2 をソースとする極大 (1,2)-DAMG による方向付けが採用されるので，s_2から t_1 への有向経路を持つので s_2 から少なくとも 1 つ以上のシンクへの有向経路を持つことがいえる．

補題 2. 極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズムによって構成されるグラフは有向閉路を持たない

証明． s_1 をソースとする極大 (1,2)-DAMG での方向付けと s_2 をソースとする極大 (1,2)-DAMG での方向付けでは有向閉路はできないのでアルゴリズム ASSD により有向閉路ができるかを考える．アルゴリズム ASSD では，s_1 をソースとする極大 (1,2)-DAMG で方向付けされていない辺のみ s_2 をソースとする極大 (1,2)-DAMG で方向付けする．有向閉路ができる場合 s_1 をソースとする極大 (1,2)-DAMG で p_i から p_j への有向経路が存在し，s_2 をソースとする極大 (1,2)-DAMG で，p_iと p_j 以外のプロセスが全て異なる p_j から p_i への有向経路が存在する．p_i と p_j が同じ 2-連結ブロック内に存在するかで場合分けして考える．

- p_iと p_j が 2-連結ブロック内に存在しない場合
  2-連結ブロックの外部にまたがって有向閉路ができる場合は一回関節点を通過してから別の関節点を通って戻ってくることになるが，戻ってくることが可能であるならばこれは 2-連結ブロック内である (図 12) ので矛盾する．

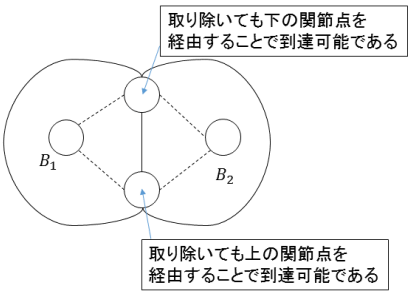


図 12　2-連結ブロックをまたがってできる有向閉路

- p_iと p_j が 2-連結ブロック内に存在する場合
  アルゴリズム ASSD では s_1 をソースとする極大 (1,2)-DAMG による方向付けはそのまま採用され，s_2 が存在しないブロックならばすべての辺が方向付けられているので，2-連結ブロックの内部で有向閉路ができる場合は 2-連結ブロック内に s_2 が存在する場合のみである．このとき，方向付けられていない

可能性がある辺は s_2 が取り除かれたことによってできた s_1 をソースとする極大 (1,2)-DAMG によって方向付けされなかった辺である．方向付けられなかった辺は s_2 を通過しなければ方向付けることができないところであるので図 13 のように，s_2 を取り除いたときに関節点となるプロセスから s_2 までの範囲となる．また，有向閉路ができる場合 s_1 をソースとす

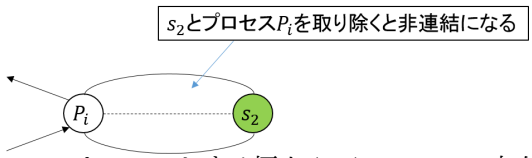


図 13　s_1 をソースとする極大 (1,2)-DAMG で方向付けされない場所

る極大 (1,2)-DAMG で構成された経路上のプロセスからそのプロセス以前の同じ経路上のプロセスへ方向付けられた場合である．方向付けられていない辺は，s_2 をソースとする極大 (1,2)-DAMG による方向付けが採用されるので s_2 に内向辺が方向付けられることはない．

よって 2連結ブロック内に有向閉路ができることはない．

補題 3. ソースは 1 つ以上の外向辺を持ち内向辺を持たない，かつシンクは 1 つ以上の内向辺を持ち外向辺を持たない

証明． s_1 をソースとする極大 (1,2)-DAMG ではソースは内向辺を持たず，シンクは外向辺を持たない．s_2 をソースとする極大 (1,2)-DAMG でもソースは内向辺を持たず，シンクは外向辺を持たない．これらからアルゴリズム ASSD により極大弱連結 (2,2)-DAMG を構成するのでソースに内向辺が方向付けられることはなく，シンクに外向辺が方向付けられることもない．よってソースは 1 つ以上の外向辺を持ち内向辺を持たない，かつシンクは 1 つ以上の内向辺を持ち外向辺を持たないので題意を満たす．

補題 4. s_1, s_2, t_1, t_2 以外は内向辺と外向辺を両方持つか，または無向辺のみを持つ

証明． s_1 をソースとする極大 (1,2)-DAMG の方向付けでは s_1, t_1, t_2 以外のプロセスは内向辺と外向辺のどちらの辺も持つのでソース，シンクにならない．同様に s_2 をソースとする極大 (1,2)-DAMG でも s_2, t_1, t_2 以外のプロセスはソース，シンクとならない．アルゴリズム ASSD では，s_1 をソースとする極大 (1,2)-DAMG で方向付けされていない辺のみ s_2 をソースとする極大 (1,2)-DAMG で方向付けする．各プロセスについて s_1 をソースとする極大 (1,2)-DAMG により方向付けられているかどうかで場合分けする．

ソース，シンクでないプロセス p_i について

1. 全ての辺が方向付けられていない場合

   s_2 をソースとする極大 (1,2)-DAMG で方向付けられるならば内向辺と外向辺を持ち，s_1 をソースとする極大 (1,2)-DAMG で方向付けられていないので内向辺と外向辺がどちらも方向付けられるので内向辺と外向辺を両方持つか，または無向辺のみを持つ．

2. それ以外の場合

   s_1 をソースとする極大 (1,2)-DAMG で方向付けられているのでソース，シンクでないプロセスであるなら内向辺と外向辺を持つ．よって内向辺と外向辺を両方持つか，または無向辺のみを持つ．

補題 5. 方向付けられていない辺を 1 つでも方向付けると条件 2,3,4 のうち 1 つ以上を満たさなくなる

証明． プロセス p_i. p_j 間の無向辺 e_{ij} とすると e_{ij} が無向辺となる場合は，全ての経路 s_1-t_1，s_1-t_2，s_2-t_1，s_2-t_2 で p_i，p_j どちらかのみが出現する場合，どちらも出現しない場合である．また，どちらも出現する場合で e_ij が無向辺となる場合は p_i, p_j がどちらもともにソースである場合，どちらもシンクである場合のみである．

1. p_i，p_j どちらかのみが出現する場合

   どちらかのみが出現する場合，出現したプロセスは関節点となる．出現したプロセスを p_i とする．p_i から p_j 方向に方向付けると p_j がシンクとなり，シンクでないようにするには p_j から p_i 以外のプロセスへ外向辺を方向付ける必要があるが方向付けた場合そのプロセスがシンクになってしまう (図 14)．これ

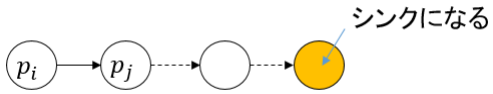


図 14　方向付けるとシンクができる

らのプロセスがシンクとならないためには p_i から方向付けられてきたプロセスのどれかに外向辺を方向付けなければならないがこのとき有向閉路ができてしまう．(図 15)

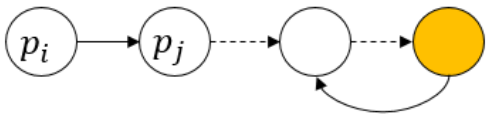


図 15　シンクでないようにすると有向閉路ができる

p_j から p_i 方向に方向付けると p_j がソースとなり，ソースでないようにするには p_j へ p_i 以外のプロセスから内向辺を方向付ける必要があるが，方向付けた場合そのプロセスがソースとなってしまう．これらのプロセスがソースとならないためには，p_i から方向付けられてきたプロセスからの内向辺を方向付けなければならないがこのとき有向閉路ができてしまう．よって p_i，p_j のどちらかのみが出現する場合，1 つでも方向付けると条件 2,4 を満たさない．

2. どちらも出現しない場合

   p_i から p_j へ方向付ける場合 p_i がソースとなり，p_j がシンクとなる．どちらかのみが出現する場合と同様にすることでソース，シンク，有向閉路のいずれかが存在するようにしか方向付けることができないので条件 2,4 を満たさない．

3. どちらも出現する場合
   - どちらもソースである場合
     ソースからソースへ方向付けると片方は内向辺を持ってしまうので条件 3 を満たさない
   - どちらもシンクである場合
     シンクからシンクへ方向付けるとっ片方は外向辺を持ってしまうので条件 3 を満たさない

よって方向付けられていない辺を 1 つでも方向付けると条件 2,3,4 を満たさなくなる.

5 つの補題より次の定理が成り立つ.

定理 1. 極大弱連結 (2,2)-DAMG 構成自己安定アルゴリズムは任意の連結ネットワーク上に極大弱連結 (2,2)-DAMG を構成する自己安定アルゴリズムである.

6 まとめと今後の課題

まとめ

- 極大強，弱連結 $(\mathcal{S},\mathcal{T})$DAMG の定義
- 極大弱連結 (2,2)-DAMG 構成アルゴリズムの提案
- 提案アルゴリズムの正当性の証明

今後の課題

- 極大強連結 (2,2)-DAMG が存在する必要十分条件
- 極大強連結 (2,2)-DAMG 構成アルゴリズムの提案
- ソース，シンクが 3 つ以上の極大 DAMG を構成するアルゴリズムの提案

参考文献

[1] 伊藤公一, 片山喜章, 和田幸一, 高橋直久, ”MANET 上の GeoCast のための DAG 構成自己安定プロトコルについて”, 電子情報通信技術研究報告, *COMP 111(195), pp. 23-30*, 2011

[2] E. W. Dijkstra, ”Self-stabilizing systems in spite of distributed control”, *Comm. ACM 17 (11), pp.103-117*, 1974.

[3] J. Ebert, ”st-ordering the vertices of biconnected graphs”, *Computing 30, pp. 19-33*, 1983.

[4] R. F. M. Aranha and C. Pandu, ”An efficient distributed algorithm for st-numbering the vertices of a biconnected graph”, *Journal of Universal Computer Science Vol. 1 (9), pp. 633-650*, 1995.

[5] P. Chaudhuri, H. Thompson, ”A self-stabilizing algorithm for st-order problem”, *The Inter- national Journal of Parallel, Emergent and Distributed Systems Vol. 23 (3), pp. 219-235*, 2008.

[6] M. H. Karaata, P. Chaudhuri, ”A Dynamic Self-Stabilizing Algorithm for Constructing a Transport Net”, *Computing 68, pp. 143-161*, 2002.

[7] 大野陽香, ”(1，1)-極大 DAG 構成自己安定アルゴリズムに関する研究”, 名古屋工業大学 修士論文, 2016

[8] 大野陽香, 片山喜章, ”深さ優先探索木によるグラフの関節点を求めるメッセージサイズ $\mathcal{O}(\log 2n)$ の自己安定アルゴリズムについて”, 平成 26 年度東海支部連合大会, 2014

[9] S. Dolev, ”Self-stabilization”, *MIT Press, Cambiridge*, 2000.

[10] 岡本圭祐, 片山喜章,”メッセージサイズ $\mathcal{O}(\log 2n)$ の深さ優先探索木生成自己安定アルゴリズムに関する研究”, 第 9 回情報科学ワークショップ,*pp. 118-128*, 2013

単位円グラフの $L(2,1)$-ラベリングのための近似アルゴリズム

*山中　寿登　　小野　廣隆
名古屋大学　　名古屋大学

1　はじめに

与えられたグラフの頂点への非負整数の割り当てをラベリングという. 特に, 任意の頂点 u, v に対して, u と v が隣接するとき $|\ell(u) - \ell(v)| \geq 2$ を, u と v の距離が 2 のとき $\ell(u) \neq \ell(v)$ を満たすような, 頂点への非負整数の割り当て ℓ を $L(2,1)$-ラベリングという. 本研究で扱う $L(2,1)$-ラベリング問題とは, 割り当て ℓ のラベル値の範囲 (つまり $\max(\ell(u)) - \min(\ell(v)) + 1$) を最小化するものである.

この問題の背景として無線の周波数割り当て問題がある. 無線の基地局 A, B, C があったとして, A と B, B と C はそれぞれ交信しているとする. また, A と C は直接的な交信はないとする. このような場合, A と B, B と C は交信するために十分離れた周波数を使用しなければならないが, B における混信を避けるため, 直接交信しない A と C も異なる周波数を使用する必要がある. 基地局を頂点とみなし, 直接交信している基地局間に辺を結ぶことで, このような状況をグラフとしてモデル化することができる. 直接交信している場合を (直接辺で結ばれていることを意味する) 距離 1, 直接的な交信はないが, 一つの基地局のみを介して交信する場合を (辺を 2 本たどることで結ばれていることを意味する) 距離 2 とみなすことで, 無線の周波数割り当て問題を $L(2,1)$-ラベリング問題として定式化することができる.

この問題は一般には NP 困難である [4]. 単位円グラフに対しても NP 完全 [3] であり, Fiala らによって 12-近似アルゴリズムが与えられている [2].

本研究では, Fiala らのアルゴリズムを解析することで近似比を 10.8 に改善し, Fiala らがパラメータとして用いたグラフの最大クリークのサイズ ω に加え, グラフの最大次数 Δ をパラメータに用いることで近似比を 10 に改善しうるアイデアを紹介する.

2　準備

2.1　単位円グラフ

2 次元平面上の頂点集合と正実数の閾値が与えられたとき, 頂点間のユークリッド距離が閾値以下ならばそのときに限り頂点が隣接であるグラフを単位円グラフという. より厳密には, 有限集合 $P \subseteq \mathbb{R}^2$ と閾値 $d \in \mathbb{R}_+$ が与えられたとき, 頂点集合 $V = P$, 辺集合 $E = \{\{u, v\} \mid u, v \in V, d_E(u, v) \leq d\}$ よりなるグラフ $G = (V, E)$ を単位円グラフという. ここで $d_E(x, y)$ は点 x, y 間のユークリッド距離である.

2.2　クリークと次数

与えられたグラフ $G = (V, E)$ に対して, 集合 $K \subseteq V$ の任意の 2 頂点が隣接しているとき, K を G のクリークとよぶ. また, グラフ G における最大クリークのサイズを ω とする.

また, 単位円グラフが与えられたとき, その最大クリークのサイズは多項式時間で得られることが Clark, Colbourn によって 1990 年に示されている [1].

単位円グラフ G の最大次数 Δ に対して, 以下の命題が成り立つ.

定理 1. 単位円グラフ G の最大クリークのサイズを ω, 最大次数を Δ としたとき, $\omega - 1 \leq \Delta \leq 6\omega - 7$ が成り立つ.

証明. 最大クリークを考えると，そのサイズが ω なのでクリーク内の頂点の次数は少なくとも $\omega - 1$ である．すなわち，$\Delta \geq \omega - 1$ が成立する．

以下では, $\Delta \leq 6\omega - 7$ を示す．点 v_0 を次数 Δ の点とし，v_0 を中心としたユークリッド距離で半径 1 の円 C について考える. 円 C の内部および円周上に存在する, v_0 を除く点は Δ 個ある. ここで円 C 内の

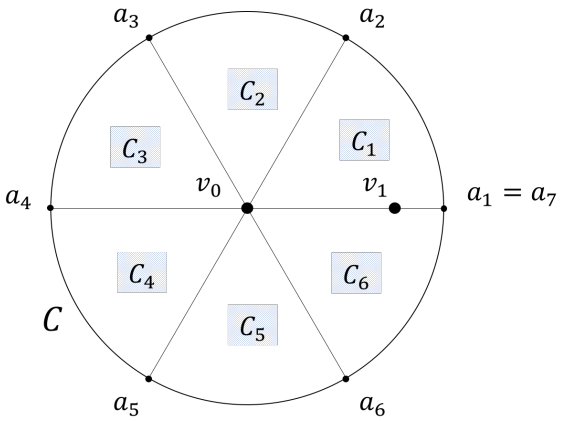


図 1　最大クリークのサイズと次数の関係について

T_{6k-5}	1~6ωの中の奇数のラベル
T_{6k-4}	6ω+1~12ωの中の奇数のラベル
T_{6k-3}	12ω+1~18ωの中の奇数のラベル
T_{6k-2}	1~6ωの中の偶数のラベル
T_{6k-1}	6ω+1~12ωの中の偶数のラベル
T_{6k}	12ω+1~18ωの中の偶数のラベル
$T_{6(k+1)-5}$	1~6ωの中の奇数のラベル

図 2　各テープに対するラベリング

ある点 v_1 に対し v_0 から半直線を引いたときの円 C との交点を a_1 とする. 図 1 のように $\angle a_i v_0 a_{i+1} = \pi/3$ $(i = 1, 2, 3, 4, 5, 6$, ただし $a_7 = a_1$ とする) となるように a_i をとる. ここで扇形 $a_i v_0 a_{i+1}$ を領域を C_i とし, 線分 $v_0 a_i$ は C_{i-1}, C_i(ただし a_1 は C_6 と C_1) の両方に含まれるものとする. また C_i に含まれる頂点同士のユークリッド距離は全て 1 以下となるので, これらはクリークをなすため, C_i には v_0 以外に高々 $\omega - 1$ 個の頂点が存在する. したがって, C_1 から C_6 にありうる v_0 を覗く頂点は高々 $6(\omega - 1) - 1$ 個となる. この最後の -1 は, v_1 が C_1, C_6 の両方に所属することによる. すなわち, $\Delta \leq 6\omega - 7$ が成立する. □

2.3　グラフラベリング

グラフラベリングを次のように定義する.

定義 1. グラフ $G = (V, E)$ の $L(p, q)$-ラベリングとは関数 $\ell : V \to \mathbb{N} \cup \{0\}$ で以下を満たすもののことを言う : 任意の頂点 u, v に対して, u と v が隣接するとき $|\ell(u) - \ell(v)| \geq p$, u と v が共通の隣接する頂点を持つとき $|\ell(u) - \ell(v)| \geq q$. また, この条件のことを $L(p, q)$ 制約と呼ぶ.

また, ラベル値の範囲を最小化するラベリングを最適なラベリングと呼び, その最小値を $\sigma_{p,q}$ で表す.

本研究では主に $p = 2$, $q = 1$ の $L(2, 1)$-ラベリングを扱うが, σ について $L(2, 1)$-ラベリングと $L(2, 2)$-ラベリングと $L(1, 1)$-ラベリングとの間には $\sigma_{2,1} \leq \sigma_{2,2} \leq 2 \cdot \sigma_{1,1}$ が成立する.

3　先行研究

本節では Fiala らによるアルゴリズムを紹介する. 単位円グラフが与えられたとき, 以下のような 12-近似アルゴリズムを与える. まず, 幅 a のテープ T_i(ただし $2/3 \leq a \leq 1/\sqrt{2}, i = 1, 2, \ldots$) に平面を x 軸と平行となるように分割する. そして, そのテープ上の単位円グラフに対してそれぞれラベリングを行い, 最後に, ラベリングされたテープを元の状態に組み合わせることで全体のラベリングを完成させる.

以下の補題 1 をもとに, 定理 2 を導く. ただし, 補題 1 に現れる $N^2[v]$ は頂点 v から距離 2 以下の頂点と v そのものを合わせた頂点の集合とする.

補題 1. ([2]) G をテープ T_i 上の単位円グラフとし, v を x 軸方向で最小の点とする. このとき, $N^2[v]$ に含まれる頂点数は高々 3ω となり, テープ T_i 上の単位円グラフに対して $L(1, 1)$-ラベリングするために必要なラベルは $\{1, 2, \ldots, 3\omega\}$ であり, $L(2, 1)$-ラベリングするために必要なラベルは $\{1, 2, \ldots, 6\omega\}$ である.

定理 2. ([2]) G を平面上の単位円グラフとし, テープ T_i に対して k を自然数として, 図 2 のようにラベリングすることで上界は 18ω となる. また, 下界は完全グラフの場合の $2\omega - 1$ となり, $\omega \geq 2$ を仮定すると近似比は 12 となる.

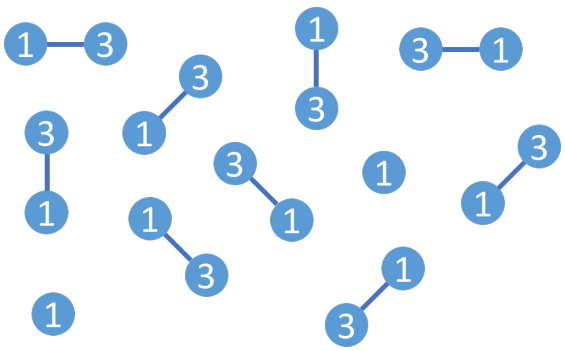


図 3　$\omega = 2$ のときのラベリング

4　近似比の改善

4.1　$\omega = 2$ のとき

Fiala らのアルゴリズムにおいて, $\omega = 2$ の場合, 下界が 3 となるのはテープ上に少なくとも 1 つ, 長さが 1 の路を含んだ状態で, 長さが 2 以上の路を含まないときである. このとき, 図 3 のようにグラフが与えられ, 最適なラベリングが可能である.

また, 上述の状態でない場合, 下界は 4 または 5 となるので, 仮に上界を $18\omega = 36$ としても近似比は最悪で 9 となり, $\omega \geq 3$ のときの近似比 10.8 より下回るので全体の近似比は 10.8 とすなる.

4.2　次数を考慮した貪欲法

貪欲法による, 単位円グラフ上のある頂点 v_0 に対するラベリングを考える. v_0 から距離が 2 以下の頂点にはすでにラベリングされており, それらの頂点は v_0 をラベリングする際の制約ができる限り被らないようにしてあるものとする. 詳細は省略するが, v_0 から距離が 1 の頂点のクリークや次数に注目することで以下のような上界が得られると予想している.

$$3\Delta + 1 + \min\{\omega\Delta + 12\omega - 13, \Delta^2 - \Delta - \omega^2 + 3\omega - 2\}.$$

また, Δ を用いたとき, 下界は $\max\{2\omega - 1, \Delta + 2\}$ である.

近似比を上界 / 下界から導く. 先行研究の式から $\omega \geq 5$ のとき近似比が 10 以下となるため $\omega = 2, 3, 4$ の場合について考えると, 定理 1 などにより, 貪欲法で近似比はそれぞれ約 2.3, 7.2, 8.8 となることを示すことができると予想している. すなわちこの予想が正しいならばアルゴリズムの近似比は 10 となる.

5　まとめと今後の展望

今回, 単位円グラフに対する $L(2,1)$-ラベリングの 12-近似アルゴリズムを紹介し, $\omega = 2$ の場合に限定して, 解析を行ったことで近似比を 10.8 まで改善することができた. 現在 4.2 節で紹介した貪欲法をベースとしたアルゴリズムにより, 近似比を 10.8 未満に改善できると予想している.

参考文献

[1] B. N. Clark, C. J. Colbourn,D. S. Johnson. *Unit disk graphs.* Discrete Mathematics, Vol. 86, pp. 165–177, 1990.

[2] J. Fiala, A. V. Fishkin, F. Fomin. *On distance constrained labeling of disk graphs.* Theoretical Computer Science, 326(1): 261–292, 2004.

[3] J. Fiala, J. Kratpchvíl, T. Kloks. *Fixed-parameter tractability of λ-colorings.* in: Proc. 25th Internat. Workshop on Graph-Theoretic Concepts in Computer Science, Ascona, Vol. 1665, Springer, Berlin, 350–363, 2002.

[4] J. R. Griggs and R. K. Yeh. *Labelling graphs with a condition at distance 2.* SIAM Journal on Discrete Mathematics, 5(4): 586–595, 1992.

単体的複体の連続変形による
分散タスクの実現可能性判定アルゴリズム

西村 進
京都大学大学院理学研究科
susumu@math.kyoto-u.ac.jp

概要

与えられた分散タスクが，非同期 READ-WRITE 共有メモリ分散システム上の無待機プロトコルとして実現できるかどうか判定するアルゴリズムを提案する．ただし，この問題は一般には決定不能問題であることが知られているので，分散タスクの一部についてその実現可能性を機械的に保証することを目標とする．

組合せトポロジーの手法を用いた分散計算の理論で確立されている非同期計算可能性定理によれば，分散タスクを単体的複体間の離散的な写像として与えた場合，これに対応する適切な連続関数が存在すればタスクを実現するプロトコルも存在する．このような連続関数を発見するため，与えられたタスクからcarrier 複体と呼ばれる単体的複体を組合せ的に構成し，carrier 複体の中でタスクの入力に相当する複体を出力に相当する複体へ段階的に変形していくことによってプロトコル実現可能性を判定する．このような複体の段階的変形は単体に対する基本的変形操作の列で離散的に表すことができる．しかしながら、その探索空間は小さなタスクであってもかなり大きくなってしまう．そこでこの探索問題を，carrier 複体中で適当な条件を満たす変形ベクトル場を発見する問題と定式化し，ハイパーグラフ及びハイパーフォレストに関する既存の組合せ的アルゴリズムを適用することによって，変形ベクトル場を効率的に発見する方法を提案する．

1　はじめに

分散計算システムのトポロジー論 [10] は，分散システムを組み合わせトポロジーの言葉を用いて離散モデル化し，トポロジー的手法を適用してプロトコル実現可能性等の分散計算に関する本質的性質を明らかにしようとするものである．この分野で特に重要な初期の結果は，Herlihy と Shavit による非同期計算可能性定理 [12] である．この定理は，与えられた分散タスク（目的とする分散計算のみたすべき入出力関係）が，非同期 READ-WRITE 共有分散メモリシステム上で無待機プロトコルを持つための必要十分条件を，トポロジーの言葉で特徴づけを行ったという点で先駆的かつ根源的な定理であり，分散タスクのプロトコル実現不可能性についてその後多くの興味深い結果 [4] を生み出す契機ともなった．また，Gafni らはこの定理中の離散的条件を，連続関数による条件に置きかえた非同期計算可能性定理を示した．[2, 20]

本研究の目的は，分散タスクが具体的に与えられたとき，そのタスクが無待機分散プロトコルを持つかどうかを判定するアルゴリズムを開発することである．残念ながら，与えられた分散タスクを実現するプロトコルが存在するかどうかはプロセス数が 3 以上のときに決定不能であることが知られている [11, 8] こともあり，この方向に関する研究はあまり行われてきていない．本研究では完璧な解にはこだわらず，実現可能なタスクの一部に関しては肯定的な判定結果を報告できるような，健全だが完全ではないアルゴリズムを与えることを目標とする．すなわち，あるタスクがこのアルゴリズムによって実現可能と判定された場合，そのタスクは実

際プロトコルを持つことを保証されるが，そうでない場合（実現可能と判定されなかった場合）[*1]はプロトコル実現の可否は確定できない．

本稿では上記のようなアルゴリズムを，幾何的直観に従って構成する．Gafni らによる非同期計算可能性定理によれば，与えられたタスクがプロトコルを持つことを示すには，タスクへの入力を出力に写す適切な連続関数を発見すればよい．このために，組合せトポロジーの手法を用いて適当な単体的複体を設定し，その中で入力に相当する単体を段階的に変形し，出力に見合うような複体を得る手続きを考える．このような複体の変形が導く連続関数によってプロトコルの存在を保証することができる．

本稿の技術的成果は以下の通りである．

- 上記のような変形の探索空間として組み合わせ的に構成される単体的複体として **carrier 複体**を導入する．carrier 複体は，分散タスクの形式的な記述である carrier 写像から導かれる半順序集合から組合せ的に自然に構成される単体的複体である．図形変形は carrier 複体内での基本変形操作の集合として実現される．これら変形操作の集合を，carrier 複体中の**変形ベクトル場**（通常のベクトル場の概念を，変形の向きに従って離散化したもの）として表す．非同期計算可能性定理の条件を満たすような連続関数を見つけるには，一定の条件をみたす変形ベクトル場をひとつ発見する必要がある．
- サイズがそれほど大きくないタスクであっても，変形ベクトル場には非常に多くの候補がある．大きな探索空間の中からなるべく効率良く適切な変形ベクトル場を発見する組合せ的アルゴリズムを提案する．このため，carrier 複体の構成要素である単体的複体をこれと等価なハイパーグラフ表現に変換し，これにハイパーグラフの縮約アルゴリズムを適用することによって，より疎な構造であるフォレストを抽出する．そして，フォレストの辺を辿って目的とする変形ベクトル場を構成する．

関連研究

Havlicek は代数幾何的手法を用いることによって，与えられたタスクの実現不可能性，すなわち対応するプロトコルが存在しないこと，を導く十分条件を与え，これを用いてタスクが実現不可能であることを判定するアルゴリズムを与えた．[9] Havlicek のアルゴリズムと本稿で提案するアルゴリズムはどちらも健全だが不完全という点では似ているが，前者と違って後者はタスクの実現可能性を判定するアルゴリズムを与えている．

ハイパーフォレストの縮約アルゴリズムは，探索空間を大幅に削減する重要な役割を担っている．このアルゴリズムは，Lovász のハイパーフォレストに関する 2 彩色可能性定理の証明の中で暗示 [18] されており，Formin, Gaspers, Saurabh, および Thomassé が [6] の中で，多項式時間アルゴリズムとして構成できることを指摘した．

1.1 本稿の構成

本稿は，以降以下のように構成される．まず 2 節で，組合せトポロジーに関する基本的事項と分散計算のトポロジー理論に関して概説する．3 節で，carrier 複体を定義し，タスクのプロトコル実現可能性判定のためのアルゴリズムを与える．この時点では，判定に必要な変形ベクトル場の選択を非決定的に行っている．4 節では，変形ベクトル場の探索を行う決定性アルゴリズムを与える．carrier 複体のハイパーグラフ表現に組合せ

[*1] 提案するアルゴリズムは任意の入力に対して必ず停止する．ただし，入力によっては計算量が膨大となり現実的な時間内で計算が終了しないこともある．

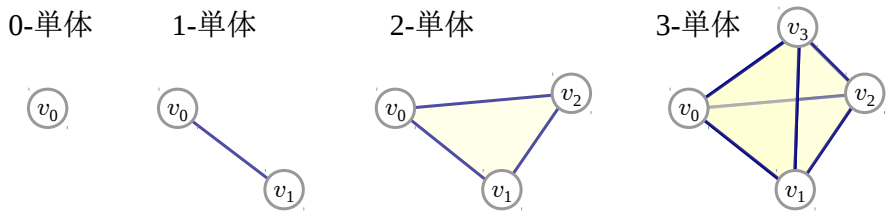


図 1: 単体と幾何的実現

的アルゴリズムを適用することで探索空間を削減できることを示す．最後に，5 節で結論を述べる．

2　分散計算のための組合せトポロジー

本節では，基礎的な組合せトポロジーとその分散計算との関連について述べる．これらの話題に関してより詳細は参考文献 [15, 10] を参照されたい．

2.1　単体的複体と細分

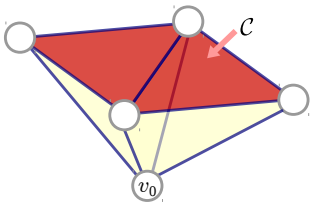
以下，V を頂点の集合とする．V の有限部分集合 σ を**単体** (simplex) という．$\#\sigma - 1$ を σ の**次元**といい，$\dim(\sigma)$ と表記する．d 次元単体 σ を **d-単体**ともいい，次元を明示して $\sigma^{(d)}$ と書く．単体 σ, τ について $\sigma \subseteq \tau$ が成り立つとき，σ は τ の**面**であるという．

単体的複体 (**複体** (complex) ともいう) $\mathcal{C}$ とは空でない単体の集合であって集合の包含関係について閉じたもの，すなわち $\sigma \in \mathcal{C}$ かつ $\emptyset \subsetneq \tau \subseteq \sigma$ ならば $\tau \in \mathcal{C}$ となるようなものをいう．複体 $\mathcal{C}$ に含まれる単体の次元のうち最大のものを $\mathcal{C}$ の次元といい，$\dim(\mathcal{C})$ と書く．次元 d の複体を **d-複体**ともいう．$V(\mathcal{C})$ で $\mathcal{C}$ に含まれる頂点の集合を表す．複体 $\mathcal{D}$ が複体 $\mathcal{C}$ の部分集合であるとき，$\mathcal{D}$ は $\mathcal{C}$ の**部分複体**であるという．単体 σ が $\mathcal{C}$ の極大面 ($\sigma \subseteq \sigma' \in \mathcal{C}$ ならば $\sigma = \sigma'$) であるとき σ は $\mathcal{C}$ の **facet** であるという．複体 $\mathcal{C}$ の facet でない単体 σ がそれを含む唯一の facet を持つとき，σ を**自由な面**と呼ぶ．複体の全ての facet が同じ次元を持つとき，その複体は **pure** であるという．本稿で扱う複体は特に言及しない限り pure である．

単体 σ について，その**閉包** $\overline{\sigma}$ を $\overline{\sigma} = \{\tau \mid \emptyset \subsetneq \tau \subseteq \sigma\}$ で定める．記法を濫用して，v と書いて 0-単体 $\{v\}$ もしくは 0-複体 $\{\{v\}\}$ を表す．

各 d-単体 $\sigma = \{v_0, v_1, \ldots, v_d\}$ $(d \geq 0)$ に対して，適当なユークリッド空間上にアフィン独立に配置された $d+1$ 個の頂点の凸包を対応させることができる．(図 1) これを σ の**幾何的実現**といい $|\sigma|$ で表す．複体 $\mathcal{C}$ の幾何的実現を $|\mathcal{C}| = \bigcup_{\sigma \in \mathcal{C}} |\sigma|$ で定める．ただし，2 つの単体が共有する面は同一の幾何的実現を持つとする．

複体 $\mathcal{C}$ と $\mathcal{D}$ が共通する単体を持たないとき，$\mathcal{C}$ と $\mathcal{D}$ の**ジョイン**を $\mathcal{C} * \mathcal{D} = \mathcal{C} \cup \mathcal{D} \cup \{\sigma \cup \tau \mid \sigma \in \mathcal{C}, \tau \in \mathcal{D}\}$ で定める．特に，$v_0 * \mathcal{C}$ は v_0 を頭頂点，$\mathcal{C}$ を底面とする $(\dim(\mathcal{C}) + 1$ 次元) の**錐** (cone) である．(右図参照．) 複体 $\mathcal{C}$ および単体 $\sigma \in \mathcal{C}$ について，**スター近傍**を $\mathrm{St}(\sigma, \mathcal{C}) = \{\tau \in \mathcal{C} \mid \sigma \cup \tau \in \mathcal{C}\}$，**リンク**を $\mathrm{Lk}(\sigma, \mathcal{C}) = \{\tau \in \mathrm{St}(\sigma, \mathcal{C}) \mid \tau \cap \sigma = \emptyset\}$ で定義する．スター近傍 $\mathrm{St}(\sigma, \mathcal{C})$ は σ を包含する facet(およびその面) 全体からなる $\mathcal{C}$ の部分複体，リンク $\mathrm{Lk}(\sigma, \mathcal{C})$ はスター近傍に属する単体のうち σ と共通部分を持たないものからなる部分複体である．特に $\sigma \in \mathcal{C}$ のとき，$\sigma * \mathrm{Lk}(\sigma, \mathcal{C}) = \mathrm{St}(\sigma, \mathcal{C})$ であることに注意せよ．



単体写像 $\mu : V(\mathcal{C}) \to V(\mathcal{D})$ とは，$\mathcal{C}$ の頂点から $\mathcal{D}$ の頂点への全域関数で
単体を単体に写す，すなわち任意の $\sigma \in \mathcal{C}$ について $\mu(\sigma) \in \mathcal{D}$ となるようなものである．

複体 $\mathcal{C}$ をさらに細かい単体で分割したものを複体の**細分**といい，$\mathrm{Div}\,\mathcal{C}$ のように表す．図 2 に代表例として 2 次元の場合の**重心細分** Bary を示す．重心細分は，各単体の重心を頂点とする単体で分割を行って得られるものである．重心細分は半順序集合と密接なつながりがあることが知られている．[1] 複体 $\mathcal{C}$ に対して，要素集合を $\mathcal{C}$ の単体全体，半順序を単体の包含関係 $\subseteq$ とする半順序集合 $F(\mathcal{C})$ が得られる．$F(\mathcal{C})$ を**面半順序** (face poset) という．逆に，半順序 $(P, \sqsubseteq)$ に対して，P の要素を頂点，長さ $d+1$ の任意の狭義単調増加列 $\{a_0 \sqsubset a_1 \sqsubset \cdots \sqsubset a_d\}$ を d-単体として持つような**順序複体** (order complex) が構成でき，これを $\Delta(P)$ で表す．面半順序 $F(\mathcal{C})$ の各要素が重心細分 $\mathrm{Bary}\,\mathcal{C}$ の頂点に一意に対応することから，複体 $\mathcal{C}$ の重心細分は順序複体 $\Delta(F(\mathcal{C}))$ によって組合せ的に表現できることがわかる．(図 2 に，2-単体 $\{0,1,2\}$ の重心細分の組合せ的対応を示した．)

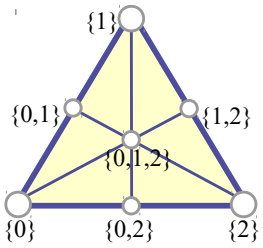


図 2: 重心細分 $\mathrm{Bary}\,\{0,1,2\}$

2.2 色付き carrier 写像による分散タスク表現

以降では，n $(n > 1)$ 個の非同期プロセスからなる READ-WRITE 共有メモリ型無待機分散システムを考える．すなわち，高々 $n-1$ 個のプロセスが故障によって停止する可能性があるとする．

本稿では，色付き分散タスク，すなわち各プロセスがプロセス固有の色 (プロセス ID) で識別される場合のみを考察する．$\Pi = \{p_0, \ldots, p_{n-1}\}$ を色の集合とする．色付きタスクにおいては，分散システム全体の状態は $(n-1)$-単体 $\{(p_1, u_1), (p_2, u_2), \ldots, (p_n, u_n)\}$ (あるいはプロセスのいくつかが故障している場合はそれらのプロセスに相当する頂点を取り除いて得られる面) として表される．ただし，単体の各頂点 (p_i, u_i) は色が p_i でそのプロセスの状態が u_i であるようなプロセスに対応する．各頂点に対して色付け関数を $\mathrm{color}((p_i, u_i)) = p_i$ で定義する．色付きタスクにおいて，全ての複体 $\mathcal{C}$ は**色付き複体**，すなわち任意の単体 $\sigma \in \mathcal{C}$ は各々異なる色からなる頂点の集合でなければならない．

分散タスクは，分散システムの入出力関係を色付き複体間の**色付き carrier 写像** $\Phi : \mathcal{I} \to 2^{\mathcal{O}}$ で形式的に定義したものである．carrier 写像 Φ は入力複体 $\mathcal{I}$ の各単体を出力複体 $\mathcal{O}$ の部分複体に写す関数であり，単調 ($\sigma \subseteq \tau$ ならば $\Phi(\sigma) \subseteq \Phi(\tau)$) かつ色を保存 (任意の $\sigma \in \mathcal{I}$ について $\mathrm{color}(\sigma) = \mathrm{color}(V(\Phi(\sigma)))$) しなければならない．色付き carrier 写像は次元を保存する，すなわち $\dim(\Phi(\sigma)) = \dim(\sigma)$ が任意の $\sigma \in \mathcal{I}$ について成り立つことに注意せよ．

2.3 非同期計算可能性定理

Herlihy と Shavit[12] によって，無待機 READ-WRITE 共有メモリモデルの計算可能性の組合せトポロジーによる特徴付けが与えられた．

定理 2.1 (非同期計算可能性定理 (Asynchronous Computability Theorem)[12])**.** 色付き分散タスク $\Phi : \mathcal{I} \to 2^{\mathcal{O}}$ が無待機 READ-WRITE 共有メモリモデルにおいてこれを実現する分散プロトコルをもつための必要十分条件は，適当な色付き細分 $\mathrm{Div}\,\mathcal{I}$ と色付き単体写像 $\mu : V(\mathrm{Div}\,\mathcal{I}) \to V(\mathcal{O})$ が存在して，$\mu(\mathrm{Div}\,(\sigma)) \subseteq \Phi(\sigma)$ が任意の $\sigma \in \mathcal{I}$ について成り立つことである．

ここで色付き細分とは，色付き複体の細分であってその細分結果もまた色付き複体であるものをいう．色付き細分で最も基本的なものは標準色付き細分 [16] である．図 3 に 2-単体の場合の標準色付き細分を示す．

Herlihy と Shavit による非同期計算可能性定理では必要十分条件は carrier 写像に対する離散的な条件として与えられるが，Gafni らはこれを連続関数に置きかえた定理が成り立つことを示した．

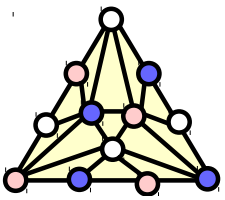


図 3: 標準色付き細分

定義 2.1. D^{k+1} を $(k+1)$ 次元円盤，S^k をその境界，すなわち k 次元球面とする．(ただし，$k \geq 0$) 複体 $\mathcal{C}$ が **k-連結**であるとは，全ての m $(0 \leq m \leq k)$ について，任意の連続関数 $f : S^m \to |\mathcal{C}|$ が連続関数 $g : D^{m+1} \to |\mathcal{C}|$ に拡張できる事をいう．

pure な n-複体 $\mathcal{C}$ は，$\mathrm{Lk}(\sigma, \mathcal{C})$ が任意の $\sigma \in \mathcal{C}$ について $(n - \dim(\sigma) - 2)$-連結である時**リンク連結**であるという．

定理 2.2 (非同期計算可能性定理, 連続版 [2, 20])**.** 色付き分散タスク $\Phi : \mathcal{I} \to 2^{\mathcal{O}}$ について，$\Phi(\sigma)$ は任意の $\sigma \in \mathcal{I}$ に対してリンク連結とする．無待機 READ-WRITE 共有メモリモデルにおいてタスク Φ を実現する分散プロトコルをもつための必要十分条件は，連続関数 $f : |\mathcal{I}| \to |\mathcal{O}|$ が存在して，$f(|\sigma|) \subseteq |\Phi(\sigma)|$ が任意の $\sigma \in \mathcal{I}$ について成り立つことである．

3　複体の変形によるプロトコル発見

分散タスクを実現するプロトコルを発見するアルゴリズムを与える．アルゴリズムは，carrier 写像から導出される単体的複体の構造を探索空間として，この中での複体の変形を発見する (非決定的) アルゴリズムとして与えられる．

3.1　Carrier 複体

分散タスクが色付き carrier 写像 $\Phi : \mathcal{I} \to 2^{\mathcal{O}}$ として与えられているとする．$\mathcal{I}$ および $\mathcal{O}$ は $(n-1)$-複体である．このcarrier 写像から自然に導かれる半順序構造から以下のような単体的複体を組合せ的に構成することができる．

定義 3.1 (carrier 複体)**.** Φ が定める半順序集合を $(\{(\sigma, \tau) \in \mathcal{I} \times \mathcal{O} \mid \tau \in \Phi(\sigma)\} \cup (\mathcal{I} \times \{\emptyset\}), \sqsubseteq)$ のように定める．ただし，半順序関係 $(\sigma, \tau) \sqsubseteq (\sigma', \tau')$ は $\sigma \subseteq \sigma'$ かつ $\tau \subseteq \tau'$ のとき成り立つものとする．

Φ の **carrier 複体**とは，上記半順序集合の要素を頂点，長さ $k+1$ の任意の狭義単調増加列 $\{(\sigma_0, \tau_0) \sqsubset (\sigma_1, \tau_1) \sqsubset \cdots \sqsubset (\sigma_k, \tau_k)\}$ $(k \geq 0)$ で $\tau_k \in \Phi(\sigma_0)$ を満たすものを k-単体として持つ複体であり，$\tilde{\Delta}(\Phi)$ と書く．

以降便宜的に，x, y, z 等で carrier 複体の頂点 (すなわち半順序集合の要素)，u, v 等で carrier 写像の入力・出力複体の頂点を表すものとする．さらに，$x = (\sigma, \tau)$ のとき，$\pi_{\mathcal{I}}(x)$ で σ を，$\pi_{\mathcal{O}}(x)$ で τ を表すものとする．

Carrier 複体 $\tilde{\Delta}(\Phi)$ は以下の性質を満たす．

- $\tilde{\Delta}(\Phi)$ は pure な n-複体で，その facet は $\dim(\sigma_n) = n - 1$ かつ $\dim(\sigma_0) = \dim(\tau_n)$ を満たす (これ以上要素を追加して長くできない) 飽和鎖 [21] $\{(\sigma_0, \emptyset) \sqsubset (\sigma_1, \tau_1) \sqsubset \cdots \sqsubset (\sigma_n, \tau_n)\}$ である．

- $\tilde{\Delta}(\Phi)$ は $\mathcal{J} = \{\{(\sigma_0, \emptyset) \sqsubset \cdots \sqsubset (\sigma_k, \emptyset)\} \mid 0 \leq k \leq n, \sigma_i \in \mathcal{I} (0 \leq i \leq k)\}$ を部分複体として含み，この $\mathcal{J}$ は $\mathcal{I}$ の重心細分 $\mathrm{Bary}\,\mathcal{I}$ と同型である．
- $\tilde{\Delta}(\Phi)$ はいくつかの $(n-1)$ 次元部分複体で分割される．すなわち，$x \in V(\mathcal{J})$ について $\tilde{\Delta}_x(\Phi) = \{\sigma \in \mathrm{St}(x, \tilde{\Delta}(\Phi)) \mid$ 任意の $y \in \sigma$ について $x \sqsubseteq y\}$ と定義するとき，$\tilde{\Delta}(\Phi) = \bigcup\{\tilde{\Delta}_x(\Phi) \mid x \in V(\mathcal{J})\}$ が成り立つ．なお，$\tilde{\Delta}_x(\Phi)$ はそれぞれ錐であり，その頭頂点は $V(\tilde{\Delta}_x(\Phi))$ の中で最小の頂点である．また，異なる x, x' に対して $\tilde{\Delta}_x(\Phi)$ と $\tilde{\Delta}_{x'}(\Phi)$ は共通の facet を持たない．

図 4 に恒等タスクの carrier 複体を示す．恒等タスクとは，入力をそのまま出力とするようなタスクであり，carrier 写像 $\Phi(\sigma) = \overline{\sigma}$ で定められる．図では特に $\mathcal{I} = \mathcal{O} = \overline{\{(p_0, 0), (p_1, 1)\}}$ のときの carrier 写像 $\Phi : \mathcal{I} \to 2^{\mathcal{O}}$ を示した．(さらに込み入ったタスクのcarrier 複体については付録 A を参照せよ．) 今後，carrier 複体の頂点 $(\{v_0, \ldots, v_l\}, \{u_0, \ldots, u_m\})$ は視覚的に読みやすいように $v_0 \cdots v_l \mid u_0 \cdots u_m$ の形で表示する．また，$\mathcal{I}$, $\mathcal{O}$ の各頂点 (p_i, w) も視覚的にわかりやすいように異なるプロセスを違う色で色分けして表示し，プロセスが持つ値 w をラベル付けして表す．(図の例ではプロセス p_0 を薄 (赤) 色で，p_1 を濃 (青) 色で色付けしている．) 図で示すように，carrier 複体は太い線で表した部分を部分複体 $\mathcal{J}$ として含んでおり，また全体は $\mathcal{J}$ の 3 つの異なる頂点をそれぞれ最小の頂点として持つ (異なる色で色づけられた) 部分複体で分割されている．

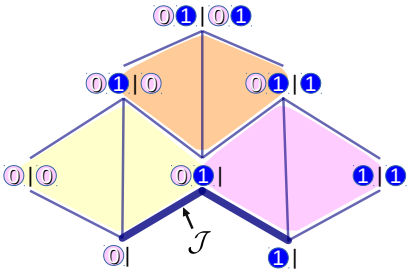


図 4: 恒等タスクのcarrier 複体

3.2 極小頂点除去による複体変形

Carrier 複体 $\tilde{\Delta}(\Phi)$ が含む部分複体 $\mathcal{J}$ の変形手続きの擬似コードを Algorithm 1 に示す．このアルゴリズムによる変形が成功したとき，分散タスクを実現するプロトコルが存在することが保証される．

Algorithm 1 複体変形による解探索の手続き

Input: $\Phi : \mathcal{I} \to \mathcal{O}$ (* タスクを定義するcarrier 写像 *)
Output: $\mathcal{K}$ (* (成功した場合) 複体の変形結果 *)

$\mathcal{J} := \{\{(\sigma_0, \emptyset) \sqsubset \cdots \sqsubset (\sigma_m, \emptyset)\} \mid \sigma_0, \ldots, \sigma_m \in \mathcal{I}, 0 \leq m \leq n\}$; $\mathcal{K} := \mathcal{J}$;
while $\mathcal{K} \cap \mathcal{J} \neq \emptyset$ **do**
  $x :=$ 部分複体 $\mathcal{K} \cap \mathcal{J}$ の頂点のうち $\mathcal{K}$ で極小な任意の頂点;
  $\mathcal{K} := \{\sigma \in \mathcal{K} \mid x \notin \sigma\} \cup \mathrm{MinvElim}(x, \mathcal{K} \cap \tilde{\Delta}_x(\Phi), \tilde{\Delta}_x(\Phi))$;
end while;
return $\mathcal{K}$

擬似コードは，以下のように，初期値となる部分複体 $\mathcal{J}$ を順次変形して最終的な変形結果を導くアルゴリズムを示しており，$\mathcal{K}$ は変形途中の複体を表す．また，変形の手続き中 $\mathcal{K}$ の次元は常に $n-1$ である．

- アルゴリズムは**極小頂点除去**の手続き MinvElim を繰り返し，$x \in V(\mathcal{K} \cap \mathcal{J})$ かつ複体 $\mathcal{K}$ で極小となるような頂点 x に対して適用する．

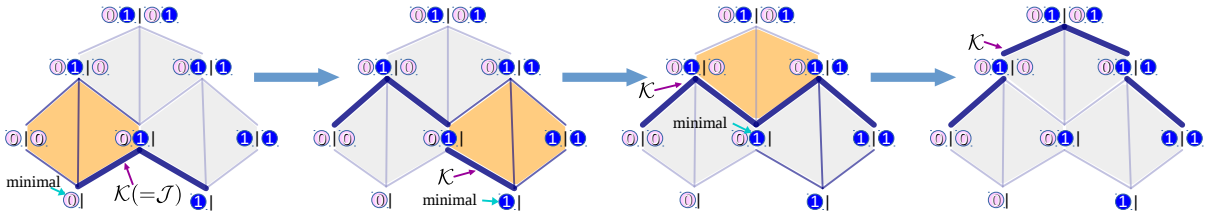


図 5: 極小頂点除去の繰り返しによる複体変形 (極小頂点除去による変形対象でない単体は灰色表示している)

- 手続き MinvElim は極小頂点 x の除去を複体の変形によって達成するが，この変形は $\Delta(\Phi)$ を分割する部分複体 $\tilde{\Delta}_x(\Phi)$ に制限して行われる．具体的には，MinvElim は部分複体 $\mathcal{K} \cap \tilde{\Delta}_x(\Phi)$ を錐の底面 $\mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ 上に変形することによって x を除去する．

極小頂点除去の手続きは，変形結果の $\mathcal{K}$ が $\mathcal{J}$ と共通部分を持たなくなるまで繰り返される．このような $\mathcal{K}$ が得られればアルゴリズムは $\mathcal{K}$ を変形結果として返す．繰り返しの途中で極小頂点を除去することができなくなった場合はアルゴリズムは失敗する．

図 5 に，複体の変形が極小頂点除去の繰り返しによってどのように達成されるかを，恒等タスクの場合を例に示す．

3.2.1 極小頂点除去アルゴリズム

Carrier 複体を分割する部分複体 $\tilde{\Delta}_x(\Phi)$ は極小元 x を頭頂点とする n 次元錐であった．この錐を構成する単体 $\sigma \in \tilde{\Delta}_x(\Phi)$ について，$\sigma \in \mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ のとき σ を**底面**とよび，$x \in \sigma$ のとき**側面**とよぶ．

極小頂点除去は，複体の変形手続きを適当な離散的に定義されたベクトル場にしたがって繰り返すことによって行われる．

定義 3.2. $\mathcal{C}$ を複体とする．単体の組 $(\alpha^{(d)}, \beta^{(d)}) \in \mathcal{C} \times \mathcal{C}$ で，適当な $\gamma^{(d+1)} \in \mathcal{C}$ について $\alpha^{(d)} \cup \beta^{(d)} = \gamma^{(d+1)}$ を満たすものを次数 d の**変形ベクトル** (deformation vector) という．($\alpha^{(d)}$ がベクトルの末尾，$\beta^{(d)}$ が先頭である．) 同じ次数 d を持つ $\mathcal{C}$ の変形ベクトルの集合 U について，任意の $(\alpha^{(d)}, \beta^{(d)}), (\alpha'^{(d)}, \beta'^{(d)}) \in U$ が $\alpha^{(d)} \neq \alpha'^{(d)}$ $\alpha^{(d)} \cup \beta^{(d)} \neq \alpha'^{(d)} \cup \beta'^{(d)}$ をみたすとき，U を $\mathcal{C}$ 上の**変形ベクトル場** (deformation vector field) とよぶ．

特に $\mathcal{C}$ が carrier 写像 $\Phi : \mathcal{I} \to 2^{\mathcal{O}}$ から導かれる carrier 複体 $\mathcal{C}$ の部分複体であるとき，$\mathcal{C}$ の変形ベクトル $(\alpha^{(d)}, \beta^{(d)})$ が $\bigcup\{\pi_{\mathcal{I}}(z) \mid z \in \alpha^{(d)}\} \supseteq \bigcup\{\pi_{\mathcal{I}}(y) \mid y \in \beta^{(d)}\}$ を満たすとき，**coherent** であるという．また，$\mathcal{C}$ 上の変形ベクトル場 U について U に属する任意の変形ベクトルが coherent であるとき，U は **coherent** であるという．

極小頂点除去のアルゴリズムは 2 種類の単体的複体に対する変形をいくつか組合せて実現する．これらの変形は幾何的にはそれぞれ，アフィン変換に対応する連続変形の離散的表現である．

簡単のため，錐 $\tilde{\Delta}_x(\Phi)$ の部分複体 $\mathcal{L}$ が変形対象であり，facet として 2-単体 $\{x, y, z\}$，変形ベクトルとして $(\{y\}, \{z\})$ を持つとしよう．ひとつ目の変形は，**側面射影** (図 6(a)) である．側面射影は錐の側面 $\{x, y\}$ を，底面と $\{y, z\}$ それ以外の側面 $\{x, z\}$ に変換する変形である．幾何的には，側面の重心点を頂点 z に写すようなアフィン変換に対応する．もうひとつの変形は**側面引き込み** (図 6(b)) である．側面引き込みは錐の側面 $\{x, y\}$ を錐の中身とともに底面と $\{y, z\}$ とそれ以外の側面 $\{x, z\}$ 上に押しつぶすように引き込み (retract)

(a) 側面射影　　(b) 側面引き込み

図 6: 側面射影 ${}_x\mathbf{P}^{\{z\}}_{\{y\}}$ と側面引き込み ${}_x\mathbf{R}^{\{z\}}_{\{y\}}$. (太線および色付きの面は変形対象の単体を，太矢印は変形ベクトルを，点線矢印は重心がアフィン変換によって写る先を示している．)

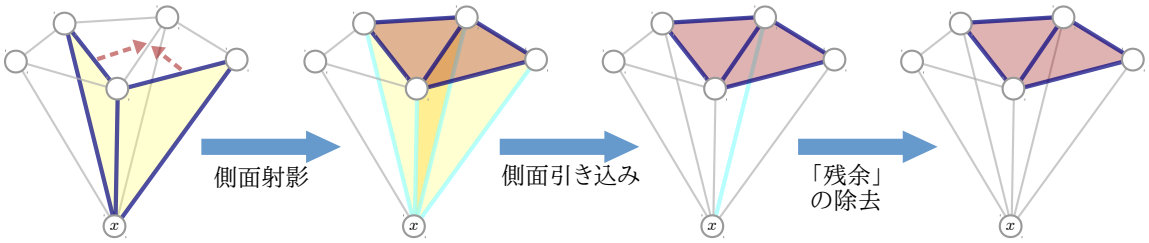


図 7: 複体変形による極小点 x の除去 ($n = 3$ の場合)

を行う変形である．幾何的には，側面の重心点および頂点 z 結ぶ線分上の点をすべて z に写すようなアフィン変換に対応する．

上記 2 種類の変形は以下のように一般的な定義を与えることができる．$\mathcal{L}$ を錐 $\tilde{\Delta}_x(\Phi)$ の部分複体，$(\alpha^{(d)}, \beta^{(d)}) \in \mathrm{Lk}(x, \tilde{\Delta}_x(\Phi)) \times \mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ を変形ベクトルとする．

- 変形ベクトル $(\alpha^{(d)}, \beta^{(d)})$ による**側面射影**は，$\mathcal{L}$ が $x * \alpha^{(d)}$ を facet として持つ $\tilde{\Delta}_x(\Phi)$ の部分複体であるとき，$\mathcal{L}$ を ${}_x\mathbf{P}^{\beta^{(d)}}_{\alpha^{(d)}}(\mathcal{L}) = \{\tau \in \mathcal{L} \cup \overline{x * (\alpha^{(d)} \cup \beta^{(d)})} \mid \tau \not\supseteq x * \alpha^{(d)}\}$ に変形する．
- 変形ベクトル $(\alpha^{(d)}, \beta^{(d)})$ による**側面引き込み**は，$\mathcal{L}$ が $x * (\alpha^{(d)} \cup \beta^{(d)})$ を facet として持ちかつ $x * \alpha^{(d)}$ が自由な面であるような $\tilde{\Delta}_x(\Phi)$ の部分複体であるとき，$\mathcal{L}$ を ${}_x\mathbf{R}^{\beta^{(d)}}_{\alpha^{(d)}}(\mathcal{L}) = \{\tau \in \mathcal{L} \cup \overline{x * (\alpha^{(d)} \cup \beta^{(d)})} \mid \tau \not\supseteq x * \alpha^{(d)}\}$ に変形する．

z を $\alpha^{(d)} \cup \{z\} = \alpha^{(d)} \cup \beta^{(d)}$ を満たす唯一の頂点としたとき，幾何的には，側面射影 ${}_x\mathbf{P}^{\beta^{(d)}}_{\alpha^{(d)}}$ は側面 $x * \alpha^{(d)}$ の重心点を z に写すようなアフィン変換により底面 $\alpha^{(d)} \cup \beta^{(d)}$ および残りの側面に射影するような変形に対応する．側面引き込み ${}_x\mathbf{R}^{\beta^{(d)}}_{\alpha^{(d)}}$ も同様に側面 $x * \alpha^{(d)}$ の重心点を z に写すようなアフィン変換に対応するが，この変形によって側面 $x * \alpha^{(d)}$ に加えて錐 $z * x * \alpha^{(d)}$ の内部が押し潰されるように錐の底面および残りの側面に引き込みが行われる．

極小頂点除去アルゴリズムを Algorithm 2 に示す．変形対象の $\mathcal{L}$ の次元は $n - 1$，これを含む carrier 複体の分割 $\tilde{\Delta}_x(\Phi)$ の次元は n である．

極小頂点除去アルゴリズムは 2 ステップの手続きからなる．

1. まず錐の底面 $\mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ 上の coherent な変形ベクトル場 U を任意に選択する．(ここでは変形ベクトル場の選択は非決定的であるが，適切な変形ベクトル場を決定的に選択する方法については

Algorithm 2 MinvElim$(x, \mathcal{L}, \tilde{\Delta}_x(\Phi))$: 極小頂点除去の手続き

Input: x (* 除去する極小頂点 *)
Input: $\mathcal{L}$ (* 変形対象となる錐側面, $(n-1)$-複体 *)
Input: $\tilde{\Delta}_x(\Phi)$ (* x でインデックスされたcarrier 複体の分割 *)
Output: $\mathcal{K}$ (* (成功した場合)x を含まない変形結果の複体 *)

$U :=$ Lk$(x, \tilde{\Delta}_x(\Phi))$上の任意の変形ベクトル場;

(* 側面射影による変形の繰り返し *)
$W := U$; $\mathcal{K} := \emptyset$;
while $W \neq \emptyset$ **do**
  $X := \{(\alpha^{(n-2)}, \beta^{(n-2)}) \in W \mid x * \alpha^{(n-2)} \in \mathcal{L}$ かつ任意の$(\alpha'^{(n-2)}, \beta'^{(n-2)}) \in W$ について
    $\alpha^{(n-2)} \subseteq \alpha'^{(n-2)} \cup \beta'^{(n-2)}$ ならば $\alpha'^{(n-2)} = \alpha^{(n-2)}$ and $\beta'^{(n-2)} = \beta^{(n-2)}\}$;
  if $X = \emptyset$ **then fail else** $(\alpha^{(n-2)}, \beta^{(n-2)}) := X$ に属する任意のベクトル **endif**;
  $\mathcal{L} := {}_x\mathbf{P}^{\beta^{(n-2)}}_{\alpha^{(n-2)}}(\mathcal{L})$;
  $W := W \setminus \{(\alpha^{(n-2)}, \beta^{(n-2)})\}$
end while;

(* 側面引き込みによる変形の繰り返し *)
for $d = n-1$ **downto** 2 **do**
  while Lk$(x, \mathcal{L})$ が $(d-1)$-単体を含む **do**
    $Y := \{(\alpha^{(d-2)}, \beta^{(d-2)}) \mid (\alpha^{(d-2)}, \beta^{(d-2)})$が coherent な変形ベクトル,
      $x * (\alpha^{(d-2)} \cup \beta^{(d-2)}) \in \mathcal{L}$, かつ $x * \alpha^{(d-2)}$ は $\mathcal{L}$ の自由な面$\}$;
    if $Y = \emptyset$ **then fail else** $(\alpha^{(d-2)}, \beta^{(d-2)}) := Y$ に属する任意のベクトル **end if**;
    $\mathcal{L} := {}_x\mathbf{R}^{\beta^{(d-2)}}_{\alpha^{(d-2)}}(\mathcal{L})$;
  end while;
end for;

if ある coherent な変形ベクトル $(\{x\}, \{z\})$ について $z =$ Lk$(x, \mathcal{L})$
then $\mathcal{K} := \{\tau \in \mathcal{L} \mid x \notin \tau\}$ **else fail**;
return $\mathcal{K}$

後に 4 節で示す．) 選択された変形ベクトル場 U からベクトルをひとつずつ取り出し，このベクトルに従った側面射影を貪欲に適用して変形していく．ただし，変形ベクトル $(\alpha^{(n-2)}, \beta^{(n-2)})$ にしたがって側面 $x * \alpha^{(n-2)}$ の射影が行えるのは，$x * \alpha^{(n-2)}$ が自由な面でありかつ U に属する他のベクトル $(\alpha'^{(n-2)}, \beta'^{(n-2)})$ による側面射影がどれも側面 $x * \alpha^{(n-2)}$ を生成しないとき，すなわち $\alpha^{(n-2)} \subseteq \alpha'^{(n-2)} \cup \beta'^{(n-2)}$ とならないときに限る．この側面射影の繰り返しにおいて適用不可能なベクトルが変形ベクトル場 U の中に残ってしまった場合，アルゴリズムは失敗する．

側面射影の繰り返しが成功したとき，その変形結果 $\mathcal{L}$ は錐の底面の部分複体 $\mathcal{L} \cap$ Lk$(x, \tilde{\Delta}_x(\Phi))$ および

側面の部分複体 $\mathrm{St}(x, \mathcal{L})$ の和となっている．このうち後者にはまだ極小頂点 x が含まれており，側面射影では除去しきれなかった「残余」部分となっている．

2. 次に上記のような「残余」複体を取り除く．$n = 2$ の場合を考えると，「残余」複体の次元は 1 であり，もしこれがひとつの 1-単体 $\{x, z\}$ のみからなり，$z \in V(\mathcal{K})$ かつ $(\{x\}, \{z\})$ が coherent な変形ベクトルである場合は単純に取り除いてよい．それ以外の場合は最小頂点除去は失敗する．幾何的には，複体の除去は**単体を潰す**操作 (辺 $|\{x, z\}|$ 全体を 1 点 z に写すアフィン変換) に対応する．
   高次元 ($n > 2$) の場合は，「残余」複体の次元をひとつずつ減らしていく．$\mathcal{L}$ が次元 dimension d $(2 \leq d \leq n-1)$ の「残余」複体を含んでいるとしよう．$\mathcal{L}$ の側面 $x * \alpha^{(d-2)}$ のうち自由なものに対して，$\mathrm{Lk}(x, \mathcal{L})$ の coherent な変形ベクトル $(\alpha^{(d-2)}, \beta^{(d-2)})$ にしたがって側面引き込みを適用する操作を，$\mathcal{L}$ から次元 d の側面がなくなるまで貪欲に繰り返す．これが成功した場合，複体の次元はひとつ減って $d-1$ となる．もし次元 d の側面が変形しきれずに残ってしまった場合は極小頂点の除去は失敗する．
   このようにして次元を順次減らしていくことによって，「残余」複体の次元を 1 とすることができ，上記の $n = 2$ の場合と同様に，単体を潰す操作によって複体の除去を行うことができる．

図 7 に，$n = 3$ の場合に極小頂点 x が除去される様子を概略図で示す．
この複体変形アルゴリズムが健全であることは，以下の定理によって示される．(証明は略す．)

定理 3.1. $\Phi : \mathcal{I} \to \mathcal{O}$ を，各 $\sigma \in \mathcal{I}$ について $\Phi(\sigma)$ がリンク連結であるような carrier 写像とする．Φ に対して複体変形アルゴリズムが成功するならば，Φ が定める分散タスクは非同期 read-write 共有メモリモデルで無待機プロトコルを持つ．

4　変形ベクトル場発見アルゴリズム

3 節で示したアルゴリズムは変形ベクトル場の選択を非決定的に行っている．非決定的選択を行う代わりに，可能なすべてのベクトル場をすべて数え上げようとすると，小さなサイズの対象であってもすぐに組合せ爆発を起こしてしまう．(たとえば，たったひとつの d-単体に対してさえも $2^d - 2$ 通りの異なる変形ベクトルが設定可能である．) さらに悪い事に，carrier 複体は往々にして多様体ではないため，標準的なグラフ・アルゴリズムを適用することもできない．

本節では，ハイパーグラフの理論で知られている技法を援用して，そう大きくないサイズの対象については，適切な変形ベクトル場を現実的な時間で特定できるような決定性アルゴリズムを提案する．

4.1　ハイパーグラフとハイパーフォレスト

ハイパーグラフ [3] $H = (V, E)$ は頂点の集合 V と**ハイパー辺**の集合 E からなり，各ハイパー辺 $e \in E$ は V の部分集合であって $\#e \geq 2$ をみたすとする．ハイパーグラフの全ての辺 $e \in E$ が $\#e = k$ をみたすとき，k **一様**であるという．(2 一様ハイパーグラフは単純グラフに相当する．) ハイパー辺 e, e' について $e \supsetneq e'$ であるとき e' は e の**縮約**であるという．また，ハイパーグラフ H' が，H のいくつかの辺をそれらを縮約したものと置きかえて得られるとき，H' は H の**縮約**であるという．

Pure な d-複体 $\mathcal{C}$ $(d \geq 1)$ は，それと等価な $(d+1)$ 一様ハイパーグラフ表現 $H = (V, E)$，ただし $V = \{\sigma^{(d-1)} \mid \sigma^{(d-1)} \in \mathcal{C}\}$ かつ $E = \{\{\sigma^{(d-1)} \in V \mid \sigma^{(d-1)} \subseteq \tau^{(d)}\} \mid \tau^{(d)} \in \mathcal{C}\}$ を持つ．つまり，各ハイ

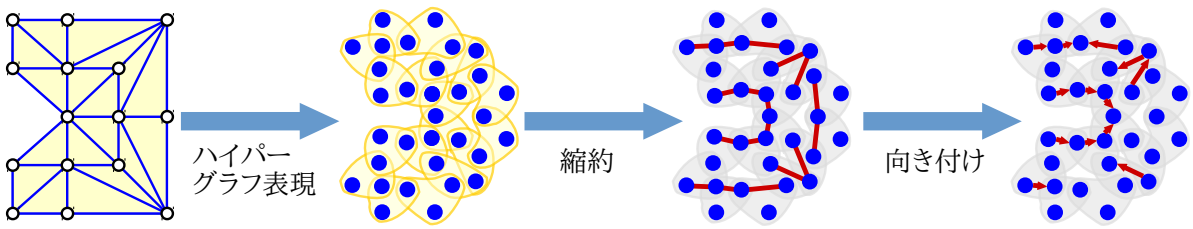


図 8: 縮約による変形ベクトル場の発見 (縮約あるいは除去されたハイパー辺は灰色で表す)

パー辺が $\mathcal{C}$ の各 facet に対応し，そのハイパー辺に含まれる $d+1$ 個の頂点が facet に含まれる $d+1$ 個の面 ($(d-1)$-単体) に対応する．特に，複体のハイパーグラフ表現に含まれる異なるハイパー辺 e と e' に対して，$\#(e \cap e') \leq 1$ が常に成り立つ．

ハイパーグラフ $F = (V, E)$ が**強 Hall 条件**をみたす，すなわち E の任意の空でない部分集合 E' に対して $\#\left(\bigcup\{e \mid e \in E'\}\right) \geq \#E' + 1$ であるとき，F は**ハイパーフォレスト**であるという．2 一様ハイパーフォレストはフォレスト（すなわち，閉路を持たない単純グラフ）である．与えられたハイパーグラフ $H = (E, V)$ がハイパーグラフかどうかは，ハイパーグラフから導かれる二部グラフに Hopcroft-Karp マッチングアルゴリズム [13] を適用することで，多項式時間 $O((\#V)^{1/2}(\#E)^2)$ で判定を行うことができる [19, 5] ことが知られている．

あるハイパーグラフの部分ハイパーグラフであるようなハイパーフォレスト全体の集合は（この集合を独立集合とする）マトロイドとなることが知られており，これを**ハイパーグラフ的マトロイド** [17, 7] という．さらに，任意のハイパーフォレスト $F = (V, E)$ をフォレスト（閉路を持たない単純グラフ）に縮約できることが Lovás[18] によって示されている．

4.2　ハイパーフォレスト縮約による変形ベクトル場の特定

いま carrier 複体に含まれる極小頂点 x を頭頂点とする錐 $\tilde{\Delta}_x(\Phi)$ について，その側面全体からなる部分複体 $\mathcal{L}$ の変形を考える．$(n-1)$-複体 $\mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ 上の変形ベクトル場の特定を以下のような手順で行う．(手続きの流れを図 8 に概略図として示す．)

H を複体 $\mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ のハイパーグラフ表現とする．このとき H は n 一様ハイパーグラフであり，その頂点は $\mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ に含まれる $(n-2)$-次元面からなる．また，$V(H)$ の部分集合として $S = \{\alpha^{(n-2)} \mid \alpha^{(n-2)} \in \mathrm{Lk}(x, \mathcal{L})\}$ および $G = \{\alpha^{(n-2)} \in \mathrm{Lk}(x, \tilde{\Delta}_x(\Phi)) \mid \pi_{\mathcal{I}}(x) \supseteq \pi_{\mathcal{I}}(z)$なる $z \in \alpha^{(n-2)}$ が存在$\}$ を定義する．S は変形の出発点となる面の集合に，G は変形が終着する面の候補の集合にそれぞれ対応する．

ハイパーグラフ表現 H を以下の手続きに従って縮約する．

1. H がハイパーフォレストでない場合，最大独立集合を求めるマトロイドの貪欲アルゴリズム [14] を適用してハイパーフォレスト F を得る．
2. ハイパーフォレスト F を前述の Lovás[18] の方法によってフォレストに縮約する．すなわち，$\#e > 2$ なるハイパー辺 e に含まれる頂点 v のうち，$v \notin S$ または $\#\{e \in F \mid v \in e\} > 1$ であってかつ，v を取り去って F を縮約しても強 Hall 条件を崩さないようなものを繰り返し貪欲に取り去って行くことによってフォレストを得る．

上記手続きの結果として，以下の性質を満たすフォレスト T を得る．

- T は複体 $\mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ を被覆する．すなわち，任意の $\gamma^{(n-1)} \in \mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ について，$\gamma^{(n-1)} = \alpha^{(n-2)} \cup \beta^{(n-2)}$ なる辺 $\{\alpha^{(n-2)}, \beta^{(n-2)}\} \in T$ が存在する．
- $S \subseteq V(T)$.

上記の縮約手続き中，条件を満たすハイパーグラフの頂点が見つからなかった場合，縮約は失敗し変形ベクトル場も与えられない．

もとのハイパーグラフ表現 H を縮約し，これよりずっと疎なフォレスト T を得ることができた．しかし T からすぐに目的とする変形ベクトル場が得られるわけではない．なぜなら，

- T は閉路を持たない無向グラフであるが，変形ベクトル場は各グラフの辺に対してそれぞれcoherentな向きが付いていなくてはならない．
- T は上述のように複体 $\mathrm{Lk}(x, \tilde{\Delta}_x(\Phi))$ を被覆しているが，変形ベクトル場もそうとは限らない．ベクトル場が余分なベクトルを含んでいると側面射影どうしが相互に依存することによって変形がうまくいかなくなることがある．

T の適切な部分グラフに向き付けをして coherent な変形ベクトル場を得るため，T をさらに次のように処理する． $\dim(\tau) = \dim(\pi_{\mathcal{I}}(x))$ を満たすような単体 $\tau \in \Phi(\pi_{\mathcal{I}}(x))$ を任意に選び，$G_\tau = \{y \in V(H) \mid \pi_{\mathcal{O}}(y) = \tau\}$ と定義する．T の辺を coherence とは逆向きに辿りながら繰り返し変形ベクトル場にベクトルを追加していく．ただし，最初変形ベクトル場は空，訪問済み頂点の集合は G_τ とする．変形ベクトル $(\alpha^{(n-2)}, \beta^{(n-2)})$ は，これがcoherent かつ次のいずれかの条件を満たすとき，新たにベクトル場に加えられる．

- z が訪問済みで y が未訪問かつ $\{y, z\} \in T$，もしくは
- z が未訪問かつ訪問済みの頂点 w について $w \in \alpha^{(n-2)} \cup \beta^{(n-2)}$,

ただしここで頂点 y と z はそれぞれ等式 $z * \alpha^{(n-2)} = y * \beta^{(n-2)} = \alpha^{(n-2)} \cup \beta^{(n-2)}$ の唯一の解とする．いずれの条件が成り立ったときも，$(n-1)$-単体 $\alpha^{(n-2)} \cup \beta^{(n-2)}$ に含まれる全ての $(n-2)$-単体が訪問済み頂点としてマークされる．ベクトル場へのベクトルの追加は，S のすべての要素が訪問済みとなるまで繰り返される．最後に，拡張されたベクトル場から変形に寄与しないベクトル（すなわち S から到達可能でないもの）を取り除いたものが最終的な結果となる．

4.3 アルゴリズムの計算量

上記の決定的アルゴリズムを用いて変形ベクトル場を特定した場合の，プロトコル発見アルゴリズムの時間計算量は以下の通りである．

定理 4.1. n プロセスの分散タスク $\Phi : \mathcal{I} \to \mathcal{O}$ に対して，N を $\mathcal{I}$ の facet の数，$M = \max\{\#\{\tau \in \Phi(\sigma) \mid \dim(\tau) = \dim(\sigma)\} \mid \sigma \in \mathcal{I}\}$, すなわち $\sigma \in \mathcal{I}$ に対する像 $\Phi(\sigma)$ に含まれる facet の最大数とする．このとき，アルゴリズムの時間計算量は高々 $O(N^{9/2} M^{7/2} \cdot n^{3/2}(2^n - 1)(n!)^{7/2})$ である．

これより，プロセス数 n が固定の場合は多項式時間であることがわかる．しかし，n を大きくすると係数が大きく増大する．

4.4 プロトタイプ実装

提案したアルゴリズムをプロトタイプ実装した．実装コード (実装言語 OCaml) は `https://github.com/CS-nishimura/cpl-deform` より入手可能である．

このプロトタイプ実装を用いて，いくつかの分散タスクに対してアルゴリズムを適用する実験を行った．まず，プロセス数 n が 3 までの場合，あまり大きくないサイズのタスク（例えば 3.1 節で示した恒等タスクや付録 A に示す擬 consensus タスク）であればすぐに結果が得られた．一方，$n = 4$ の場合は問題によって計算時間が大きく異なった．恒等タスクのようなごく小さなタスクであれば $n = 4$ であってもすぐに結果が得られたが，少し大きなタスク ($N = 1$, $M = 75$) については計算が終了しなかった．計算量の項 $M^{7/2}$ の影響と考えられる．$n \geq 5$ の場合は，タスクの大きさによらず計算が終了しなかった．

5 まとめ

与えられた分散タスクを実現する無待機分散プロトコルが実現可能であることを，非同期計算可能性定理の連続化版 [2, 20] の条件に合致する連続関数を発見することによって判定する健全なアルゴリズムを提案した．そもそもこの問題は計算不可能問題なので，提案するアルゴリズムの目的は（プロトコル実現可能な）一部のタスクについてその実現可能性を機械的に保証することである．素朴な方法で解空間全体を探索すると組合せ爆発を起こしてしまうが，連続関数の探索を，与えられたタスクから組合せ的に導かれるcarrier 複体の中で適切な変形ベクトル場を見つける問題に帰着し，さらにハイパーグラフに関する既存の組合せ的アルゴリズムで解空間を縮小することで，ある程度のサイズのタスクまでは現実的な時間で判定を行えることを示した．

謝辞

本研究は JSPS 科研費 16K00016 の助成を受けたものです．

参考文献

[1] Björner. Topological methods. In R. Graham, M. Gröstel, and L. Lovász, editors, *Handbook of Combinatorics*, pages 1819–1872. North-Holland, 1995.

[2] Elizabeth Borowsky and Eli Gafni. A simple algorithmically reasoned characterization of wait-free computations. In *Proc. of the 16th ACM Symposium on Principles of Distributed Computing*, pages 189–198, 1997.

[3] Alain Bretto. *Hyper Graph Theory: An Introduction*. Springer, 2013.

[4] Faith Fich and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2-3):121–163, 2003.

[5] Herbert Fleischner and Stefan Szeider. Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. Technical Report TR00-049, Electronic Colloquium on Computational Complexity, 2000.

[6] Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Stéphan Thomassé. A linear vertex kernel for maximum internal spanning tree. *J. Comput. Syst. Sci.*, 79(1):1–6, 2013.

[7] András Frank, Tamás Király, and Matthias Kriesell. On decomposing a hypergraph into k connected sub-hypergraphs. *Discrete Applied Mathematics*, 131(2):373–383, 2003.

[8] Eli Gafni and Elias Koutsoupias. Three-processor tasks are undecidable. *SIAM J. Comput.*, 28(3):970–983, 1999.

[9] John Havlicek. Computable obstructions to wait-free computability. *Distributed Computing*, 13(2):59–83, 2000.

[10] Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013.

[11] Maurice Herlihy and Sergio Rajsbaum. The decidability of distributed decision tasks (extended abstract). In *STOC'97: Proc. of the 29th annual ACM symposium on theory of computing*, 1997.

[12] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, 1999.

[13] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[14] Bernhad Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 4th edition, 2008.

[15] Dmitry Kozlov. *Combinatorial Algebraic Topology*. Springer, 2008.

[16] Dmitry N. Kozlov. Chromatic subdivision of a simplicial complex. *Homology, Homotopy and Applications*, 14(2):197–209, 2012.

[17] M. Lorea. Hypergraphes et matroides. *Cah. Cent. étud. Rech. Opér.*, 17:289–291, 1975.

[18] L. Lovász. A generalization of König's theorem. *Acta Math. Acad. Sci. Hungar.*, 21:443–446, 1970.

[19] M.D. Plummer and L. Lovász. *Matching Theory*, volume 29 of *North-Holland Mathematics Studies*. Elsevier Science, 1986.

[20] Vikram Saraph, Maurice Herlihy, and Eli Gafni. An algorithmic approach to the asynchronous computability theorem. arXiv:1703.08525, March 2017.

[21] Richard P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 2nd edition, 2012.

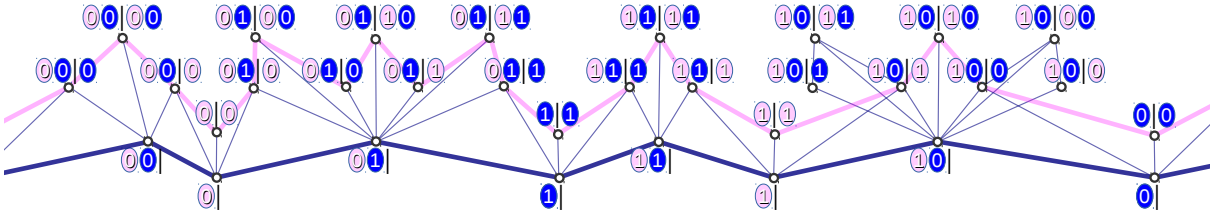


図 10: 擬 consensus タスクのcarrier 複体 (左右端で折り返す)

付録 A　擬 consensus タスクとcarrier 複体

2 つのプロセスからなる分散システムの擬 consensus タスクとは，プロセス p_0 と p_1 がそれぞれ 0 もしくは 1 どちらか任意の値を初期値として実行を開始し，初期値のどちらか一方の値に合意するか，もしくは合意するのを諦めて p_1 は 0，p_0 は 1(この逆の組合せは不可) を最終的に出力するような（色付き）タスクであり，形式的には次の carrier 写像 $\Phi : \mathcal{I} \to \mathcal{O}$ で定義される．

$$\Phi(\tau) = \begin{cases} \mathcal{O} & \text{if } \tau = \{(p_0, v_0), (p_1, v_1)\} \text{ かつ } v_0 \neq v_1, \\ \overline{\tau} & \text{それ以外} \end{cases}$$

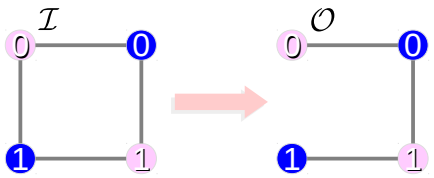


図 9: 擬 consensus タスク

ただし，タスクの入力複体は $\mathcal{I} = \bigcup\{\overline{\{(p_0, v_0), (p_1, v_1)\}} \mid v_0, v_1 \in \{0, 1\}\}$ で，出力複体は $\mathcal{O} = \bigcup\{\overline{\{(p_0, v_0), (p_1, v_1)\}} \mid (v_0, v_1) \in \{(0, 0), (1, 1), (1, 0)\}\}$ で定義される．

図 9 に示すように，擬 consensus タスクは幾何的には，四辺形の 4 つの辺を，これからひとつの辺を取り去った 3 つの辺からなる図形へ写す carrier 写像として捉えることができる．図 10 に擬 consensus タスクの（2 次元の）carrier 複体を図示する．

大域公平性を仮定した個体群プロトコルモデルにおける k 分割アルゴリズム

安見　嘉人*, 北村　直暉†, 大下　福仁‡, 泉　泰介†
* 奈良工業高等専門学校, † 名古屋工業大学 ‡ 奈良先端科学技術大学院大学

Abstract

In this paper, we consider a uniform k-partition problem in a population protocol model. The uniform k-partition probrem divides a population into k groups of the same size. For this problem, we study symmetric protocols with designated initial states under the global fairness. As a result, we propose a protocol that solves the uniform k-partition problem with $3k - 2$ states.

1 Introduction

A population protocol model [2] is an abstract model that represents computation on a network of low-performance devices. We refer to such devices as agents and a set of agents as a population. Agents can update their states by interacting with other agents, and proceed with computation by repeating the pairwise interactions. The population protocol model can be applied to many systems such as sensor networks and molecular robot networks. For example, one may construct sensor networks to monitor wild birds by attaching sensors to them. In this system, sensors collect and process data based on pairwise interactions when two sensors (or birds) come sufficiently close to each other. Another example is a system of low-performance molecular robots [11]. In this system, a large number of molecular robots compose a network inside a human body and discriminate the physical condition. To realize such systems, many protocols have been proposed as bulding blocks in the population protocol model [5]. For example, they include leader election protocols [1, 8, 10, 12, 13, 14], counting protocols [4, 6, 7], and majority protocols [3, 9].

In this paper, we focus on a uniform k-partition problem, which divides a population into k groups of the same size. The uniform k-partition problem has many applications. For example, we can reduce energy consumption by switching on some groups and switching off the others. In another example, we can assign a different task to each group and make agents execute multiple tasks at the same time. This can be regarded as differentiation of a population in the sense that initially identical agents are eventually divided into k groups and execute different tasks.

As a prior work, Yasumi et al. studied a uniform k-partition problem for $k = 2$ [15]. They proposed space-optimal protocols on various assumptions for the uniform bipartition probrem.

In this paper, we consider the uniform k-partition problem for $k \geq 2$. In particular, for this problem, we consider symmetric protocols with designated initial states under the global fairness. In symmetric protocols, when two agents in the same state interact, they transit to the same state. Such protocols do not require a mechanism to break symmetry among agents and hence can be applied to various systems. Designated initial states mean that all agents have the same initial state in the initial configuration. A global fairness is an assumption on interaction patterns of agents. (the definition is given in Section 2). As a result, we propose a protocol that solves a uniform k-partition problem under the global fairness. This protocol requires $3k - 2$ states for each agent.

2 Definitions

2.1 Population Protcol Model

A population A is defined as a collection of pairwise interacting agents. A protocol is defined as $P = (Q, \delta)$, where Q is a set of possible states of agents and δ is a set of transitions on Q. Each transition in δ is described in the form $(p, q) \to (p', q')$, which means that, when an agent in state p and an agent in state q interact, they change their states to p' and q' respectively. In this paper, only deterministic

protocols are considered. For protocol $P = (Q, \delta)$, if every transition $(p, q) \to (p', q')$ in δ satisfies $(p = q \wedge p' = q') \vee p \neq q$, P is symmetric.

A global state of a population is called a configuration. A configuration is defined as a vector of (local) states of all agents. We define $s(a, C)$ as the state of agent a at configuration C. When C is clear from the context, we simply write $s(a)$. If configuration C' is obtained from configuration C by a single transition of a pair of agents, we say $C \to C'$. For configurations C and C', if there is a sequence of configurations $C = C_0, C_1, \cdots, C_m = C'$ that satisfies $C_i \to C_{i+1}$ for any i $(0 \leq i < m)$, we say C' is reachable from C, denoted by $C \xrightarrow{*} C'$.

If an infinite sequence of configurations $E = C_0, C_1, C_2, \ldots$ satisfies $C_i \to C_{i+1}$ for any i $(i \geq 0)$, E is an execution of a protocol. An execution E is globally fair if, for every pair of configurations C and C' such that $C \to C'$, C' occurs infinitely often when C occurs infinitely often.

We consider protocols with designated initial states, that is, all agents have the same initial state in the initial configuration. No agent knows the total number of agents in the initial configuration.

2.2 Uniform k-Partition Problem

Let $f : Q \to \{1, 2, \ldots, k\}$ be a function that maps a state of an agent to an integer $i (1 \leq i \leq k)$. We define a group number of $a \in A$ as $f(s(a))$. We say agent $a \in A$ belongs to the i-th group if $f(s(a)) = i$.

Configuration C is stable if there is a partition $\{G_1, G_2, \ldots, G_k\}$ of A that satisfies the following condition:

1. $||G_i| - |G_j|| \leq 1$ for any distinct i and j, and
2. For all C^* such that $C \xrightarrow{*} C^*$, each agent in G_i belongs to the i-th group at C^*.

An execution $E = C_0, C_1, C_2, \ldots$ solves the uniform k-partition problem if there is a stable configuration C_t in E. If each execution E of protocol P solves the uniform k-partition problem, we say protocol P solves the uniform k-partition problem. The main objective of this paper is to minimize the number of states. When protocol P requires x states, we say P is a protocol with x states.

3 Uniform k-partition protocol

In this section, we propose a symmetric uniform k-partition protocol with designated initial states under global fairness.

In this protocol, a set of agent states are divided into four subsets, i.e., $Q = I \cup G \cup M \cup D$, where $I = \{initial, initial'\}$, $G = \{g_1, g_2, \ldots, g_k\}$, $M = \{m_2, m_3, \ldots, m_{k-1}\}$, and $D = \{d_1, d_2, \ldots, d_{k-2}\}$. The designated initial state of agents is *initial*. State g_i in G indicates that the agent belongs to the i-th group, that is, $f(g_i) = i$ holds for any $g_i \in G$. For other state s, we define $f(s)$ as follows:

- $f(ini) = 1$ holds for any $ini \in I$.
- $f(d_i) = 1$ holds for any $d_i \in D$.
- $f(m_i) = i$ holds for any $m_i \in M$.

We say an agent is free if its state is in I. We define $\overline{initial} = initial'$ and $\overline{initial'} = initial$. The basic strategy of the protocol is as follows: First two free agents transit to states g_1 and m_2. After that, for each i $(2 \leq i \leq k-2)$, when an agent in state m_i and a free agent interact, they transit to states m_{i+1} and g_i, respectively. Lastly, when a agent in state m_{k-1} and a free agent interact, they transit to states g_k and g_{k-1}. By this behavior, k free agents can change their states to $g_1, g_2, \ldots, g_k$. That is, the size of each group is increased by one. To realize this, the protocol includes the following transitions.

1. $(initial, initial) \to (initial', initial')$
2. $(initial', initial') \to (initial, initial)$
3. $(d_i, ini) \to (d_i, \overline{ini})$ $(d_i \in D$ and $ini \in I)$
4. $(g_i, ini) \to (g_i, \overline{ini})$ $(g_i \in G$ and $ini \in I)$
5. $(initial, initial') \to (g_1, m_2)$
6. $(ini, m_i) \to (g_i, m_{i+1})$ $(ini \in I$ and $2 \leq i \leq k-2)$
7. $(ini, m_{k-1}) \to (g_{k-1}, g_k)$ $(ini \in I)$

First we explain transitions 1 to 5, which make two free agents transit to states g_1 and m_2. Recall that all agents are in state *initial* in the initial configuration. Since we consider symmetric protocols, two

agents in state *initial* cannot transit to states g_1 and m_2 at one interaction. This is the reason why we introduce state *initial*$'$. Each agent in state *initial* (resp., *initial*$'$) transits to *initial*$'$ (resp., *initial*) when it interacts with an agent in state in $I \cup D \cup G$ (except for interaction between one in state *initial* and one in state *initial*$'$). Transition 5 implies, when agents in states *initial* and *initial*$'$ interact, they become g_1 and m_2, respectively. From global fairness, if at least two free agents and no agents in a state in M exist, two free agents eventually enter states *initial* and *initial*$'$, respectively, and then enter states g_1 and m_2 by interaction. Transition 6 implies, when a free agent and an agent in state m_i interact, they become g_i and m_{i+1}, respectively. By these transitions, free agents transit to states $g_1, \ldots, g_{k-2}$ one by one. After that, from transition 7, when a free agent and an agent in state m_{k-1} interact, they become g_{k-1} and g_k, respectively. From this behavior, the size of each group is increased by one.

However, it is possible that $\lceil n/k \rceil$ or more agents in state m_1 appear, where n is the number of agents. In this case, the above transitions do not achieve a uniform k-partition. For example, in the case of $n = 12$ and $k = 4$, if four agents enter state m_1, agents can transit to states g_1, g_2, m_3, g_1, g_2, m_3, g_1 ,g_2, m_3, g_1, g_2, m_3. To slove this probrem, we introduce states in D and add the following transitions.

8. $(m_i, m_j) \to (d_{i-1}, d_{j-1})(2 \leq i, j \leq k-1)$
9. $(d_i, g_i) \to (d_{i-1}, initial)(2 \leq i \leq k-2)$
10. $(d_1, g_1) \to (initial, initial)$

By transition 8, when two agents in states in m_i and m_j interact, they transit to states in d_{i-1} and d_{j-1}, respectively. Intuitively, an agent in state d_i makes agents in $g_1, g_2, \ldots, g_i$ go back to state *initial*. Recall that an agent in state m_{i+1} can enter state d_i and an agent in state m_{i+1} has made agents in states $g_1, g_2, \ldots, g_i$. This means an agent in state d_i initializes agents that it makes enter states $g_1, g_2, \ldots, g_i$. More concretely, an agent in a state in D works as follows:

- For $2 \leq i \leq k-2$, when agents in states d_i and g_i interact, they become d_{i-1} and *initial* by transition 9, respectively.
- After that, from transition 10, when agents in states d_1 and g_1 interact, they become *initial*.

Clearly, agents can repeatedly enter state g_i and go back to *initial* many times. However, after an agent enters state g_k, one set of agents in states $g_1, \ldots, g_k$ never go back to *initial*. Thus, if there are h agents in state g_k, the number of agents in state g_i is at least h for each i. In addition, when there are h agents in state g_k and $n - kh \geq k$ holds, there is an execution that makes some agent enter state g_k. This implies, from the global fairness, some agent eventually enters state g_k. When $n - kh = r < k$ holds, there is an execution that makes the remaining agents transit to $g_1, g_2, \ldots, m_r$. From the global fairness, the remaining agents eventually enter these states. In this configuration, agents achieve a uniform k-partition and after that all agents never change their states.

Theorem **1** *When the number of agents is at least three, there exists a symmetric protocol with* $3k-2$ *states and designated initial states that solves the uniform* k*-partition problem under global fairness.*

4 Conclusion

In this paper, we proposed a uniform k-partition protocol in a population protocol model. An interesting future research is to clarify the relation between the uniform k-partition problem and other problems such as counting, leader election, and majority.

References

[1] Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In: Proc. of International Conference on Distributed Computing in Sensor Systems. pp. 63–74 (2005)

[2] Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Proc. of the 23rd annual ACM symposium on Principles of Distributed Computing. pp. 290–299 (2004)

[3] Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. Distributed Computing 21(2), 87–102 (2008)

[4] Aspnes, J., Beauquier, J., Burman, J., Sohier, D.: Time and space optimal counting in population protocols. In: Proc. of International Conference on Principles of Distributed Systems. pp. 13:1–13:17 (2016)

[5] Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Middleware for Network Eccentric and Mobile Applications. pp. 97–120 (2009)

[6] Beauquier, J., Burman, J., Claviere, S., Sohier, D.: Space-optimal counting in population protocols. In: Proc. of International Symposium on Distributed Computing. pp. 631–646 (2015)

[7] Beauquier, J., Clement, J., Messika, S., Rosaz, L., Rozoy, B.: Self-stabilizing counting in mobile sensor networks with a base station. In: Proc. of International Symposium on Distributed Computing. pp. 63–76 (2007)

[8] Cai, S., Izumi, T., Wada, K.: How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. Theory of Computing Systems 50(3), 433–445 (2012)

[9] Gasieniec, L., Hamilton, D., Martin, R., Spirakis, P.G., Stachowiak, G.: Deterministic population protocols for exact majority and plurality. In: Proc. of International Conference on Principles of Distributed Systems. pp. 14:1–14:14 (2016)

[10] Izumi, T.: On space and time complexity of loosely-stabilizing leader election. In: Proc. of International Colloquium on Structural Information and Communication Complexity. pp. 299–312 (2015)

[11] Murata, S., Konagaya, A., Kobayashi, S., Saito, H., Hagiya, M.: Molecular robotics: A new paradigm for artifacts. New Generation Computing 31(1), 27–45 (2013)

[12] Sudo, Y., Masuzawa, T., Datta, A.K., Larmore, L.L.: The same speed timer in population protocols. In: Proc. of International Conference on Distributed Computing Systems. pp. 252–261 (2016)

[13] Sudo, Y., Nakamura, J., Yamauchi, Y., Ooshita, F., Kakugawa, H., Masuzawa, T.: Loosely-stabilizing leader election in a population protocol model. Theoretical Computer Science 444, 100–112 (2012)

[14] Sudo, Y., Ooshita, F., Kakugawa, H., Masuzawa, T.: Loosely-stabilizing leader election on arbitrary graphs in population protocols without identifiers nor random numbers. In: Proc. of International Conference on Principles of Distributed Systems. pp. 14:1–14:16 (2015)

[15] 安見嘉人, 大下福仁, 山口賢一, 井上美智子: 個体群プロトコルにおける定数スペース半数分割アルゴリズム. The Institute of Electoronics, Information and Communication Engineers (2017)

個体群プロトコルにおける
分割問題の一般化と空間複雑性について

名古屋工業大学　海野友希　北村直樹　泉泰介

2017 年 8 月 29 日

1　はじめに

個体群プロトコル [1] とは，計算資源が非常に制限された小型デバイスによって構成される，受動的モバイルセンサネットワークの理論モデルの一つである．個体群プロトコルの個体群はエージェントの集合によって構成され，各エージェントは他のエージェントに十分近く接近した時に通信を行うことができる．

本研究では，個体群プロトコルモデル上における分割問題を考える．分割問題とは，エージェントの集合を個体群プロトコルによって複数のグループに分割する問題である．一般的な設定では各グループの構成比が与えられた時，その比に従って複数のグループに分割することが要求される．先行研究では，各グループに含まれるエージェントの数が等しい N 分割を状態数 3N-2 で行うプロトコルが考案されている [2]．また，このプロトコルを用いて，任意の整数構成比に対して分割を行うプロトコルも構成することができる．例えば，2:1 の比にエージェント集合を分割したいときは，等しいサイズでの 3 分割を行い，そのうちの 2 つのグループを同一とみなせば良い．しかし，このプロトコルでは比率の総和と等しい数のグループに一度分割する必要があるため，分割するグループの数に対して多くの状態数が必要となってしまう．そこで，本稿では，分割するグループの比率の総和が 2^k となる時により少ない状態数で分割を行うことができるプロトコルを提案する．

2　諸定義

2.1　個体群プロトコル

個体群プロトコルとは，受動的モバイルセンサネットワークの理論モデルの一つである．個体群プロトコルでは個体群は N 個の有限状態機械で構成され，スケジューラによって選択された 2 つのエージェントが，プロトコルに従ってインタラクトし互いの状態を遷移させる．個体群プロトコルの個体群は，個体群を形成するエージェント集合を V，インタラクト可能なエージェント同士をつなぐ辺集合を E とした無向グラフ $G = (V, E)$ で表すことができる．この無向グラフ G をコミュニケーショングラフという．

個体群プロトコルは $PP = (X, Y, Q, I, O, \delta)$ で表される.X はプロトコル開始時に各ノードに与えられる入力文字の集合を表し，Y は各エージェントが出力する文字の集合を表す．$I : X \to Q$ は入力アルファベットから状態集合への写像である．各エージェントの初期状態はこの I によって決定される．$O : Q \to Y$ は状態集合から出力アルファベットへの写像である．個体群プロトコルの出力 Out は $Out = \{O(v) | v \in V\}$ で表される.$\delta : Q \times Q \to Q \times Q$ は状態遷移関数を表し，インタラクトする 2 つのエージェントはこの関数に従って状

態を遷移させる．後述する分割問題では入力アルファベット，入力関数は使用しないため以後省略する．

ある時点での個体群のすべてのエージェントの状態の集合を状況 C とする．状況 C におけるあるエージェント $a \in V$ の状態は $s(a, C)$ で表される．文脈上 C が明確である場合 C は省略し，$s(a)$ とする．ある 2 つのノード $u, v \in V$ のインタラクション $e \in E$ によって状況 C が状況 C'に変化する時，C は C'に遷移可能といい，$C \to C'$と表す．また，2 つの状況 C, C'の間に遷移可能な状況列 $C \to C_0 \to C_1 \to C_2 \to ..... \to C_m$が存在する時，$C'$は C から到達可能といい，$C \to^* C'$と表す．

ある無限の状況の系列 $E = C_0, C_1, C_2, , , , (C_i \to C_{i+1})$ が個体群プロトコルの実行である．実行 E において，ある状況 C が無限回現れ，C から遷移可能なすべての状況が同じく無限回 E に現れる時，この実行 E は公平である．個体群プロトコルにおける計算とは，公平な実行である．

本稿で提案するプロトコルは全体公平性を仮定している．全体公平性とは，エージェントのインタラクトに対する過程であり，この仮定のもとでは以下の 2 つの性質が存在する．

1. コミュニケーショングラフが完全
2. すべての遷移可能な状況についてスタベーションが起こらない. すなわち，$C \to^* C'$ であり，かつ C への遷移が無限回生じるときは，C' への遷移も無限回生じる

また，以下の 3 つの制約も仮定する．

1. 対称性
2. 匿名性
3. 初期状態が一意

対称性とは，プロトコルの状態遷移関数に関する制約である．この条件のもとでは，状態遷移関数のルールはインタラクト前の 2 つのエージェントが同じ状態の時，インタラクト後の 2 つのエージェントの状態も同じでなければならない．匿名性とは，エージェントの識別子に関する制約であり，この条件のもとではエージェントはユニークな識別子を持たない．初期状態が一意とは，エージェントの初期状態に関する制約条件であり，この条件のもとではすべてのエージェントの初期状態は等しい．

2.2 分割問題

分割問題とは，与えられた個体群を個体群プロトコルによって複数のグループに分割する問題である．分割問題では，出力アルファベット $Y = \{G_1, G_2, G_3, , , , G_N\}$ をエージェントの分割先グループ名の集合とし，各 $G_i \in G$ に所属するエージェントの数を $|G_i|$ とする．そして，ベクトル $\boldsymbol{A_C} = \{|G_1|, |G_2|, |G_3|, \cdots, |G_N|\}$ を状況 C における各グループの比率とする．この Y に各エージェントの状態を出力関数 O によって写像する．この時，O は単射である必要はない．所望の分割を $a_1 : a_2 : a_3 : ... : a_N$とした時，個体群プロトコル P の公平な実行 E 内に，すべての遷移可能な状況 C'にたいして $O(C) = O(C')$ となり，$\frac{A_{C\,i}}{gcd(A_{C\,1}, A_{C\,2}, \cdots, A_{C\,N})} = a_i (1 \leq i \leq N)$ が成り立つ時，P は分割問題を解くとする．

3　$1:2^k-1$ 分割を行うプロトコル

本節では，個体群に対して二分割を木状に繰り返して行うことで $1:2^k-1$ 分割を得るプロトコルを提案する．個体群の二分割は以下の個体群プロトコル $PP_{2-divide}$ で行うことができる．

$$
\begin{aligned}
PP_{2-divide} &= (Y_{2-divide}, Q_{2-divide}, O_{2-divide}, \delta_{2-divide}) \\
Y_{2-divide} &= \{G_1, G_2\} \\
Q_{2-divide} &= \{0, 1, A, B\} \\
O_{2-divide} &= \{A \to G_1, B \to G_2\} \\
\delta_{2-divide} &= \{00 \to 11, 11 \to 00, 01 \to AB, 1B \to 0B\}
\end{aligned}
$$

$PP_{2-divide}$では，出力状況は全エージェントの半分が状態 A を持ち，残りのエージェントが B を持つ．

一般にある複数の個体群プロトコルが逐次的に実行可能であるかは非自明である．これは，個体群プロトコルではプロトコルの終了を検知できないためである．ある 2 つの個体群プロトコル A, B を仮定し，A によって生成された出力状況 C_Aを B の入力状況として動作させることを考える．この時，C_Aを構成する各エージェントの状態が，一度エージェントがその状態に遷移した後に他の状態に遷移しない状態ならば，B は A の終了を検知することなく処理を行う事ができる．言い換えると，エージェントが状態によって自身に対する A の処理が終了したことを検知できる場合，A の後に他の個体群プロトコルを実行することができる．個体群プロトコルでは，出力状況は一度に形成されるのではなく，徐々に形成される．このため，A の遷移関数に出力状況を構成する状態を用いた遷移が存在する場合，B によって A の処理が終わったエージェントの状態が遷移し A の処理が正しく行われない可能性がある．しかし，この問題は，A の遷移関数において A の状態から B によって遷移する状態を同一視することで解決することができる．

次に，個体群を二分割するプロトコルが上記の性質を満たしているかを考える．$\delta_{2-divide}$より，一度 A または B に遷移したエージェントは再び他の状態に遷移することはないため，複数の $PP_{2-divide}$を逐次的に実行することが可能である．このため，図 1 のように分割を木状に繰り返し，木の葉に A または B を割り振ることで，$1:2^k-1$ 分割を得ることができる．図 1 では $k=3$ とした．

二分割を木状に繰り返し $1:2^k-1$ 分割を得る場合に必要な状態数を考える．$PP_{2-divide}$より，一つの個体群を 1 : 1 の比率で二分割するために必要な状態数は 4 である．この 4 状態 $0, 1, A, B$ の役割は, それぞれ初期状態，初期状態と対称的に変化する状態，そして出力状態である．二分割を逐次的にに動作させる時，最初の二分割以外は，他の二分割の出力状況を初期状況とするため，新しく初期状態を追加する必要はない．また，2 つの出力状態の内一つはすでに利用した出力状態を使用できるため，最初の二分割以外の二分割で必要となる状態数は 2 となる．よって，m 個の二分割を逐次的に動作させるために必要な状態数は，$4+2(m-1)$ となる．図 1 より，$1:2^k-1$ の比率に分割するためには木の深さ分の分割が必要になることがわかる．よって，$m=\log k$ となり，必要な状態数は $4+2(\log k-1)$ となる．

4　二分割以外への拡張

$1:2^k-1$ の二分割を構成する際には，二分割を他の二分割によって生成された 2 つの個体群の内一方にのみ行っていたが，2 つの個体群のそれぞれに二分割を行うことも同様に可能である．この方法を用いると以下

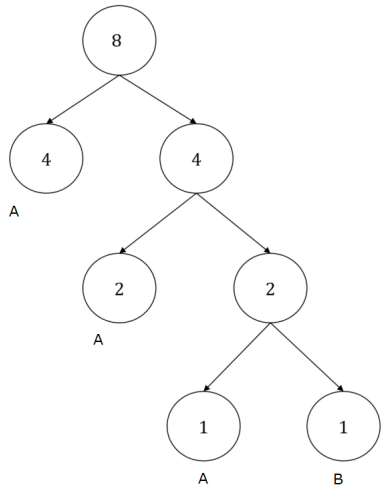


図 1　k=3 の場合の分割木

のような分割が可能になる．

$$a_1 : a_2 : a_3 : ... : a_n \sum_{i=1}^{n} a_i = 2^k$$

上記の分割を行う際に必要となる状態数を考える．$a_1 : a_2 : a_3 : ... : a_n \sum_{i=1}^{n} a_i = 2^k$の分割では，比の個数が増える二分割と比率を変える二分割の二種類が考えられる．例えば，2 : 2 の状態から 1 : 1 : 2 となる二分割と，1 : 3 となる二分割である．この時，比率を変える二分割で必要となる状態数は上述したように 2 状態である．これは，二分割において必要な初期状態と出力状態の内一つを既存の状態で代用できるためである．しかし，比の個数が増える二分割では，出力状態を新たに 2 つ追加する必要がある．よって，この時に必要となる状態数は 3 である．

上記より，$a_1 : a_2 : a_3 : ... : a_n \sum_{i=1}^{n} a_i = 2^k$の分割を行うために必要な状態数は，逐次的に動作させる二分割の個数を m，比の個数を N とすると，以下のようになる．

$$4 + 2(m - 1) + N - 2$$

ここで，逐次的に動作させる二分割の個数をカウントすることを考える．分割を構成する木の根は 2^kであるため，2 進数で表すとビット列に 1 が一つだけ含まれている．二分割を行うと 1 であるビットが一つから二つに増える．この関係は明らかにすべてのノードと子ノードにおいて成り立つ．よって，二分割の個数はすべての比の要素に含まれる 1 であるビットの合計から 1 を引いたものと等しくなる．以上より，ビット列を与えた時そのビット列に含まれる 1 の個数を返す関数を F とした時，必要となる状態数は以下のようになる．

$$4 + 2(\sum_{i=1}^{N} F(a_i) - 2) + N - 2$$

5　評価

本節では，本稿で提案したプロトコルが先行研究よりも状態数が少なくなることを確認する．このために，提案プロトコルにおいて最も必要な状態数が多くなる分割を考える．提案プロトコルによって生成される分

割木は二分木であり，枝分かれ一回が一回の二分割を表しているため，最も二分割の回数が多くなる分割は比率が等しい 2^k分割である．この時必要な状態数は，$\sum_{i=1}^{N} F(a_i) = N$ であるため以下のようになる．

$$\begin{aligned} \text{必要な状態数 } x &= 4 + 2(\sum_{i=1}^{N} F(a_i) - 2) + N - 2 \\ &= 4 + 2(N-2) + N - 2 \\ &= 4 + 2N - 4 + N - 2 \\ &= 3N - 2 \end{aligned}$$

以上より，提案プロトコルの必要状態数の上界が先行研究で必要な状態数と一致するため，比率が異なる分割においてこのプロトコルは先行研究よりも状態数を減らすことができる．

6 まとめ

本稿では，グループの比率が異なる分割問題を比率の総和が 2^kになる場合にのみ，より少ない状態数で解くことができるプロトコルを提案した．現在は総和が 2^k以外の場合への一般化と個体群が比率の総和で割り切れない場合に発生する誤差についての解析を進めている．

参考文献

[1] Angluin, D., Aspnes, J., Diamadi, Z., Fischer,M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Proc. of the 23rd annual ACM symposium on Principles of Distributed Computing. pp. 290299 (2004)

[2] 安見嘉人，北村直樹，大下福仁，泉泰介: 大域公平性を仮定した個体群プロトコルモデルにおける k 分割アルゴリズム (未発表).

Brief Announcement: Fast Aggregation in Population Protocols

Ryota Eguchi Taisuke Izumi

Nagoya Institute of Technology

Abstract

The coalescence protocol plays an important role in the population protocol model. The conceptual structure of the protocol is for two agents holding two non-zero values a, b respectively to take a transition $(a, b) \to (a + b, 0)$, where $+$ is an arbitrary commutative binary operation. Obviously, it eventually aggregates the sum of all initial values. In this paper, we present a fast coalescence protocol that converges in $O(\sqrt{n} \log^2 n)$ parallel time with high probability in the model with an initial leader (equivalently, the model with a base station), which achieves an substantial speed-up compared with the naive implementation taking $\Omega(n)$ time.

Introduction A *passively-mobile* system, which is an abstract notion of wireless ad-hoc networks, consists of a collection of moving objects (called *agents*) in a certain region, with computing devices that do not control over how they move. Despite the restrictions on communication range, memory, and computational power caused by the mobility requirement, the devices must execute cooperatively some task through tiny local computation and short-range communication with other devices located nearby. Typical examples of passively-mobile systems are the network of smart devices attached to cars or animals. *Population protocol* is one of the promising models for such a system, which is first introduced by Angluin et al. [2]. A Population protocol consists of anonymous and identical n agents, which are defined as deterministic state machines. The communication among agents is performed by pairwise interactions, where two interacting agents change their states following a transition function (protocol) deployed to all agents. An execution of a population protocol is a sequence of pairwise interactions. In the basic model, the scheduling of interactions is worst-case but guaranteed to be *fair*, which means that if in the infinitely-many interactions every two agents interact infinitely often.

The recent trends of this model are to design fast protocols for popular problems (e.g. leader election, majority) converging in $O(polylog(n))$ time, and to reveal trade-offs between time and space for several problems. To measure time in the runs of the population protocol models, the (uniform) probabilistic scheduler is often assumed. In the model, two agents interacting at each step are selected at random uniformly and independently. In the literature of this model, time complexity is defined as the number of interactions divided by the number of agents n, and space complexity is the number of states used by each agent.

We consider some abstract problem, called the *aggregation* problem. Precisely, the aggregation problem is defined by any monoid $(X, +)$, where initially each agent i has a value $x_i \in X$, and eventually one specific agent must output the value of $s = \sum_i x_i$. This problem can be solved by the traditional *coalescence* protocol, whose transition rule for two agents with values a and b respectively is specified by $(a, b) \to (a + b, 0)$. One can see that, the standard coalescence protocol needs $\Theta(n)$ time for convergence, since the probability that the last two agents having non-zero values interact is $1/n^2$.

In this paper, we present a new coalescence protocol. It achieves $O(\sqrt{n} \log^2 n)$-convergence time in the special model with existence of one unique leader (equivalently, the model with a base station). On the space complexity side, agents (including the leader) uses $O(|X|^3)$ states.

Problem Statement Let $(X, +)$ be an arbitrary commutative monoid whose identity element is zero (where $+$ is not necessarily the standard arithmetic sum), and $\hat{X} = X \setminus \{0\}$. In the aggregation problem for $(X, +)$, each agent i initially has a value $x_i \in X$, and the goal of the task is that the leader computes the value $s = \sum_i x_i$. More precisely, we assume that the leader equips an output register storing a value in X. The value of the output register must be converged and stabilized into s. Note that the leader does not have to detect the termination of an execution, and is allowed to update answers multiple times. The computation time of the aggregation problem is defined as the time taken until the convergence of the output register.

Outline of Our Algorithm Our algorithm utilizes several algorithmic tools proposed in past literature as building blocks. Before the presentation of our protocol, we illustrate three tools. The first algorithm called *epidemics* (or propagation) is a straightforward subroutine used in many algorithms. The abstract structure of the epidemics is as follows: At first there are at least one agent with value v, which wishes to propagate v to all other agents, and the other agents initially with value $\perp$. The transition rule is $(v, \perp)$ or $(\perp, v)$ to (v, v). The analysis by Angluin et al. [3] shows that under the random scheduler the epidemics algorithm finishes within $O(\log n)$ parallel time with high probability.

The second tool is a synchronization mechanism called *phase clock* which counts approximately $O(\log n)$ time or $O(\log^2 n)$ time. The phase clock is first presented in the paper by Angluin et al. [3]. The phase clock is mainly introduced for a unique leader to detect the end of the epidemics (i.e. $O(\log n)$ time), and by a simple extension, it is also possible to count $O(\log^2 n)$ time [4]. A non-trivial advantage of the phase clock mechanism is that it uses only $O(1)$ states per agent.

The third tool is synthetic coin flips due to Alistarh et al. [1], which provides the accessibility of private random bits to each agent. It gives a coin flipping mechanism with reasonably small bias to the agents. The randomness of the synthetic coin flips is extracted from the random interaction-pattern of the scheduler, and thus it works only on the random scheduler.

The idea of our algorithm is very simple: The bottleneck of the standard coalescence algorithm is the situation where the number of agents with non-zero values becomes small. If only m agents have non-zero values, an interaction selected by the scheduler gets no progress of the algorithm with probability

$1 - \Theta((m/n)^2)$. In the naive coalescence algorithm, spending $O(\sqrt{n}\text{polylog}(n))$ parallel time, $\Theta(\sqrt{n})$ agents still have non-zero values. To accelerate the following coalescence process, when only $O(\sqrt{n})$ agents have non-zero values, we utilizes another mechanism, called *sequential absorption*. The sequential absorption first chooses an agent (which is not leader) with a non-zero value as an absorption agent only spending $O(\log n)$ parallel time. This process is achieved by utilizing the phase clock and the synthetic coin flips: At each phase, the agents with non-zero values flip the synthetic coin, the agents which get value 1 start epidemics, and the epidemics kill the agents with value 0. The number of phases to elect one unique absorption agent is $O(\log n)$, thus the total time to elect the agent is $O(\log^2 n)$. The absorption agent runs the epidemics its value, and immediately become an agent with value zero. The value reaches to the leader within $O(\log n)$ time. Repeating this procedure $\Theta(\sqrt{n})$ times, we can complete the aggregation. Since both the election and epidemics take $O(\log^2 n)$ time, the total running time of the sequential absorption is $O(\sqrt{n}\log^2 n)$. The remaining issue is to combine those two algorithms. While the sequential composition is obviously correct, it requires the timer for (exactly or approximately) counting $\Theta(\sqrt{n})$ parallel time. To avoid consuming extra memory space, instead we choose fair composition, that is, simply running them concurrently. This composition does not affect the correctness of our protocol, since the absorption agent behave following way so that the value of the sum does not change: When the absorption agent with value x_i detect its uniqueness by the phase clock counting $O(\log^2 n)$ time, it immediately change its state to the value zero, and thus x_i is never aggregated in the standard coalescence side. Here we present our main theorem. Note that our algorithm have a low probability of error, that is conversely the algorithm convergences only with high probability.

Theorem 1. *Our algorithm solves an aggregation problem for $(X, +)$ in expected $O(\sqrt{n}\log^2 n)$ time using $O(|X^3|)$ states per agent, with high probability.*

Discussion and Research Direction In [2], authors show the simple coalescence protocol can compute semilinear predicates, which are exact characterization of the basic population protocol model, and thus our protocol computes the predicates in $O(\sqrt{n}\log^2 n)$ time. However for computation of semilinear predicates with leader, there is much faster protocol presented by Angluin et al. [3] which converges in $O(\log^4 n)$ time with high probability. We believe that due to its simplicity and generality, there are some applications of our algorithm in population protocol models.

References

[1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in population protocols. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2560–2579. SIAM, 2017.

[2] Dana Angluin, James Aspnes, Zoë Diamadi, MichaelJ. Fischer, and Rene Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[3] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, September 2008.

[4] Leszek Gasieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. *arXiv preprint arXiv:1704.07649*, 2017.

個体群プロトコルモデルにおける緩安定リーダ選挙の空間複雑性について

片岡　大輝　泉　泰介

近年，インターネットやモバイルのデバイス，スマートフォンなどが普及している中で，移動体通信による分散システムは見時間身近な存在になりつつある．移動体通信のシステムのモデル化における大きな要因の一つは時間とともにネットワークの構造（トポロジ，参加ノード）が変化しうることであるが，そのようなシステムの動的変化をモデル化した分散アルゴリズムの通信モデルが提案されている．特に代表的なモデルに個体群プロトコルモデルがある．

個体群プロトコルは，n 台のエージェントが互いに通信を行うモデルである．個体群プロトコルにおいては，スケジューラは 1 対のエージェントを選択し，選択されたエージェント対は互いに情報をやり取りすることで自身の状態を更新する．

本研究では，個体群プロトコルモデル上でのリーダ選挙問題を考える．自己安定アルゴリズムは，どんな初期状態からもいくつかの正当な状況に収束し，それ以降その状況を維持することが保証されてる．自己安定リーダ選挙問題は，正確なエージェント数 n が既知のときにメモリ複雑度が $\Omega(\log n)$ ビット必要であることが証明されている [1]．

緩自己安定は，自己安定の制約を弱めたものとして提案された [3]．緩自己安定アルゴリズムは，どんな初期状態からも正当な状況に収束し，それ以降その状況を十分高い確率で維持することが保証されてる．自己安定アルゴリズムとの違いは，十分に小さい確率で正当な状況から外れることを許している点にある．緩自己安定リーダ選挙問題は，エージェント数 n に対して上界 N のみが既知のときにメモリ複雑度が $\Theta(\log N)$ ビットとなるようなアルゴリズムが知られている [2] [3]．

またその一方で，エージェント数の上界 N に対して，エージェント数 n が $N \geq n$ であるような任意のシステムで正しく動作するような緩安定リーダ選挙アルゴリズムのメモリ複雑度が $\Omega(\log N)$ ビットとなることも示されている [2]．

本研究では，エージェント数 n に対して上界 N のみが既知ではなく，下界が既知である場合について，緩安定リーダ選挙プロトコルのビット複雑度が下げられるかどうかを議論する．既知の下界 [2] の証明においては，アルゴリズムが $n = 2$ の場合においても正しく動作するという事実が重要な役割を果たしており，n についての下界を仮定した場合 (すなわち，十分 N に近い n の値に対してのみアルゴリズムが正しく動作することを保証する場合)，その命題は成立しない．本研究では，この疑問を否定的に解決する．すなわち，たとえ n の値が完全に既知であったとしても，緩安定リーダ選挙アルゴリズムのビット複雑度が $\Omega(\log N)$ ビットとなることを示す．

参考文献

[1] Shukai Cai, Taisuke Izumi, and Koichi Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory of Computing Systems*, Vol. 50, No. 3, pp. 433–445, 2012.

[2] Taisuke Izumi. On space and time complexity of loosely-stabilizing leader election. In *International Colloquium on Structural Information and Communication Complexity*, pp. 299–312. Springer, 2015.

[3] Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely-stabilizing leader election in population protocol model. In *SIROCCO*, pp. 295–308. Springer, 2009.

CPA アルゴリズムのランダムグラフにおける局所ビザンチン故障耐性について

早川　駿　泉　泰介

インターネットに代表される計算機ネットワークが社会基盤として必要不可欠となった現在において，故障耐性を有する分散システムの設計が重要性を増している．特に不特定多数が参加するオープンなシステムにおいて，悪意を持った計算ノード（ビザンチン故障）に対する耐性をシステムが有することは実用的にも重要である．

任意の数のノードがビザンチン故障を起こし得るシステムでは，故障耐性の実現は自明に不可能であるので，一般にはビザンチン故障耐性を考える場合は故障数の上限が設定されることになる．一方で，たとえ故障数の上限 f が全体のノード数 n に比して十分に小さいとしても，ネットワークトポロジが頂点数 f 未満の切断頂点集合を持つ場合，故障ノードによりネットワークの分断が生じるため，ビザンチン故障耐性の実現は不可能となる．この問題を回避するアプローチの一つとして，f-局所ビザンチン故障と呼ばれるモデルが提案されている，このモデルにおいては，各ノードの隣接ノードのうち，高々 f 個がビザンチン故障ノードであり得るという制限が課される．もちろん，f-局所故障モデルにおいても，次数が f より小さいノードが存在した場合，やはりネットワークの分断が生じ得る．そのため一般的には f はある程度 (最小次数よりも) 小さい値が設定される．その一方で，f-局所故障モデルは全体での故障の総数に関しては制限がなく，この点においては旧来のモデルよりも緩い仮定が課されている．

f-局所ビザンチン故障モデルにおける既存アルゴリズムの一つとして，ネットワーク上の全正常ノードに正しくメッセージを配信するブロードキャスト問題を解く CPA アルゴリズムが知られている [1,2,3]．ただし，このアルゴリズムは任意のトポロジで正しく動作することは保証されていない．アルゴリズムが正しく動作するトポロジの条件に関して，いくつかの結果が知られている [1,2]. 本研究では，Erdos-Renyi ランダムグラフ $G(n,p)$ における，CPA アルゴリズムの成功確率を解析することを目指す．ランダムグラフにおけるの多くの事象の発生がそうであるように，CPA アルゴリズムの正当な動作についても，p の増加に伴う相転移現象が発生することが期待される．本研究では，シミュレーション実験を通して，実際に相転移が生じることの観察，またその確率の閾値がパラメタ n および f にどのように依存しているかについての検討を行う．

参考文献

[1] Akira Ichimura, Maiko Shigeno, ”A new parameter for a broadcast algorithm with locally bounded Byzantine faults”, Information Processing Let-

ters, Volume 110, Issues 12-13, 15 June 2010, Pages 514-517

[2] Lewis Tseng, Nitin Vaidya, Vartika Bhandari, "Broadcast Using Certified Propagation Algorithm in Presence of Byzantine Faults", Technical Report, September 20, 2012

[3] Andrzej Pelc, David Peleg, "Broadcasting with locally bounded Byzantine faults", Information Processing Letters, Volume 93, Issue 3, 14 February 2005, Pages 109-115

On Directed Covering and Domination Problems

Tesshu Hanaka* Naomi Nishimura Hirotaka Ono

Abstract

In this paper, we study covering and domination problems on directed graphs. Although undirected Vertex Cover and Edge Dominating Set are well-studied classical graph problems, the directed versions have not been studied much due to the lack of clear definitions.

We give natural definitions for Directed r-In (Out) Vertex Cover and Directed (p, q)-Edge Dominating Set as directed Vertex Cover and Edge Dominating Set. For these problems, we show that

- Directed r-In (Out) Vertex Cover and Directed (p, q)-Edge Dominating Set are NP-complete on planar directed acyclic graphs except when $r = 1$ or $(p, q) = (0, 0)$,
- if $r \geq 2$, Directed r-In (Out) Vertex Cover is $W[2]$-hard and $c \ln k$-inapproximable on directed acyclic graphs,
- if either p or q is greater than 1, Directed (p, q)-Edge Dominating Set is $W[2]$-hard and $c \ln k$-inapproximable on directed acyclic graphs,
- all problems can be solved in polynomial time on trees, and
- Directed $(0, 1), (1, 0), (1, 1)$-Edge Dominating Set are fixed-parameter tractable in general graphs.

The first result implies that (directed) r-Dominating Set on directed line graphs is NP-complete even if $r = 1$.

1 Introduction

Covering and domination problems are well-studied problems in theory and in applications of graph algorithms, for example, Vertex Cover [17], Dominating Set [17] and Edge Dominating Set [26]. However, almost all of these problems are studied on undirected graphs. In particular, Vertex Cover and Edge Dominating Set on directed graphs have not been studied although there are some results on directed Dominating Set [11, 7, 23, 16]. This seems surprising, but maybe one reason might be that it is difficult to expand the definition naturally to directed graphs due to the unclear relationship between "direction" and "domination".

In this paper, we study directed versions of Vertex Cover and Edge Dominating Set. First, we give formal definitions of directed Vertex Cover and directed Edge Dominating Set. In the definitions, we consider several scenarios that reflect how the selected set influences edges via directed edges. It should be noted that the definition follows from r-Dominating Set [8, 13, 23]. These definitions are also motivated by economic network analysis. We mention applications of these problems in Section 1.2.

In a directed graph, vertex v is said to *in-cover* every incoming edge (u, v) and *out-cover* every outgoing edge (v, u) for some u. A vertex v is also said to *r-in-cover* all edges

*Kyushu University, Email: 3EC15004S@s.kyushu-u.ac.jp

in the directed path to v of length at most r. Similarly, v is said to *r-out-cover* all edges in the directed path from v. Here, for a path $v_1, v_2, \ldots, v_\ell$, the *length* of the path is defined as the number of edges, that is, $\ell - 1$. In particular, if $r = 0$, a vertex is not considered to cover any edge. Then Directed r-In (Out) Vertex Cover is the following problem.

Definition 1.1 Directed r-In (Out) Vertex Cover (r-In (Out) VC) *is the problem that given a directed graph $G = (V, E)$ and two positive integers k and r, determines whether there exists a vertex subset $S \subseteq V$ of size at most k such that every edge in E is r-in (out)-covered by S. Such S is called an* r-in (out)-vertex cover.

Furthermore, we define Directed (p, q)-Edge Dominating Set. An edge $e = (u, v)$ is said to (p, q)-*dominate* itself and all edges that vertex u p-in-covers and vertex v q-out-covers. In particular, edge (u, v) is said to $(p, 0)$-*dominate* (resp., $(0, q)$-*dominate*) itself and all edges p-in-covered by u (resp., q-out-covered by v).

Then Directed (p, q)-Edge Dominating Set is defined as follows.

Definition 1.2 Directed (p, q)-Edge Dominating Set ((p, q)-EDS) *is the problem that given a directed graph $G = (V, E)$, one positive integer k, and two non-negative integers p, q, determines whether there exists an edge subset $K \subseteq E$ of size at most k such that every edge is (p, q)-dominated by K. Such K is called a* (p, q)-edge dominating set.

The undirected Edge Dominating Set problem is Dominating Set on (undirected) line graphs. We can see the same relationship between Directed $(0, 1)$-Edge Dominating Set and Dominating Set on directed line graphs. For a directed graph, a *directed line graph* is deined as follows:

Definition 1.3 ([18]) *A* directed line graph *of $G = (V, E)$ is $L(G) = (E, E_2)$ such that*

$$E_2 = \{((x, y), (z, w)) | (x, y), (z, w) \in E \wedge y = z\}.$$

It is obvious that a directed $(0, 1)$-edge dominating set on a directed graph G corresponds to a (directed) dominating set on the line graph of G. Furthermore, Directed $(1, 1)$-Edge Dominating Set corresponds to undirected Dominating Set on an underlying undirected graph of a directed line graph. These relations imply that our definition of Directed (p, q)-Edge Dominating Set is quite natural from the viewpoint of the line graph operation.

One interesting aspect of directed versions, but not undirected versions, is the asymmetry of the problem structures. For Directed r-In Vertex Cover, a vertex v in-covers only (u, v) when $r = 1$. Thus, a 1-in vertex cover is the set of all vertices whose in-degree is at least one. Therefore, it is trivial that Directed 1-In (Out) Vertex Cover is solvable in linear time, while undirected Vertex Cover is NP-complete. On the other hand, Directed $(1, 1)$-Edge Dominating Set, in a sense, corresponds to (undirected) Edge Dominating Set. For the optimization version, Edge Dominating Set is equivalent to Minimum Maximal Matching [26]. However, Directed $(1, 1)$-Edge Dominating Set does not necessarily correspond to matching on the undirected graphs underlying directed graphs due to the asymmetry of domination.

For Directed (p, q)-Edge Dominating Set, there exists another source of asymmetry. That is, we can consider the case in which p and q are different. In the case in which $(p, q) = (0, 1)$, edge (u, v) dominates itself and edges out-covered by v. Although Directed $(0, 1)$-Edge Dominating Set is similar to Directed 1-Out Vertex Cover, surprisingly, it is NP-complete on directed acyclic graphs.

Graph class	Tree	Planar DAG of bounded-degree	DAG	General
1-In (Out) VC	-	-	-	$O(n)$
r-In (Out) VC ($r \geq 2$)	$O(n^4)$	NP-c	$W[2]$-h	$W[2]$-h
$(0,1),(1,0)$-EDS	$O(n^4)$	NP-c	NP-c	$2^{O(k)}n$
$(1,1)$-EDS	$O(n^4)$	NP-c	NP-c	$2^{O(k)}n$
(p,q)-EDS (p or $q \geq 2$)	$O(n^4)$	NP-c	$W[2]$-h	$W[2]$-h

Table 1: Our results for graph classes. NP-c and $W[2]$-h stand for NP-complete and $W[2]$-hard, respectively.

1.1 Our Contributions

Table 1 shows our results. In this paper, we first give hardness results for Directed r-In (Out) Vertex Cover and Directed (p,q)-Edge Dominating Set on restricted graphs, even on directed acyclic planar graphs of bounded-degree. The hardness on directed acyclic graphs implies that we cannot design parameterized algorithms with respect to *directed treewidth* [19] and *DAG-width* [1] unless P=NP. The fact that Directed $(0,q)$-Edge Dominating Set is NP-complete even if $q = 1$ implies that (directed) r-Dominating Set on directed line graphs is NP-complete even if $r = 1$. Moreover, we prove that Directed r-In (Out) Vertex Cover is $W[2]$-hard and $c \ln k$-inapproximable on directed acyclic graphs when $r \geq 2$, and Directed (p,q)-Edge Dominating Set is $W[2]$-hard and $c \ln k$-inapproximable on directed acyclic graphs when either p or q is greater than 1. These results hold even if there are no multiple edges or loops.

On the other hand, we obtain algorithms for certain cases, including algorithms for all problems when restricted to trees, for any values of p, q, and r. The interplay among distance, direction, and domination results in a complex dynamic programming solution, running in $O(n^4)$ time. Because an edge can either dominate or be dominated by edges outside of a subtree depending on how it is directed, at each step of the algorithm we need to maintain extensive information not only about the subtree itself but also potential outside influence.

We show that Directed $(0,1),(1,0),(1,1)$-Edge Dominating Set is fixed-parameter tractable with respect to k. In particular, we give $2^{O(k)}n$-time algorithms. We emphasize that the running time of these algorithms is single exponential in k and linear in n. Moreover, our fixed-parameter algorithms are based on dynamic programming on a tree decomposition. Thus, we also show that Directed $(0,1),(1,0),(1,1)$-Edge Dominating Set can be solved in linear time on graphs whose underlying undirected graphs have bounded treewidth. Note that given a directed graph G and its underlying undirected graph G^*, the directed treewidth of G is no greater than its DAG-width which, in turn, is no greater than the treewidth of G^*[1].

1.2 Motivation and Application

As practical motivation, a number of network models employ directed graphs. For example, directed graphs are used to represent economic networks in which vertices correspond to industries and edges correspond to transactions of money or materials between industries [5, 24].

Recently, economists have used graph algorithms to analyze these economic networks in terms of graph structures in order to find critical industries and transactions [21, 22, 20]. Based on the analyses, they discuss which kinds of economic policies should be adopted, and so on. However, there are some problems. Such analyses in economics are based on

undirected graph algorithms instead of directed graph algorithms; they first transform directed graphs to undirected graphs, and then apply undirected graph algorithms to the graphs thus obtained. This is because there are many more results on graph optimization on undirected graphs than on directed graphs. Of course, such substitute algorithms might extract some information from the processed graph, but some important information is definitely lost. For example, when we would like to find a critical transaction in an economic network, the edge direction is clearly essential.

The theoretical motivation is a relationship between directed Dominating Set and Directed (p, q)-Edge Dominating Set. As we mentioned above, Directed $(0, 1)$-Edge Dominating Set is directed Dominating Set on directed line graphs and Directed $(1, 1)$-Edge Dominating Set is undirected Dominating Set on an underlying undirected graph of a directed line graph. Directed line graphs are well-studied for DNA sequencing and have some useful properties and characterizations [18, 2]. As for combinatorial problems on graphs, (directed) Hamiltonian Path on directed line graphs can be solved in time $O(n^2 + m^2)$ [3] while Hamiltonian Path on undirected line graphs is NP-complete [4]. Therefore, some directed problems could be easier than the undirected versions on line graphs. Unfortunately, however, our results show that directed Dominating Set and the distance version, that is, directed r-Dominating Set, remain NP-complete even on directed line graphs.

1.3 Related problems

One of the most famous covering problems is Vertex Cover. This is a classical NP-complete problem on undirected graphs, known to be fixed-parameter tractable [11]. In terms of graph parameters, the size of the minimum vertex cover of G is called the *vertex cover number* of G. For any graph, it is easily seen that vertex cover number is greater than or equal to the treewidth [15].

Edge Dominating Set is the problem that given an undirected graph $G = (V, E)$ and an integer k, determines whether there exists a set of edges X of size at most k such that any edge in $E \setminus X$ has at least one incident edge in X. This problem is NP-complete even on bipartite, planar, and bounded-degree graphs [26], but fixed-parameter tractable in general [14]. As we have seen, the Edge Dominating Set problem is equivalent to Dominating Set on line graphs. Moreover, the (optimization) Edge Dominating Set problem is equivalent to Minimum Maximal Matching [26].

Dominating Set is a classical domination problem. This problem is known to be $\Omega(\log n)$-inapproximable, but $O(\log n)$-approximable by a simple greedy algorithm on general graphs [9]. With respect to parameterized complexity, Dominating Set is $W[2]$-complete, unlike Vertex Cover and Edge Dominating Set [10]. Therefore, this problem is well-studied on restricted graphs. Recently, Dawar et al. [8] and Drange et al. [13] considered fixed-parameter tractability and the existence of problem kernels for some sparse graph classes. Their results include the distance version, that is, r-Dominating Set. This approach was generalized to directed graphs because the directed Dominating Set problem is also $W[2]$-complete [23].

The remainder of this paper is organized as follows. In Section 2, we first give basic terminology, notions, and definitions. In Section 3, we show the hardness results of the problems. In Section 4, we give polynomial-time algorithms on trees and fixed-parameter algorithms on general graphs.

2 Preliminaries

In this section, we give notation and definitions. Let $G = (V, E)$ be a directed graph where $|V| = n$ and $|E| = m$. A vertex u is called an *in-neighbor* of v if there exists an edge (u, v) and a vertex w is called an *out-neighbor* of v if there exists an edge (v, w). Moreover, the sets of in (out)-neighbors of v are denoted by $N^{in}(v)$ (resp., $N^{out}(v)$). The number of in (out)-neighbor vertices of v is called the *in (out)-degree* and denoted by $indeg(v) := |N^{in}(v)|$ (resp., $outdeg(v) := |N^{out}(v)|$).

For two vertices u, v, the *distance* from u to v is defined as the number of edges in the shortest path from u to v, denoted by $dist(u, v)$. A vertex u such that $dist(u, v)$ is at most r is called an *r-in-neighbor* of v and a vertex w such that $dist(v, w)$ is at most r is called an *r-out-neighbor* of v. The sets of r-in (out)-neighbors of v are denoted by $N_r^{in}(v)$ (resp., $N_r^{out}(v)$). Note that $N_r^{in}(v) = N^{in}(v)$ and $N_r^{out}(v) = N^{out}(v)$ when $r = 1$.

In an undirected graph G^*, a set of edges such that no edges share an endpoint is called a *matching*. Furthermore, a matching is *maximal* if no proper superset is a matching. An edge dominating set is the edge set E' such that every edge in $E \setminus E'$ is adjacent to at least one edge in E'. Therefore, a maximal matching is an edge dominating set. As for graph parameters, we denote the *treewidth* of G^* by $\mathbf{tw}(G^*)$. The definitions of treewidth and tree decompositions can be found in Section J of the appendix.

A directed graph G is called a *directed acyclic graph* (*DAG*) if G has no directed cycle and a *planar graph* if it can be embedded in the plane without any edges crossing. We mention results on such restricted graphs.

3 Hardness results

In this section, we discuss the hardness of Directed r-In (Out) Vertex Cover and Directed (p, q)-Edge Dominating Set.

3.1 Directed $(0, 1), (1, 0)$-Edge Dominating Set

We first show that Directed $(0, 1), (1, 0)$-Edge Dominating Set is NP-complete. Although Directed $(0, 1)$-Edge Dominating Set is very similar to 1-Out Vertex Cover, there is a large gap in terms of time complexity.

To show this, we introduce a variant of the SAT problem. Let $(X, \mathcal{C})$ be an instance I of SAT, where $X = \{x_1, x_2, \ldots, n\}$ is the set of variables and $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ is the set of clauses. We consider a bipartite graph $G_I = (X \cup \mathcal{C}, E)$, where $E = \{\{x, C\} \mid x \in X, C \in \mathcal{C}$ such that $x \in C$ or $\bar{x} \in C\}$. An instance I of SAT is called *planar* if G_I is planar. Much is known concerning the planar version of SAT. For example, 3SAT is known to be NP-complete even if the instance is restricted to being planar. The restricted version of 3SAT is called Planar 3SAT.

Here, we consider the another restriction of Planar 3SAT. In the restricted instances, each literal appears at most twice, that is, $\forall y \in X \cup \bar{X} : |\{C \in \mathcal{C} \mid y \in C\}| \leq 2$. Instead, the size of each clause is relaxed to be not exactly three but at most three. We call this version Planar At-Most3SAT(L2). We can show the following. Here, (A) next to a theorem means the proof can be found in Section A of the appendix.

Lemma 3.1 (A) Planar At-Most3SAT(L2) *is NP-complete.*

By using Lemma 3.1, we can obtain Theorem 3.2.

Theorem 3.2 Directed $(0, 1), (1, 0)$-Edge Dominating Set *is NP-complete on directed planar graphs such that $indeg(v) + outdeg(v) \leq 3$ holds for any vertex.*

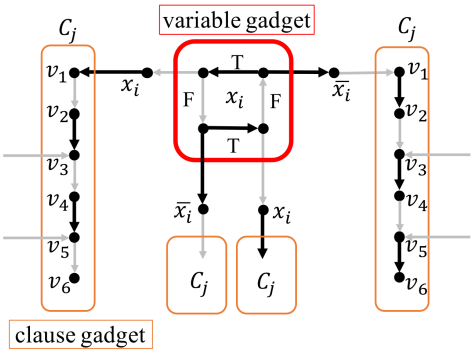


Figure 1: Constructed graph of the reduction from 3SAT to $(0,1)$-EDS (Thick edges are included in the solution.)

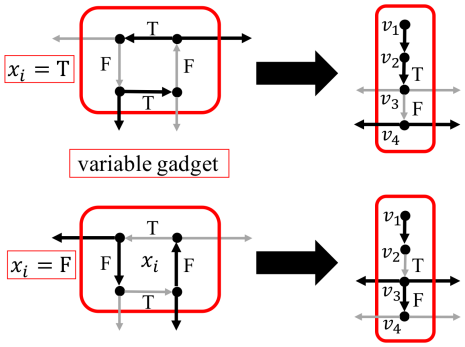


Figure 2: Replacing a cycle by a directed path for a variable's gadget (Thick edges are included in the solution.)

Proof: We only consider Directed $(0,1)$-Edge Dominating Set as the other proof is similar. This problem is clearly in NP. Thus, we show the hardness. The reduction is from Planar At-Most3SAT(L2).

Let n be the number of variables, m be the number of clauses, and l be the number of literals in an input Φ for Planar At-Most3SAT(L2). Then, we construct a graph as in Figure 1. First, we create n cycles of length four corresponding to the variables in Φ and m paths of length five corresponding to the clauses in Φ. For a variable's gadget, if we include the two horizontal edges in the $(0,1)$-edge dominating set, it corresponds to setting the variable to true in Φ. Otherwise, we include the two vertical edges, which corresponds to setting the variable to false. Note that the size of a minimum $(0,1)$-edge dominating set for a cycle of length four is two.

We connect each clause gadget to the variable gadgets corresponding to the literals in the clause, as follows. For $v_1, v_2, \ldots, v_6$ the vertices in the clause gadget, each of v_1, v_3, and v_5 is connected by a path of length two, called a *linking path*, to one of the vertices in a variable gadget. We can observe that there are l linking paths in the constructed graph. For each variable, there are at most two occurrences of true literals and at most two of false literals. Because the variable gadget has four vertices corresponding to literals, by connecting each vertex in the variable gadget to a clause gadget, for any vertex v in the constructed graph, $indeg(v) + outdeg(v) \leq 3$.

Finally, we conclude this proof by obtaining the following lemma.

Lemma 3.3 (B) *An input Φ for* Planar At-Most3SAT(L2) *has a satisfying truth assignment if and only if there exists a $(0,1)$-edge dominating set of size $2n+l+2m$ in a constructed graph.*

By replacing each variable gadget by a path v_1, v_2, v_3, v_4 of length three and connecting vertex v_3 and true literals in a clause, and vertex v_4 and false literals (See Figure 2), we can also show that Directed $(0,1)$-Edge Dominating Set is NP-complete on directed acyclic planar graphs of bounded degree. Note that edge (v_1, v_2) is contained in any $(0,1)$-edge dominating set. Moreover, including edge (v_2, v_3) in the $(0,1)$-edge dominating set corresponds to setting the variable to true and including edge (v_3, v_4) corresponds to setting the variable to false.

Corollary 3.4 Directed $(0,1),(1,0)$-Edge Dominating Set *is NP-complete on directed acyclic planar graphs such that $indeg(v) + outdeg(v) \leq 4$ holds for any vertex.*

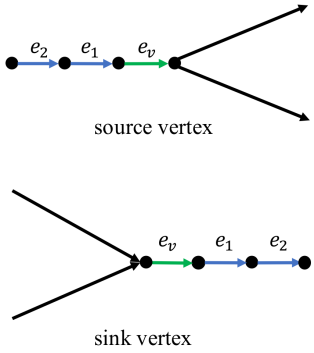


Figure 3: Vertex gadgets for source and sink in the reduction to $(1,1)$-EDS

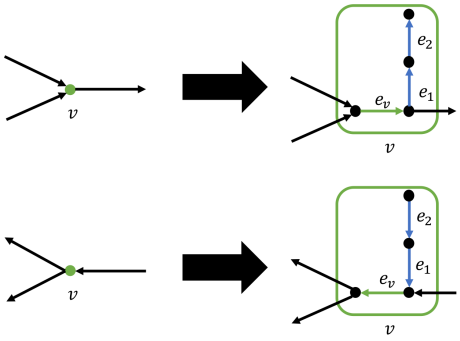


Figure 4: Vertex gadgets for other vertices in the reduction to $(1,1)$-EDS

3.2 Directed $(1,1)$-Edge Dominating Set

As for Directed $(1,1)$-Edge Dominating Set, we obtain a stronger result in terms of a degree constraint. To show this, we first introduce a variant of planar graphs. A graph is *planar almost cubic* if it is planar, there are exactly two vertices of degree two, and the degree of all other vertices is three. We show that Vertex Cover remains NP-complete on planar almost cubic graphs.

Lemma 3.5 (C) Vertex Cover *on planar almost cubic graphs is NP-complete.*

By using Lemma 3.5, we show the following theorem.

Theorem 3.6 Directed $(1,1)$-Edge Dominating Set *is NP-complete on directed acyclic planar graphs such that* $indeg(v) + outdeg(v) \leq 3$ *holds for any vertex.*

Proof: Since Directed $(1,1)$-Edge Dominating Set clearly belongs to NP, we prove the hardness. We show a reduction from Vertex Cover on planar almost cubic graphs. Suppose that we are given an instance (G,k) of Vertex Cover. For an undirected planar almost cubic graph G, we choose two vertices with degree two in G as source and sink vertices. We then arrange each vertex in a horizontal line such that the two vertices of degree two become ends of the line and orient every edge from left to right. Note that there exist exactly one source vertex such that the in-degree is zero and out-degree is two and exactly one sink vertex such that the in-degree is two and out-degree is zero. For other vertices v, it holds that $indeg(v) = 1$ and $outdeg(v) = 2$ or $indeg(v) = 2$ and $outdeg(v) = 1$. Each oriented edge corresponding to an edge in G is called an *original edge*.

Next, we attach paths of length three to the source vertex and the sink vertex as a vertex gadget as in Figure 3. Moreover, we replace any other vertex by a path of length three consisting of e_v, e_1, e_2 as in Figure 4. An edge e_v in G' corresponds to vertex v in G. Let G' be the constructed graph. Since we only replace vertices in G by paths, G' remains planar and acyclic and for any vertex v in G', $indeg(v) + outdeg(v) \leq 3$. Then the following lemma completes the proof.

Lemma 3.7 (D) *An instance* (G,k) *of* Vertex Cover *is a yes-instance if and only if an instance* $(G', n+k)$ *of* Directed $(1,1)$-Edge Dominating Set *is a yes-instance.*

We also obtain the following result on the distance-generalized version.

Corollary 3.8 (E) Directed (p,q)-Edge Dominating Set *is NP-complete on directed acyclic planar graphs such that* $indeg(v) + outdeg(v) \leq 3$ *holds for any vertex when* $p, q \geq 1$.

3.3 Distance generalization

In this subsection, we consider the distance-generalized versions as with Corollary 3.8. We first show that Directed r-In (Out) Vertex Cover and Directed $(0,q),(p,0)$-Edge Dominating Set are NP-complete on directed acyclic planar graphs of bounded degree.

Theorem 3.9 (F) *When r, p and q are greater than 1,* Directed r-In (Out) Vertex Cover *and* Directed $(0,q),(p,0)$-Edge Dominating Set *are NP-complete on directed acyclic planar graphs such that $indeg(v) + outdeg(v) \leq 4$ holds for any vertex v.*

From Theorems 3.2 and 3.9, we can conclude directed r-Dominating Set on directed line graphs is NP-complete.

Corollary 3.10 *The (directed) r-*Dominating Set *problem is NP-complete on directed line graphs even if $r = 1$.*

Finally, we show that Directed r-In (Out) Vertex Cover and Directed (p,q)-Edge Dominating Set are $W[2]$-hard on directed acyclic graphs by a reduction from Set Cover, which is $W[2]$-complete and $\Omega(\log n)$-inapproximable [12, 9].

Theorem 3.11 (G) Directed r-In (Out) Vertex Cover *is $W[2]$-hard on directed acyclic graphs when $r \geq 2$.* Directed (p,q)-Edge Dominating Set *is $W[2]$-hard on directed acyclic graphs when $p \geq 2$ or $q \geq 2$. For these problems, there is no polynomial-time $c \ln k$-approximation algorithm for any constant $c < 1$ unless P=NP, where k is the size of an optimal solution, though they can be approximated within ratio $O(\log n)$ by a greedy algorithm.*

4 Algorithms

In this section, we give polynomial-time algorithms for Directed r-In (Out) Vertex Cover and Directed (p,q)-Edge Dominating Set on trees and fixed-parameter algorithms for Directed $(0,1),(1,0),(1,1)$-Edge Dominating Set on general graphs.

4.1 Algorithms on Trees

We solve Directed (p,q)-Edge Dominating Set by dynamic programming on a graph G for which the underlying undirected graph is a tree, which we can root at an arbitrary vertex; henceforth we use $\hat{G}$ to denote such a rooted tree. When we use the terms *parent*, *child*, *ancestor*, and *descendant*, we are referring to the relationships between vertices in $\hat{G}$.

We first extend the definition of distance to specify distances between vertices and edges. For an edge $e = (u,v)$ and vertices w and x, we define $dist(w,e)$ to be $dist(w,u)$ and $dist(e,x)$ to be $dist(v,x)$. Moreover, for two edges $e = (u,v)$ and $f = (x,y)$, we define $dist(e,f)$ to be $dist(v,x)$. An edge e *i-in-dominates* (or just *in-dominates*) all edges f such that $dist(f,e) \leq i$ and an edge e *j-out-dominates* (or just *out-dominates*) all edges f such that $dist(e,f) \leq j$. In a directed path containing edges e and f, the edges (not including e and f) traversed along the path are *between* e and f. If there are k edges between e and f, then e $(k+1)$-out-dominates f and f $(k+1)$-in-dominates e.

In $\hat{G}$, we use T_v to denote the subtree rooted at the vertex v, and $G[T_v]$ to denote the subgraph of (the directed graph) G induced on the vertices in T_v. We call $G[T_v]$ the *subtree of G rooted at v* and use $conn(v)$ to denote the edge connecting v to its parent, if it has one. We refer to a vertex v as an *out-vertex* if $conn(v)$ is directed from v to its parent and a *in-vertex* if $conn(v)$ is directed from v's parent to v. If v is the root of $\hat{G}$,

it is neither an out-vertex nor an in-vertex. We use $same(v)$ and $diff(v)$ to denote the sets of children of v that are out-vertices and in-vertices, respectively, if v is an out-vertex and that are in-vertices and out-vertices, respectively, if v is an in-vertex. Furthermore, we use $ST(v)$ to denote the set of subtrees rooted at vertices in $same(v)$ and $DT(v)$ to denote the set of subtrees rooted at vertices in $diff(v)$; these are considered to be two different *types of subtrees*. In addition, we use C_s to denote the set of edges between v and vertices in $same(v)$, and C_d to denote the set of edges between v and vertices in $diff(v)$; just as there are two types of subtrees, we consider these set to constitute two types of *connecting edges*.

Our dynamic-programming algorithm processes vertices in an order such that a vertex v is processed after all its descendants, where we use information about the subtrees rooted at the children of v to determine how to dominate edges in $G[T_v]$. We store not only the sizes of edge dominating sets, but also the sizes of edge dominating sets defined in terms of their *reach* and *deficit*, which are measures of the impact of edges inside a subtree in the domination of edges outside the subtree and the impact of edges outside a subtree in the domination of edges inside the subtree.

To see how edges in subtrees rooted at children of v can have an impact on each other, suppose v has two children w and x such that w is an out-vertex and x is a in-vertex. Furthermore, consider an edge e_w in $G[T_w]$ such that $dist(e_w, w) = i$ and an edge e_x in $G[T_x]$ such that $dist(x, e_x) = j$. We can form a directed path that starts at e_w and traverses the edges (w, v) and (v, x) to end at e_x. Since the number of edges between e_w and e_x is $i + j + 2$, this means that e_w $(i + j + 3)$-out-dominates e_x and that e_x $(i + j + 3)$-in-dominates e.

To determine the reach of a set of edges K in $G[T_v]$, we first determine the shortest distance i from an edge in K to v, if v is an out-vertex, or the shortest distance i from v to an edge in K, if v is an in-vertex. When v is an endpoint of an edge in K (that is, $i = 0$), that edge will be able to q-out-dominate an edge outside of $G[T_v]$, if v is an out-vertex, or p-in-dominate an edge outside of $G[T_v]$, if v is an in-vertex. We thus define $maxreach(v) = q$ for each out-vertex v and $maxreach(v) = p$ for each in-vertex v. More generally, we define the *reach of K beyond $G[T_v]$* to be $maxreach(v) - i$.

To measure which edges depend on outside edges for domination, we define the *deficit of K within $G[T_v]$* to be maximum over $dist(e, v)$ (respectively, $dist(v, e)$) over all edges e in $G[T_v]$ not (p, q)-dominated by any edge in K, for v an out-vertex (respectively, in-vertex). Since the edge between v and its parent is the outside edge that can cover the largest deficit, we set $maxdeficit(v) = p$ for v an out-vertex and $maxdeficit(v) = q$ for v a in-vertex. We refer to all edges e with $dist(e, v) \leq d$ (respectively, $dist(v, e) \leq d$) to be *edges of deficit of most d* in $G[T_v]$, for v an out-vertex (respectively, an in-vertex). Should an edge outside a subtree have sufficient reach to dominate all edges of deficit at most d, we will say that the edge *covers the deficit*.

Using these concepts, we say that a set of edges K is a *reach-r-deficit-d edge dominating set* for $G[T_v]$ if the reach of K beyond $G[T_v]$ is r, and K (p, q)-dominates $G[T_v \backslash J]$ where J is the set of edges of deficit at most d in $G[T_v]$. In our algorithm, we use $D[v, r, d]$ to store the minimum number of edges in a reach-r-deficit-d edge dominating set for $G[T_v]$.

When processing a vertex v, we determine $D[v, r, d]$ for values of r and d in the ranges $0 \leq r \leq maxreach(v)$ and $0 \leq d \leq maxdeficit(v)$. For the base cases, for each leaf v in $\hat{G}$, we set $D[v, r, d] = 0$ for all values of r and d. To determine the value of $D[v, r, d]$, we will consider all possible options for adding edges between v and its children to K, a reach-r-deficit-d edge dominating set for $G[T_v]$, as the choice of edges of K in the subtrees rooted at the children of v will be represented by already-computed table entries.

The computation of the table entries depends on the following lemmas.

Lemma 4.1 *The reach of K beyond $G[T_v]$ is $maxreach(v)$ if and only if $K \cap C_s \neq \emptyset$.*

Lemma 4.2 *If $K \cap C_s = \emptyset$, the reach of K beyond $G[T_v]$ is one less than the maximum over all vertices $u \in same(v)$ of the reach of K restricted to $G[T_u]$.*

Lemma 4.3 *For any child u of v, $conn(u)$ covers a deficit of $maxdeficit(u)$ in $G[T_u]$.*

Lemma 4.4 *For any child u of v, if $conn(u)$ is not included in K, then the maximum possible deficit within $G[T_v]$ that can be covered by K is $maxdeficit(u) - 1$.*

Lemma 4.5 *For any child u of v, if $conn(u)$ is not included in K, the deficit in $G[T_v]$ will be covered by any single connecting edge of the opposite type. Thus, if $K \cap C_d \neq \emptyset$, $d = 0$.*

The appendix contains the complete proofs of Theorems 4.6 and 4.7.

Theorem 4.6 (H) *There is an algorithm that solves* Directed (p, q)-Edge Dominating Set *on trees in $O(n^4)$-time.*

Theorem 4.7 (I) *There is an algorithm that solves* Directed r-In (Out) Vertex Cover *on trees in $O(n^4)$-time.*

4.2 Fixed-parameter Algorithm for Directed $(1, 1)$-Edge Dominating Set

In this subsection, we give a $2^{O(k)}n$-time algorithm for Directed $(1, 1)$-Edge Dominating Set. First, we obtain the following lemmas and theorem.

Lemma 4.8 *Given a directed graph G, let G^* be the underlying undirected graph of G and s be the minimum size of* Directed $(1, 1)$-Edge Dominating Set *on G. Then the following inequality holds: $\mathbf{tw}(G^*) \leq 2s$.*

Proof: Let G^* be an undirected graph, $\mathbf{tw}(G^*)$ be the treewidth of G^*, and $\mathbf{vc}(G^*)$ be the size of minimum vertex cover. Then we have $\mathbf{tw}(G^*) \leq \mathbf{vc}(G^*)$ [15].

Let M^* be a minimum maximal matching in G^*. A minimum $(1, 1)$-edge dominating set in G is an (not necessarily minimum) edge dominating set in G^*. If not, there is an edge not dominated by the $(1, 1)$-edge dominating set in G. Moreover, for any edge dominating set D in undirected graphs, $|D| \geq |M^*|$ holds because a minimum maximal matching is a minimum edge dominating set [26]. Therefore, $s \geq |M^*|$ holds.

On the other hand, we have a well-known result that for any maximal matching M, $\mathbf{vc}(G^*) \leq 2|M|$ [17]. Moreover, we already know that $\mathbf{tw}(G^*) \leq \mathbf{vc}(G^*)$ holds. Finally, we can obtain $\mathbf{tw}(G^*) \leq 2s$.

Lemma 4.9 (J) *Given a directed graph G, let G^* be the underlying undirected graph of G. Then given a tree decomposition of G^* of width at most ℓ, there exists an algorithm that solves* Directed $(1, 1)$-Edge Dominating Set *in $25^\ell \ell^{O(1)}n$-time.*

Theorem 4.10 ([6]) *There exists an algorithm that, given an n-vertex graph G and an integer ℓ, in time $2^{O(\ell)}n$ either outputs that the treewidth of G is larger than ℓ, or constructs a tree decomposition of G of width at most $5\ell + 4$.*

Finally, we show a fixed-parameter algorithm for Directed $(1, 1)$-Edge Dominating Set.

Theorem 4.11 *Given an instance* (G, k) *of* Directed $(1,1)$-Edge Dominating Set, *it can be solved in* $2^{O(k)}n$*-time.*

Proof: Given an instance (G, k), we first determine whether the treewidth of G^* is at most $2k$ in $2^{O(k)}n$-time by using Theorem 4.10. If $\mathbf{tw}(G^*) > 2k$, we conclude that it is a no-instance by Lemma 4.8. Otherwise, we use the $25^{\ell}\ell^{O(1)}n$-time algorithm based on a tree decomposition of width at most $10k + 4$ obtained by Theorem 4.10. Therefore, the total running time is $2^{O(k)}n + 25^{10k+4}(10k + 4)^{O(1)}n = 2^{O(k)}n$.

Thus, Directed $(1,1)$-Edge Dominating Set is fixed-parameter tractable with respect to k. We emphasize that the running time of this algorithm is single exponential in k and linear in n. In the same way, we can prove Directed $(0,1),(1,0)$-Edge Dominating Set is fixed-parameter tractable with respect to k.

Theorem 4.12 (K) *Given an instance* (G, k) *of* Directed $(0,1),(1,0)$-Edge Dominating Set, *it can be solved in* $2^{O(k)}n$*-time.*

References

[1] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, J. Obdržálek: The DAG-width of directed graphs, Journal of Combinatorial Theory, Series B, 102(4), pp. 900–923 (2012)

[2] J. Blazewicz, A. Hertz, D. Kobler, D. de Werra: On some properties of DNA graphs. Discrete Applied Mathematics, 98(1), pp. 1–19 (1999)

[3] J. Blazewicz, M. Kasprzak, B. Leroy-Beaulieu, D. de Werra: Finding Hamiltonian circuits in quasi-adjoint graphs, Discrete Applied Mathematics. 156(13), pp. 2573–2580 (2008)

[4] A. A. Bertossi: The edge Hamiltonian path problem is NP-complete. Information Processing Letters, 13(4), pp. 157–159 (1981)

[5] F. Blochl, F. J. Theis, F. Vega-Redondo, E. O'N. Fisher: Vertex centralities in input-output networks reveal the structure of modern economies. Physical Review E, 83(4), 046127 (2011)

[6] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov and M. Pilipczuk: A $c^k n$ 5-Approximation Algorithm for Treewidth. SIAM Journal on Computing, 45(2), pp. 317–378 (2016)

[7] M. Chlebík, J. Chlebíková: Approximation hardness of dominating set problems in bounded degree graphs. Information and Computation, 206(11), pp. 1264–1275 (2008)

[8] A. Dawar, S. Kreutzer: Domination Problems in Nowhere-Dense Classes. In: *Proceedings of the ARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS2009)*, pp. 157–168 (2009)

[9] I. Dinur, D. Steurer: Analytical Approach to Parallel Repetition. In: *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing (STOC2014)*, pp. 624–633 (2014)

[10] R. G. Downey, M. R. Fellows: Fixed-parameter tractability and completeness, Congr. Numer., 87 (1992), pp. 161–187.

[11] R. G. Downey, M. R. Fellows: Parameterized computational feasibility. In: *Proceedings of the Second Cornell Workshop on Feasible Mathematics. Feasible Mathematics II*, pp. 219–244 (1995)

[12] R. G. Downey, M.R. Fellows: Parameterized Complexity. Monographs in Computer Science, Springer, Berlin (1999)

[13] P. G. Drange, M. Dregi, F. V. Fomin, S. Kreutzer, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, F. Reidl and F. S. Villaamil, S. Saurabh, S. Siebertz, S. Sikdar: Kernelization and Sparseness: the case of Dominating Set. In: *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, pp. 31:1–31:14 (2016)

[14] H. Fernau: Edge Dominating Set: Efficient Enumeration-Based Exact Algorithms. In: *Parameterized and Exact Computation: Second International Workshop, IWPEC 2006*, pp. 142–153 (2006)

[15] J. Fiala, P. A. Golovach, J. Kratochvíl: Parameterized complexity of coloring problems: Treewidth versus vertex cover. Theoretical Computer Science, 412(23), pp. 2513–2523 (2011)

[16] R. Ganian, P. Hliněný, J. Kneis, A. Lange, J. Obdržálek, P. Rossmanith: Digraph width measures in parameterized algorithmics. Discrete Applied Mathematics, 168(11), pp. 88–107 (2014)

[17] M. R. Garey, D. S. Johnson: Computers and Intracdtability: A Guide to the Theory of NP-Completeness. W H Freeman & Co (Sd) (1979)

[18] F. Harary, R. Z. Norman: Some properties of line digraphs. Rendiconti del Circolo Matematico di Palermo, 9(2), pp. 161–168 (1960)

[19] T. Johnson, N. Robertson, P.D. Seymour, R, Thomas: Directed Tree-Width. Journal of Combinatorial Theory, Series B, 82(1), pp. 138–154 (2001)

[20] S. Kagawa, S. Suh, K. Hubacek, T. Wiedmann, K. Nansai, J. Minx: CO_2 emission clusters within global supply chain networks: Implications for climate change mitigation. Global Environmental Change, 35, pp. 486–496 (2015)

[21] S. Kagawa, Sangwon Suh, Yasushi Kondo, Keisuke Nansai: Identifying environmentally important supply chain clusters in the automobile industry. Economic Systems Research, 25(3), pp. 265–286 (2013)

[22] S. Kagawa, S. Okamoto, S. Suh, Y. Kondo, K. Nansai: Finding environmentally important industry clusters: Multiway cut approach using nonnegative matrix factorization. Social Networks, 35(3), pp. 423–438 (2013)

[23] S. Kreutzer, S. Tazari: Directed Nowhere Dense Classes of Graphs. In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2012)*, pp. 1552–1562 (2012)

[24] M. L. Lahr, E. Dietzenbacher: Input-Output Analysis: Frontiers and Extensions. Palgrave Macmillan UK (2001)

[25] B. Mohar: Face Covers and the Genus Problem for Apex Graphs, Journal of Combinatorial Theory, Series B, 82(1), pp.102–117 (2001)

[26] M. Yannakakis, F. Gavril: Edge Dominating Sets in Graphs. SIAM Journal on Applied Mathematics, 38(3), pp. 364-372 (1980)

A Proof of Lemma 3.1

This problem clearly belongs to NP. We show the hardness. The hardness is shown by a reduction from Planar 3SAT. Let $I = (X, \mathcal{C})$ be an instance of Planar 3SAT. If all the literals appear at most twice, we do not need to do anything. Otherwise, there is a literal that appears at least three times. We assume that the literal x is positive, without loss of generality. Let ℓ be the total number of appearances of x and $\bar{x}$. We then number the x's and $\bar{x}$'s according to the order of appearance. Next, we create ℓ new variables $x^{(1)}, x^{(2)}, \ldots, x^{(\ell)}$, and new clauses $C^x_1 = \{x^{(1)}, \bar{x}^{(2)}\}, C^x_2 = \{x^{(2)}, \bar{x}^{(3)}\}, \ldots, C^x_{\ell-1} = \{x^{(\ell-1)}, \bar{x}^{(\ell)}\}, C^x_\ell = \{x^{(\ell)}, \bar{x}^{(1)}\}$. These new clauses are introduced in order to guarantee that all $x^{(i)}$'s takes same value (1 or 0). Furthermore, for C in which i-th x or $\bar{x}$ appears, we define C' to be the same as C except that x or $\bar{x}$ is replaced with $x^{(i)}$ or $\bar{x}^{(i)}$. Note that $x^{(i)}$ (or $\bar{x}^{(i)}$) appears only in C' and C^x_i (resp., C^x_{i+1}), that is, at most twice. It is obvious that the original instance is satisfiable if and only if the new instance is, where x's and $\bar{x}$'s are replaced with $x^{(i)}$ or $\bar{x}^{(i)}$, $i = 1, 2, \ldots, \ell$. By doing this replacement for all the variables, we obtain a new equivalent SAT instance I', whose clauses contain at most three literals and literals appear at most twice.

We then show that the new instance preserves planarity. We give a planar drawing of $G_{I'}$ based on the planar drawing of G_I. That is, we replace vertex x with the cycle $(x^{(1)}, C^x_1, x^{(2)}, \ldots, x^{(\ell)}, C^x_\ell, x^{(1)})$. Obviously, a cycle has a planar drawing. Edges between x's and C's are rewired according to the order of appearance of x's. This does not yield any crossing. See figures 5 and 6. This completes the proof. □

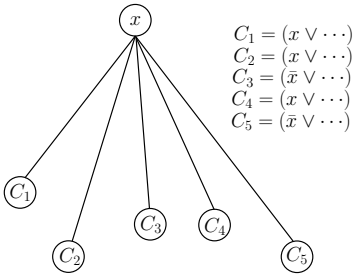


Figure 5: Original clauses where x's appear

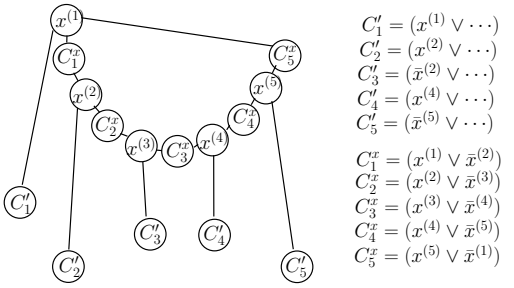


Figure 6: Replaced clauses

B Proof of Lemma 3.3

Given a truth assignment for Φ, we take the corresponding vertical or horizontal pair of edges in each variable gadget since we need exactly two edges in order to dominate the cycle. For each linking path, we add to the $(0, 1)$-edge dominating set either the edge incident on a variable gadget if that edge is not dominated, and otherwise the other edge on the linking path. For a clause gadget, if a literal in the clause is true, an edge adjacent to the vertex corresponding to the literal is dominated. Note that if at least one literal is true, the clause gadget can be dominated by only two edges; otherwise we need three edges to dominate it. Because every clause is true and there are n variable gadgets, l linking paths, and m clauses, we can dominate all edges by $2n + l + 2m$ edges.

Conversely, suppose that we are given a $(0, 1)$-edge dominating set of size $2n + l + 2m$. We need at least two edges to dominate each variable gadget and at least one edge to dominate each linking path. Because there are n variable gadgets and l linking paths, all clause gadgets must be dominated by at most $2m$ edges. Note that we need at least

two edges to dominate a clause gadget. Two edges will suffice only when the $(0,1)$-edge dominating set contains at least one adjacent edge in a linking path. This means that each clause has at least one true literal. Moreover we can observe that a $(0,1)$-edge dominating set contains exactly two, one, and two edges in each variable gadget, linking path and clause gadget, respectively. In a variable gadget, either the vertical or horizontal pair of edges is in the $(0,1)$-edge dominating set. Thus, we give an assignment for each variable such that if the horizontal pair is included, we assign true, and otherwise false. Because each clause has at least one true literal, this is a satisfying truth assignment. □

C Proof of Lemma 3.5

We show a reduction from Vertex Cover on planar cubic graphs, which is NP-complete [25]. Given a planar cubic graph $G = (V, E)$, choose an edge (u, v) and insert two vertices w and x to (u, v), that is, change (u, v) to a path consisting of u, w, x, v. Let $G' = (V', E')$ be the constructed graph where $V' = V \cup \{w, x\}$ and $E' = (E \setminus \{(u, v)\}) \cup \{(u, w), (w, x), (x, v)\}$. Note that the degree of w and x is two, and the degree of any other vertices is three. Since we only replace an edge by a path of length three in a planar graph, G' remains planar. Thus, G' is a planar almost cubic graph.

Then, we show that an instance (G, k) of Vertex Cover on planar cubic graphs is a yes-instance if and only if an instance $(G', k + 1)$ of Vertex Cover on planar almost cubic graphs is a yes-instance.

Let C be a vertex cover of size at most k in G. Then, C covers every edge in $E\setminus\{(u, v)\}$ and at least one of (u, w) and (x, v) in G' since at least one of u and v is in C. If both of (u, w) and (x, v) are covered by C, we add either w or x. Because (w, x) is the only edge not covered, we can obtain a vertex cover of size at most $k + 1$. Otherwise, without loss of generality, we suppose that (u, w) is not covered. Now, the only edges not covered by C are (u, w) and (w, x). Therefore, by setting $C' = C \cup \{w\}$, we obtain a vertex cover in G' of size at most $k + 1$.

Conversely, let C' be a vertex cover of size at most $k + 1$ in G'. To cover edge (w, x), C' contains at least one of w and x. First, we suppose C' contains exactly one of w and x. Without loss of generality, we suppose that $w \in C'$ and $x \notin C'$. Then we can observe that $v \in C'$ in order to cover (x, v). Thus $C' \setminus \{w\}$ is vertex cover of size at most k in G because v covers (u, v) in G. If C' contains both x and w, we can replace x by v because (w, x) is covered by w. Then, $(C' \setminus \{w, x\}) \cup \{v\}$ is a vertex cover of size at most k in G because v covers (u, v) in G. □

D Proof of Lemma 3.7

Let $G = (V, E)$ and C be a vertex cover of size at most k in G. In G', we add each e_1 to the solution set in order to dominate e_2 and e_v for each vertex gadget. We also add e_v's such that $v \in C$ in G. Because edge e_v corresponds to v in G, each edge e_v dominates all original edges. Therefore, such a solution set is a $(1, 1)$-edge dominating set and its size is $n + k$.

Conversely, let K be a $(1, 1)$-edge dominating set of size at most $n + k$ in G'. To dominate e_2, K contains either e_1 or e_2 for each vertex gadget. If K contains e_2 for some gadgets, e_2 can be replaced by e_1 and maintain a $(1, 1)$-edge dominating set. Thus, we can assume that K contains all e_1's and does not contain any e_2's. Note that every e_v is dominated by e_1. From this fact, if K includes some original edges, we can exclude them because they can only dominate e_v's. Therefore, we can suppose K only contains e_1's and e_v's. Because the set of e_v's in K dominates all original edges, the set of vertices in G

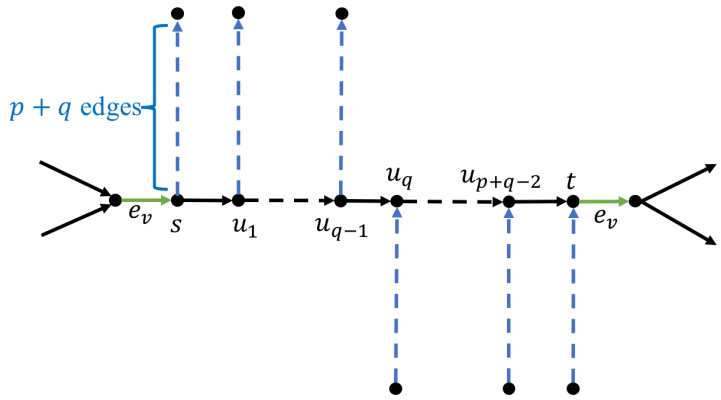


Figure 7: Replacing an edge and attaching path gadgets for the reduction to (p, q)-EDS

corresponding to e_v's in K covers all edges in G. Thus, this is vertex cover for G. Since the size of K is at most $n + k$ and the number of e_1's included in K is n, the number of e_v's in K is at most k. Therefore, the size of the vertex cover is at most k. □

E Proof of Corollary 3.8

For graph G' in Theorem 3.6, let $G' = (V', E')$, and E_v, E_1 and E_2 be the sets of e_v's, e_1's and e_2's, respectively. First, we remove every edge in $E_1 \cup E_2$ from G'. Then, we replace each original edge (s, t) by a path of length $p + q - 1$, consisting of $s, u_1, u_2, \ldots, u_{p+q-2}, t$. We consider the edge (u_{q-1}, u_q) to correspond to an edge in E.

Next, we attach a path of length $p + q$, called a *path gadget*, to each u_i as in Figure 7. As with Directed $(1, 1)$-Edge Dominating Set, for each edge $(w, x) \in E_v$ except for the source and the sink, if $indeg(w) = 1$, we attach a path gadget to w. Otherwise, that is, if $outdeg(x) = 1$, we attach it to x. Finally, we attach a path gadget to each of the source and the sink.

Let G'' be the created graph. Because we only attach the paths to vertices satisfying $indeg(v) + outdeg(v) \leq 2$, it holds that $indeg(v) + outdeg(v) \leq 3$ for any v. Furthermore, G'' remains planar and acyclic.

Then, we show that an instance (G, k) of Vertex Cover is a yes-instance if and only if an instance $(G'', (p + q - 1)n + k)$ of Directed (p, q)-Edge Dominating Set is a yes-instance. The rest of the argument is the same as that in the proof for Directed $(1, 1)$-Edge Dominating Set. □

F Proof of Theorem 3.9

We show a reduction from Vertex Cover on planar cubic graphs, which is NP-complete [25], to Directed r-Out Vertex Cover.

Given an instance $(G = (V, E), k)$ of Vertex Cover, we create a graph G'. First, we replace each edge $e = (u, w)$ in E by an edge gadget as in Figure 8. We insert a center vertex v_e^1 in each edge e, that is, we replace one edge $e = (u, w)$ by two directed edges

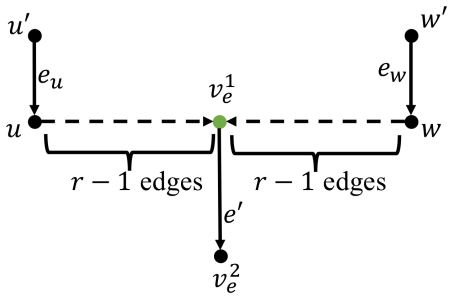


Figure 8: Replacing $e = (u, v)$ by a gadget for r-VC

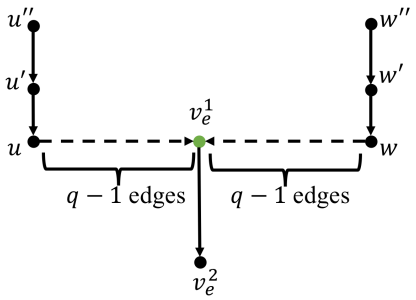


Figure 9: Replacing $e = (u, v)$ by a gadget for (p, q)-EDS

(u, v_e^1) and (w, v_e^1). We denote the set of v_e^1's by V_e^1. Moreover, for each e in E we add a vertex v_e^2 and an edge $e' = (v_e^1, v_e^2)$. Let V_e^2 be the set of v_e^2's and E' be the set of e''s.

Next, we replace (u, v_e^1) by a directed path from u to v_e^1 of length $r-1$ and (w, v_e^1) by a directed path from w to v_e^1 of length $r-1$. Let V_P be the set of vertices in the directed paths except for u, v_e^1, and w and E_P be the set of edges in the directed paths.

As a vertex gadget, we attach an edge $e_v = (v', v)$ to every v. We denote the set of v''s by V' and the set of e_v's by E_v.

Let $G' = (V \cup V' \cup V_e^1 \cup V_e^2 \cup V_P, E' \cup E_v \cup E_P)$ be the created graph. Note that G' remains planar and acyclic. Moreover, for any vertex v in G', $indeg(v) + outdeg(v) \leq 4$ holds because we only replace each edge by a path and attach edges.

Now, we show that an instance (G, k) of Vertex Cover is a yes-instance if and only if an instance $(G', n+k)$ of Directed r-Out Vertex Cover is a yes-instance. Suppose that we are given a vertex cover $C = \{v_1, \ldots, v_k\} \subseteq V$ of size k. Let $C' = V' \cup C$. Then the set V' covers all edges in $E_v \cup E_P$. Furthermore, since C is a vertex cover in G, it covers every edge in E' due to the construction. Thus, C' is an r-out-vertex cover of size at most $n + k$.

Conversely, suppose that we are given an r-out-vertex cover C' of size $n+k$. Note that C' always includes V' in order to cover each edge $e_v = (v', v)$.

Because V' covers every edge in $E_v \cup E_P$, and all edges covered by $V_e^1 \cup V_e^2 \cup V_P$ are covered by V, if C' includes a vertex in $V_e^1 \cup V_e^2 \cup V_P$, we can replace it by a vertex in V while maintaining the coverage of each edge by C'. Let $C = C' \setminus V' \subseteq V$. Then C is a vertex cover in G of size k because C covers all edges in E' corresponding to E and $|V'| = n$.

By attaching a new edge (v'', v') for each v' and a corresponding additional vertex v'' and setting $r = q$ (See Figure 9), we can similarly obtain the proof for Directed $(0, q), (p, 0)$-Edge Dominating Set. □

G Proof of Theorem 3.11

To prove Theorem 3.11, we introduce the Set Cover problem as follows.

Definition G.1 (Set Cover) Set Cover *is the problem that given a collection of sets* $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ *defined over a domain of elements* $U = \{e_1, e_2, \ldots, e_n\}$ *and an integer* k*, determines whether there exists a subcollection* $\mathcal{S}' \subseteq \mathcal{S}$ *of size at most* k *such that* $\bigcup_{S \in \mathcal{S}'} S = U$.

Set Cover is known to be $W[2]$-complete and $\Omega(\log n)$-inapproximable [12, 9].

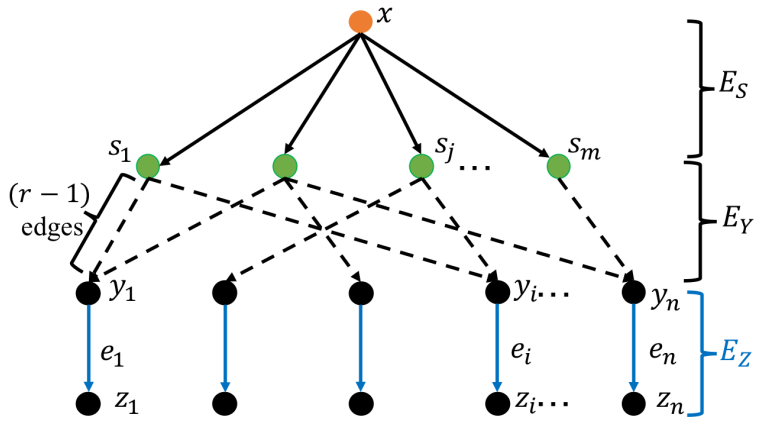


Figure 10: Constructed graph of the reduction from Set Cover to Directed r-Out Vertex Cover

G.1 Directed r-Out Vertex Cover

We show that there is a parameterized reduction from Set Cover to Directed r-In (Out) Vertex Cover. In this proof, we focus on Directed r-Out Vertex Cover since we can also prove the other case by a slight modification. If $k = m$, it is trivial. Hence, we suppose that $k < m$. Without loss of generality, suppose that every $S \in \mathcal{S}$ is not empty.

Given an instance of Set Cover, that is, $(\mathcal{S}, U, k)$, we create a graph $G = (V, E)$ where $V = \{x\} \cup V_S \cup V_Y \cup V_Z$ and $E = E_S \cup E_Y \cup E_Z$ by the following method (See Figure 10). Let x be a *super vertex* and $V_S = \{s_1, s_2, \ldots, s_m\}$ be a vertex set corresponding to $\mathcal{S}$ in Set Cover. Then we connect x and each $s_j \in V_S$ by adding edge (x, s_j) where $1 \leq j \leq m$. We denote the set of (x, s_j)'s by E_S.

Let $V_Y = \{y_1, \ldots, y_n\}$ and $V_Z = \{z_1, \ldots, z_n\}$. Then, for each element e_i of Set Cover, we create the corresponding edge (y_i, z_i) for $1 \leq i \leq n$. Finally, we connect each s_j and y_i by a path of length $r-1$ from s_j to y_i if $S_j \in \mathcal{S}$ covers element $e_i \in U$. We denote the set of (y_i, z_i)'s by E_Z and the set of edges in the paths of length $r-1$ by E_Y. Obviously, G is directed acyclic due to the construction. Constructing the graph can be done in time polynomial in the size of the input to Set Cover. By reversing the orientation of every edge, we can also prove that Directed r-In Vertex Cover is $W[2]$-hard.

Then we show that an instance of Set Cover $(\mathcal{S}, U, k)$ is a yes-instance if and only if the instance of Directed r-Out Vertex Cover $(G, r, k+1)$ is a yes-instance, where G is the graph created by the above procedure.

If $(\mathcal{S}, U, k)$ is a yes-instance of Set Cover, let $\mathcal{S}^* = \{S_{j_1}, S_{j_2}, \ldots, S_{j_k}\}$ be a solution where $\{j_1, j_2, \ldots, j_k\} \subseteq \{1, 2, \ldots, m\}$. In G, we select $C = \{s_{j_1}, s_{j_2}, \ldots, s_{j_k}, x\}$. Then we can immediately confirm that C covers every edge in E since x covers all edges in $E \setminus E_Z$.

Conversely, if (G, r, k) is a yes-instance of Directed r-Out Vertex Cover, let C be a solution of size $k+1$. Note that C contains x because every edge in E_S is covered by only vertex x. If C includes a vertex u_j on the path from s_j to z_i, then because every edge covered by u_j is covered by s_j, we can replace u_j by vertex s_j if $s_j \in V_S \setminus C$, and any vertex $s \in V_S \setminus C$ otherwise, keeping every edge covered by C.

Therefore, we assume that $C \setminus \{x\} \subseteq V_S$ and let $C = \{s_{j_1}, s_{j_2}, \ldots, s_{j_k}, x\}$. Then we

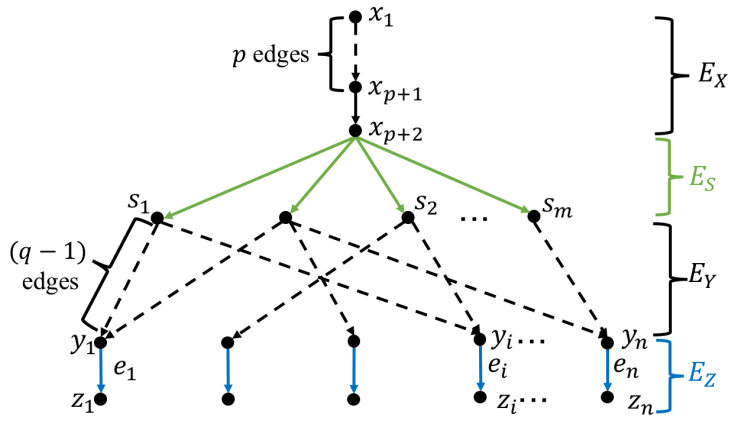


Figure 11: Constructed graph of the reduction from Set Cover to Directed (p, q)-Edge Dominating Set

choose $\mathcal{S}^* = \{S_{j_1}, S_{j_2}, \ldots, S_{j_k}\} \subseteq \mathcal{S}$. Note that element e_i is covered by S_j if and only if edge $e_i = (y_i, z_i)$ is covered by s_j in G since vertex x does not cover any edge in U. From the construction of G, we conclude that $\bigcup_{S \in \mathcal{S}^*} S = U$, and hence $(\mathcal{S}, U, k)$ is a yes-instance.

For Set Cover, there is no polynomial-time algorithm whose approximation ratio is better than $c \ln |U|$ with any $c < 1$ unless P=NP. Since the parameter k of Set Cover corresponds to $(k+1)$ of an r-out vertex cover, it is hard to find an r-out vertex cover with size at most $(k+1)c \ln |U|$. This $|U|$ is below bounded by k, and this implies that r-Out Vertex Cover is hard to approximate within $c \ln k$ with any $c < 1$.

G.2 Directed (p, q)-Edge Dominating Set

By almost the same method, we obtain the proof for Directed (p, q)-Edge Dominating Set. That is, we replace a super vertex by a path of length $p + 1$.

We show that there is a parameterized reduction from Set Cover to Directed (p, q)-Edge Dominating Set. In this proof, we show only the case in which $q \geq 2$ since we can also prove the other case by a slight modification.

Given an instance of Set Cover, that is, $(\mathcal{S}, U, k)$, we create graph G by the following method (See Figure 11). This graph is similar to a constructed graph for Directed r-In (Out) Vertex Cover. We replace only r by $q - 1$ and x by a path $x_1, x_2, \ldots, x_{p+2}$ of length $p+1$. Let E_X be an edge set of the path. Note that each edge (x_{p+2}, s_j) corresponds to $S_j \in \mathcal{S}$. The rest of the proof is almost the same as Directed r-In (Out) Vertex Cover.

We show that an instance of Set Cover $(\mathcal{S}, U, k)$ is a yes-instance if and only if the instance of Directed (p, q)-Edge Dominating Set $(G, k + 1)$ is a yes-instance, where G is the graph created by the above procedure.

Given a yes-instance of Set Cover $(\mathcal{S}, U, k)$, let $\mathcal{S}^* = \{S_{j_1}, \ldots, S_{j_k}\}$ be a solution where $\{j_1, \ldots, j_\ell\} \subseteq \{1, \ldots, m\}$. In graph G, we select the edge set $K = \{(x_{p+2}, s_{j_1}), \ldots, (x_{p+2}, s_{j_k}), (x_{p+1}, x_{p+2})\}$. Then we can immediately confirm that edge (x_{p+1}, x_{p+2}) dominates every edge in $E \setminus E_Z$ and $K \setminus \{(x_{p+1}, x_{p+2})\}$ dominates every edge in E_Z.

Conversely, if $(G, k+1)$ is yes-instance of Directed (p,q)-Edge Dominating Set, let K be a solution of size $k+1$. Note that K contains at least one edge in E_X and we can assume that K only contains one edge (x_{p+1}, x_{p+2}) because it dominates every edge dominated by $e \in E_X \setminus \{(x_{p+1}, x_{p+2})\}$.

If there is edge $e \in (E \setminus E_X \setminus E_S) \cap K$ on the path from s_j to z_i, we can replace e by $(x_{p+2}, s_j) \in E_S$ if it is not in K, otherwise by any edge $e_s \in E_S$, keeping every edge of G dominated by K. That reason is why every edge in $E \setminus E_Z$ is already dominated by (x_{p+1}, x_{p+2}). Therefore, we assume that $K \subseteq E_S$.

Let $K = \{(x_{p+2}, s_{j_1}), (x_{p+2}, s_{j_2}), \ldots, (x_{p+2}, s_{j_k}), (x_{p+1}, x_{p+2})\}$ be a solution of size $k+1$. We then choose $\mathcal{S}^* = \{S_{j_1}, S_{j_2}, \ldots, S_{j_k}\} \subseteq \mathcal{S}$. Note that element e_i is covered by S_j if and only if edge (y_i, z_i) is dominated by (x_{p+2}, s_j) in G. From the construction of G, we conclude that $\bigcup_{S \in \mathcal{S}^*} S = U$, and hence $(\mathcal{S}, U, k)$ is yes-instance.

By reversing the orientation of every edge, we can also prove the case in which $p \geq 2$.

For Set Cover, there is no polynomial-time algorithm whose approximation ratio is better than $c \ln |U|$ with any $c < 1$ unless P=NP. Since the parameter k of Set Cover corresponds to $(k+1)$ of a (p,q)-edge dominating set, it is hard to find an (p,q)-edge dominating set with size at most $(k+1)c \ln |U|$. This $|U|$ is below bounded by k, and this implies that (p,q)-Edge Dominating Set is hard to approximate within $c \ln k$ with any $c < 1$.

Since we can describe Directed r-In (Out) Vertex Cover and Directed (p,q)-Edge Dominating Set as Set Cover, they can be approximated within ratio $O(\log n)$ by a greedy algorithm. □

H Proof of Theorem 4.6

The lemma below results from the fact that the size of the minimum edge dominating set of a smaller subgraph is never bigger than the size of a minimum edge dominating set for a larger subgraph.

Lemma H.1 *The following properties hold:*

1. $D[u, r, d] \leq D[v, r, d]$ *for* u *a descendant of* v,
2. $D[v, r, d] \leq D[v, r', d]$ *for* $r \leq r'$, *and*
3. $D[v, r, d] \leq D[v, r, d']$ *for* $d \geq d'$.

Corollary H.2 *For a given value of* v, *the minimum values will be* $D[v, 0, maxdeficit(v)]$; *the solution to* Directed (p,q)-Edge Dominating Set *will be* $D[v, 0, 0]$, *for* v *the root of* $\hat{G}$.

Before detailing the calculation of $D[v, r, d]$, we first consider the roles of C_s, $ST(v)$, C_d, and $DT(v)$. For both types of subtrees and connecting edges, a single edge may cover the deficit for all the subtrees and connecting edges of the opposite type. However, the roles of the two types of subtrees and connecting edges are not symmetric. Specifically, since edges in C_d cannot form directed paths with $conn(v)$, only edges in C_s and $ST(v)$ will have an impact on the reach r and deficit d for v. In contrast, in the choice of edges for K in $G[T_v]$, all edges in C_d and all deficits in trees in $DT(v)$ must be covered.

We first observe in the following two lemmas that a single edge in C_s will suffice to ensure that $r = maxreach(v)$, and that if there is no edge in $K \cap C_s$, then it suffices for a single subtree in $ST(v)$ to have reach $r+1$. In our calculations, this implies that the reach for every other tree in $ST(v)$ can be assumed to be 0 (or for all to have reach 0, if $K \cap C_s \neq \emptyset$).

Lemma H.3 *The reach of K beyond $G[T_v]$ is $maxreach(v)$ if and only if $K \cap C_s \neq \emptyset$.*

Proof: Clearly, an edge in $K \cap C_s$ can $maxreach(v)$-dominate edges outside of $G[T_v]$.

If instead $K \cap C_s = \emptyset$, then the reach of K beyond $G[T_v]$ is strictly less than $maxreach(v)$, since it will be one less than the maximum over all $u \in same(v)$ of the reach of K beyond $G[T_u]$, and since $u \in same(v)$, and $maxreach(u) = maxreach(v)$.

Lemma H.4 *If $K \cap C_s = \emptyset$, the reach of K beyond $G[T_v]$ is one less than the maximum over all vertices $u \in same(v)$ of the reach of K restricted to $G[T_u]$.*

To determine deficit, we first make a few observations that apply to subtrees of both types. For any child u of v, if $conn(u)$ is in K, then we can assume that the deficit within $G[T_u]$ is $maxdeficit(u)$. Furthermore, if $conn(u)$ is not in K, then the maximum deficit possible within $G[T_u]$ is $maxdeficit(u) - 1$. This is summarized in the following two lemmas.

Lemma H.5 *For any child u of v, $conn(u)$ covers a deficit of $maxdeficit(u)$ in $G[T_u]$.*

Lemma H.6 *For any child u of v, if $conn(u)$ is not included in K, then the maximum possible deficit within $G[T_v]$ that can be covered by K is $maxdeficit(u) - 1$.*

We make use of the following terminology to capture the idea of determining whether or not to include $conn(u)$ in K, where the deficit can be an arbitrary value j. We define $min(u, j) = \min\{1 + D[u, 0, maxdeficit(u)], D[u, 0, j]\}$; the first case represents choosing to include $conn(u)$ and the second case not to include $conn(u)$. If the latter is the smaller for all $u \in same(v)$ ($u \in diff(v)$, respectively), we may choose to add an arbitrary edge in C_s (c_d, respectively) to cover the deficit for subtrees of the opposite type. Accordingly, we define $\alpha_s(j) = 1$ if there exists $u \in same(v)$ such that $1 + D[u, 0, maxdeficit(u)] \leq D[u, 0, j]$, and 0 otherwise. We define $\alpha_d(j)$ similarly, for $u \in diff(v)$.
As a consequence of Lemma H.6, we observe that the reach of a connecting edge of the opposite type is sufficient to cover any deficit.

Lemma H.7 *For any child u of v, if $conn(u)$ is not included in K, the deficit in $G[T_v]$ will be covered by any single connecting edge of the opposite type. Thus, if $K \cap C_d \neq \emptyset$, $d = 0$.*

We consider using a subtree T_u in $DT(v)$ to cover the deficit in a subtree rooted at a vertex w in $same(v)$. In this case, the reach beyond $G[T_x]$ must be two greater than the deficit in $G[T_w]$, due to the need for the path to pass through $conn(x)$ and $conn(w)$ en route to $G[T_x]$. This reach for a single subtree in $DT(v)$ will cover the deficit for all subtrees in $ST(v)$.

The analogous result holds for the reach of a single subtree in $ST(v)$ covering the deficit for all subtrees rooted at vertices in $DT(v)$. The key difference is that for the subtrees in $ST(v)$, there remains the option of not covering the deficit. No such option exists for subtrees in $DT(v)$, for which all deficits must be covered.

To determine the value of $D[v, r, d]$, we will consider all possible options for adding edges between v and its children to K, a reach-r-deficit-d edge dominating set for $G[T_v]$, as the choice of edges of K in the subtrees rooted at the children of v will be represented by already-computed table entries. We demonstrate how to compute $D[v, r, d]$ as four different cases, depending on the values of d and r.
Case 1: $r = maxreach(v)$ and $d > 0$

Since $r = maxreach(v)$, by Lemma H.3 $K \cap C_s \neq \emptyset$, and since $d > 0$, by Lemma H.7 $K \cap C_d = \emptyset$.

Since K contains at least one edge in C_s, by Lemma H.7, all edges in C_d are covered, as well as deficits of $maxdeficit(u) - 1$ for each $u \in diff(v)$, which by Lemma H.6 is the maximum possible.

Any $u \in same(v)$ for which $conn(u) \in K$ will contribute $D[u, 0, maxdeficit(u)]$, by Lemma H.5.

If for $u \in same(v)$, $conn(u) \notin K$, then $G[T_u]$ can have a deficit of $d - 1$, which in conjunction with $conn(u)$ will result in deficit d. Thus, for each $u \in same(v)$, the contribution is $min(u, d - 1)$. We may need to add an arbtirary edge in C_s to ensure $r = maxreach(v)$, as indicated by $\alpha_s(d - 1)$.

$D[v, r, d] = \sum_{u \in diff(v)} D[u, 0, maxdeficit(u)-1] + \sum_{u \in same(v)} min(u, d-1) + \alpha_s(d-1)$.

Case 2: $r < maxreach(v)$ and $d > 0$

Since $r < maxreach(v)$, by Lemma H.3 $K \cap C_s = \emptyset$, and since $d > 0$, by Lemma H.7 $K \cap C_d = \emptyset$.

To ensure that the edges in C_d are covered and that the deficit of all trees in $DT(v)$ are covered, we make use of an edge in the subtree in $ST(v)$ that gives rise to reach r. We consider all choices of $w \in same(v)$, for a contribution of $D[w, r + 1, d - 1]$ for that choice, and $D[u, 0, d - 1]$ for all $u \in same(v) \backslash \{w\}$. For each $u \in diff(v)$, the contribution will be $D[u, 0, r - 1]$.

Thus,

$D[v, r, d] = min_{w \in same(v)} \{D[w, r + 1, d - 1] + \sum_{u \in same(v) \backslash \{w\}} D[u, 0, d - 1]\}$
$+ \sum_{u \in diff(v)} D[u, 0, r - 1]$.

Case 3: $r = maxreach(v)$ and $d = 0$

Since $r = maxreach(v)$, by Lemma H.3 $K \cap C_s \neq \emptyset$. Any $u \in same(v)$ for which $conn(u) \in K$ will contribute $1 + D[u, 0, maxdeficit(u)]$, by Lemma H.5.

The edges in C_s cover all deficits of trees T_u in $DT(v)$ up to a deficit of $maxdeficit(u) - 1$; the only other option for a tree in $DT(v)$ is to include $conn(u)$ in K to cover a deficit of up to $maxdeficit(u)$.

To cover the deficits of trees in $ST(v)$, we consider the minimum over all choices of j in the range from 0 to $maxdeficit(v) - 1$ as the deficit for any $T_u \in ST(v)$ such that $conn(u) \notin K$. When $j = maxdeficit(v) - 1$, an edge in C_d must be in K. For all other values of j, one tree in $DT(v)$ must have reach $j + 2$, and all others will have reach 0.

We set $D[v, r, d] = \min_{j \in \{0, \ldots, maxdeficit(v)-1\}} Cost(j)$ for $Cost(j)$ as defined below.

$Cost(maxdeficit(v) - 1) = \sum_{u \in same(v) \cup diff(v)} min(u, maxdeficit(u) - 1)$
$+ \alpha_s(maxdeficit(u) - 1) + \alpha_d(maxdeficit(u) - 1)$

For any value of j in the range from 0 to $maxdeficit(v) - 2$,

$Cost(j) = \min_{w \in diff(v)} \{D[w, j + 2, maxdeficit(w) - 1]$
$+ \sum_{u \in diff(v) \backslash \{w\}} D[u, 0, maxdeficit(u) - 1] + \sum_{u \in same(v)} min(u, j) + \alpha_s(j)\}$.

Case 4: $r < maxreach(v)$ and $d = 0$

Since $r < maxreach(v)$, by Lemma H.3 $K \cap C_s = \emptyset$.

If K does not contain any edge in C_d, we need to cover the deficits in trees in $ST(v)$ and $DT(v)$ as well as the edges in C_s and C_d, all without being able to select any of the connecting edges. Since an edge in a subtree in $DT(v)$ can cover the deficit for all trees in $ST(v)$ (as well as all edges in C_s), we consider all trees in $ST(v)$ to have the same deficit, j. We then require a single subtree in $DT(v)$ to have reach $j + 2$, with the rest having reach 0. We consider all possible values of j, $0 \leq j \leq maxdeficit(v) - 2$, and all choices of a subtree in $DT(v)$ to have sufficient reach.

Similarly, we need to ensure that the edges in C_d are covered and that the deficits of all trees in $DT(v)$ are covered. This will be accomplished by an edge in a subtree in $ST(v)$,

which is also the one that results in reach r (the rest will have reach 0). Thus all subtrees in $DT(v)$ will have the same deficit, $r-2$. This immediately implies that $r \geq 2$ in this case. We consider all possible choices of subtrees in $ST(v)$.

If instead K contains any edge in C_d, then by Lemma H.7, K covers the deficits in all subtrees in $ST(v)$, as well as all edges in C_s. To cover the edges of C_d and the deficits in all trees in $DT(v)$, we consider $D[w, r+1, maxdeficit(w)-1]$, for all possibilities of $w \in same(v)$, and $D[u, 0, maxdeficit(u)-1]$ for all $u \in same(v)\backslash\{w\}$ (only one tree needs to contribute to the reach of r for v). Since this reach will cover a deficit of $r-1$ in any tree in $DT(v)$, the contribution for each $u \in diff(v)$ will be $min(u, r-1)$, with the possible addition of an arbitrary edge in C_d (represented by $\alpha_d(r-1)$.

We set $D[v, r, d] = \min_{j \in \{0,\ldots,maxdeficit(v)-1\}} Cost(j)$ for $Cost(j)$ as defined below; the value $j = maxdeficit(v) - 1$ handles the case in which K contains an edge in C_d. Thus,

$Cost(maxdeficit(v)-1) = \min_{w \in same(v)}\{D[w, r+1, maxdeficit(w)-1] +$
$\sum_{u \in same(v)\backslash\{w\}} D[u, 0, maxdeficit(u)-1] + \sum_{u \in diff(v)} min(u, r-1) + \alpha_d(r-1)\}$.

For any value of j in the range from 0 to $maxdeficit(v)-2$,

$Cost(j) = \min_{w \in same(v), x \in diff(v)}\{D[w, r+1, j] + \sum_{u \in same(v)\backslash\{w\}} D[u, 0, j] + D[x, j+2, r-2] + \sum_{u \in diff(v)\backslash\{x\}} D[u, 0, r-2]\}$.

It is not difficult to see that the algorithm can be executed in polynomial time, since there are $O(n^3)$ table entries to fill, each of which can be filled in time $O(n^4)$.

I Proof of Theorem 4.7

As we will be r to denote reach, here we will consider the (renamed) Directed q-Out Vertex Cover. The proof for Directed q-In Vertex Cover, which is similar, has been omitted. We assume that $q \geq 2$ because the case in which $q = 1$ is trivial in general graphs.

As with Directed (p, q)-Edge Dominating Set, we use $D[v, r, d]$ to store the minimum number of vertices in a reach-r-deficit-d vertex cover for $G[T_v]$.

When processing a vertex v, we determine $D[v, r, d]$ for values of r and d in the ranges $0 \leq r \leq q$ and $0 \leq d \leq q$. For the base cases, for each leaf v and its parent u in $\hat{G}$, if there is an edge (v, u) from v to its parent u, we must include v in the solution in order to cover (v, u). Thus, we define $D[v, r, d]$ as follows:

$$D[v, r, d] = \begin{cases} 1 & (1 \leq r \leq q \wedge d = 0) \\ +\infty & (\text{otherwise}). \end{cases}$$

If there is an edge (u, v) from parent u for a leaf v, there is no reachable path from v to any vertex. Moreover, v is not included in any minimum r-out vertex cover because v does not cover any edge. Thus, we define $D[v, r, d]$ as follows:

$$D[v, r, d] = \begin{cases} 0 & (r = 0 \wedge 0 \leq d \leq q-1) \\ +\infty & (\text{otherwise}). \end{cases}$$

As with Directed (p, q)-Edge Dominating Set, the lemma below results from the fact that the size of the minimum r-out vertex cover of a smaller subgraph is never bigger than the size of a minimum r-out vertex cover for a larger subgraph.

Lemma I.1 *The following properties hold:*

1. $D[u, r, d] \leq D[v, r, d]$ *for* u *a descendant of* v,
2. $D[v, r, d] \leq D[v, r', d]$ *for* $r \leq r'$, *and*

3. $D[v, r, d] \leq D[v, r, d']$ *for* $d \geq d'$.

Corollary I.2 *For a given value of* v*, the minimum values will be* $D[v, 0, r]$*; the solution to* Directed r-Out Vertex Cover *will be* $D[v, 0, 0]$*, for* v *the root of* $\hat{G}$.

For each v, we define the recursive formulas. We consider two types of vertices.

Case 1: v such that there is an edge (v, u) to parent u.
In this case, if $d > 0$ or $r = 0$, we set $D[v, r, d] = +\infty$ because uncovered edges in $G[T_v]$ and (v, u) are never covered henceforth. Otherwise, we consider two cases:

1. When $r = q$, v must be included in the solution. Therefore, we set $D[v, r, d] = \sum_{u \in diff(v)} D[u, 0, q-1] + \sum_{w \in same(v)} D[w, 1, 0] + 1$.

   Because v and any vertex above v in a tree never cover any edge (w, v) for $w \in same(v)$, we need the reach of 1 in $D[w, 1, 0]$.

2. When $1 \leq r \leq q-1$, v must not be included in the solution. Therefore, we set $D[v, r, d] = \min_{x \in same(v)} \{D[x, r+1, 0] + \sum_{u \in diff(v)} D[u, 0, r-1] + \sum_{w \in same(v) \setminus \{x\}} D[w, 1, 0]\}$.

   In the recursive formula, every edge in $DT(v)$ is covered by x. As with Case 1-1, we need the reach of 1 in $D[w, 1, 0]$ since v and any vertex above v in a tree never cover any edge (w, v).

Case 2: v such that there is an edge (u, v) from parent u.
In this case, if $r > 0$, we set $D[v, r, d] = +\infty$ because v does not reach any vertex above itself in $G[T_v]$. Otherwise, we consider two cases:

1. When $d > 0$, v must not be included in the solution. Therefore, we set $D[v, r, d] = \sum_{u \in diff(v)} D[u, 1, 0] + \sum_{w \in same(v)} D[w, 0, d-1]$.

   Because v and any vertex above v in a tree never cover any edge (u, v) for $u \in diff(v)$, we need the reach of 1 in $D[u, 1, 0]$.

2. When $d = 0$, we set

$$D[v, r, d] = \min \Bigg\{ \sum_{u \in diff(v)} D[u, 1, 0] + \sum_{w \in same(v)} D[w, 0, q-1] + 1,$$
$$\min_{0 \leq j \leq q-2} \Big[\min_{x \in diff(v)} \{D[x, j+2, 0] + \sum_{u \in diff(v) \setminus \{x\}} D[u, 1, 0] + \sum_{w \in same(v)} D[w, 0, j]\} \Big] \Bigg\}.$$

   The first part is the case in which v is included in the solution. Since v q-covers edges in $ST(v)$, we sum up $D[w, 0, q]$ for $w \in same(v)$. The second part is the other case, that is, v is not included in the solution. In this case, we consider every possible combination of the reach of $DT(v)$ and the deficit of $ST(v)$ such that every edge in $G[T_v]$ is covered. Then we set a minimum possible solution in $G[T_v]$ as $D[v, r, d]$.

It is not difficult to see that the algorithm can be executed in polynomial time, since there are $O(n^3)$ table entries to fill, each of which can be filled in time $O(n^4)$.

J Proof of Lemma 4.9

We first give the definition of a tree decomposition.

Definition J.1 *A* tree decomposition *of an undirected graph $G = (V, E)$ is defined as a pair $\langle \mathcal{X}, T \rangle$, where $\mathcal{X} = \{X_1, X_2, \ldots, X_N \subseteq V\}$, and T is a tree whose nodes are labeled by $I \in \{1, 2, \ldots, N\}$, such that*

1. $\bigcup_{i \in I} X_i = V$.

2. *For all $\{u, v\} \in E$, there exists an X_i such that $\{u, v\} \subseteq X_i$.*

3. *For all $i, j, k \in I$, if j lies on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

In the following, we call T a *decomposition tree*, and we use the term "nodes" (not "vertices") for the elements of T to avoid confusion. Moreover, we call a subset of V corresponding to a node $i \in I$ a *bag* and denote it by X_i. The *width* of a tree decomposition $\langle \mathcal{X}, T \rangle$ is defined by $\max_{i \in I} |X_i| - 1$, and the *treewidth* of G, denoted by $\mathbf{tw}(G)$, is the minimum width over all tree decompositions of G.

We also use a very useful type of tree decomposition for use in algorithms, called a *nice tree decomposition* introduced by Kloks [29]. More precisely, it is a special binary tree decomposition which has four types of nodes, named *leaf*, *introduce vertex*, *forget* and *join*. A variant of the notion, using a new type of node named *introduce edge*, was introduced by Cygan et al. [27].

Definition J.2 *A tree decomposition $\langle \mathcal{X}, T \rangle$ is called a* nice tree decomposition *if it satisfies the following:*

1. *T is rooted at a designated node $r \in I$ satisfying $|X_r| = 0$, called the* root node.

2. *Each node of the tree T has at most two children.*

3. *Each node in T has one of the following five types:*

- *A* leaf *node i which has no children and its bag X_i satisfies $|X_i| = 0$.*
- *An* introduce vertex *node i has one child j with $X_i = X_j \cup \{v\}$ for a vertex $v \in V$.*
- *An* introduce edge *node i has one child j and labeled with an edge $(u, v) \in E$ where $u, v \in X_i = X_j$.*
- *A* forget *node i has one child j and satisfies $X_i = X_j \setminus \{v\}$ for a vertex $v \in V$.*
- *A* join *node i has two children nodes j_1, j_2 and satisfies $X_i = X_{j_1} = X_{j_2}$.*

We can transform any tree decomposition to a nice tree decomposition with $O(n)$ nodes in linear time [28]. Without loss of generality, we can assume that the parent node of an introduce edge (u, v) node is an introduce edge (u, w), introduce edge (v, x), forget u, or forget v node for some w or x [28].

By using dynamic programming based on a tree decomposition of width ℓ, we prove Lemma 4.9. In this proof, for vertex v, (u, v) is called an *incoming edge* of v and (v, u) is called an *outgoing edge* of v. Let D be the solution set. We fill table entries for each node. For a representation of the state of v in a bag, we define the *coloring* function:

$$c : V \to \{\mathbf{0}, \mathbf{0}_{in}, \mathbf{0}_{out}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}.$$

Each element of $\{\mathbf{0}, \mathbf{0}_{in}, \mathbf{0}_{out}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$ is called a *state* and has the following meanings:

- $\mathbf{0}$: v is an endpoint of an edge not included in D.

- $\mathbf{0}_{in}$: v is an endpoint of an edge not included in D, but will become an endpoint of an incoming edge of v included in D.
- $\mathbf{0}_{out}$: v is an endpoint of an edge not included in D, but will become an endpoint of an outgoing edge of v included in D.
- $\mathbf{0}_{inout}$: v is an endpoint of an edge not included in D, but will become an endpoint of both an incoming edge and an outgoing edge of v included in D.
- $\mathbf{0}^{in}_{inout}$: v is an endpoint of an incoming edge of v included in D and will also become an endpoint of an outgoing edge of v included in D.
- $\mathbf{0}^{out}_{inout}$: v is an endpoint of an outgoing edge of v included in D and will also become an endpoint of an incoming edge of v included in D.
- $\mathbf{1}_{in}$: v is an endpoint of an incoming edge of v included in D.
- $\mathbf{1}_{out}$: v is an endpoint of an outgoing edge of v included in D.
- $\mathbf{1}_{inout}$: v is an endpoint of both an incoming edge and an outgoing edge of v included in D.

Let $\Sigma = \{\mathbf{0}, \mathbf{0}_{in}, \mathbf{0}_{out}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$. Given two vertex sets V and W, we denote their colorings by $c_V \in \Sigma^{|V|}$ and $c_W \in \Sigma^{|W|}$, respectively. Suppose that $c_V = (c(v_1), \ldots, c(v_{|V|}))$ and $c_W = (c(w_1), \ldots, c(w_{|W|}))$, where $v_i \in V$ and $w_i \in W$. Then we define the *concatenation* $c_V \times c_W \in \Sigma^{|V|+|W|}$ of c_V and c_W as the coloring $(c(v_1), \ldots, c(v_{|V|}), c(w_1), \ldots, c(w_{|W|}))$.

Suppose that $W \subseteq V$. Then we define the *separation* $c_V \setminus c_W \in \Sigma^{|V|-|W|}$ as the coloring $(c(v_{i_1}), \ldots, c(v_{i_{|V|-|W|}}))$, where $v_{i_1}, \ldots, v_{i_{|V|-|W|}} \in V \setminus W$.

Given a tree decomposition $\langle \mathcal{X}, T \rangle$, we define a subgraph $G_i = (V_i, E_i)$ for each node i where V_i is the union of all bags X_j with $j = i$ or j a descendant of i in T, and $E_i \subseteq E$ is the set of edges introduced in the subtree rooted at node i. Then we define a *partial solution* in node i as a subset of E_i that possibly dominates every edge in an induced subgraph of V_i.

For each node i, we define the function:

$$f_i : \Sigma^{|X_i|} \to \mathbb{N} \cup \{+\infty\}.$$

This function's value represents the minimum size of a possible partial solution in node i. Let $c_i \in \Sigma^{|X_i|}$ be a coloring of node i. If $f_i(c_i) = +\infty$, it means that the coloring c_i is invalid. We obtain the minimum size of the solution by computing $f_i(c_i)$ by dynamic programming on a tree decomposition. For simplicity, we sometimes denote c_i by c.

Leaf node: In leaf nodes, we define $f_i(\emptyset) := 0$.

Introduce vertex v node: If $c(v) \in \{\mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$, we set $f_i(c) := +\infty$. That is because there is no edge incident to v in node i. Note that an introduce vertex node only adds v, and does not add any edge. Thus, this contradicts the state of v because each state in $\{\mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$ represents that v is incident on at least one edge.

Otherwise, we set $f_i(c) := f_j(c \setminus \{c(v)\})$, that is, we set $f_i(c)$ as the value of f_j for the same coloring as c in its child j.

(u,v)	$\mathbf{0}$	$\mathbf{0}_{in}$	$\mathbf{0}_{out}$	$\mathbf{0}_{inout}$	$\mathbf{0}^{in}_{inout}$	$\mathbf{0}^{out}_{inout}$	$\mathbf{1}_{in}$	$\mathbf{1}_{out}$	$\mathbf{1}_{inout}$
$\mathbf{0}$	-	-	(1)	(1)	(1)	(1)	-	(1)	(1)
$\mathbf{0}_{in}$	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
$\mathbf{0}_{out}$	-	-	(1)	(1)	(1)	(1)	-	(1)	(1)
$\mathbf{0}_{inout}$	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
$\mathbf{0}^{in}_{inout}$	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
$\mathbf{0}^{out}_{inout}$	(1)	(1)	(1)	(1)	(2)	(1)	(3)	(1)	(4)
$\mathbf{1}_{in}$	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
$\mathbf{1}_{out}$	-	-	(1)	(1)	(5)	(1)	(6)	(1)	(7)
$\mathbf{1}_{inout}$	(1)	(1)	(1)	(1)	(8)	(1)	(9)	(1)	(10)

Table 2: Entries indicate case numbers that apply for the possible states of u and v for the introduce node for the edge (u,v). The labels for the rows correspond to $c(u)$ and the labels for the columns correspond to $c(v)$.

Introduce edge (u,v) node: Table 2 shows all cases of the possible states of u and v for the introduce node for the edge (u,v).

Case 1. In case 1 (See Table 2), we cannot add edge (u,v) to D because if we add it, this contradicts the states of u and v. For example, if $c(u) = \mathbf{0}$ and $c(v) = \mathbf{0}_{out}$, u is not an endpoint of edge in D and v is an endpoint of an incoming edge of v. That is why if we add (u,v) to the solution, it contradicts the states of u and v. The other cases are similar. Therefore, we set $f_i(c) := f_j(c)$ since the size of a partial solution does not change.

Case 2. If $c(u) = \mathbf{0}^{out}_{inout}$ and $c(v) = \mathbf{0}^{in}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{0}^{in}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}^{in}_{inout}\})\}. \end{aligned}$$

In a child node j, if $c_j(u), c_j(v) = \mathbf{0}_{inout}$, u and v do not have any edge in the solution yet. If we add (u,v) to the solution in the introduce edge (u,v) node, the states of u and v are changed to $\mathbf{0}^{out}_{inout}$ and $\mathbf{0}^{in}_{inout}$ since u becomes incident to an outgoing edge in the solution and v becomes incident to an incoming edge. The second and third equations are similar cases in which the states of u and v are changed to $\mathbf{0}^{out}_{inout}$ and $\mathbf{0}^{in}_{inout}$. For the last equation, u and v already have an outgoing and incoming edge in the solution, respectively. Moreover, edge (u,v) is dominated even if it is not included in the solution. Thus, we set $f_i(c \times \{c(u)\} \times \{c(v)\})$ as described above. Similar arguments apply to the remaining cases below.

Case 3. If $c(u) = \mathbf{0}^{out}_{inout}$ and $c(v) = \mathbf{1}_{in}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{0}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{1}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{1}_{in}\})\}. \end{aligned}$$

Case 4. If $c(u) = \mathbf{0}^{out}_{inout}$ and $c(v) = \mathbf{1}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{1}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{1}_{inout}\})\}. \end{aligned}$$

Case 5. If $c(u) = \mathbf{1}_{out}$ and $c(v) = \mathbf{0}^{in}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{0}^{in}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{0}^{in}_{inout}\})\}. \end{aligned}$$

Case 6. In the case in which $c(u) = \mathbf{1}_{out}$ and $c(v) = \mathbf{1}_{in}$, if we do not add (u, v) to the solution, it is never dominated. Thus, we must add edge (u, v) to the solution in any case. Thus we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{0}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{1}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{0}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{1}_{in}\}) + 1\}. \end{aligned}$$

Case 7. If $c(u) = \mathbf{1}_{out}$ and $c(v) = \mathbf{1}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{1}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{1}_{inout}\})\}. \end{aligned}$$

Case 8. If $c(u) = \mathbf{1}_{inout}$ and $c(v) = \mathbf{0}^{in}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}^{in}_{inout}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{in}_{inout}\} \times \{\mathbf{0}^{in}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{0}^{in}_{inout}\})\}. \end{aligned}$$

Case 9. If $c(u) = \mathbf{1}_{inout}$ and $c(v) = \mathbf{1}_{in}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}_{out}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{1}_{out}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{0}_{out}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{1}_{out}\})\}. \end{aligned}$$

Case 10. If $c(u) = \mathbf{1}_{inout}$ and $c(v) = \mathbf{1}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}^{in}_{inout}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{in}_{inout}\} \times \{\mathbf{1}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{1}_{inout}\})\}. \end{aligned}$$

For each entry denoted by "-" in Table 2, we set $f_i(c) := +\infty$ since edge (u, v) is never dominated.

	$\mathbf{0}$	$\mathbf{0}_{in}$	$\mathbf{0}_{out}$	$\mathbf{0}_{inout}$	$\mathbf{0}^{in}_{inout}$	$\mathbf{0}^{out}_{inout}$	$\mathbf{1}_{in}$	$\mathbf{1}_{out}$	$\mathbf{1}_{inout}$
$\mathbf{0}$	$\mathbf{0}$								
$\mathbf{0}_{in}$		$\mathbf{0}_{in}$					$\mathbf{1}_{in}$		
$\mathbf{0}_{out}$			$\mathbf{0}_{out}$					$\mathbf{1}_{out}$	
$\mathbf{0}_{inout}$				$\mathbf{0}_{inout}$	$\mathbf{0}^{in}_{inout}$	$\mathbf{0}^{out}_{inout}$			$\mathbf{1}_{inout}$
$\mathbf{0}^{in}_{inout}$				$\mathbf{0}^{in}_{inout}$	$\mathbf{0}^{in}_{inout}$	$\mathbf{1}_{inout}$			$\mathbf{1}_{inout}$
$\mathbf{0}^{out}_{inout}$				$\mathbf{0}^{out}_{inout}$	$\mathbf{1}_{inout}$	$\mathbf{0}^{out}_{inout}$			$\mathbf{1}_{inout}$
$\mathbf{1}_{in}$		$\mathbf{1}_{in}$					$\mathbf{1}_{in}$		
$\mathbf{1}_{out}$			$\mathbf{1}_{out}$					$\mathbf{1}_{out}$	
$\mathbf{1}_{inout}$				$\mathbf{1}_{inout}$	$\mathbf{1}_{inout}$	$\mathbf{1}_{inout}$			$\mathbf{1}_{inout}$

Table 3: Entries indicate $c_i(v)$ for possible combinations of $c_{j_1}(v)$ and $c_{j_2}(v)$ for v in a join node i which has two children node j_1 and j_2.

Forget v node: Suppose that $X_i = X_j \setminus \{v\}$ for a forget node i and its child j. Let $c_j = c \times \{c_j(v)\}$ and D be a set of c_j's such that one of these sets of conditions holds:

1. $c_j(v) = \mathbf{0}$ and $\forall u \in N^{in}(v) \cap X_j, c(u) \in \{\mathbf{0}_{in}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{inout}\}$ and $\forall w \in N^{out}(v) \cap X_j, c(w) \in \{\mathbf{0}_{out}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$,
2. $c_j(v) = \mathbf{1}_{in}$ and $\forall u \in N^{in}(v) \cap X_j, c(u) \in \{\mathbf{0}_{in}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$,
3. $c_j(v) = \mathbf{1}_{out}$ and $\forall w \in N^{out}(v) \cap X_j, c(w) \in \{\mathbf{0}_{out}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$,
4. $c_j(v) = \mathbf{1}_{inout}$.

The conditions of set D mean that every edge incident to v is dominated. In condition 2, note that any $(u, v) \in E$ such that $c_j(u) = \mathbf{1}_{out}$ is included in K in the introduce edge (u, v) node. Similarly, in condition 3, any $(u, v) \in E$ such that $c_j(u) = \mathbf{1}_{in}$ is included in K. Then, we set $f_i(c) := \min_{c_j \in D} f_j(c_j)$.

Join node: We assume that a join node i has two children nodes j_1, j_2. According to Table 3, we have 25 combinations of states for each v in a join node. Let D' be a tuple of two colorings c_{j_1}, c_{j_2} such that for each vertex v, $c_{j_1}(v)$ and $c_{j_2}(v)$ satisfy the condition of Table 3. Then we set the recursive formula as follows:

$$f_i(c) \quad := \min_{c_{j_1}, c_{j_2} \in D'} f_{j_1}(c_{j_1}) + f_{j_2}(c_{j_2}).$$

Note that there is no edge (u, v) for $u, v \in X_i$ in G_i, due to the assumption of a tree decomposition, that is, edge (u, v) is introduced an introduce edge node above i.

Time analysis In each introduce vertex, introduce edge, and forget node, we can compute every recursive formula in $9^\ell \ell^{O(1)}$-time. For each join node, we have 25 combinations of states for each v, and hence it takes $25^\ell \ell^{O(1)}$-time to compute every recursive formula. Therefore, the total running time is $25^\ell \ell^{O(1)} n$.

K Proof of Theorem 4.12

In this section, we show that Directed $(0, 1)$-Edge Dominating Set is fixed-parameter tractable with respect to k. The proof is almost the same as Directed $(1, 1)$-Edge Dominating Set. First, we can immediately obtain the following lemma because the size

(u,v)	$\mathbf{0}$	$\mathbf{0}_{in}$	$\mathbf{0}_{out}$	$\mathbf{0}_{inout}$	$\mathbf{0}^{in}_{inout}$	$\mathbf{0}^{out}_{inout}$	$\mathbf{1}_{in}$	$\mathbf{1}_{out}$	$\mathbf{1}_{inout}$
$\mathbf{0}$	-	-	-	-	-	-	-	-	-
$\mathbf{0}_{in}$	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
$\mathbf{0}_{out}$	-	-	-	-	-	-	-	-	-
$\mathbf{0}_{inout}$	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
$\mathbf{0}^{in}_{inout}$	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
$\mathbf{0}^{out}_{inout}$	(1)	(1)	(1)	(1)	(2)	(1)	(3)	(1)	(4)
$\mathbf{1}_{in}$	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
$\mathbf{1}_{out}$	-	-	-	-	(5)	-	(6)	-	(7)
$\mathbf{1}_{inout}$	(1)	(1)	(1)	(1)	(8)	(1)	(9)	(1)	(10)

Table 4: Entries indicate case numbers that apply for possible states of u and v for the introduce node for the edge (u,v). The labels for the rows correspond to $c(u)$ and the labels for the columns correspond to $c(v)$.

of minimum Directed $(0,1)$-Edge Dominating Set is larger than that of Directed $(1,1)$-Edge Dominating Set.

Lemma K.1 *Given a directed graph G, let G^* be an undirected graph underlying in G and s' be the minimum size of* Directed $(0,1)$-Edge Dominating Set *on G. Then the following inequality holds: $\mathbf{tw}(G^*) \leq 2s'$.*

If there is an algorithm that solves Directed $(0,1)$-Edge Dominating Set in $25^{\ell}\ell^{O(1)}n$-time, we conclude Directed $(0,1)$-Edge Dominating Set can be solved in $2^{O(k)}n$-time by the same way as Theorem 4.11.

Lemma K.2 *Given a directed graph G, let G^* be an undirected graph underlying in G. Then given a tree decomposition of G^* of width at most ℓ, there exists an algorithm that solves* Directed $(0,1)$-Edge Dominating Set *in $25^{\ell}\ell^{O(1)}n$-time.*

Proof: We only change the recursive formulas of introduce edge and forget nodes in Lemma 4.9.

Introduce edge (u,v) node: Table 4 shows all cases of state of u and v in an introduce edge (u,v) node.

Case 1. In case 1 (See Table 2), we cannot add edge (u,v) to D because if we add it, this contradicts the states of u and v. For example, if $c(u) = \mathbf{0}$ and $c(v) = \mathbf{0}_{out}$, u is not an endpoint of edge in D and v is an endpoint of an incoming edge of v. That is why if we add (u,v) to the solution, it contradicts the states of u and v. The other cases are similar. Therefore, we set $f_i(c) := f_j(c)$ since the size of a partial solution does not change.

Case 2. If $c(u) = \mathbf{0}^{out}_{inout}$ and $c(v) = \mathbf{0}^{in}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{0}^{in}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}^{in}_{inout}\})\}. \end{aligned}$$

Case 3. If $c(u) = \mathbf{0}^{out}_{inout}$ and $c(v) = \mathbf{1}_{in}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{0}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{1}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{1}_{in}\})\}. \end{aligned}$$

Case 4. If $c(u) = \mathbf{0}^{out}_{inout}$ and $c(v) = \mathbf{1}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{inout}\} \times \{\mathbf{1}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{1}_{inout}\})\}. \end{aligned}$$

Case 5. In the case in which $c(u) = \mathbf{1}_{out}$ and $c(v) = \mathbf{0}^{in}_{inout}$, if we do not add (u, v) to the solution, it is never dominated. Thus, we must add edge (u, v) to the solution in any case. Thus we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{0}^{in}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{0}^{in}_{inout}\}) + 1\}. \end{aligned}$$

Case 6. In the case in which $c(u) = \mathbf{1}_{out}$ and $c(v) = \mathbf{1}_{in}$, if we do not add (u, v) to the solution, it is never dominated. Thus, we must add edge (u, v) to the solution in any case. Thus we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{0}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{1}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{0}_{in}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{1}_{in}\}) + 1\}. \end{aligned}$$

Case 7. In the case in which $c(u) = \mathbf{1}_{out}$ and $c(v) = \mathbf{1}_{inout}$, if we do not add (u, v) to the solution, it is never dominated. Thus, we must add edge (u, v) to the solution in any case. Thus we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}_{out}\} \times \{\mathbf{1}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{out}\} \times \{\mathbf{1}_{inout}\}) + 1\}. \end{aligned}$$

Case 8. If $c(u) = \mathbf{1}_{inout}$ and $c(v) = \mathbf{0}^{in}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}^{in}_{inout}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{in}_{inout}\} \times \{\mathbf{0}^{in}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{0}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{0}^{in}_{inout}\})\}. \end{aligned}$$

Case 9. If $c(u) = \mathbf{1}_{inout}$ and $c(v) = \mathbf{1}_{in}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{0}_{out}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{out}_{inout}\} \times \{\mathbf{1}_{out}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{0}_{out}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{1}_{out}\})\}. \end{aligned}$$

Case 10. If $c(u) = \mathbf{1}_{inout}$ and $c(v) = \mathbf{1}_{inout}$, we define as follows:

$$\begin{aligned} f_i(c \times \{c(u)\} \times \{c(v)\}) \quad := \min\{ \quad & f_j(c \times \{\mathbf{0}^{in}_{inout}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{0}^{in}_{inout}\} \times \{\mathbf{1}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{0}^{out}_{inout}\}) + 1, \\ & f_j(c \times \{\mathbf{1}_{inout}\} \times \{\mathbf{1}_{inout}\})\}. \end{aligned}$$

For the case we denote by "-" in Table 2, we set $f_i(c) := +\infty$ since edge (u, v) is never dominated.

Forget v node: Suppose that $X_i = X_j \setminus \{v\}$ for a forget node i and its child j. Let $c_j = c \times \{c_j(v)\}$ and D be a set of c_j's such that one of these sets of conditions holds:

1. $c_j(v) \in \{\mathbf{1}_{in}, \mathbf{1}_{inout}\}$ and $\forall u \in N^{in}(v) \cap X_i, c(u) \in \{\mathbf{0}_{in}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$,
2. $c_j(v) = \mathbf{1}_{out}$ and $\forall u \in N^{in}(v) \cap X_i, c(u) \in \{\mathbf{0}_{in}, \mathbf{0}_{inout}, \mathbf{0}^{in}_{inout}, \mathbf{0}^{out}_{inout}, \mathbf{1}_{out}, \mathbf{1}_{inout}\}$ and $\forall w \in N^{out}(v) \cap X_i, c(w) \in \{\mathbf{0}^{in}_{inout}, \mathbf{1}_{in}, \mathbf{1}_{inout}\}$.

The conditions of set D mean that every edge incident to v is dominated. Then, we set $f_i(c) := \min_{c_j \in D} f_j(c_j)$.

The running time of this algorithm is dominated by join nodes. Therefore, this algorithm runs in $25^{\ell}\ell^{O(1)}n$-time.

The following literature is cited in the appendix.

References

[27] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M, van Rooij, J. O. Wojtaszczyk: Solving connectivity problems parameterized by treewidth in single exponential time. In: *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS2011)*, pp. 150–159 (2011)

[28] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh: Parameterized Algorithms, Springer International Publishing (2015)

[29] T. Kloks: Treewidth, Computations and Approximations. Lecture Notes in Computer Science, 842, Springer-Verlag Berlin Heidelberg (1994)

ビザンチン環境における認証機能付き白板を用いたモバイルエージェント非同期集合アルゴリズム

土田 将司 *　　大下 福仁 *　　井上 美智子 *

1　はじめに

1.1　本研究の背景

近年，多数のコンピュータ (以下，ノード) を用いて大規模分散システムを構築し，運用する事例が増えている．しかしながらシステムが大規模化するにつれ，ノードの保守作業，アップデート作業などが複雑化するという問題が生じている．そこでモバイルエージェント (以下，エージェント) と呼ばれる，分散システム上を自律的に移動し，様々なタスクを行うソフトウェアが注目を集めている．エージェントはそれ自身が情報の収集，解析を行いながらノード間を移動するため，ノード自体は他ノードとの情報交換を行う必要がなくなり，分散システムの設計が容易になる．また，複数のエージェントが協調動作することで，より効率的に分散システムを運用することができるため，複数のエージェントを協調させるアルゴリズムの開発が盛んに行われている．

エージェントを協調動作させるための基本的タスクの１つとして，集合問題が考えられている．集合問題とは，初期状況において分散システム上の任意のノードに散在しているエージェントを，有限時間内に同一のノードに集合させる問題である．例えばエージェントが１つのノードに集合することで，エージェント間で情報交換を行うことができ，協調動作を行いやすくなる．

しかしエージェント自体が分散システム上を移動するため，システムエラーが生じてエージェントの動作が停止したり，クラッキングされ意図しない動作をすることを考える必要がある．上記のような，アルゴリズムに従わずに任意の動作をする故障をビザンチン故障とし，ビザンチン故障を起こしたエージェントをビザンチンエージェントとする．本研究では，このビザンチンエージェントの動作によらず，正常にアルゴリズムを実行するエージェントが１つのノードに集合できるアルゴリズムを提案する．

1.2　関連研究

集合問題はエージェントを協調させるための基本タスクであり，これまでに多くの研究が行われている [6][7]．表１に本研究と主な関連研究の成果の比較を示す．これらの研究は，様々な環境において，集合問題の解決可能性や，解決可能な場合に必要となるコスト (時間，移動量，メモリ量等) を明らかにすることを目的としている．そのため，エージェントの同期性や匿名性が異なる様々な環境において多くの研究がなされている．

エージェントが同期して動作する場合，集合問題に対する決定性アルゴリズムが多数研究されている [1][2][8]．ノード上のメモリ (以下，白板) を利用できず，エージェントに固有の ID がない場合，対称性を破壊することが不可能なことから集合を実現できないグラフが存在する．そのため文献 [1][2] ではエージェントに固有の ID を仮定し，任意のグラフで集合できるアルゴリズムを提案している．文献 [1] では，ノード数を n，エージェントの最小 ID の長さを l としたとき，白板を用いずに 2 エージェントを $\tilde{O}(n^5 l)$ 時間で出会わせるアルゴリズムを提案している．また，ビザンチンエージェントが存在する場合に集合

*奈良先端科学技術大学院大学

表 1: 関連研究の比較 (n はネットワーク中のノード数，l はエージェントの最小 ID の長さ，λ はエージェントの最大 ID の長さ，m は辺の数，f_u はビザンチンエージェント数の上限，D は初期位置における 2 エージェント間の距離，$poly()$ は引数の多項式，f はビザンチンエージェント数を表す).

	同期性	グラフ	ビザンチン	白板	終了判定	時間複雑度 [1]
[1]	同期	任意	なし	なし	可能	$\tilde{O}(n^5 l)$
[2]	同期	任意	あり	なし	可能	$\tilde{O}(n^9 \lambda)$
[3]	同期	任意	あり	認証付き	可能	$O(f_u m)$
[4]	非同期	直線	なし	なし	可能	$O((D+\lambda)^3)$
[4]	非同期	リング	なし	なし	可能	$O((n\lambda))$
[5]	非同期	任意	なし	あり	可能	$poly(n, l)$
自明な手法	非同期	任意	なし	あり	可能	$O(m)$
提案	非同期	任意	あり	認証付き	不可	$O(m + fn)$

を実現するアルゴリズムも多数研究されている．文献 [2] では，エージェントの最長 ID の長さを λ としたとき，$\tilde{O}(n^9\lambda)$ 時間で集合できる，ビザンチン故障耐性を持ったアルゴリズムを提案している．文献 [8] では，文献 [2] で未解決問題として残されていた，正常エージェントが集合を行うために必要な正常エージェントの個数を明らかにしている．さらに，文献 [3] では認証機能付き白板を用いることにより，ビザンチンエージェント数の上限を f_u，グラフ中の辺の数を m とした時に $O(f_u m)$ 時間での集合を実現している．ここで認証機能付き白板とは，認証機能によって各エージェント専用の領域を設けた白板である．また，認証機能を用いて，各エージェントが生成した情報に署名を与えることも可能である．この認証機能付き白板が各ノードに存在すると仮定し，各正常エージェントはコンセンサスアルゴリズムを認証機能付き白板を用いてシミュレートすることで集合地点の算出を行っている．

エージェントが同期せずに動作する場合では，同期ネットワークに比べ，仮定を追加しなければ集合が難しくなる．文献 [4] では非同期ネットワークにおいて，ビザンチンエージェントを考慮せず，2 エージェントの集合問題について考察している．この文献 [4] では無限直線グラフにおいて，各エージェントは固有の ID を持ち，グラフ中の辺の中でもエージェント同士が出会うことができると仮定しており，この場合，初期状況における 2 エージェント間の距離を D としたとき，$O((D+\lambda)^3)$ 回の移動で集合ができることが示されている．更にグラフが未知のリンググラフである場合では $O(n\lambda)$ 回の移動で集合できるアルゴリズムを提案している．また，同様の仮定において 2 エージェントの集合問題について考察した文献として [5] が挙げられる．文献 [5] では任意の匿名ネットワークにおいて，グラフサイズとエージェントの最小 ID の長さによる多項式時間での集合を実現している．

ノードに白板が存在する場合，非同期ネットワーク環境においても集合に要する時間を大きく削減できる事が知られている．例えば，エージェントが固有の ID を持つ場合，各エージェントはアルゴリズムを開始したノードの白板に ID を書き込み，白板を用いてネットワークを一周することで，全エージェントの全ての ID 及び書き込まれたノードの位置を知ることができる．そのため最小 ID のエージェントの ID が書き込まれたノードに集まることで $O(m)$ 回の移動で集合を実現できる (表 1 に自明な手法として記載)．しかしビザンチンエージェントが存在する場合，偽の情報が白板に書き込まれ，集合場所を

[1] 同期モデルだと時間複雑度を表す．非同期モデルだと 1 エージェントの移動数を表す．

偽る動作を考える必要がある．

また，文献 [9] では本研究とは異なるビザンチンエージェントの能力を仮定しているが，ビザンチンエージェントが存在する円環状のネットワークトポロジ及びメッシュ状のネットワークトポロジにおいて，非同期環境での集合を実現している．文献 [9] で用いているビザンチンエージェントモデルでは，正常エージェントはビザンチンエージェントを区別することができ，正常エージェントとビザンチンエージェントがお互いに同一ノードに存在できず，また，辺でお互いにすれ違うこともできない．このようなモデルにおいて，各エージェントは $O(n)$ 回の移動で集合できるアルゴリズムが提案されている．

他に集合問題でエージェントの故障を考慮した文献として [10] が挙げられる．文献 [10] で扱っているネットワークモデルは，各エージェントごとに移動速度が異なるが一定の速度で動くことができる．すなわち，各エージェントが同レートのクロックを所持しているが，他エージェントの移動に必要なクロック数を把握できないモデルである．この文献 [10] では，エージェントがノードもしくは辺で故障し停止する場合について考察している．エージェントのメモリが残ったまま停止する場合と，エージェントのメモリ自体が消えて停止する場合の 2 つの停止故障について多項式時間で集合できるアルゴリズムを提案している．

1.3 本研究の成果

本稿では非同期ネットワークにおいて，ビザンチン環境下での集合を実現するための新たなアルゴリズムを提案する．本研究で考察するモデルは文献 [3] で考えられているモデルを非同期ネットワークとしたものである．すなわち，任意の非同期ネットワーク上にビザンチンエージェントが存在し，各ノードに白板が存在するモデルである．現在用いられている分散システムの大半は非同期分散システムであり，非同期ネットワークでの集合を実現することで，同期分散システムだけでなく，より多様な分散システムでもエージェントを集合させることが可能となる．提案するアルゴリズムは非同期ネットワークにおいて，終了を宣言することはできないが，分散システム上に存在するビザンチンエージェント数を f とすると，各エージェントが高々$2m+4n+10fn = O(m+fn)$ 回の移動で集合を実現する．ビザンチンエージェントが存在せず，白板を用いることができる場合では，自明な手法にて $O(m)$ 回の移動で集合が可能である．本稿で提案するアルゴリズムは，自明な手法にビザンチンエージェント 1 体あたり $O(f)$ 回の移動を追加することでビザンチン故障耐性を実現している．

2 諸定義

2.1 分散システム

本稿ではノード集合を V，辺集合を E として，分散システムをグラフ $G = (V, E)$ として表す．ネットワークのノード数を $n = |V|$，辺の数を $m = |E|$ とする．また，ノードの ID が匿名なシステムとする．ノード $u, v \in V$ 間に辺 $(u, v) \in E$ が存在するとき，u と v は隣接していると呼ぶ．ノード v の隣接ノードの集合を $N_v = \{u|(u, v) \in E\}$ で表す．ノード v の次数を $d(v) = |N_v|$ で表す．各ノード v に接続する辺はポート番号で識別可能であり，ノード v での辺 (v, u) のポート番号を $\lambda_v(v, u)$ と表す．また，同一ノードのポート番号は各ノードで相異なる，すなわち $\lambda_v(v, u) \neq \lambda_v(v, w)(u \neq w)$ が成り立つ．

各ノード $v \in V$ にはエージェントが情報を残せる白板が存在すると仮定する．白板上には各エージェントが書き込み可能な領域が割り当てられており，エージェントはその領域に対してのみ情報を書き込むことができる．一方，各エージェントは他エージェントの領域も含めて白板上の全ての情報を参照することができる．

2.2 モバイルエージェント

分散システム中に複数のエージェントが存在し，エージェントの集合を $A = \{a_1, a_2, \cdots, a_k\}$ とする．ここで k はエージェント数である．各エージェントは固有の ID を持つ．エージェント a_i の ID を ID_i と表す．また，各エージェントはノード数 n，及びエージェント数 k を知らないものとする．

各エージェントは状態機械 (S, δ) としてモデル化される．S はエージェントの状態集合を表し，各エージェントの状態はエージェントメモリ内の変数の値によって決定される．状態遷移関数 δ は，エージェントの状態，訪問中のノードの状態 (白板の内容)，訪問時に通ったポート番号から，エージェントの次状態，ノードの次状態，ノードに滞在するか移動するか，移動する場合は移動先のポート番号を決定する関数である．

本稿ではエージェントは非同期的に動作する．すなわち，各エージェントの内部計算に要する時間及びノード間の移動に要する時間は，有限ではあるが既知の上限は仮定しない．初期状況では全てのエージェントは任意のノード上で待機状態として存在する．各エージェントは任意のタイミングで起動状態へ移行し，アルゴリズムを実行することができる．起動状態のエージェント a_i がノード v で待機状態のエージェント a_j に遭遇した場合，a_i は v での動作を行う前に a_j を起動状態に移行させることができる．各エージェントは滞在中のノード v において 1 回の不可分操作として，内部計算と高々1 回の白板への書き込みまたは読み込みができる．

2.3 署名

各エージェント a_i はそれぞれ署名関数を保持しており，自身の ID である ID_i と，滞在中のノード v を用いて署名付きマーカーを作成することができる．エージェント a_i の署名関数である $Sign_{i,v}()$ によって出力されるマーカーを $marker_{i,v}$ とする．エージェント a_i はノード v に滞在しているとき，$Sign_{i,v}()$ のみ実行することができ，$Sign_{j,v'}()$ $(i \neq j$ または $v \neq v')$ を実行できない．したがってエージェント a_i が $Sign_{i,v}()$ にて作成したマーカー $marker_{i,v}$ は，エージェント a_i がノード v で作成したマーカーであると保証することができる．エージェント a_j は滞在中のノードが v のときのみ，$maker_{i,v}$ が v で作成されたことを認識でき，$v'(v' \neq v)$ に滞在しているときは v' 以外のノードで作成されたことだけを認識できる．

2.4 ビザンチンエージェント

ビザンチンエージェントは，アルゴリズムとは関係なく任意の動作を行うことができる．ただし，自身の ID を変更，偽装することはできない．また，エージェント a_i がビザンチンエージェントであっても，ノード v において $Sign_{j,v'}()(i \neq j$ または $v \neq v')$ を実行することができず，$marker_{j,v'}$ $(i \neq j$ または $v \neq v')$ を作成することはできない．白板においても他エージェントの領域への情報の書き込み，削除は行うことができない．分散システム上にビザンチンエージェントは f 体存在し，エージェントはビザンチンエージェント数 f を知らないものとする．

2.5 集合問題

モバイルエージェント集合問題とは，全ての正常エージェントを有限時間内に一つのノードに集合させる問題である．初期状況では，エージェントはネットワーク上の任意のノードに散在しており，一つのノードに複数のエージェントが存在することもある．本稿ではアルゴリズムの性能を示すため，最も移動数の多い正常エージェントが集合に要した移動数を評価する．

2.6 手続き DFS

本節ではアルゴリズム中で用いる手続き Depth-First search(DFS) を説明する．この手続きにより，エージェントは全てのノードを訪れることができる．

任意のノードからエージェントが DFS を開始した時，ノードに印を施し，未探索の辺を探索する．探索する際は訪れるポート番号を探索済みとしてノード上の白板の領域に記述しておく．印の施されていないノードに訪れた際，新たに印を施し，未探索の辺を探索する．既に印の施されているノードを訪れた際は前のノードに戻り，別の未探索の辺の探索を行う．全ての辺を探索し，DFS を開始したノードに戻ると手続きを終了する．この手続きではグラフ中の全ての辺を 2 回訪れるため，どのノードから DFS を始めたとしても必要な移動回数は $2m$ 回である．また，ノードに施す印及び，探索済みのポート番号の記録等は白板上の自身の領域のみで行うことができるため，ビザンチンエージェントの影響を受けない．一度 DFS による探索を終えると DFS 木を構成できる．そのため 2 回目以降のネットワーク探索は DFS 木を用いて $2n$ 回の移動で行うことができる．

3　アルゴリズム

3.1　アルゴリズムの概要

本アルゴリズムは非同期ネットワークにおいて，ビザンチンエージェントが存在する場合でも全ての正常エージェントの集合を実現する．アルゴリズムの方針としては以下の通りである．

各エージェント a_i はノード v_{start} でアルゴリズムを実行開始したとき，アルゴリズムを開始したことを示す情報 $marker_{i,v_{start}}$(以下，スタート地点マーカー)を作成する．このスタート地点マーカーはエージェントの ID 及び，作成されたノードの署名が含まれている．そのため，各エージェントは $marker_{i,v_{start}}$ を確認した際，a_i が v_{start} で作成したスタート地点マーカーであると認識できる．このスタート地点マーカーを全エージェント間で共有し，最小 ID のエージェントがスタート地点マーカーを作成したノードを集合地点とすることで集合を行う．

エージェント a_i はスタート地点マーカーを共有するために，DFS を行いながら全てのノードに自身のスタート地点マーカーをコピーする．DFS 中，待機状態で存在するエージェント a_j に遭遇した場合，a_j のアルゴリズムを実行させる．また，他エージェントのスタート地点マーカーを目撃した場合，そのスタート地点マーカーを自身のローカル変数で保存しておく．

a_i が DFS を終え，v_{start} に帰った後，a_i は集めたスタート地点マーカーより，最小 ID のエージェント a_{min} が作成したマーカー $marker_{min,v_{min}}$ を参照する．エージェント a_i はこの $marker_{min,v_{min}}$ が作成されたノード v_{min} を集合地点とすることで集合を実現する．

しかし，ビザンチンエージェント a_b が存在する場合，a_b が偽のマーカーの書き込みや削除を繰り返し，集合を妨げる可能性がある．集合を妨げる動作として，ビザンチンエージェントが自身のスタート地点マーカーを設置しない，一部の正常エージェントにのみスタート地点マーカーを見えるように書き込み・削除を行う，複数のスタート地点マーカーを作成するなどが考えられる．そこで本アルゴリズムでは，ビザンチンエージェントと判明している ID を除く，最小 ID のエージェントが作成したスタート地点マーカーの共有を各エージェント間で行う．これを実現するためには，最小 ID のエージェントが作成したスタート地点マーカー及びビザンチンエージェントの ID を記したリストを共有するといった手法が考えられる．ここで，集合を行うノードは最小 ID のエージェント a_{min} がスタート地点マーカーを作成したノード v_{min} なので，$marker_{min,v_{min}}$ さえ全正常エージェントで共有できれば集合ができる．したがって，スタート地点マーカーを作成しないビザンチンエージェントによって集合地点が撹乱されることはない．

最小 ID のエージェントが作成したスタート地点マーカーの共有は，各エージェントがそのスタート地点マーカーを全ノードにコピーすることで行う．全ノードにコピーすることにより，全エージェントは任意のノードに滞在していたとしても，最小 ID のエージェントが作成したスタート地点マーカーを共有することができる．また，全正常エージェントが

この動作を行うことにより，一部の正常エージェントのみが確認し，その時点で最小 ID であるビザンチンエージェントのスタート地点マーカーも全正常エージェント間で共有できる．共有する際，全正常エージェントは共有するマーカーをコピーする．したがって同一ノードで同一 ID のエージェントによって複数スタート地点マーカーを作成されたとしても，マーカーをコピーしたのと同様の状態となり，問題なく集合地点を算出する際に利用できる．

ビザンチンエージェントの ID を記したリストの共有だが，リスト自体を共有するのではなく，ビザンチンエージェントが作成したスタート地点マーカーの共有を行う．本稿で用いるスタート地点マーカーは署名が施されており，他エージェントのスタート地点マーカーを偽装することはできない．また，正常エージェントが作成する $marker_{i,v_{start}}$ は，アルゴリズム実行開始時に 1 度だけ作成する．したがって，異なるノードで作成されたスタート地点マーカー $marker_{b,v}$ と $marker_{b,v'}$ が存在する場合，正常エージェントは a_b がビザンチンエージェントであると判別することができる．a_b がビザンチンエージェントと判明した場合，ブラックリストに ID_b を加える．ブラックリストとは各エージェントがビザンチンエージェントだと判定したエージェント ID を保存する変数である．正常エージェント a_i はブラックリストに登録されたエージェントのマーカー全てを無視する．また，本研究では非同期ネットワークを考えているため，a_i が一度集合地点を算出したとしても，その後他エージェントが共有するマーカーによって集合地点を変更しなければならない場合がある．そのため，a_i は集合地点を算出した後も滞在しているノードの白板を逐一監視し，新しいマーカーが書き込まれていないか確認を行う．

ブラックリストに含まれていない ID のうち，最小 ID のエージェントが作成したスタート地点マーカー $marker_{min,v_{min}}$ を参照し，常にそのノードに移動することにより，集合を実現する．

Algorithm 1 エージェント a_i のアルゴリズム．v は a_i が滞在しているノードを示す．

1: $marker_{i,v} = Sign_{i,v}()$
2: **while** DFS が未完了 **do**
3: 未実行エージェントに遭遇したら実行状態へ
4: $v.wb[ID_i].T = \{marker_{i,v}\}$
5: $a_i.All = a_i.All \cup \bigcup_{id} v.wb[id].T$
6: DFS の次のノードへ移動
7: **end while**
8: DFS の結果よりネットワークトポロジを算出
9: **while** $True$ **do**
10: $a_i.All = a_i.All \cup \bigcup_{id} v.wb[id].T$
11: **if** $\exists x_1, x_2 (x_1 \neq x_2) : x_1, x_2 \in a_i.All \wedge writer(x_1) == writer(x_2) \wedge writer(x_1) \notin a_i.Blacklist$ **then**
12: $a_i.X = \{x_1, x_2\}$
13: **while** 一周が未完了 **do**
14: $v.wb[ID_i].T = v.wb[ID_i].T \cup a_i.X$
15: 次のノードへ移動
16: **end while**
17: $a_i.Blacklist = a_i.Blacklist \cup \{writer(x_1)\}$
18: **else**
19: $min\_tmp = min\{writer(t) : t \in a_i.All \wedge writer(t) \notin a_i.Blacklist\}$
20: **if** $a_i.min > min\_tmp \ || \ a_i.min \in a_i.Blacklist$ **then**
21: $a_i.t_{min} = t \ s.t. \ t \in a_i.All \wedge writer(t) == min\_tmp$
22: $a_i.min = writer(a_i.t_{min})$
23: **while** 一周が未完了 **do**
24: $v.wb[ID_i].T = v.wb[ID_i].T \cup \{a_i.t_{min}\}$
25: 次のノードへ移動
26: **end while**
27: $a_i.t_{min}$が書かれたノードへ移動
28: **end if**
29: **end if**
30: **end while**

3.2 アルゴリズムの詳細

本アルゴリズムの疑似コードを Algorithm 1 に示す．各エージェントは初期状況では任意のノード v に待機状態で存在する．$v.wb[ID_i]$ はエージェント a_i のみが書き込むことのできるノード v の白板上の領域である．ノード v でアルゴリズム実行状態となったエージェント a_i は，スタート地点マーカーとして $marker_{i,v} = Sign_{i,v}()$ を 1 行目で作成する．ここで，ローカル変数として $a_i.Blacklist, a_i.All, a_i.min$ をプロセス p_i が管理しているとする．$a_i.Blacklist$ はエージェント a_i がビザンチンエージェントと判定したエージェント ID を格納する変数である．$a_i.All$ はエージェント a_i が目撃したスタート地点マーカーを全て記録する変数である．$a_i.min$ は a_i が最小 ID であると判定したエージェント ID を格納する変数である．これらの変数は初期値として $a_i.Blacklist = \emptyset, a_i.All = \emptyset, a_i.min = \infty$ とする．また，関数 $writer(marker_{i,v})$ は $marker_{i,v}$ を作成したエージェント ID，すなわち i を返す関数である．

エージェント a_i はスタート地点マーカーを作成した後，他エージェントに自身のスタート地点マーカーを知らせるため，DFS を行いながら全てのノードに $marker_{i,v}$ をコピーする (4 行目)．DFS を行なっている際，待機状態で存在する他エージェントに遭遇した場合，実行状態へと移行させる．同時に，確認した他エージェントのスタート地点マーカーを $a_i.All$ に加える (5 行目)．また，DFS によって全てのノード，辺を訪問するため，ネットワークのトポロジを把握し，マップを作成することができる．そのため，a_i は DFS 終了後からは作成したマップを用いて高々 $2n$ 回の移動でネットワークを巡回できる．

DFS による 1 周後，a_i は $a_i.All$ に収集したスタート地点マーカーを検証し，集合地点を算出するステップへ移行する．このステップにおいて，エージェント a_i は，$a_i.Blacklist$ に含まれていない ID のうち，最小 ID だと判定したエージェントによって作成されたスタート地点マーカーを変数 $a_i.t_{min}$ に格納する．エージェント a_i は $a_i.t_{min}$ を逐一更新し，$a_i.t_{min}$ の情報が作成されたノードを常に集合地点とする．

9 行目以降は集めたマーカー及び滞在中のノードにコピーされたマーカーより，ビザンチンエージェントが作成したマーカーを判別し，集合地点の算出を行う．1 エージェントが複数マーカーを作成していた場合には，そのエージェントをビザンチンエージェントと判定する．また，ビザンチンエージェントと判定していない ID のうち，最小 ID のエージェントがマーカーを作成したノードを集合地点として算出する動作を無限に繰り返す．集合地点の算出を終えた後も，エージェント a_i が滞在しているノードに他エージェントがスタート地点マーカーをコピーしてくることを考慮し，10 行目で a_i が滞在するノード v に書き込まれたスタート地点マーカーを $a_i.All$ に加える．

11 行目から 17 行目でビザンチンエージェントが作成したマーカーを判別し，$a_i.Blacklist$ への登録及び，全ノードにマーカーをコピーし，他エージェントにビザンチンエージェントだと教える動作を行う．$a_i.All$ の中で，異なるマーカー x_1, x_2 だが，$writer(x_1) == writer(x_2) \wedge write(x_1) \notin a_i.Blacklist$ となる情報が存在するか確認する．正常エージェントはアルゴリズムを開始した時のみ，開始したノードでマーカーを作成する．そのため，$writer(x_1) == writer(x_2)$ となる異なるノードで作成されたマーカー x_1 及び x_2 が存在すると，x_1 を作成したエージェント a_x はビザンチンエージェントと判断できる．正常エージェント a_i は，a_x がビザンチンエージェントであると判定した時，他エージェントに a_x がビザンチンエージェントであると知らせるため，13 行目から 16 行目で全てのノードに x_1 及び x_2 をコピーする．全てのノードにコピーすると，他ノードを集合地点として算出した正常エージェントも x_1 及び x_2 を参照できるようになり，a_x をビザンチンエージェントと判別することができる．全てのノードにコピー後，a_i は $writer(x_1)$ を $a_i.Blacklist$ に加える．a_i は $a_i.Blacklist$ に登録された ID の情報を今後無視する．上記の動作を $a_i.All$ の中から $writer(x_1) == writer(x_2) \wedge write(x_1) \notin a_i.Blacklist$ となる異なるマーカー x_1, x_2 が存在しなくなるまで行う．

19 行目からは集合地点の算出及び集合地点への移動を行っている．$a_i.All$ 内の情報で，$a_i.Blacklist$ に保存されていない ID かつ，最小の $writer(t)$ となるスタート地点マーカーを算出し，そのスタート地点マーカーが書かれたノードを集合地点とするため，最小の $writer(t)$ を min_tmp に代入する．a_i は最小 ID だと判定した ID ID_{min} を常に $a_i.min$ に保存し，ID_{min} のエージェントがマーカーを作成したノードを集合地点とする．そのため，$a_i.min > min\_tmp$ である場合，または，今まで最小 ID だと a_i が判断していた ID が $a_i.Blacklist$ に含まれた場合，$a_i.min$ を更新するとともに，集合地点の変更動作を行う．$writer(t) == min\_tmp$ となるマーカー t を新たに $a_i.t_{min}$ とし，全てのノードに $a_i.t_{min}$ をコピーする．全てのノードに $a_i.t_{min}$ をコピーすることで，他ノードを集合地点と算出しているエージェントも $a_i.t_{min}$ を参照可能となり，a_i が最小 ID だと認識したエージェントのスタート地点マーカーを他エージェントと共有することができる．もし $min\_tmp == writer(a_i.t_{min})$ である場合，a_i が滞在しているノードが集合地点であるため，何も動作はしない．

全ノードにコピー後，ネットワークを 1 周しながら $a_i.t_{min}$ が作成されたノードを探索する．$a_i.t_{min}$ にはノードの署名も含まれており，a_i が各ノードで滞在している時，$a_i.t_{min}$ が作成されたノードか否かを判別することができる．そのため全てのノードを巡回すると，必ず $a_i.t_{min}$ が作成されたノードを発見することができる．$a_i.t_{min}$ が作成されたノードを発見した場合，そのノードを新たな集合地点として移動する．

この動作を繰り返すことにより，最終的に全正常エージェントは同一の最小 ID エージェントが作成したスタート地点マーカーを参照するようになり，同一の集合地点を算出し，集合する．

3.3 アルゴリズムの正当性

補題 1. 正常エージェント a_i は正常エージェント a_j を $a_i.Blacklist$ に加えることはない．

Proof. 正常エージェント a_j はアルゴリズムを開始したノード v にて，スタート地点マーカー $marker_{j,v} = Sign_{j,v}()$ を一度だけ作成する．加えて，任意のビザンチンエージェント a_b は a_j のスタート地点マーカーをコピーすることができるが，署名が施されているため改変することができない．すなわち，a_j が正常エージェントである限り $marker_{j,v'} = Sign_{j,v'}()$ $(v \neq v')$ は分散システム上に存在しない．

正常エージェント a_i は 2 箇所以上で作成された同一 ID エージェント a_b のスタート地点マーカー $marker_{b,v}$ 及び $marker_{b,v'}$ $(v \neq v')$ を確認した場合，$a_i.Blacklist$ にエージェント a_b の ID ID_b を加える．しかし正常エージェント a_j は一度しかスタート地点マーカーを作成しないため，$a_i.Blacklist$ に ID_j が入ることはない． □

補題 2. 正常エージェント a_i において，a_i がアルゴリズム開始時に行う DFS 終了後，$marker_{x,v} \in a_i.All \wedge writer(marker_{x,v}) \notin a_i.Blacklist$ となるスタート地点マーカー $marker_{x,v}$ が少なくとも 1 つ存在する．

Proof. 正常エージェント a_i は自身のスタート地点マーカー $marker_{i,v_{start}}$ を含む，アルゴリズム実行中に確認したスタート地点マーカー全てを $a_i.All$ に保存する．補題 1 より，正常エージェント a_i は $a_i.Blacklist$ に正常エージェントの ID を保存することはない．加えて，a_i 自身も正常エージェントであるため，$a_i.Blacklist$ に ID_i が保存されることはない．したがって a_i がアルゴリズム開始時に行う DFS 終了後，$marker_{i,v_{start}} \in a_i.All \wedge writer(marker_{i,v_{start}}) \notin a_i.Blacklist$ であり，補題が成立する． □

補題 3. 正常エージェント a_i が最初に集合地点の算出を行った後，$a_i.min$ の更新回数は高々$2f$ 回である．

Proof. 正常エージェント a_i がアルゴリズム開始時に DFS にてネットワークを一周した際に，少なくとも全ての正常エージェントのスタート地点マーカーを確認することができる．したがって，a_i が最初に

集合地点の算出を行った際，$a_i.min$ は正常エージェントの中で最小のエージェント ID，もしくはビザンチンエージェントの ID となる．この状態から a_i が $a_i.min$ を更新するのは，$a_i.min > writer(t)$ となるスタート地点マーカー t を目撃した場合，もしくは $a_i.min$ を作成したエージェント a_b がビザンチンエージェントと判明した場合である．$a_i.min > writer(t)$ となるスタート地点マーカー t を目撃した場合，a_i は新たに集合地点の算出を行い，$a_i.min$ を更新する．このとき，$a_i.min > writer(t)$ となるスタート地点マーカーを a_i に目撃させることができるのはビザンチンエージェントのみである．また，$a_i.min$ を作成したエージェント a_b がビザンチンエージェントと判明した場合では，ビザンチンエージェントと判明した ID 以外で集合地点の再計算を行い，$a_i.min$ を更新する．したがって，$a_i.min$ の更新は，1 ビザンチンエージェントにつき最大で 2 度起こりうるため，高々$2f$ 回更新される． □

補題 4. 正常エージェント a_i，a_j の $a_i.min$ と $a_j.min$ は最後の更新を行った後，等しくなる．

Proof. 背理法にて証明する．一般性を失わず，正常エージェント a_i, a_j が $a_i.min, a_j.min$ を最後に更新した後，$a_i.min = x, a_j.min = y, x < y$ であると仮定する．

$a_i.min < a_j.min$ であるためには a_i と a_j は異なるスタート地点マーカー $marker_{x,v}$ 及び $marker_{y,v'}$ をそれぞれ最小 ID が作成したスタート地点マーカーと認識する必要がある．エージェント a_i が $marker_{x,v}$ を最小 ID が作成したスタート地点マーカーだと認識した場合，a_i は全てのノードに $marker_{x,v}$ をコピーする．その後，a_j は a_i によって全ノードにコピーされたスタート地点マーカー $marker_{x,v}$ を確認する．ここで $x < y$ であると仮定しているため，$a_j.min \leq x$ となる．したがって，a_i, a_j が $a_i.min$ と $a_j.min$ の最後の更新を行った後，$a_i.min < a_j.min$ となる状況は起こり得ず，仮定した状況は矛盾する．よって補題 4 は成り立つ． □

補題 5. 全ての正常エージェントは有限時間内に *1* つのノードに集合する．

Proof. 正常エージェント a_i は $a_i.Blacklist$ に含まれていない ID で，最小 ID のエージェント a_{min} によって $marker_{min,v}$ が作成されたノード v を集合地点とする．補題 2 より，少なくとも 1 箇所は集合地点となるノードが存在する．また，補題 4 より，正常エージェント a_i 及び a_j の $a_i.min$ と $a_j.min$ は最後の更新後，等しくなる．加えて，補題 3 より，a_i と a_j が最初に集合地点の算出を行った後，$a_i.min$ と $a_j.min$ の更新回数は高々$2f$ 回である．すなわち，全ての正常エージェント間で同一のスタート地点マーカー $marker_{min,v}$ を参照し，同一のノード v を集合地点として算出し，有限時間内に集合する． □

定理 1. 正常エージェントは高々$2m + 4n + 10fn$ 回の移動で集合する．

Proof. 補題 5 より，正常エージェントは有限時間内に 1 つのノードを集合地点として算出し，集合する．以降では，各エージェントの移動回数を考える．最初にアルゴリズム実行状態に移行した正常エージェント a_i は，DFS を行い全ノードを訪問する．DFS は $2m$ の移動回数で分散システムを 1 周することができる．その後，a_i は集合地点を算出し，最小 ID のエージェントが作成したスタート地点マーカーを全ノードにコピーし，集合地点のノードへ移動する．この移動は a_i が最初の DFS でマップを作成しているため，全ノードにコピーする動作は高々$2n$ 回の移動，集合地点への移動も高々$2n$ 回の移動で行う．

a_i は $a_i.min$ を更新する度，全ノードへスタート地点マーカーをコピーするために，マップを用いて高々$2n$ 回の移動で 1 周し，新たな集合地点へ高々$2n$ 回で移動する．補題 3 より，ビザンチンエージェントによって $a_i.min$ は高々$2f$ 回更新される．また，ビザンチンエージェントの ID が判明した場合，他エージェントにビザンチンエージェントの ID を知らせるため，全ノードへスタート地点マーカーを高々$2n$ 回の移動でコピーする．したがって，a_i は高々$2m+2n+2n+2f(2n+2n)+f\times 2n = 2m+4n+10fn$

回の移動で集合することができる. □

参考文献

[1] A. Ta-Shma and U. Zwick, “Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences,” ACM Transactions on Algorithms (TALG), vol.10, no.3, pp.12:1–15, 2014.

[2] Y. Dieudonné, A. Pelc, and D. Peleg, “Gathering despite mischief,” ACM Transactions on Algorithms (TALG), vol.11, no.1, pp.1:1–28, 2014.

[3] M. Tsuchida, F. Ooshita, and M. Inoue, “Byzantine gathering in networks with authenticated whiteboards,” International Workshop on Algorithms and Computation-Springer, pp.106–118 2017.

[4] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro, “Asynchronous deterministic rendezvous in graphs,” Theoretical Computer Science, vol.355, no.3, pp.315–326, 2006.

[5] Y. Dieudonné, A. Pelc, and V. Villain, “How to meet asynchronously at polynomial cost,” SIAM Journal on Computing, vol.44, no.3, pp.844–867, 2015.

[6] E. Kranakis, D. Krizanc, and E. Markou, “The mobile agent rendezvous problem in the ring,” Synthesis Lectures on Distributed Computing Theory, vol.1, no.1, pp.1–122, 2010.

[7] A. Pelc, “Deterministic rendezvous in networks: A comprehensive survey,” Networks, vol.59, no.3, pp.331–347, 2012.

[8] S. Bouchard, Y. Dieudonné, and B. Ducourthial, “Byzantine gathering in networks,” Distributed Computing, vol.29, no.6, pp.435–457, 2016.

[9] S. Das, F.L. Luccio, and E. Markou, “Mobile agents rendezvous in spite of a malicious agent,” International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotic-sSpringer, pp.211–224 2015.

[10] A. Pelc, “Deterministic gathering with crash faults,” arXiv preprint arXiv:1704.08880, pp.1–27, 2017.

Group exploration of dynamic tori

Tsuyoshi GOTOH†, Yuichi SUDO†, Fukuhito OOSHITA††, Hirotsugu KAKUGAWA†, and Toshimitsu MASUZAWA†

† Graduate School of Information and Science, Osaka University
Yamadaoka 1-5, Suita-si, Osaka, 565-871 Japan
†† Nara Institute of Science and Technology
takayamacho 8961-5, Ikoma-si, Nara, 630-0192 Japan

Abstract Mobile agents (agents) are activities which can move autonomously in a networked system and execute actions at visited nodes. One of the most fundamental problems of agents is exploration, which requires that each node should be visited by at least one agent. For a long time, researchers focus on exploration of static networks. However, exploration of dynamic networks come to be studied recently. In this paper, we consider exploration of a dynamic torus under some constraint on the dynamics (or topology changes). An $n \times n$ torus is considered as a collection of n row rings and n column rings. The constraint on the dynamics is that each the ring should be 1-interval connected, which allows at most one link is missing at any time in each the ring. On this $n \times n$ dynamic torus, we propose exploration algorithms with and without the link presence detection: with the link presence detection, an agent can detect which incident links are missing (if exist) before determining its next move. Specifically, we prove for exploration of the $n \times n$ dynamic torus that, without the link presence detection, $n+1$ agents are necessary and sufficient, and, with the link presence detection, $\lceil n/2 \rceil + 1$ agents are necessary and sufficient.

Key words exploration, dynamic network, dynamic torus, 1-interval connected graph, link presence detection

1. Introduction

Mobile agents (agents) have attracted a lot of attention [3] recently. An agent is a mobile entity (e.g., a software program or a vehicle) which can move autonomously in a networked system and execute actions at visited nodes. In addition, agents can collect information at visited nodes or exchange the information with other agents existing at the visited nodes and they use the information for their execution. An agent can be considered as encapsulation of data and actions; it executes an action at a node and move to another node with the result of the action. From an application perspective, an agent is used for *network maintenance*, *electronic commerce* and *security*.

Exploration is one of the most fundamental problems of agents for accomplishing such applications. Exploration has been studied on many kinds of settings [6], for example, a network with a distinct node labeling or without node labeling (anonymous network), exploration with termination or *perpetual exploration* where agents continue to explore perpetually, and from the point of the number of agents, exploration by one agent or by multiple agents. In most of the previous works is static, that is, network assumed that a network topology does not change or no fault occurs (at nodes and links). Some works considers *fault-tolerant exploration*. Exploration of a network with a black hole (a kind of fault models) is considered in [2].

A network whose topology changes as time proceeds is called a *dynamic network* [4], [11]. This network is used to model a mobile ad-hoc network and a transportation network where topology change happens frequently and is not anomaly but an important feature of these networks. A commonly used model of dynamic networks is *an evolving graph* [7]. It regards a dynamic network as a sequence of (disconnected) static networks. An evolving graph is called *1-time interval connected* when every static network composing it is connected. Moreover, if there is always a consistent connected subgraph on static networks in any T interval, it is called *T-interval connected*.

1.1 Related works

Several works consider exploration of a dynamic network. As a randomized approach, [1] used random walk to explore such a fast changing network. Others consider deterministic exploration of a dynamic network. Flocchini et al. [8] consider a carrier network as a dynamic network model where links appear and disappear depending on the periodic move-

ments of mobile carriers among the nodes. Ilcinkas et al. [10] consider exploration of *T-interval connected rings* by one agent in two cases. One is that a schedule of link change is known a priori to agents and the other is that there is a bound on intervals of link change called δ-appearance. Ilcinkas et al. [9] consider exploration of *T-interval connected cactuses* (consist of rings and trees) by one agent with a priori knowledge of a schedule of link change. The knowledge of topology change can be used when a link of a network represents for instance a public transportation network. However, there exists a case where agents cannot use a schedule of topology change, for example, a mobile ad hoc network or a transportation network where movement of carriers are not scheduled a priori.

Di Luna et al. [5] consider exploration of 1-interval connected rings by a group of agents without a priori knowledge of a schedule of link disappearance. They called this *live exploration*. Moreover, as a main effort, it takes difference of synchrony into account. Specifically, they considered exploration in three cases, fully synchronous, semi synchronous and asynchronous networks.

1.2 Contribution

In this paper, we consider live exploration of *1-interval connected torus* by a group of agents under the settings of full synchrony, distinct node labeling and the requirement of termination detection. A torus can be considered as a collection of row rings and column rings. Informally speaking, a dynamic torus considered in this paper requires that each ring should be 1-interval connected, which guarantees 1-interval connectedness of the whole torus. One may think such a restriction on dynamics looks far-fetched or impractical. However, let us imagine a case where there are mobile obstacles (is not always harmful to a network) in a network whose movements are restricted. For example, mobile vehicles for maintenance of transportation networks or another group of agents which execute other tasks on the same network. In such a case, a schedule of link disappearance may be restricted and not be known to agents. Another important contribution of our work is that we clarify an impact of *link presence detection*: an agent can detect whether incident links of the current node are present or not before determining its next move.

Without the link presence detection, we prove that $n+1$ agents are necessary and sufficient for $n \geqq 3$ and $n+2$ agents are necessary and sufficient for $n = 2$ to explore $n \times n$ dynamic tori. Moreover, we show that the optimal time complexity to explore the dynamic tori is $O(n^2)$ when the number of agents is $n+c$ for any constant $c \geqq 2$. With the link presence detection, we prove that $\lceil n/2 \rceil + 1$ agents are necessary and sufficient for $n = 3, n \geqq 5$ and $\lceil n/2 \rceil + 2$ agents are necessary and sufficient for $n = 4$ to explore the $n \times n$ dynamic tori. Moreover, we show that the optimal time complexity to explore the dynamic tori is $\Theta(n^2)$ when the number of agents is $\lceil n/2 \rceil + c$ for any constant $c \geqq 3$.

2. Preliminaries

2.1 Network model

A network is modeled by a $n \times n$ torus $T = (V, E)$, where $n \geqq 2$ is the size of a torus, V is a set of nodes $\{v_{i,j} \mid 0 \leqq i, j \leqq n-1\}$ and E is a set of links $\{(v_{i,j}, v_{i+1 \bmod n,j}), (v_{i,j}, v_{i,j+1 \bmod n}) \mid 0 \leqq i, j \leqq n-1\}$. Each link is bidirectional; we use $(v_{i,j}, v_{i+1 \bmod n,j})$ (resp, $(v_{i,j}, v_{i,j+1 \bmod n})$) and $(v_{i+1 \bmod n,j}, v_{i,j})$ (resp, $(v_{i,j+1 \bmod n}, v_{i,j})$) as the same link. Intuitively, a torus T consists of n row rings and n column rings. A row ring R_i (resp, a column ring C_j) is an induced subgraph of T, where its node set is $\{v_{i,j} \mid 0 \leqq j \leqq n-1\}$ (resp, $\{v_{i,j} \mid 0 \leqq i \leqq n-1\}$). We denote $\mathcal{R} = \{R_i \mid 0 \leqq i \leqq n-1\}$ and $\mathcal{C} = \{C_j \mid 0 \leqq j \leqq n-1\}$. Each ring of $\mathcal{R}$ and $\mathcal{C}$ is a discrete time-varying graph where, at each time step $t \in \mathbb{N}$, called a *round*, one of its links may be missing. Thus, torus T is a discrete time-varying graph where, at each round $t \in N$ from each ring of $\mathcal{R}$ and $\mathcal{C}$, one of its links may be missing. Since each of the row and column rings are 1-interval connected (or connected at any round), the dynamic torus T is also 1-interval connected [4].

Each node $v_{i,j}$ is labeled by a pair (i, j) of its row number i and column number j and each incident link of $v_{i,j}$ is locally labeled as follows $(v_{i,j}, v_{i,j+1 \bmod n})$, $(v_{i,j}, v_{i,j-1 \bmod n})$, $(v_{i,j}, v_{i+1 \bmod n,j})$, $(v_{i,j}, v_{i-1 \bmod n,j})$ are labeled by *right*, *left*, *down*, *up* respectively. The local link labels of a torus are globally consistent.

Each node has two kinds of rooms (memory space for agents), a waiting room and a transporting room. A node v has one waiting room $room_v$ and four transporting rooms $right_v$, $left_v$, up_v, $down_v$, which are attached to the *right*, *left*, *up*, and *down* links respectively. The transporting room is for an agent that is about to leave v using the corresponding link and the waiting room is for other agents staying at v.

2.2 Mobile agent model

The mobile agent model is based on [5]. In the network, there exists a set $A = \{a_0, \ldots, a_{k-1}\}$ of k computational entities, called agents, which have local memory and computational capabilities. An agent is *mobile*; it can move from a node to its neighboring node through a link. An agent has the knowledge of torus size n. An agent can use the node label and the link label of the current node for its execution; its execution can depend on its location (the label (i, j) of the current node). An arbitrary number of agents can reside on

waiting room $room_v$ of a node v and at most one agent can reside at the same time on each transporting room of v. An agent can detect whether another agent resides on each room of the current node. However, it cannot detect the number of agents in $room_v$. An agent has no tool to communicate with other agents.

The execution of agents is fully synchronized, that is, all the agents execute their actions simultaneously at each round. The execution of agents in each round is divided into three phases, **Look**, **Compute** and **Move**. We consider two kinds of models in this paper, the model with the link presence detection and without the link presence detection. In each of the two models, the action of agent a_i of each phase is as follows.

Look : Agent a_i gets the node label and its current room ($room_v$ or transporting rooms) and checks, for each room, whether another agent exists on the room or not. In the model with the link presence detection, the agent also checks whether, for each of the four links incident to the current node, the link is present or not. We call the information collected in this phase a *snapshot*.

Compute : Based on the snapshot and the content of its local memory, a_i decides whether it moves from the current node or not, and, if a_i decides to move, it also decides the direction it moves to. Then, if a_i decides move, it tries to move to the transporting room corresponding to the chosen direction. Move to a transporting room is implemented by mutual exclusion, that is, if multiple agents try to move to the same transporting room at the same round, only one of them succeeds to move to the room, and the others fail and it stays at $room_v$ of the current node.

Move : If a_i is at a transporting room, it executes this phase. Otherwise, a_i does nothing until the end of this phase at the waiting room. If the link corresponding to its current transporting room is present, a_i moves through the link and reaches the waiting room of the neighboring node at the beginning of the next round. Otherwise, it stays at the current room of the current node. As mentioned before, the access to the transporting room is implemented by mutual exclusion, that is, at most one agent can move in each direction of each link at each round.

Each action is executed synchronously, that is, after all the agents finish actions of the t-th round, they start actions of the $(t+1)$-th round.

We define *configuration* V_t as a multiset (or bag) of k nodes where agents exist at the beginning of round t. The number of the same node contained in V_t corresponds to the number of agents staying at the node. Note that V_t denotes the current nodes of agents and does not denote the current rooms of agents. We call configuration V_0 an initial configuration. Without loss of generarity, in V_0, we can assume that each agent is in a waiting room of some node. This is because each agent at a transporting room moves to $room_v$ during **Computation** at first round.

3. Execution on 1-interval connected rings

In this section, we present two methods on 1-interval connected rings that are used for exploration of a dynamic torus. In the method, an agent uses the procedure Move($dir \mid p_1 : s_1; p_2 : s_2; \ldots; p_\ell : s_\ell$) where $dir \in \{right, left, down, up, nil\}$, p_i is a predicate and s_i is a state. If an agent changes its state into **Return**, it returns from a procedure. Procedure Move makes an agent move in the direction dir and change its state depending on the predicate satisfied first when evaluating them from p_1 to p_ℓ. If dir is nil, an agent stays at $room_v$. Notice that the methods work in the models with and without the link presence detection. Hereafter, we use the following variables.

- $Ftime$: the number of rounds since Algorithm 1, 2 or 5 is called last.
- $current$: the current node of an agent.

We say that an agent a *catches* another agent a' at round t and a' is *caught* by a when there is a' at round t on the transporting room corresponding to the direction where a moves forward at round $t-1$ after a successes to move.

Exploration: Algorithm 1 is a pseudo code of exploration of 1-interval connected rings borrowed from [5], where we assume that the port labeling of the ring is globally consistent (or it can be considered as a column ring of the torus). Algorithm 1 is executed by two or more agents from arbitrary initial configurations. At first, agent a_i moves in the direction *up*. If a_i catches another agent a_j, a_i bounces (moves in the direction *down*) and they do not change their direction after that. Agent a_i stops its execution at the $3n$-th round from the start of ExplorationUp and, by the $3n$-th round, agents complete exploration. In [5], authors proved that, only in the case where there are two agents, Algorithm 1 explores 1-interval connected rings. So, we show that, in the case where there are three or more agents, Algorithm 1 explores 1-interval connected rings.

Algorithm 1 ExplorationUp

```
1: In state Init:
2:   Move(up | Ftime ≧ 3n : Return; catches : Bounce; caught : Forward);
3: In state Bounce:
4:   Move(down | Ftime ≧ 3n : Return);
5: In state Forward:
6:   Move(up | Ftime ≧ 3n : Return);
```

Lemma 3.1 *By* ExplorationUp*, two or more agents on a column ring explore the column ring by the* $3n$*-th round from the start of* ExplorationUp*.*

Proof 3.1 *We first show that two agents explore the column ring by Algorithm 1. If* a_i *succeeds to move forward* n *times, it completes exploration. If* a_i *catches another agent* a_j *by the* $2n$*-th round from the start of* ExplorationUp*,* a_i *bounces. This guarantees that they complete exploration after at most* n *additional rounds. If* a_i *neither successes to move* n *times nor catches another agent by the* $2n$*-th round,* a_i *is definitely caught by another agent by the* $2n$*-th round from the start of* ExplorationUp*. If it happens, they complete exploration after at most* n *additional rounds. Thus, by the* $3n$*-th round from the start of* ExplorationUp*, agents complete exploration of 1-interval connected rings.*

Next, we show that three or more agents explore the column ring by Algorithm 1. Similarly, by the $2n$*-th round, an agent (say* a'_i*) catches another agent (say* a'_j*) and agents complete exploration otherwise. After* a'_i *catches* a'_j*,* a'_i *moves in the direction down and* a'_j *moves in the direction up and they do not change their direction. Since at most one agent can move in each direction of each link at each round,* a'_i *and* a'_j *may not complete exploration by the* n*-th round when* a'_i *or* a'_j *fails to move by the mutual exclusion rule. However, at this time, another agent succeeds to move if there is a link. Then, we rename the agent* a'_i *(resp,* a'_j*) when* a'_i *(resp,* a'_j*) fails to move by the mutual exclusion rule and, by* n*-th round, agents complete exploration of the column ring.*

Arrangement on a specific node: Algorithm 2 is the pseudo code which aims to locate one of two agents or two of three or more agents on a specific node in a 1-interval connected ring from arbitrary initial configurations (Algorithm 2 is our work). In the algorithm, we assume that the ring is a column ring of the torus and so each node is labeled with its row and column numbers. The specific node where the algorithm aims to locate an agent is the node in the row i (or in R_i). Let v be the specific node. By the $3n$-th round from the start of ArrangementUp(i), agent behavior is same as that in Algorithm 1 except that an agents changes its state to **Wait** when it reaches v. Once an agent changes its state **Wait**, it keeps its state and location until the end of Algorithm 2. At the $3n$-th round from the start of ArrangementUp(i), unless a_i reaches v, it starts the second exploration that is the exactly same as the first one. Agents stop their execution at the $6n$-th round from the start of ArrangementUp(i). ArrangementLeft(j) for a row ring is similarly defined by replacing *up*, *down* and i to *left*, *right* and j. A correctness proof is as follows.

Algorithm 2 ArrangementUp(i)

1: In state **Init**:
2: Move(*up* | *Ftime* $\geqq 3n$: **Init'**; *current* $\in R_i$: **Wait**; *catches* : **Bounce**; *caught* : **Forward**);
3: In state **Bounce**:
4: Move(*down* | *Ftime* $\geqq 3n$: **Init'**; *current* $\in R_i$: **Wait**);
5: In state **Forward**:
6: Move(*up* | *Ftime* $\geqq 3n$: **Init'**; *current* $\in R_i$: **Wait**);
7: In state **Init'**:
8: Move(*up* | *Ftime* $\geqq 6n$: **Return**; *current* $\in R_i$: **Wait**; *catches* : **Bounce'**; *caught* : **Forward'**);
9: In state **Bounce'**:
10: Move(*down* | *Ftime* $\geqq 6n$: **Return**; *current* $\in R_i$: **Wait**);
11: In state **Forward'**:
12: Move(*up* | *Ftime* $\geqq 6n$: **Return**; *current* $\in R_i$: **Wait**);
13: In state **Wait**:
14: Move(*nil* | *Ftime* $\geqq 6n$: **Return**);

Lemma 3.2 *By* ArrangementUp(i) *(resp,* ArrangementLeft(j)*), one of two agents or two of three or more agents on a column (resp, row) ring reach and stay at specific node of the column (resp, row) ring by the* $6n$*-th round from the start of* ArrangementUp(i) *(resp,* ArrangementLeft(j)*).*

Proof 3.2 *By the* $3n$*-th round from the start of* ArrangementUp(i) *round, agents complete exploration. Hence, one of the agents reaches and stays at* v*. Similarly, by the* $6n$*-th round from the start of* ArrangementUp(i)*, another agent must visit* v *if there are three or more agents. Thus, one of two agents or two of three or more agents on a column ring reach and stay at specific node of the column ring by the* $6n$*-th round from the start of* ArrangementUp(i)*. Similarly, the same thing holds for* ArrangementLeft(j)

In the following sections, we use Algorithms 1 and 2 as tools. Note that, as ExplorationUp (resp, Arrangement(dir, i)) finishes in exactly $3n$-th (resp, $6n$-th) rounds, agents can detect their termination at the same time.

4. Exploration without the link presence detection in tori

In this section, we consider the case without the link presence detection. In this case, we show that $n+1$ agents are necessary and sufficient to explore the $n \times n$ dynamic torus T. Specifically, we first prove that a group of n (or less) agents cannot explore the dynamic torus, and we give the algorithm by which a group of $n+1$ (or more) agents can explore the torus where $n \geqq 3$ in $O(n^3)$ rounds. Furthermore, we give a faster algorithm with a group of $n+2$ (or more) agents, which explores the torus within $O(n^2)$ rounds. This is optimal time complexity; a group of $n + O(1)$ agents cannot

explore the torus T within $o(n^2)$ rounds as we will show in the last of Section 4. 1. In the following, we first show the impossibility results when $k \leqq n$ and when $k = 3$ and $n = 2$ (Recall that k is the number of agents). In Section 4. 1, we give an algorithm for $k \geqq n + 2$, and prove the optimality of it in terms of time complexity. In Section 4. 2, we give an algorithm for $k \geqq n + 1$.

Lemma 4.1 *Without the link presence detection, a group of n (or less) agents cannot explore the $n \times n$ dynamic torus.*

Proof 4.1 *Suppose that there is an agent on each node of a diagonal line of T ($v_{0,0}$, $v_{1,1}$, ..., $v_{n-1,n-1}$) in V_0. In this case, an adversary can delete the links which agents choose in* **Compute** *phase since there is only one agent on each ring of $\mathcal{R}$ and $\mathcal{C}$. Then, all the agents cannot leave the current node forever.*

Lemma 4.2 *Without the link presence detection, for $n = 2$, a group of three agents cannot explore a dynamic torus.*

Proof 4.2 *Suppose that there are three agents, on $v_{0,0}$, $v_{0,1}$ and $v_{1,0}$. We show that this configuration does not change by any algorithm. Let a_0 be the agent on $v_{0,0}$, a_1 be the agent on $v_{0,1}$ and a_2 be the agent on $v_{1,0}$. The execution of agents is classified into three cases, one is that a_0 moves in the row direction, another is that a_0 moves in the column direction, the other is that a_0 does not move. In the first case, an adversary stops a_2 and, if a_1 moves in the row direction, the adversary let a_0 and a_1 move (consequently the locations of a_0 and a_1 are swapped), otherwise (or a_1 does not move or moves in the column ring), the adversary makes a_0 and a_1 stay at their current nodes. Thus, the configuration does not change. Similarly, in the second case, the configuration does not change. In the third case, the adversary makes both a_1 and a_2 stay at their current nodes. Then, the configuration does not change. Thus, the configuration does not change by any algorithm and a group of three agents cannot explore a dynamic torus for $n = 2$.*

4. 1 By $n + 2$ or more agents

Algorithm 3 is an exploration algorithm of a dynamic torus by $n + 2$ or more agents.

Algorithm 3 Exploration by n+2 agents

```
1: for j = 0 to n − 1
2:     ArrangementLeft(j);
3:     ExplorationUp;
4: end for
```

At line 2, an agent executes ArrangementLeft(j) where j is a column number; an agent moves to C_j in the direction *left*. Since there are $n + 2$ agents, there exist at least two rings of $\mathcal{R}$ with two agents or at least one ring of $\mathcal{R}$ with three agents. Hence, after line 2 finished, there exist at least two agents on C_j. Since there is two agents on C_j, ring C_j is explored by ExplorationUp at line 3. By repeating this from C_0 to C_{n-1}, agents complete exploration. This algorithm obviously explores a dynamic torus with $n+2$ or more agents.

Lemma 4.3 *Without the link presence detection, for $n \geqq 2$, a group of $n + 2$ agents explore a $n \times n$ dynamic torus in $O(n^2)$ rounds by Algorithm 3.*

Proof 4.3 **Correctness:** *It is obvious that $n + 2$ agents can explore a dynamic torus by Algorithm 3.*

Time evaluation: *It takes $6n$ and $3n$ rounds to execute* ArrangementLeft(j) *and* ExplorationUp *respectively and the number of repetition at line 2 is n times. From these, $n + 2$ agents explore the $n \times n$ dynamic torus in $9n^2$ rounds by Algorithm 3.*

Corollary 1 *Without the link presence detection, for $n = 2$, $n + 2$ (four) agents are necessary and sufficient for exploration of the $n \times n$ dynamic torus.*

From Lemma 4.3, the following theorem holds.

Theorem 4.1 *Without the link presence detection, for $n \geqq 2$ and for any constant $c \geqq 2$, $\Theta(n^2)$ rounds are necessary and sufficient for exploration of the $n \times n$ dynamic torus with $n+c$ agents.*

Proof 4.4 *Sufficiency: It is from Lemma 4.3.*

Necessity: Suppose that there are $n+c$ agents in a dynamic torus where $c \geqq 2$. Since the adversary can make n agents stay at their current nodes by the argument in the proof of Lemma 4.1 and the number of nodes in a dynamic torus is n^2, it takes n^2/c rounds to visit all the nodes with $n + c$ agents.

4. 2 By $n + 1$ agents

Algorithm 4 is an exploration algorithm of an $n \times n$ dynamic torus by $n + 1$ or more agents.

Like Algorithm 3, Algorithm 4 makes two agents move on C_j and explore it for each j ($0 \leqq j \leqq n - 1$) one by one. At line 2, an agent executes ArrangementLeft(j) where j is the column number of a destination node. Since there are $n + 1$ agents, there exists at least one ring of $\mathcal{R}$ with two or more agents. Hence, when ArrangementLeft(j) at line 2 finishes, there exists at least one agent on C_j.

Then, an agent executes ArrangementUp(i mod n) and ArrangementLeft(j) $n + 2$ times in the for-loop at lines

Algorithm 4 Exploration by $n+1$ agents

```
1: for j = 0 to n − 1
2:    ArrangementLeft(j);
3:    for i = 0 to n + 1
4:       ArrangementUp(i mod n);
5:       ArrangementLeft(j);
6:    end for
7:    ExplorationUp;
8: end for
```

3-6; in the first n times, each agent tries to move to a destination in R_i for each i $(0 \leqq i \leqq n-1)$ one by one, then, it moves again to R_0 in the $n+1$-th loop and to R_1 in the $n+2$-th loop. Note that, an agent which reaches C_j stays at C_j during the for-loop in lines 3-6.

When the for-loop at lines 3-6 finishes, there are at least two agents on C_j and these agents explore C_j. By repeating this from C_0 to C_{n-1}, agents complete exploration of the $n \times n$ dynamic torus.

Lemma 4.4 *Without the link presence detection, for $n \geqq 3$, a group of $n+1$ agents explore the $n \times n$ dynamic torus in $O(n^3)$ rounds by Algorithm 4.*

Proof 4.5 Correctness: *Without loss of generality, it suffices to show that the agents explores ring C_0 during the first execution of the for-loop at lines 1-13, that is, $j = 0$. To prove that, it suffices to show that two or more agents exist on ring C_0 at the start of the first execution of line 7 because two agents can explore a single ring by* ExplorationUp. *Furthermore, since at least one agent reaches C_0 by* ArrangementLeft(j) *at line 2, it suffices to show that one or more agents reach C_0 during the execution of the for-loop at lines 3-6 when only one agent is on C_0.*

Let C'_0 be an induced subgraph of T where its node set is $\{v_{i,j} \mid 0 \leqq i \leqq n-1, 1 \leqq j \leqq n-1\}$, i.e. $C'_0 = T \backslash C_0$ and let R'_i be an induced subgraph of C'_0 where its node set is $\{v_{i,j} \mid 1 \leqq j \leqq n-1\}$. We show that by contradiction. We assume that no agent reaches C_0 from C'_0 at any round t during the execution of the for-loop at lines 3-6 when only one agent is on C_0.

Obviously, since only one agent reaches C_0 during ArrangementLeft(j) *at line 2 and no agent reaches C_0 during* ArrangementLeft(j) *at line 5, there is one agent on each R'_i at the start and end of each* ArrangementLeft(j) *at line 2 and 5. Note that every agent on C'_0 does not moves in the direction right by* ArrangementLeft(j) *at line 5 since an agent moves in the direction left by* ArrangementLeft(j) *at line 5 unless it catches another agent (if it happens, one agent reaches C_0). Then, at the start of each* ArrangementUp$(i \bmod n)$, *there is one agent on each R'_i. At this time, we show that no agent moves by* ArrangementUp$(i \bmod n)$ *at line 4. At first, by the construction of* ArrangementUp(i), *an agent on R'_i does not move by* ArrangementUp$(i \bmod n)$. *Then, an agent on R'_{i+1} also does not move by* ArrangementUp$(i \bmod n)$ *since one agent stays at R'_i (if the agent on R'_{i+1} moves, there are two agents on R'_i). Similarly, every agent on each R'_i does not move by* ArrangementUp$(i \bmod n)$. *Thus, no agent moves by* ArrangementUp$(i \bmod n)$ *at line 4. Moreover, since no agent moves by* ArrangementUp$(i \bmod n)$ *at line 4 and there is one agent on each R'_i, there is one agent on $v_{i,j}$ and there is one agent on each column ring in C'_0 other than C_j (there are two agents on C_j).*

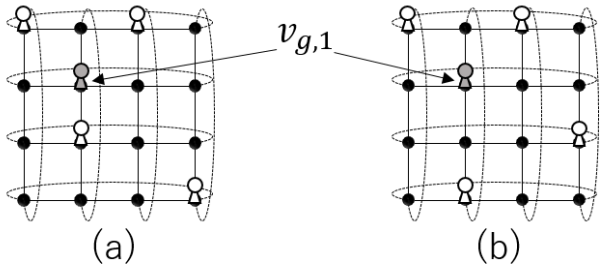


図 1 impossible case for $n = 4$

Let g be a row number where $0 \leqq g \leqq n-1$ and an agent is on $v_{g,1}$ in C_1. At the start of g-th for-loop at lines 3-6, two agents are on C_1 since no agent moves by ArrangementUp$(g \bmod n)$ *at line 4. In the case where the other agent on C_1 is on the node other than $v_{g+1 \bmod n,1}$, one of two agents on C_1 moves to $R'_{g+1 \bmod n}$ by* ArrangementUp$(g+1 \bmod n)$ *at line 4 during the $g+1$-th for-loop at lines 3-6. It is contradiction since no agent moves by* ArrangementUp$(i \bmod n)$. *In other case where the other agent on C_1 is on $v_{g+1 \bmod n,1}$, one of two agents on C_1 moves to $R'_{g+2 \bmod n}$ by* ArrangementUp$(g+2 \bmod n)$ *at line 4 during the $g+2$-th for-loop at lines 3-6. It is contradiction. Hence, one or more agents reach C_0 during the execution of the for-loop at lines 3-6. Thus, agents explores ring C_0 during the first execution of the for-loop at lines 1-13 and explore each C_j during the j-th execution of the for-loop at lines 1-13.*

Time evaluation: *As each execution of lines 4 and 5 takes $12n$ rounds and the number of repetition of the for-loop in lines 3-6 is $n+2$, each execution of the for-loop at lines 1-8 takes $12n(n+2)+9n = 12n^2+33n$ rounds. As the number of repetition of the for-loop in lines in 1-8 is n, the total number of rounds is $12n^3+33n^2$.*

From Lemmas 4.1 and 4.4, we get the following theorem.

Theorem 4.2 *Without the link presence detection, for $n \geqq 3$, $n+1$ agents are necessary and sufficient for exploration of the $n \times n$ dynamic torus.*

5. Exploration with the link presence detection in tori

In this section, we consider the case with the link presence detection and show that the link detection has an considerable influence on the number of agents required to explore the dynamic torus: we show that $\lceil n/2 \rceil + 1$ agents are necessary and sufficient to explore the $n \times n$ dynamic torus T. Specifically, we prove that a group of $\lceil n/2 \rceil$ (or less) agents cannot explore the torus, and a group $\lceil n/2 \rceil + 1$ (or more) agents can explore the torus for $n \geqq 5$. Furthermore, we give an faster algorithm with a group of $\lceil n/2 \rceil + 2$ (or more) agents, which explore the torus within $O(n^2)$ rounds. This is optimal time complexity; a group of $\lceil n/2 \rceil + O(1)$ agents cannot explore the torus T within $o(n^2)$ rounds. In the following, we first show the impossibility results for the case of $k \leqq \lceil n/2 \rceil$ and $n \geqq 3$ and the case of $k = 3$ and $n = 4$. Second, we give an algorithm for $k \geqq \lceil n/2 \rceil + 2$ and $n \geqq 3$, and prove the optimality of its time complexity. Third, we give an algorithm for $k \geqq \lceil n/2 \rceil + 1$ and $n \geqq 5$. Finally, we give an algorithm for $k = 3$ and $n = 3$.

If $n = 2$, one agent is sufficient and (obviously) necessary to explore a dynamic torus. When $n = 2$, each node is connected with each neighboring node by two links and one of the links is always present. With the link presence detection, the agent can always choose the present link to move to the neighboring node. Thus, one agent is sufficient to explore the 2×2 dynamic torus.

5.1 Impossibility results

First, we show a building block of the proof of the impossibility result: an adversary can forever contain an agent in a 2×2 subgrid consisting of four nodes, $v_{i,j}$, $v_{i+1,j}$, $v_{i,j+1}$ and $v_{i+1,j+1}$. Suppose that an agent is in the subgrid and is located at node v. By removing two links connecting v to the nodes outside the subgrid, the adversary can prevent the agent from going out from the subgrid. So the agent can move only in the subgrid. By repeating the argument, we can show that the agent can be contained in the subgrid forever. Notice that when another agent comes into the subgrid, we allow the newly coming agent to go out from the subgrid.

Now, we prove the following lemma.

Lemma 5.1 *With the link presence detection, for $n \geqq 3$, a group of $\lceil n/2 \rceil$ agents cannot explore the $n \times n$ dynamic torus.*

Proof 5.1 *We consider two cases, one is that n is even and the other is that n is odd. First, we consider the case where n is even. Let a_i ($0 \leqq i \leqq n/2 - 1$) be $n/2$ agents in the torus and suppose a_i is located at $v_{2i,2i}$ in V_0. In this case, by the above building block, a_i can be forever contained in the subgrid consisting of $v_{2i,2i}$, $v_{2i+1,2i}$, $v_{2i,2i+1}$ and $v_{2i+1,2i+1}$. Thus, when n is even, a group of $\lceil n/2 \rceil$ agents cannot explore the dynamic torus.*

Next, we consider the case where n is odd. Let a_i ($0 \leqq i \leqq (n-1)/2$) be $(n+1)/2$ agents in the torus and suppose a_j ($0 \leqq j \leqq (n-3)/2$) is located at $v_{2j,2j}$ and $a_{(n-1)/2}$ is located at any node other than $v_{n-1,n-1}$. By the building block, a_j can be forever contained in the subgrid consisting of $v_{2j,2j}$, $v_{2j+1,2j}$, $v_{2j,2j+1}$ and $v_{2j+1,2j+1}$. Agent $a_{(n-1)/2}$ can go through the subgrids but can be prevented from visiting $v_{n-1,n-1}$ by removing the link to $v_{n-1,n-1}$ by removing the link to $v_{n-1,n-1}$ every time it comes to a neighboring node of $v_{n-1,n-1}$. Consequently, any agent can never visit $v_{n-1,n-1}$ and thus a group of $\lceil n/2 \rceil$ agents fails to explore the torus.

Next, we prove the following lemma for $n = 4$.

Lemma 5.2 *With the link presence detection, for $n = 4$, a group of three ($\lceil n/2 \rceil + 1$) agents cannot explore the 4×4 (or $n \times n$) dynamic torus.*

Proof 5.2 *Let $V_{0,0} = \{v_{0,0}, v_{0,1}, v_{1,0}, v_{1,1}\}$, $V_{0,1} = \{v_{0,2}, v_{0,3}, v_{1,2}, v_{1,3}\}$, $V_{1,0} = \{v_{2,0}, v_{2,1}, v_{3,0}, v_{3,1}\}$ and $V_{1,1} = \{v_{2,2}, v_{2,3}, v_{3,2}, v_{3,3}\}$, and suppose that in each $V_{0,0}$, $V_{0,1}$ and $V_{1,0}$ contains one agent. We show that any algorithm may not change the situation where each $V_{0,0}$, $V_{0,1}$ and $V_{1,0}$ contains one agent. Let a_0 be the agent in $V_{0,0}$, a_1 be the agent in $V_{0,1}$ and a_2 be the agent in $V_{1,0}$.*

If a_0 and a_1 are on different row rings (e.g., a_0 is on $v_{0,0}$ and a_1 on $v_{1,2}$) or they are on neighboring nodes (e.g., a_0 is on $v_{0,1}$ and a_1 is on $v_{0,2}$), an adversary can easily prevent both a_0 and a_1 from moving to $V_{0,1}$ and $V_{0,0}$ respectively. The same argument holds for a_0 and a_2.

So it suffices to consider the case where a_0 and a_1 (resp, a_2) are on the same R_i (resp, C_j) but not on neighboring nodes. Without loss of generality, we assume that a_0 is on $v_{0,0}$ and a_1 (resp, a_2) is on $v_{0,2}$ (resp, $v_{2,0}$).

At first, an adversary simulates the agents execution when no link is deleted. Then, depending on that simulation, the adversary determines whether it deletes a link and, if so, which link it deletes.

case1: *If a_1 (resp, a_2) moves in the direction right or down (resp, down or right), the adversary deletes $(v_{0,0}, v_{0,3})$ (resp, $(v_{0,0}, v_{3,0})$) to prevent a_0 from moving to $V_{0,1}$ (resp, $V_{1,0}$). Notice that a_1 (resp, a_1) is moving in $V_{0,1}$ (resp, $V_{1,0}$).*

case2: *If a_1 (resp, a_2) moves in the direction left (resp,*

up), the adversary considers execution of a_0. If a_0 moves in the direction left (resp, up), the adversary does nothing, which allows a_0 to move to $V_{0,1}$ (resp, $V_{1,0}$) and a_1 (resp, a_2) to move to $V_{0,0}$. Otherwise, the adversary deletes $(v_{0,1}, v_{0,2})$. (resp, $(v_{1,0}, v_{2,0})$ to prevent a_1 (resp, a_2) from moving to $V_{0,0}$.

case3: *If a_1 (resp, a_2) moves in the direction up (resp, left), the adversary deletes $(v_{0,2}, v_{3,2})$ (resp, $(v_{2,0}, v_{2,0})$) to prevent a_1 (resp, a_2) from moving to $V_{1,1}$. At the same time, the adversary removes $(v_{0,0}, v_{0,3})$ (resp, $(v_{0,0}, v_{3,0})$ to prevent a_0 from moving to $V_{0,1}$ (resp, $V_{1,0}$).*

From the above argument, we can show each $V_{0,0}$, $V_{0,1}$ and $V_{1,0}$ contains one agent in any configuration. Thus, for $n = 4$ and $k = 3$, a group of three ($\lceil n/2 \rceil + 1$) agents cannot explore the dynamic torus.

5.2 By $\lceil n/2 \rceil + 2$ agents

With the link presence detection, at each round t, an agent can move in the direction *right* or *left* (if it wants to do) since each row ring has at most one missing link. This also holds for the direction *down* or *up*.

A main idea of exploration is that each agent moves to a neighboring node (if necessary) so that it is in a column ring with an odd column number, and executes an algorithm similar to Algorithm 3 or Algorithm 4.

ArrangementV(j) in Algorithm 6 makes agents move one step vertically (if necessary) to row rings with even (resp, odd) row numbers when j is even (resp, odd). The only exception is the case that both n and j are odd: the algorithm allows agents to move R_0 in addition to row rings with odd row numbers. Note that at most one agent can move in each direction of each link at each round. So, an agent executes Algorithm 5 in four rounds to guarantee that there are two column rings with two agents or is one column ring with three or more agents after the execution of Algorithm 5.

Algorithm 5 ArrangementV(i)

1: Let $v_{p,q}$ be *current*;
2: **if** (j and q are *even*) or (j and q are *odd*) or ((n and j are *odd*) and ($q = 0$)) **then** Move(*nil* | *Ftime* $\geqq 4$: **Return**);
3: **else if** (($v_{p,q}, v_{p,q-1}$) exists) **then** Move(*up* | *Ftime* $\geqq 4$: **Return**);
4: **else** Move(*down* | *Ftime* $\geqq 4$: **Return**);

This is used as ArrangementV(j) where *row* denotes that an agent moves in the row direction. We also use ArrangementH(i) by replacing *right* to *down*, *left* to *up* and i to j for arranging agents on column rings with even or odd column numbers.

Algorithm 6 is an exploration algorithm of the $n \times n$ dynamic torus by $\lceil n/2 \rceil + 2$ (or more) agents.

Algorithm 6 Exploration by $\lceil n/2 \rceil + 2$ agents

1: **for** $j = 0$ to $n - 1$
2: ArrangementV(0);
3: ArrangementLeft(j);
4: ExplorationUp;
5: **end for**

Like the algorithms in Section 4, Algorithm 6 makes two agents move on and explore each C_j one by one. At line 2, an agent executes ArrangementV(0); an agent moves to a row ring with an even row number. Then, at line 3, an agent executes ArrangementLeft(j) where j is a column number; an agent moves to C_j in the direction *left*. Since there are $\lceil n/2 \rceil + 2$ agents, there exists at least two row rings of $\mathcal{R}$ with two agents or at least one row ring of $\mathcal{R}$ with three agents. Hence, when line 3 finished, there exist at least two agents on C_j. Since there is two agents on C_j, ring C_j is explored by ExplorationUp at line 4. By repeating this from C_0 to C_{n-1}, agents complete exploration. This algorithm obviously explores the $n \times n$ dynamic torus with $\lceil n/2 \rceil + 2$ or more agents.

The following lemma holds.

Lemma 5.3 *With the link presence detection, for $n \geqq 2$, a group of $\lceil n/2 \rceil + 2$ agents explore the $n \times n$ dynamic torus in $O(n^2)$ rounds by Algorithm 6.*

Proof 5.3 **Correctness:** *It is obvious that $\lceil n/2 \rceil + 2$ agents can explore a dynamic torus by Algorithm 6.*

Time evaluation: *It takes 4, $6n$ and $3n$ rounds to execute* ArrangementC(0), ArrangementLeft(j) *and* ExplorationUp *respectively and the number of repetition of the for-loop at lines 1-5 is n times. From these, It takes $(9n + 4)n = 9n^2 + 4n$ rounds that is $O(n^2)$ rounds.*

Corollary 2 *With the link presence detection, for $n = 4$, $\lceil n/2 \rceil + 2$ (four) agents are necessary and sufficient for exploration of the $n \times n$ dynamic torus.*

From Lemma 5.3, the following theorem holds.

Theorem 5.1 *With the link presence detection, for $n \geqq 3$ and for any constant $c \geqq 3$, $\Theta(n^2)$ rounds are necessary and sufficient for exploration of the $n \times n$ dynamic torus with $\lceil n/2 \rceil + c$ agents.*

Proof 5.4 *Sufficiency: It is from Lemma 5.3.*

Necessity: Suppose that there are $\lceil n/2 \rceil + c$ agents in the dynamic torus where $c \geqq 3$. Since an adversary can make at most $\lceil n/2 \rceil$ agents stay at their current node by Lemma 5.1 and the number of nodes of the dynamic torus is n^2, it takes

n^2/c rounds to visit all the nodes with $\lceil n/2 \rceil + c$ agents.

5.3 By $\lceil n/2 \rceil + 1$ agents

Algorithm 6 is an exploration algorithm of a dynamic torus by $n/2 + 1$ or more agents.

Algorithm 7 Exploration by $\lceil n/2 \rceil + 1$ agents

```
1: for j = 0 to n − 1
2:    ArrangementC(0);
3:    ArrangementLeft(j);
4:    for i = 0 to ⌊(n − 1)/2⌋, 0 and 1
5:       ArrangementR(j);
6:       ArrangementUp(2i mod n);
7:       ArrangementC(0);
8:       ArrangementLeft(j);
9:    end for
10:   ExplorationUp;
11: end for
```

Like above algorithms, Algorithm 7 makes two agents move on and explore each C_j.

At line 2, an agent executes ArrangementR(0) to move to an row ring with an even row number. At line 3, an agent executes ArrangementLeft(j) where j is the column number of a destination node. Since there are $\lceil n/2 \rceil + 1$ agents, there exists at least one ring of $\mathcal{R}$ with two or more agents. Hence, after line 3 finished, there exists at least one agent on C_j.

Then, an agent executes a for-loop at lines 4-8. A set of execution ArrangementR(j) at line 5 and ArrangementUp($i \bmod n$) at line 6 works like ArrangementUp($i \bmod n$) at line 4 of Algorithm 4 and a set of execution of ArrangementC(0) at line 6 and ArrangementLeft(j) at line 7 works like ArrangementLeft(j) at line 5 of Algorithm 4. The agent executes the for-loop at lines 4-8 for $\lceil n/2 \rceil + 2$ times; in the first $\lceil n/2 \rceil$ times, an agent moves to a destination $R_2 i$ from $i = 0$ to $\lfloor (n-1)/2 \rfloor$, then, it moves to R_0 at the $\lfloor (n-1)/2 \rfloor + 1$-th for-loop at lines 4-8 and to R_2 at the $\lfloor (n-1)/2 \rfloor + 2$-th for-loop at lines 4-8. Note that, an agent which reaches C_j stays at C_j until the for-loop at lines 4-8 finishes.

After the end of the for-loop at lines 4-8 finishes, there are at least two agents on C_j and these agents explore C_j. By repeating this from C_0 to C_{n-1}, agents complete exploration of a dynamic torus.

Lemma 5.4 *With the link presence detection, for $n \geqq 5$, a group of $n + 1$ agents explore a dynamic torus in $O(n^3)$ rounds by Algorithm 7.*

Proof 5.5 Correctness: *The correctness proof of Algorithm 7 is similar to one of Algorithm 4.*

Time evaluation:

since they halts within constant time. As each execution of line 6 and 7 takes $12n$ rounds and the number of repetition of line 4 is $n + 2$, each execution of line 1 except for ArrangementC(0) *and* ArrangementR(j) *takes $12n(n+2)+8n = 12n^2+32n$ rounds. As the number of repetition of line 1 is n, the total number of rounds is $12n^3+32n^2$, that is, $O(n^3)$.*

5.4 By three agents for $n = 3$

When $n = 3$ and $k = 3$, agents cannot explore a dynamic torus by Algorithm 7. In this case, we use Algorithm 8 for exploration of a dynamic torus.

Algorithm 8 Exploration by three agents for $n = 3$

```
1: for j = 0 to 2
2:    ArrangementC(0);
3:    ArrangementLeft(j);
4:    ArrangementR(j);
5:    current is v_{p,q};
6:    r = p;
7:    move to v_{1,q};
8:    ArrangementLeft(j);
9:    ArrangementR(j);
10:   Let v_{p,q} be current;
11:   if (p = 1) then
12:      if (r = 0) then
13:         wait one round if (v_{1,q}, v_{2,q}) is present, otherwise, move
      to v_{0,q};
14:      else
15:         wait one round if (v_{1,q}, v_{0,q}) is present, otherwise, move
      to v_{2,q};
16:   else if (p = r) then
17:      if (r = 0) then
18:         move to v_{1,q} if (v_{1,q}, v_{0,q}) is present, otherwise, move
      to v_{2,q};
19:      else
20:         move to v_{1,q} if (v_{1,q}, v_{2,q}) is present, otherwise, move
      to v_{0,q};
21:   ArrangementLeft(j);
22: end for
```

Like above algorithms, Algorithm 8 makes two agents move on and explore each C_j.

The main idea of Algorithm 8 is that two agents on the same ring are on the same node at least once during the following executions. Suppose that two agents on C_0. At first, agents first move to $v_{0,0}$ or $v_{2,0}$ by Arrangement'($row, 0$). Then, they move to $v_{1,0}$. The agent which reaches $v_{1,0}$ from $v_{0,0}$ (resp, $v_{2,0}$) stays at $v_{1,0}$ if $(v_{1,0}, v_{2,0})$ (resp, $(v_{1,0}, v_{0,0})$) is present and returns to $v_{0,0}$ (resp, $v_{2,0}$) otherwise. The agent which does not reach $v_{1,0}$ from $v_{0,0}$ (resp, $v_{2,0}$) moves to $v_{1,0}$ if $(v_{1,0}, v_{2,0})$ (resp, $(v_{1,0}, v_{0,0})$)

is present and moves to $v_{2,0}$ otherwise. After each execution, agents execute ArrangementLeft(j) so that one of them reaches C_j if they stay at the same node and execute Arrangement(row, j) so that agents which do not reach C_j move to the same column ring.

Lemma 5.5 *With the link presence detection, for $n = 3$, a group of three agents explore a dynamic torus by Algorithm 8.*

Proof 5.6 Correctness: *It suffices that two agents on C_j are truly on the same node at least once during above execution. If two agents on C_j are not on the same node by* Arrangement($row, 0$)*, they are on $v_{0,j}$ and $v_{2,j}$. Let a_0 be the agent on $v_{0,j}$ and a_1 be the agent on $v_{2,j}$. Then, they execute second action, that is, they move to $v_{1,j}$. If both $(v_{1,j}, v_{0,j})$ and $(v_{1,j}, v_{2,j})$ are present, both a_0 and a_1 reach $v_{1,j}$. Without loss of generality, suppose that $(v_{1,j}, v_{2,j})$ is not present. At this time, a_0 reaches $v_{1,j}$ and a_1 stays at $v_{2,0}$. Finally, they execute last action. At this time, a_0 and a_1 move to $v_{0,j}$ if $(v_{1,j}, v_{2,j})$ is not present and a_0 stays at $v_{1,j}$ and a_1 moves to $v_{1,j}$ otherwise. In both cases, a_0 and a_1 reach the same node. Thus, two agents on C_j are truly on the same node at least once during above execution.*

By Lemmas 5.1, 5.4 and 5.5, we get the following theorem.

Theorem 5.2 *With the link presence detection, for $n \geqq 5$ and $n = 3$, $\lceil n/2 \rceil + 1$ agents are necessary and sufficient for exploration of a dynamic torus.*

6. Conclusion

We proposed exploration algorithms of a dynamic torus for the case without and with the link presence detection. This dynamic model looks little far-fetched and unpractical because of rules of link change but as mentioned before, such a link change can happen in a network with mobile obstacles. In this paper, we consider a dynamic torus labeled by their column and row number. So, we will consider exploration a dynamic torus with labeled agent or without labeling as a future work.

Reference

[1] Chen Avin, Michal Kouckỳ, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). *International Colloquium on Automata, languages and programming*, pages 121–132, 2008.

[2] Balasingham Balamohan, Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Exploring an unknown dangerous graph with a constant number of tokens. *Theoretical Computer Science*, 610:169–181, 2016.

[3] Jiannong Cao and Sajal Kumar Das. *Mobile Agents in Networking and Distributed Computing.* John Wiley & Sons, 2012.

[4] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[5] Giuseppe Antonio Di Luna, Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Live exploration of dynamic rings. In *International Conference on Distributed Computing Systems*, pages 570–579. IEEE, 2016.

[6] Panagiota Fatourou. Mobile agents in distributed computing: Network exploration. *Bulletin of the EATCS no*, 109:54–69, 2013.

[7] Afonso Ferreira. Building a reference combinatorial model for manets. *IEEE Network*, 18(5):24–29, 2004.

[8] Paola Flocchini, Bernard Mans, and Nicola Santoro. On the exploration of time-varying networks. *Theoretical Computer Science*, 469:53–68, 2013.

[9] David Ilcinkas, Ralf Klasing, and Ahmed Mouhamadou Wade. Exploration of constantly connected dynamic graphs based on cactuses. In *International Colloquium on Structural Information and Communication Complexity*, pages 250–262. Springer, 2014.

[10] David Ilcinkas and Ahmed Mouhamadou Wade. Exploration of the t-interval-connected dynamic graphs: The case of the ring. In *International Colloquium on Structual Information and Communication Complexity*, pages 13–23. Springer, 2013.

[11] Fabian Kuhn and Rotem Oshman. Dynamic networks: models and algorithms. *ACM SIGACT News*, 42(1):82–96, 2011.

二次元グリッド平面における自律分散ロボットによる一点包囲アルゴリズムについて

A Study on an Algorithm for Surrounding the Point with Autonomous Mobile Robots in 2D Grid Plane

大藪匡記 [1] 金鎔煥 [1] 片山喜章 [1]
Masaki Oyabu Yonghwan Kim Yoshiaki Katayama

名古屋工業大学大学院 [1]
Graduate School of Engineering, Nagoya Institute of Technology

1 まえがき

近年，自律分散システムに関する研究が盛んにおこなわれており，自律分散ロボット群の研究もそのひとつである．自律分散ロボットとは，自律的に動作するロボットであり，これらのロボットの集合を自律分散ロボット群と言う．これらのロボットが協調的に動作することで全体で一つの目標を達成する．本稿では，自律分散ロボット群を計算論的な観点からとらえ，その協調問題を扱う．ロボットは他のロボットの位置を観測 (Look) し，観測結果を入力にアルゴリズムにしたがって行き先を計算 (Compute) し，その行き先に移動 (Move) するというシンプルなサイクルを繰り返し問題を解く．たとえば一点集合問題とは，任意の位置に配置されたロボットがある一点に集合する問題である．ロボットの観測や移動の能力や知り得る情報と問題の可解性について研究されている [1–5]．自律分散ロボット群の制御に関する理論的研究においては，ロボットモデルと問題の可解性 (本稿では一点包囲問題) の関係を明らかにすることが研究者の興味の一つである．

ロボットを表現する方法として，体積・面積を持たない点ロボットと，体積・面積を有するファットロボット (fat robots) の 2 種類が扱われている．文献 [6] は，ファットロボットの集合問題を理論的に扱った最初の研究である．ロボットが他のロボットの観測の障害物となるような不透明なロボットの視界とロボット同士の衝突を次のように定義し，ロボットが重なっていない任意の初期状況から非同期に動作する 3 台または 4 台のロボットの集合問題を解くアルゴリズムを提案している．

- **不透明なロボットの視界**:r_i, r_j を円盤で表現されるロボットとする．次の二つの条件を満たす点 p, q が存在するとき，ロボット r_i は r_j を見ることができるという．
  1. 点 p, q はそれぞれロボット r_i および r_j の円盤の内部
  2. 線分 pq が他のロボットを含まない
- **ロボット同士の衝突**:あるロボットが他のロボットに接触したときロボットは双方とも移動を終了する．

ファットロボットの集合問題を解くアルゴリズムが文献 [8–15] で提案されている．文献 [6–9, 12] では，連続平面におけるファットロボットの集合問題について研究されている．文献 [7] では，連続平面と離散平面におけるファットロボットの集合について研究されている．文献 [10, 11] も離散平面におけるファットロボットの集合問題を扱っている．文献 [10] では格子平面の格子サイズを 1 としたときの，ロボットの半径によって次の 3 つのモデルを定義した．

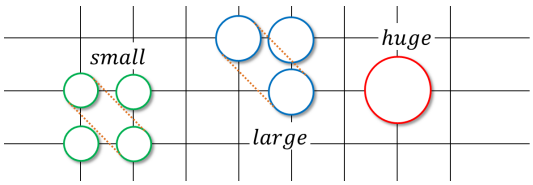


図 1: ロボットの半径 *radius* と格子サイズの関係を示す例

- $small : radius < \frac{1}{4}\sqrt{2}$
- $large : \frac{1}{4}\sqrt{2} \leq radius < \frac{1}{2}$
- $huge : radius \geq \frac{1}{2}$

モデル *small* では図 1 に例を示すように，ロボットの大きさが十分に小さく斜めに移動したとき他のロボットに接触することはない．一方でモデル *large* では，2 台のロボットが隣り合う格子点に存在することはできるが，斜め移動する際に接触の可能性があるモデルであり，このモデルでは，ロボットの接触を避けるようにアルゴリズムを設計する必要がある．さらに，モデル *huge* では，隣り合う格子点にロボットが存在することも不可能である．文献 [7, 11] はともにロボットのサイズが *small* であったのに対し，文献 [10] では *small*,*large* の両方のモデルに対してアルゴリズムが提案されている．文献 [13] では，ロボットの大きさが *large*，ロボットの台数を知っていて，半同期で動作し，共通座標系を持たないロボットモデルに対してアルゴリズムが提案されている．文献 [14] では離散空間について平面格子，六角格子，立方格子を考え，ロボットのサイズ，ロボットの総数に関する知識，スケジューラの非同期または半同期の組み合わせに関して集合問題を解くアルゴリズムが提案されている．文献 [15] では，三次元グリッド空間における集合問題を定義し，ロボットの台数を知らず，非同期に動作し，共通座標系を持ち，距離$\sqrt{2}$ 以内にある点に移動可能なロボットモデルに対して，集合すべき点を中心とした正八面体と立方体のそれぞれを構成するアルゴリズムが提案されている．

集合問題の他に，自律分散ロボット群を用いる問題として形状形成問題がある．形状形成問題とは任意の位置に配置されたロボットを決められた形に配置する問

題である [16–20]．文献 [16] では，あらかじめ決められた点の集合 F と，$|R| = |F|$ であるような点ロボットの集合 R が与えられ，|F|に属する全ての点にロボットが 1 台存在するようにロボットを移動させる問題である $EmbeddedPatternFormationProblem(EPF$ 問題) について，キラリティに合意を持たない場合の非可解な初期状態と EPF 問題を解くアルゴリズムが提案されている．文献 [17] では，非同期に動作する 4 台のロボットがロボットの初期位置から形成する正方形を求め，正方形を形成する．形状形成問題の一つにロボットを円状に並べる円形成問題が存在する [18–20]．円形成問題とはロボットを同一円周上に並べる問題で，特にロボット同士の距離を均等にする問題が注目されている．文献 [18] では全てのロボットを円周上に移動させるパートと円周上のロボット同士の間隔を均等にするパートの二つのパートに分かれたアルゴリズムによって円形成問題を解いている．文献 [19] ではキラリティに合意を持つロボットについて，アルゴリズムの実行中にロボットの追加，削除，再配置が可能な円形成アルゴリズムが提案されている．文献 [20] ではファットロボットによる円形成アルゴリズムが提案されている．ロボットは座標系についての合意を持たず，非同期に動作する．また，ロボットは透明で，全視界である．離散平面において円はひし形となる．ひし形上に全てのロボットを隙間なく配置する場合，ロボットの台数は離散的となり，離散平面における円形成問題は扱われていなかった．本稿で扱う一点包囲問題は，離散平面において円形成問題を再定義した問題であり，半同期に動作する透明で半径が $small$ のファットロボットによる一点包囲問題を解くアルゴリズムを提案する．

2　モデルと問題定義

2.1　ロボットモデル

二次元直行座標系 $\mathcal{F}$ は，単位長さ，原点 O，x と y で識別され互いに直行する 2 つの座標軸とその方向 ($direction$)，正と負で識別される座標軸の向き (orientations) で定義される．格子 $\mathcal{G}$ を $\mathcal{F}$ の全ての整数座標とする．$u = (u_x, u_y)$ および $v = (v_x, v_y)$ を $\mathcal{G}$ 上の点としたとき，u と v の距離 $dist(u, v)$ を $dist(u, v) = |u_x - v_x| + |u_y - v_y|$ とする ($L_1 - norm$, マンハッタン距離)．

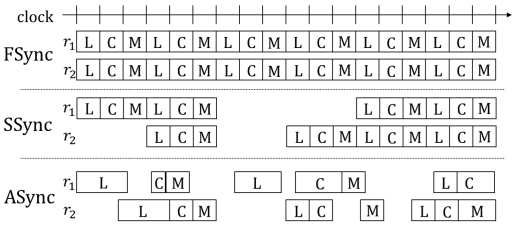


図 2: ロボットの実行モデル FSYNC,SSYNC,ASYNC の実行例．図中の L,C,M はそれぞれ L:Look,C:Compute,M:Move を表している．モデルによりサイクルを開始するタイミングと開始するロボット，および各状態にかかる時間が異なる．

n 台のロボットの集合 $\mathcal{R}$ を $\mathcal{R} = \{r_0, r_1, r_2, \ldots, r_{n-1}\}$ とする．ロボットは $\mathcal{G}$ 上の点に存在し，$\mathcal{G}$ 上の点間を離散的に移動する計算能力を持ったデバイスである．ロボットは区別不可能で同じアルゴリズムを実行し，記憶領域を持たない．また，ロボットは共通の座標系を持たないが，共通の原点，右回り方向を知っており，自身の持つ座標系において自分の座標を認識することができる．$u = (u_x, u_y), v = (v_x, v_y)$ を $\mathcal{G}$ 上の 2 点としたとき，$(u_x v_y - u_y v_x) < 0$ を満たすとき原点を中心に v は u の右回りに位置するといい，u は v の左回りに位置するという．また，$(u_x v_y - u_y v_x) = 0$ を満たすとき，u, v および原点 O は同一直線上に位置するという．ロボットが一回で移動できる点は，自身の存在する点とその周囲 8 点である．ただし，ロボット同士の衝突は許されない，つまり 2 台のロボットが同一直線上をすれ違う移動と複数のロボットが同時に同一の点へ移動してしまうことは許されない．

各ロボットの動作は次に説明する 4 つの状態，Wait,Look,Compute,Move のうち後者の 3 つの状態を 1 サイクルで実行する．Wait は任意の状態間に挿入可能な状態である．

1. **Wait:**ロボットは待機状態．ロボットは無制限に待機することはできない．開始時点ではすべてのロボットが $Wait$ 状態である．
2. **Look:**ロボットは他のロボットの位置座標を得る．$Look$ の結果，ロボットは他のロボットの位置の集合 $\mathcal{C}$ を得る．
3. **Compute:**ロボットはアルゴリズムに従って行き先を計算する．ロボットは無記憶であるため，アルゴリズムの入力は直前の $Look$ で得た $\mathcal{C}$ のみである．
4. **Move:**$Compute$ 状態において計算した行き先に向かって移動する．

各ロボットがサイクルを実行する際の同期の程度によって次の 3 つのモデルが定義される (図 2)．

- **完全同期 (fully-synchronous, FSYNC):**すべてのロボットが完全に同期した時計を用いてアルゴリズムを実行できると仮定する．すなわち，すべてのロボットは同じ時刻にアルゴリズムを開始し，その後，同じ時刻に，$Look, Compute, Move$ をそれぞれ実行する．
- **半同期 (semi-synchronous, SSYNC):**FSYNC において，サイクルを実行しないロボットの存在を許すモデルである．ただし，すべてのロボットは無限回サイクルを実行すると仮定する．
- **非同期 (asynchronous, ASYNC):**ロボットの実行に関する仮定を一切置かないモデルである．各ロボットのサイクルの開始時刻や $Look, Compute, Move$ にかかる時間の上限も与えられない．そのため，ロボットが移動中に他のロボットに観測されることもある．

これらの 3 つの同期モデルに関して，SSYNC は FSYNC の実行を含み，ASYNC は SSYNC，FSYNC の実行を含む．つまり，ASYNC で，ある問題を解くアルゴリズム

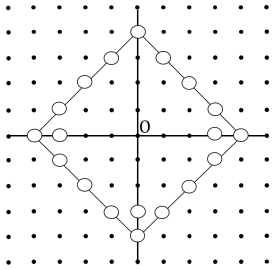


図 3: 最大距離 4 で原点を包囲する問題で包囲達成時の例

が存在する場合，そのアルゴリズムは SSYNC，FSYNC のロボットに対しても，その問題を正しく解くアルゴリズムである．SSYNC と FSYNC のモデル関係も同様である．本稿ではロボットは SSYNC で動作するものとする．ロボットのサイズについて，本稿ではロボットのサイズは *small* とする．つまりロボットは斜めに移動可能である．ロボットの視野範囲について，ロボット観測範囲に制限を設けない全視界と，ロボットの視野に制限を持たせる制限視野が考えられる．本稿ではロボットはマンハッタン距離で 2 以内にある点を観測できる制限視野を持ったロボットである．本稿で扱うロボットは透明である．つまりロボットの観測が他のロボットによって妨げられることはない．

2.2 一点包囲問題の定義

本稿では，あらかじめ決められた 1 点 (原点 O) を隙間なくロボットを配置することで包囲する一点包囲問題を扱う．ロボットが隙間なく原点を囲える距離を L_{max} とし，原点からの距離 L_{max} 以内で，原点から各ロボットまでのマンハッタン距離 (以下，単に距離という場合はマンハッタン距離を表す) の総和が最大になるようにする．このとき，図 3 の例に示すように，ロボットが包囲を達成したときの形はひし形と，その内側にロボットが数台貼りついているものとなる．a_i を原点からの距離が i である点の数とすると，a_i は以下のようになる．

$$\begin{cases} a_0 = 1 \\ a_i = 4i (i > 0) \end{cases}$$

また，n 台のロボットが包囲を達成したとき，$a_i + 4 = a_{i+1}$ より L_{max} は以下の式で求まる．

$$L_{max} = \lfloor \tfrac{n}{4} \rfloor (n \geq 4)$$

本稿では $n \geq 12$ と仮定する．従って $L_{max} \geq 3$ となる．本稿で扱う一点包囲問題を L_{max} を用いて以下のように定義する．

定義 2.1. (一点包囲問題).n 台のロボットが次の条件を満たしているとき，原点を中心とする包囲問題を解いたという．

- $\forall r_i \in \mathcal{R}, L_{max} - 1 \leq dist(O, r_i) \leq L_{max}$
- $\{\forall p \in \mathcal{G} | dist(O, p) = L_{max}\} \subseteq \mathcal{R}$
- 全てのロボットは停止している

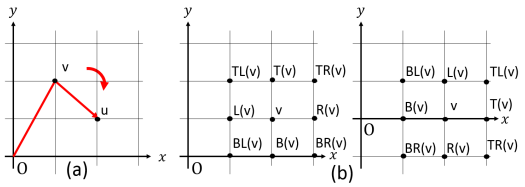


図 4: (a):点 v, u の位置関係 (b):点 v とその周囲の 8 点の例.

2.3 諸定義

$u = (u_x, u_y), v = (v_x, v_y)$ を $\mathcal{G}$ 上の 2 点としたとき，$(u_x v_y - u_y v_x) < 0$ を満たすとき原点を中心に v は u の右回りに位置するといい，u は v の左回りに位置するという．また，$(u_x v_y - u_y v_x) = 0$ を満たすとき，u, v および原点 O は同一直線上に位置するという．(図 4(a)) 本稿で扱うロボットの視野 $M(v)$ を以下のように定義する．

定義 2.2. (点集合$M(v)$).$\mathcal{G}$ 上の任意の点 v について，次の条件を満たす点 u の集合を $M(v)$ とする．

- $M(v) = \{u | dist(v, u) \leq 2\}$

本稿で扱うロボットの移動可能な点の集合 $N(v)$ を以下のように定義する．

定義 2.3. (点集合$N(v)$).$\mathcal{G}$ 上の任意の点 $v = (v_x, v_y)$ について，以下の点を満たす点 $u = (u_x, u_y)$ の集合を $N(v)$ とする．

- $N(v) = \{(u_x, u_y) | (|u_x - v_x| \leq 1) \wedge (|u_y - v_y| \leq 1)\}$

ある $v \in \mathcal{G}$ に対して，$N(v)$ に含まれる v を除く各点 u について以下の通り名前をつける (図 4(b) 参照)．ただし，点 v が軸上の点かそうでないかで各点の定義が異なる．

v が軸上の点の場合.

- $T(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) + 1) \wedge (v_x u_y - v_y u_x = 0)$
- $B(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) - 1) \wedge (v_x u_y - v_y u_x = 0)$
- $L(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) + 1) \wedge (v_x u_y - v_y u_x > 0)$
- $R(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) + 1) \wedge (v_x u_y - v_y u_x < 0)$
- $TL(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) + 2) \wedge (v_x u_y - v_y u_x > 0)$
- $TR(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) + 2) \wedge (v_x u_y - v_y u_x < 0)$
- $BL(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v)) \wedge (v_x u_y - v_y u_x > 0)$
- $BR(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v)) \wedge (v_x u_y - v_y u_x < 0)$

v が軸上の点でない場合.

- $T(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) + 1) \wedge (v_x u_y - v_y u_x > 0)$
- $B(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) - 1) \wedge (v_x u_y - v_y u_x < 0)$
- $L(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) - 1) \wedge (v_x u_y - v_y u_x > 0)$
- $R(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) + 1) \wedge (v_x u_y - v_y u_x < 0)$
- $TL(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v)) \wedge (v_x u_y - v_y u_x > 0)$
- $TR(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) + 2)$
- $BL(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v) - 2)$
- $BR(v) : (dist(\mathrm{O}, u) = dist(\mathrm{O}, v)) \wedge (v_x u_y - v_y u_x < 0)$

2.4 アルゴリズムの記述

本稿では，アルゴリズムの実行を以下の形式（ガード付きアクション）で表す．

$$< label >::< guard >\rightarrow< action >$$

各アクションはラベル付けされており，各アクションのガード $< guard >$ は Look の結果からなる論理式で表される．ロボットはガードが真の場合のみ命令文 $< action >$ を実行する．

3 提案アルゴリズム

提案アルゴリズムについての説明と証明を行う．

3.1 提案アルゴリズムの説明

本稿では共通の原点と共通の右回りを各ロボットが持っているというロボットモデルにおいて，一点包囲問題を解くアルゴリズムを提案する．初期条件として，すべてのロボットが台数 n を知っていると仮定する．提案アルゴリズムでは，ロボットは原点からの距離により異なるアルゴリズムに従う．原点からの距離が L_{max} の点と比べて原点に遠い点に存在するロボットが原点からの距離が L_{max} の点に移動可能である点を，原点からの距離が $L_{max}+1$ であり，またその点から原点からの距離を変えずに右回りに一回移動した場合，その移動した先の点が軸上の点であるという条件を満たす 4 点のみとし，またその 4 点から移動できる原点からの距離が L_{max} の点も，直近の軸上の点に限定する．原点からの距離が L_{max} の点と比べて原点に遠い点に存在するロボットは，右回りに移動することで軸上に乗り，軸沿いに原点からの距離が $L_{max}+1$ の点に移動し，原点からの距離が $L_{max}+1$ の軸上の点から右回りに移動することで原点からの距離が L_{max} の軸上の点に移動可能な点へ移動する．また軸上に向かって右回りに移動する途中で原点からの距離が $L_{max}+1$ の点に移動した場合は原点からの距離を変えずに右回りに移動することで原点からの距離が L_{max} の軸上の点に移動可能な点への移動する．一方，原点からの距離が $L_{max}-1$ の点と比べて原点に近い点に存在するロボットは直近の原点からの距離が $L_{max}-1$ の点に移動しようとする．原点からの距離が L_{max} の点に存在するロボットは原点からの距離を変えずに右回りに移動し，原点からの距離が $L_{max}-1$ の軸上以外の点に存在するロボットは原点からの距離を変えずに右回りに移動する．原点に存在するロボットは，自身の持つ座標系で x 軸正の方向に存在する点へ移動する．しかし，この動作のみの場合以下の問題点がある．

1. 初期配置で原点からの距離が L_{max} の点と比べて原点に遠い点に多くのロボットが存在する場合原点からの距離が L_{max} の点すべてがロボットで埋まり，原点からの距離が L_{max} の点と比べて原点に遠い点に存在するロボットが原点からの距離が L_{max} の点, 原点からの距離が $L_{max}-1$ の点に移動できずデッドロックが発生する．
2. 初期配置で原点からの距離が $L_{max}-1$ の点と比べて原点に近い点に多くのロボットが存在する場合原点からの距離が $L_{max}-1$ の点すべてがロボットで埋まり，原点からの距離が $L_{max}-1$ の点と比べて原点に近い点に存在するロボットが原点からの距離が L_{max} の点, 原点からの距離が $L_{max}-1$ の点に移動できずデッドロックが発生する．

これらの問題点を解決するために原点からの距離が L_{max} の点から原点からの距離が $L_{max}-1$ の点への移動を可能にし，また原点からの距離が $L_{max}-1$ の点から原点からの距離が L_{max} の点への移動を可能にする．定義 2.1 の通り，原点からの距離が L_{max} の点すべてにロボットが存在する必要がある．そのため，原点からの距離が L_{max} の点上のロボットが原点からの距離が $L_{max}-1$ の点へ移動する必要がある場合は，1 番の問題点である原点からの距離が L_{max} の点と比べて原点に遠い点に存在するロボットが原点からの距離が L_{max} の点に移動できない場合である．そこで原点からの距離が L_{max} の点上のロボットが原点からの距離が $L_{max}-1$ の点へ移動する条件は，原点からの距離が L_{max} の点に移動可能な原点からの距離が L_{max+1} の点にロボットが存在し，またその点の直近の原点からの距離が L_{max} の軸上の点にロボットが存在する場合とする．このとき原点からの距離が L_{max} の軸上の点に存在するロボットが直近の原点からの距離が $L_{max}-1$ の軸上の点へ移動を行う．2 番の問題点を解決するために，原点からの距離が $L_{max}-1$ の点に存在するロボットの原点からの距離が L_{max} の点への移動を可能にする．ただし，そのような移動が可能な点を原点からの距離が $L_{max}-1$ の軸上の 4 点に限定し，移動先の原点からの距離が L_{max} の点も直近の軸上の点のみとする．また，原点からの距離が $L_{max}-1$ の点と比べて原点に近い点に存在するロボットの動作が直近の原点からの距離が $L_{max}-1$ の点に向かって移動し原点からの距離が $L_{max}-1$ の点に移動するという動作のみである場合，直近の原点からの距離が $L_{max}-1$ の点にロボットが存在するとき，原点からの距離が $L_{max}-1$ の点に移動することができない．そのため，原点からの距離が $L_{max}-3$ 以下のロボットは右回りに直近の原点からの距離が $L_{max}-1$ の点に向かって直進するが，原点からの距離が $L_{max}-2$ の点に到達した場合は，直近の原点からの距離が $L_{max}-1$ の点にロボットが存在していて移動できなかった場合は原点からの距離を変えずに右回りに移動することで，空いている原点からの距離が $L_{max}-1$ の点を探し，空いている原点からの距離が $L_{max}-1$ の点に移動するようにする．以上の戦略を実現したアルゴリズムが Algorithm1,2,3,4,5,6,7,8 であり，各点の動作は図 11 のようになる．Algorithm1,2,3,4,5,6,7,8 中の述語について以下のように定義する．

---

a を $\mathcal{G}$ 上の点 (a_x, a_y) とする．
Predicates:
$a \in \mathcal{C}, occupy(a) \equiv true$
$(a_x = 0 \vee a_y = 0), axis(a) \equiv true$

$< guard >$ は順に評価され，真になった $< guard >$ の $< action >$ が実行される．どの $< guard >$ も真にならないとき，ロボットはその場で静止する．

Algorithm 1 点 $r_i(dist(\mathrm{O}, r_i) \geq L_{max} + 2)$ 上に存在するロボットのアルゴリズム

actions:
1:A1-1 ::$\neg occupy(B(r_i)) \wedge axis(r_i) \rightarrow$ 行き先を $B(r_i)$
2:A1-2 ::$\neg occupy(B(r_i)) \wedge axis(B(r_i)) \wedge \neg occupy(BR(r_i)) \rightarrow$ 行き先を $B(r_i)$
3:A1-3 ::$\neg occupy(B(r_i)) \wedge \neg axis(B(r_i)) \wedge \neg occupy(L(r_i)) \wedge dist(O, B(r_i)) = L_{max} + 1 \rightarrow$ 行き先を $B(r_i)$
4:A1-4 ::$\neg occupy(B(r_i)) \wedge \neg axis(B(r_i)) \wedge dist(O, B(r_i)) \neq L_{max} + 1 \rightarrow$ 行き先を $B(r_i)$

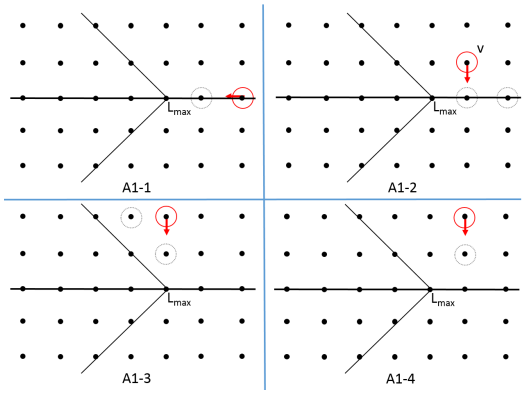


図 5: Algorithm1

Algorithm 2 点 $r_i(dist(\mathrm{O}, r_i) = L_{max} + 1)$ 上に存在するロボットのアルゴリズム

actions:
1:A2-1 ::$\neg occupy(BR(r_i)) \wedge \neg axis(BR(r_i)) \rightarrow$ 行き先を $BR(r_i)$
2:A2-2 ::$\neg occupy(B(r_i)) \wedge axis(BR(r_i)) \rightarrow$ 行き先を $B(r_i)$

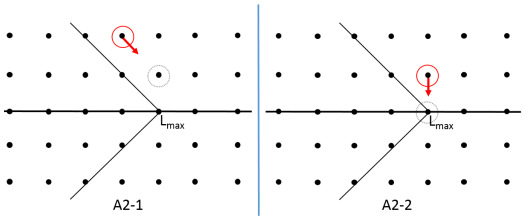


図 6: Algorithm2

Algorithm 3 点 $r_i(dist(\mathrm{O}, r_i) = L_{max})$ 上に存在するロボットのアルゴリズム

actions:
1:A3-1 ::$\neg occupy(BR(r_i)) \wedge \neg axis(BR(r_i)) \rightarrow$ 行き先を $BR(r_i)$
2:A3-2 ::$\neg occupy(BR(r_i)) \wedge axis(BR(r_i)) \wedge \neg occupy(B(r_i)) \wedge \neg occupy(R(r_i)) \rightarrow$ 行き先を $BR(r_i)$
3:A3-3 ::$\neg occupy(B(r_i)) \wedge axis(r_i) \wedge ocuupy(L(r_i)) \rightarrow$ 行き先を $B(r_i)$

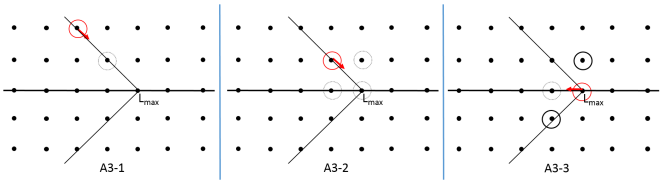


図 7: Algorithm3

Algorithm 4 点 $r_i(dist(\mathrm{O}, r_i) = L_{max} - 1)$ 上に存在するロボットのアルゴリズム

actions:
1:A4-1 ::$\neg occupy(BR(r_i)) \wedge \neg occupy(B(r_i)) \wedge \neg axis(r_i) \wedge \neg axis(BR(r_i)) \rightarrow$ 行き先を $BR(r_i)$
2:A4-2 ::$\neg occupy(BR(r_i)) \wedge \neg occupy(R(R(r_i))) \wedge axis(BR(r_i)) \rightarrow$ 行き先を $BR(r_i)$
3:A4-3 ::$\neg occupy(BR(r_i)) \wedge \neg occupy(B(r_i)) \wedge axis(r_i) \wedge occupy(T(r_i)) \wedge occupy(TL(r_i)) \rightarrow$ 行き先を $BR(r_i)$
4:A4-4 ::$\neg occupy(T(r_i)) \wedge axis(r_i) \wedge \neg occupy(TL(r_i)) \rightarrow$ 行き先を $T(r_i)$

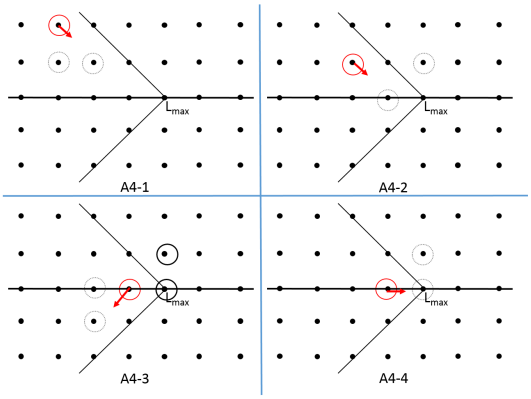


図 8: Algorithm4

Algorithm 5 点 $r_i(dist(\mathrm{O}, r_i) = L_{max} - 2)$ 上に存在するロボットのアルゴリズム

actions:
1:A5-1 ::$\neg occupy(R(r_i)) \rightarrow$ 行き先を $R(r_i)$
2:A5-2 ::$\neg occupy(BR(r_i)) \rightarrow$ 行き先を $BR(r_i)$

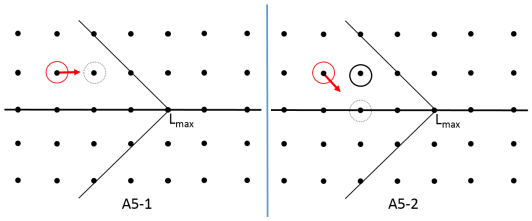


図 9: Algorithm5

Algorithm 6 点 $r_i(dist(\mathrm{O}, r_i) = L_{max} - 3 \wedge dist(\mathrm{O}, r_i) \neq 0)$ 上に存在するロボットのアルゴリズム

actions:
1:A6-1 ::$\neg occupy(R(r_i)) \wedge \neg occupy(T(r_i)) \rightarrow$ 行き先を $R(r_i)$

Algorithm 7 点 $r_i(dist(\mathrm{O}, r_i) \leq L_{max} - 4 \wedge dist(\mathrm{O}, r_i) \neq 0)$ 上に存在するロボットのアルゴリズム

actions:
1:A7-1 ::$\neg occupy(R(r_i)) \rightarrow$ 行き先を $R(r_i)$

Algorithm 8 点 $r_i(dist(\mathrm{O}, r_i) = 0)$ 上に存在するロボットのアルゴリズム

actions:
1:A8-1 ::$\neg occupy((1,0)) \wedge \neg occupy((0,1)) \wedge \neg occupy((-1,0)) \wedge \neg occupy((0,-1)) \rightarrow$ 行き先を $(1,0)$

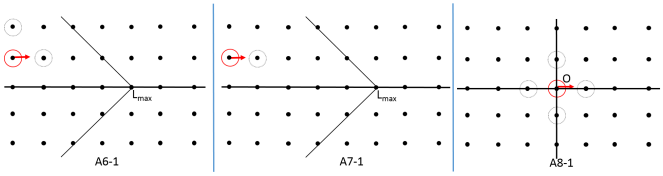


図 10: Algorithm6,7,8

3.2 正当性の証明

アルゴリズムの正当性の証明を行う．正当性の証明をするためにいくつかの定義をする．

定義 3.1. (点集合 $\mathcal{L}, \mathcal{S}$). 原点からの距離が L_{max} の点の集合を $\mathcal{L}$ とし，原点からの距離が $L_{max} - 1$ の点の集合を $\mathcal{S}$ とする (図 12).

定義 3.2. (*Entry Points Set*). 以下を満たす $\mathcal{G}$ 上の点 p を *Entry Point* と呼び，その集合を $\mathcal{EP}$ と表す (図 13).

$$\mathcal{EP} = \{\forall p | (dist(\mathrm{O}, p) = L_{max} + 1) \wedge axis(BR(p))\}$$

補題 3.1. 複数ロボットの移動による同一点での衝突は発生しない．

証明 . 本稿で提案するアルゴリズムの戦略によって発生する可能性のある，複数台のロボットが同時に同一の点

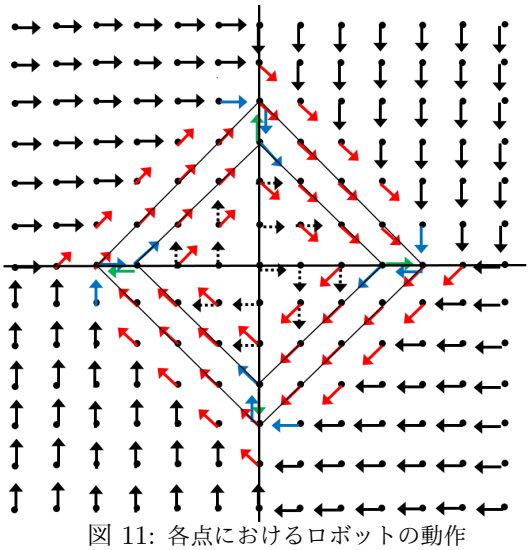


図 11: 各点におけるロボットの動作

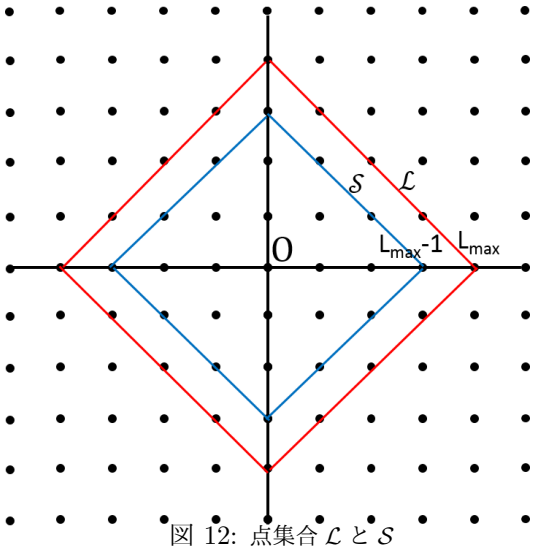


図 12: 点集合 $\mathcal{L}$ と $\mathcal{S}$

に移動することで起きる衝突は以下の 7通りである (図 14,15,16,17).

(A) 原点からの距離が $L_{max} + 2$ 以上である軸上の点に存在し，軸上に沿って原点方向へ移動するロボットと原点からの距離が同じである軸上以外の点に存在し，軸上の点に移動するロボットの 2台のロボットの衝突

(B) 原点からの距離が $L_{max} + 1$ の点に存在し，原点からの距離を $L_{max} + 1$ から変えずに右回りに移動するロボットと原点からの距離が $L_{max} + 2$ の軸上以外の点に存在し，原点からの距離が $L_{max} + 1$ の点に移動するロボットの 2台のロボットの衝突

(C) 原点からの距離が $L_{max} + 1$ の点に存在し，原点からの距離が L_{max} の軸上の点に移動するロボットと原点からの距離が L_{max} の軸上以外の点に存在し，原点からの距離が L_{max} の軸上の点に右回りに移動するロボット，原点からの距離が $L_{max} - 1$ の軸上の点に存在し，原点からの距離が L_{max} の軸上の点に移動するロボットの 3台，または 3台のうち 2台のロ

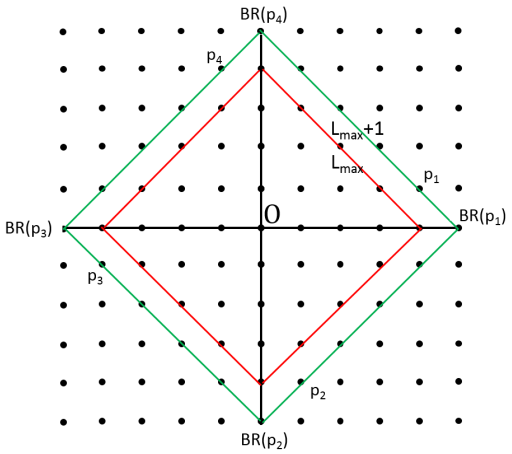


図 13: *Entry Points Set* $\mathcal{EP} = \{p_1, p_2, p_3, p_4\}$

ボットの衝突

(D) 原点からの距離が L_{max} の軸上の点に存在し，原点からの距離が $L_{max}-1$ の軸上の点に移動するロボットと原点からの距離が $L_{max}-1$ の軸上以外の点に存在し，原点からの距離が $L_{max}-1$ の軸上の点に右回りに移動するロボットの *2* 台のロボットの衝突

(E) 原点からの距離が $L_{max}-1$ の点に存在し，原点からの距離を $L_{max}-1$ から変えずに右回りに移動するロボットと原点からの距離が $L_{max}-2$ の点に存在し，原点からの距離が $L_{max}-1$ の軸上以外の点に移動するロボットの *2* 台のロボットの衝突

(F) 原点からの距離が $L_{max}-2$ の点に存在し，原点からの距離を $L_{max}-2$ から変えずに右回りに移動するロボットと原点からの距離が $L_{max}-3$ の点に存在し，原点からの距離が $L_{max}-2$ の軸上以外の点に移動するロボットの *2* 台のロボットの衝突

(G) 原点からの距離が $L_{max}-2$ の点に存在し，原点からの距離を $L_{max}-2$ から変えずに右回りに移動するロボットと原点に存在し，原点からの距離が 1 の点に移動するロボットの *2* 台のロボットの衝突

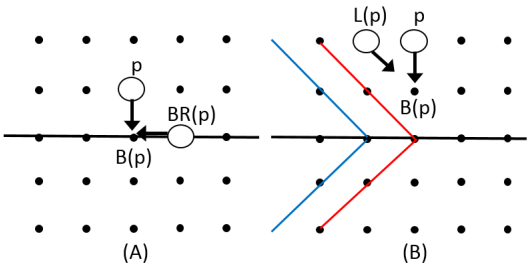


図 14: パターン (A),(B)

ただし，*(A)*〜*(G)* 全ての場合において，各ロボットが移動しようとしている点にはロボットが存在しないものとする．*(A)* の場合，軸上以外の点 p に存在するロボットは点 $BR(p)$ が軸上の点であり点 $BR(p)$ にロボットが存在するとき点 $B(p)$ への移動をしないで停止することで軸上の点 $BR(p)$ に存在するロボットを優先させ，衝突を回避する．*(B)* の場合，原点からの距離が $L_{max}+2$ の軸上以外の点 p に存在するロボットは点 $L(p)$ にロボットが存在するとき点 $B(p)$ への移動をしないで停止することで原点からの距離が $L_{max}+1$ の点 $L(p)$ に存在するロボットを優先させ，衝突を回避する．ただし，点 $B(p)$ が軸上の点である場合は *(B)* の衝突は発生せず，*(A)* のパターンの衝突が発生する可能性があるため，*(A)* のパターンの衝突回避を行う．*(C)* の場合，原点からの距離が L_{max} の軸上以外の点 p に存在するロボットは点 $BR(p)$ が軸上の点であり，また点 $B(p)$，点 $R(p)$ の二点のどちらか一点でもロボットが存在するとき点 $BR(p)$ への移動をしないで停止することで，原点からの距離が $L_{max}-1$ の軸上の点 $B(p)$, 原点からの距離が $L_{max}+1$ の点 $R(p)$ に存在するロボットを優先させる．原点からの距離が $L_{max}-1$ の軸上の点 p' に存在するロボットは点 $TL(p')$ にロボットが存在するとき点 $T(p')$ への移動をしないで停止することで，原点からの距離が $L_{max}+1$ の点 $TL(p')$，つまり点 $R(p)$ に存在するロボットを優先させ，衝突を回避する．*(D)* の場合，互いの存在を認識することができないが，原点からの距離が $L_{max}-1$ の点 p に存在するロボットは点 $R(R(p))$ に存在するロボットを認識することができる．点 $R(R(p))$ は，原点からの距離が L_{max} の軸上の点を点 p' としたとき点 $L(p')$ にあたり，点 $L(p')$ にロボットが存在する場合のみ点 p' に存在するロボットは点 $B(p')$ へと移動する．そのため，点 p に存在するロボットは点 $R(R(p))$ にロボットが存在するとき点 $BR(p)$ への移動をしないで停止することで点 p' に存在するロボットを優先させることで衝突を回避する．*(E)* の場合，原点からの距離が $L_{max}-1$ の点 p に存在するロボットは点 $B(p)$ にロボットが存在するとき点 $BR(p)$ への移動をしないで停止することで原点からの距離が $L_{max}-2$ の点 $B(p)$ に存在するロボットを優先させ，衝突を回避する．*(F)* の場合，原点からの距離が $L_{max}-3$ の点 p に存在するロボットは点 $T(p)$ にロボットが存在するとき点 $R(p)$ への移動をしないで停止することで原点からの距離が $L_{max}-2$ の点 $T(p)$ に存在するロボットを優先させ，衝突を回避する．*(G)* のパターンの衝突は，$L_{max}=3$ のときのみ発生する可能性がある．$L_{max} \geq 4$ の場合，原点からの距離が 1 である

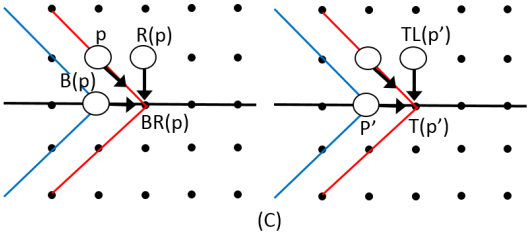


図 15: パターン (C)

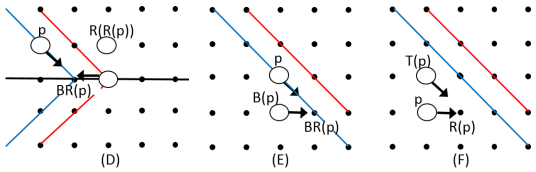


図 16: パターン (D),(E),(F)

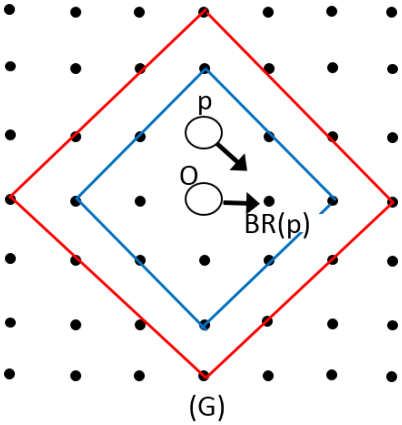


図 17: パターン (G)

点は原点からの距離が $L_{max}-3$ 以下の点となり直近の原点からの距離が $L_{max}-1$ の点に向かって直進するが，$L_{max}=3$ のときは原点からの距離が 1 である点は原点からの距離が $L_{max}-2$ の点となるため，原点からの距離が 1 である点 p に存在するロボットは原点からの距離を変えずに右回り，つまり $BR(p)$ に移動する可能性があり，原点に存在するロボットと衝突する可能性がある．そこで原点に存在するロボットは，原点からの距離が 1 である点に *1* 台もロボットが存在しない場合，つまり隣り合う *4* 点すべてにロボットが存在しない場合に移動を行うようにすることで衝突を回避する．よって *(A)* ～*(G)* 全ての場合において衝突は発生しないつまり，複数ロボットの移動による同一点での衝突は発生しない．

□

補題 3.2. 点 $ep\in\mathcal{EP}$ に存在するロボットはいつかは $\mathcal{L}$ 上の点に移動することができる．

証明 ．点 *ep* に存在するロボットは原点からの距離が $L_{max}+1$ であり，かつ点 *BR(ep)* が軸上の点であるため，$A2-2$ の動作以外は行うことができない．また，点 *ep* の条件を満たす点は $\mathcal{G}$ 上に 4 点存在する (図 13)．点 *ep* 上に存在するロボットの動作は点 *B(ep)* に存在するロボットにだけ依存するので，本証明では，点 *B(ep)* 上のロボットの移動のみに着目して証明を行う．点 *B(ep)* に存在するロボットは，点 *ep* にロボットが存在するとき以下の条件の両方を満たしている間，移動することはできない．

- 条件 1 : 点 *BR(B(ep))* にロボットが存在する ($A3-1$ より)
- 条件 2 : 点 *BL(ep)* にロボットが存在する ($A3-3$ より)

本証明では，上記のどちらかの条件が満たされ変更されない場合，必ず別の条件が解除されることを示すことで，点 *B(ep)* の存在するロボットがいずれ移動し，その結果，点 *ep* のロボットが $\mathcal{L}$ 上に移動することを示す．点 *BR(B(ep))* は $\mathcal{L}$ 上の点であり，$A3-1$ から $\mathcal{L}$ 上の点が全てロボットで埋まっていてどの $\mathcal{L}$ 上のロボットも移動できない場合条件 1 は自明に満たされ続ける．なので条件 1 を永遠に満たし続けると仮定し，$\mathcal{L}$ 上の点が全てロボットで埋まっている場合と $\mathcal{L}$ 上にロボットの存在しない点が存在する場合の二通りを考える．初めに $\mathcal{L}$ 上の点が全てロボットで埋まっている場合を仮定する．このとき，$\mathcal{L}$ 上には $4L_{max}$ 台のロボットが存在し，L_{max} の仮定から $\mathcal{L}$ 上以外には高々三台のみロボットが存在する．点 *B(ep)* にロボットが存在するため，点 *BL(ep)* に存在するロボットは $A4-4$ の動作を行うことはできない．よって $\mathcal{L}$ 上の点が全てロボットで埋まっているとき，必ず $A4-3$ の動作を行うことができることを示す．$A4-3$ の動作を行うためには点 *BR(BL(ep))* と点 *B(BL(ep))* にロボットが存在せず，かつ点 *T(BL(ep))* と点 *TL(BL(ep))* にロボットが存在する必要がある．*T(BL(ep))* は点 *B(ep)* であり，点 *TL(BL(ep))* は点 *ep* であるため，どちらの点にもロボットが存在する．なので点 *BL(ep)* に存在するロボットが $A4-3$ の動作を行うことができない場合は点 *BR(BL(ep))* と点 *B(BL(ep))* のどちらか一点，または両方の点にロボットが存在する場合である (図 18)．点 *BR(BL(ep))* と点 *B(BL(ep))* の両方の点にロボットが存在すると仮定すると，点 *ep* と点 *BL(ep)*，また点 *BR(BL(ep))* と点 *B(BL(ep))* の 4 点にロボットが存在することになる．これら 4 点は $\mathcal{L}$ 上以外の点であるため，$\mathcal{L}$ 上以外には高々三台のみロボットが存在することと矛盾する．よって点 *ep* と点 *BL(ep)* にロボットが存在しかつ $\mathcal{L}$ 上の点が全てロボットで埋まっているとき，点 *BR(BL(ep))* と点 *B(BL(ep))* の両方の点にロボットが存在することはない．点 *BR(BL(ep))* にロボットが存在すると仮定する．点 *BR(BL(ep))* に存在するロボットは $\mathcal{S}$ 上を右回りに移動しようとする $(A4-1,A4-2),A4-1,A4-2$ どちらの場合も他の点に存在するロボットによって移動を行えない可能性があるが，そのような点は $\mathcal{L}$ 上以外の点でありかつ点 *ep*，点 *BL(ep)* ではない．$\mathcal{L}$ 上以外には高々三台のみロボットが存在し，$\mathcal{L}$ 上以外には点 *ep*，点 *BL(ep)*，点 *BR(BL(ep))* の三点にロボットが存在するため，点 *BR(BL(ep))* に存在するロボットの移動を妨げるロボットは存在しない．よって点 *BR(BL(ep))* に存在するロボットは必ず右回りに移動できる．なので点 *BR(BL(ep))* に永遠にロボットが存在することはないため，点 *BL(ep)* に存在するロボットは $A4-3$ の動作を行うことができ，条件 2 を満たさなくなる．点 *B(BL(ep))* にロボットが存在すると仮定する．点 *B(BL(ep))* に存在するロボットは $A5-1$ より直近の $\mathcal{S}$ 上の軸上でない点にロボットが存在しない場合その $\mathcal{S}$ 上の点に移動する．ここで $\mathcal{L}$ 上以外には点 *ep* と点 *BL(ep)*，点 *B(BL(ep))* にロボットが存在するため，直近の $\mathcal{S}$ 上の軸上でない点

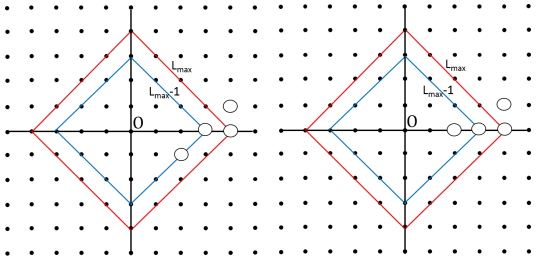


図 18: $\mathcal{S}$ の軸上の点に存在するロボットが移動できない場合

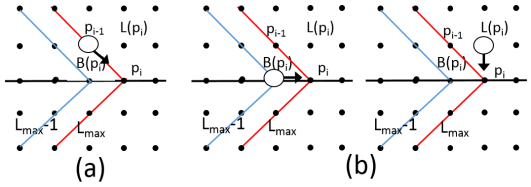


図 19: (a):ロボットの存在しない点 b_i に点 b_{i-1} に存在するロボットが移動 (b):ロボットの存在しない点 b_i に $\mathcal{L}$ 上以外に存在するロボットが移動し，$\mathcal{L}$ 上のロボットの存在しない点の数が減少

にロボットが存在することはない．よって点 $B(BL(ep))$ に存在するロボットは必ず直近の $\mathcal{S}$ 上の軸上でない点に移動できるため，点 $B(BL(ep))$ に永遠にロボットが存在することはない．またその直近の $\mathcal{S}$ 上の軸上でない点は点 $BR(BL(ep))$ であるが，点 $BR(BL(ep))$ に存在するロボットは必ず右回りに移動できるため，点 $B(BL(ep))$ に存在するロボットが $\mathcal{S}$ 上に移動することで点 $BL(ep)$ に存在するロボットが $A4-3$ の動作を行えない状態になることはない．以上から，$\mathcal{L}$ 上の点が全てロボットで埋まっている場合，必ず点 $BL(ep)$ に存在するロボットは移動できるため，条件 2 を満たさなくなる．次に $\mathcal{L}$ 上の点にロボットが存在しない点がある場合を仮定する．もし点 $BR(B(ep))$ にロボットが存在しない場合，条件 1 を満たしていないため，点 $BR(B(ep))$ にロボットが存在する場合を仮定する．ここで，説明のため $\mathcal{G}$ 上の特定の点 p_i を始点として，原点からの距離が p_i と同じである点を時計回りの順で並べた列 $B_{p_i}=(b_1,b_2,\cdots,b_m)$(ただし，$b_{i+1}=BR(b_i)$ である) を定義する．B_{r_i} の場合，$B_{r_i}=(r_i,BR(r_i),BR(BR(r_i)),\cdots)$ となり，列の定義から $|B_{r_i}|=4*dist(\mathrm{O},p_i)$ となる．

以下点 $B(ep)$ を b_1 とし，b_1 を始点とした列 B_{b_1} について考える．

条件 1 を満たしているため，b_2 にはロボットが存在する．そのため，$b_i(i\geq 3)$ について考える．b_i が軸上以外の点のとき，b_i にロボットが存在せず，b_i-1 にロボットが存在するとき b_i-1 に存在するロボットは b_i に移動する $(A3-1)$．そのため，$i\leq L_{max}$ のときロボットの存在しない点が b_2 となる．このとき b_1 に存在するロボットが移動でき，条件 1 を永遠に満たすことと矛盾する．

以降 $i>L_{max}$ とする．b_i にロボットの存在せず，また b_i が軸上以外の点のとき，ロボットの存在しない点は必ず左回りに移動し，軸上の点がロボットの存在しない点となる．

b_i にロボットの存在せず，また b_i が軸上の点のとき，点 $L(b_i)$ と点 $B(b_i)$ の両方の点にロボットが存在しない場合，$A3-2$ より b_i-1 に存在するロボットは b_i に移動する (図 19(a))．

次にロボットの存在しない点 b_i が軸上の点であり，点 $L(b_i)$ または点 $B(b_i)$ にロボットが存在する場合を考える．このとき $A2-2$,$A4-4$ より点 $L(b_i)$ または点 $B(b_i)$ に存在するロボット一台が点 b_i に移動する (図 19(b))．よって $\mathcal{L}$ 上のロボットの存在しない点の数は減少する．

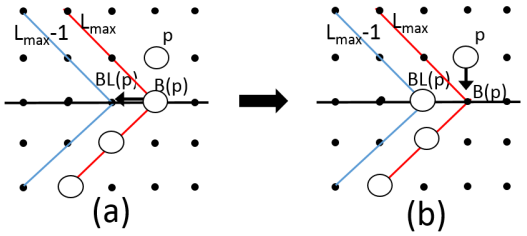


図 20: (a) 点 $B(p)$ に存在するロボットが点 $BL(p)$ に移動し，$\mathcal{L}$ 上のロボットの存在しない点の数が一時的に増加 (b):点 p に存在するロボットが点 $B(p)$ に移動し，結果的に $\mathcal{L}$ 上のロボットの存在しない点の数は変化しない

ここで，点 ep でない点 $p\in\mathcal{EP}$ にロボットが存在するとき，点 $B(p)$ のロボットが $A3-3$ より $\mathcal{S}$ の軸上の点に移動し $\mathcal{L}$ 上のロボットの存在しない点の数が増加することがある (図 20(a))．しかしこのとき点 p に存在するロボットが必ず点 $B(p)$ に移動するため結果的に $\mathcal{L}$ 上のロボットの存在しない点の数は変化しない (図 20(b))．

以上から，ロボットの存在しない点 b_i が軸上の点のとき，その点に $L(b_i)$ または点 $B(b_i)$ に存在するロボット一台が移動して $\mathcal{L}$ 上のロボットの存在しない点の数が減少するか，ロボットの存在しない点が b_i-1 に移動する．ロボットの台数は $\mathcal{L}$ 上の点の数以上なため，$L(b_i)$ または点 $B(b_i)$ に存在するロボットが $\mathcal{L}$ 上に移動することが繰り返し起きた場合いずれ $\mathcal{L}$ 上の点が全てロボットで埋まる．このとき，必ず条件 2 を満たさなくなる．

一方もし $L(b_i)$ または点 $B(b_i)$ に存在するロボットの $\mathcal{L}$ への移動が起きず，ロボットの存在しない点が移動していき，点 $b_{L_{max}+1}$ でも $L(b_{L_{max}+1})$ または点 $B(b_{L_{max}+1})$ に存在するロボットの点 $b_{L_{max}+1}$ への移動が起きなかった場合，ロボットの存在しない点は $b_{L_{max}}$ に移動し，$i\leq L_{max}$ のときロボットの存在しない点は必ず b_2 まで移動し，条件 1 を永遠に満たすことと矛盾する．

以上から点 ep にロボットが存在するとき，いつかは条件 1 または条件 2 を満たさなくなるので，b_1 つまり点 $B(ep)$ に存在するロボットはいつかは移動できる．よって点 ep に存在するロボットはいつかは点 $B(ep)$，つまり $\mathcal{L}$ 上に移動することができる．

□

補題 3.3. 原点からの距離が $L_{max}+1$ 以上の点に存在するロボットは点 ep にいつかは移動可能である

証明． 原点からの距離が $L_{max}+1$ の点 $(\neq ep)$ に存在するロボットは $A2-1$ より移動先の点にロボットがいなければ点 ep に向かって右回りに移動する．補題 3.2 から点 ep に存在するロボットはいつか点 $B(ep)$ に移動するので，右回りに一回移動すると点 ep に到達する点に存在するロボットは点 ep に移動できる．次に右回りに二回移動すると点 ep に到達する点に存在するロボットは，移動先の点が，右回りに一回移動すると点 ep に到達する点であるため，同様に点 ep に移動できる．以下同様にして原点からの距離が $L_{max}+1$ の点 $(\neq ep)$ に存在するロボットはいつかは点 ep に移動できる (図 21(a))．

原点からの距離が $L_{max}+2$ 以上の軸上の点に存在す

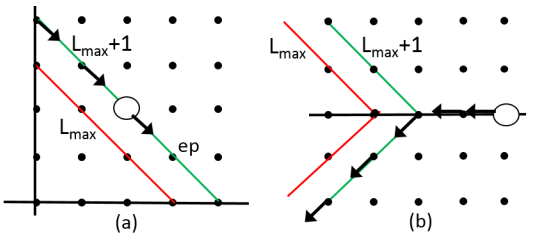


図 21: (a):原点からの距離が $L_{max}+1$ の点に存在するロボットの点 ep に向かって右回りの移動 (b):原点からの距離が $L_{max}+2$ 以上の軸上の点に存在するロボットの原点方向への移動

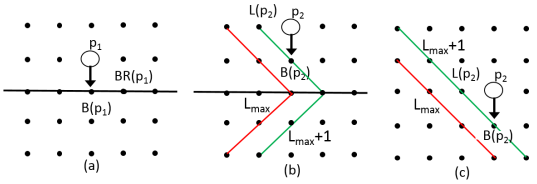


図 22: (a):原点からの距離が $L_{max}+2$ 以上の点 p_1 に存在するロボットの動作 (b),(c):原点からの距離が $L_{max}+1$ の点 p_2 に存在するロボットの動作．(b) の点 $B(p_2)$ は点 ep

るロボットは $A1-1$ より移動先の点にロボットがいなければ軸上に沿って原点方向に移動する．原点からの距離が $L_{max}+1$ の軸上の点に存在するロボットは点 ep に移動できるので，原点からの距離が $L_{max}+2$ の軸上の点に存在するロボットも点 ep に移動できる．以下同様にして原点からの距離が $L_{max}+2$ 以上の軸上の点に存在するロボットはいつかは点 ep に移動できる (図 21(b)).

原点からの距離が $L_{max}+2$ 以上の軸上以外の点であり，移動先の点が軸上の点である点を点 p_1 とする．点 p_1 に存在するロボットは $A1-2$ より移動先の点 $B(p_1)$ と，移動先の点に移動する可能性のある軸上の点 $BR(p_1)$ にロボットがいなければ点 $B(p_1)$ に移動する．軸上のロボットはいつかは点 ep に移動するので，点 p_1 に存在するロボットはいつかは点 ep に移動できる (図 22(a)).

原点からの距離が $L_{max}+2$ 以上の軸上以外の点であり，移動先の点が，原点からの距離が $L_{max}+1$ でありかつ軸上以外の点である点 p_2 に存在するロボットは $A1-3$ より移動先の点 $B(p_2)$ と，移動先の点に移動する可能性のある原点からの距離が $L_{max}+1$ の点 $L(p_2)$ にロボットがいなければ点 $B(p_2)$ に移動する．移動先の点が点 ep ならば点 ep に存在するロボットはいつかは点 $B(ep)$ に移動し，点 ep でない場合も原点からの距離が $L_{max}+1$ の点に存在するロボットはいつかは点 ep に移動するため，点 p_2 に存在するロボットはいつかは点 ep に移動できる (図 22$(b),(c)$).

原点からの距離が $L_{max}+2$ 以上の軸上以外の点であり，移動先の点が原点からの距離が $L_{max}+2$ 以上の軸上以外の点である点 p_3 に存在するロボットは $A1-4$ より移動先の点 $B(p_3)$ にロボットがいなければ移動先の点に移動する．そのためいつかは点 p_1 または点 p_2 のいずれかに移動できるため点 p_3 に存在するロボットは点 ep に移動できる (図 23).

以上から原点からの距離が $L_{max}+1$ 以上の点 (ただ

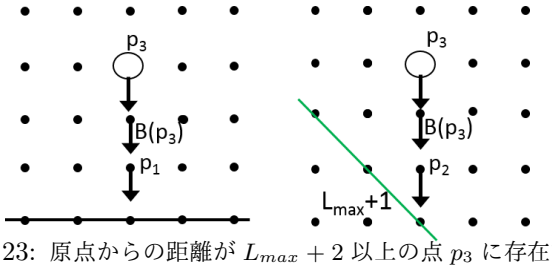


図 23: 原点からの距離が $L_{max}+2$ 以上の点 p_3 に存在するロボットの動作

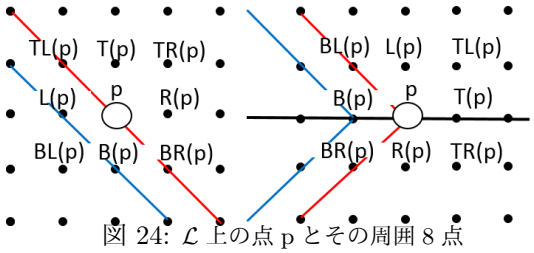


図 24: $\mathcal{L}$ 上の点 p とその周囲 8 点

し点 ep を除く) に存在するロボットは点 ep にいつかは移動できる．

□

補題 3.4. $\mathcal{L}$ 上のロボットは原点からの距離が L_{max} の点，または $L_{max}-1$ の点以外の点に移動することはない

証明 . $\mathcal{L}$ 上の点 p に存在するロボットは $A3-1$,$A3-2$,$A3-3$ より点 $BR(p)$ または点 $B(p)$ に移動する．図 24 より点 $BR(p)$ は $\mathcal{L}$ 上の点，点 $B(p)$ は $\mathcal{S}$ 上の点であるため，$\mathcal{L}$ 上のロボットは原点からの距離が $L_{max}+1$ 以上の点，または $L_{max}-2$ 以下の点に移動することはない．

□

補題 3.5. $\mathcal{S}$ 上のロボットは原点からの距離が L_{max} の点，または $L_{max}-1$ の点以外の点に移動することはない．

証明 . $\mathcal{S}$ 上の点 p に存在するロボットは $A4-1$,$A4-2$,$A4-3$,$A4-4$ より点 $BR(p)$ または点 $T(p)$ に移動する．図 25 より点 $BR(p)$ は $\mathcal{S}$ 上の点，点 $T(p)$ は $\mathcal{L}$ 上の点であるため，$\mathcal{S}$ 上のロボットは原点からの距離が $L_{max}+1$ 以上の点，または $L_{max}-2$ 以下の点に移動することはない．

□

補題 3.2,3.3,3.4,3.5 から以下の系が成り立つ．

系 3.1. 原点からの距離が $L_{max}+1$ 以上の点に存在するロボットの台数は減少し増加することはない．

補題 3.6. 原点からの距離が $L_{max}-2$ の点に存在するロボットはいつかは $\mathcal{S}$ 上に移動できる．

証明 . 系 3.1 より原点からの距離が $L_{max}+1$ 以上の点に存在するロボットはいつかいなくなるため，以下原点からの距離が $L_{max}+1$ 以上の点にロボットは存在しないものとする．

原点からの距離が $L_{max}-2$ の点に存在するロボットは $A5-1$ より直近の $\mathcal{S}$ 上に移動する．また，$A5-2$ より直近の $\mathcal{S}$ 上に移動できない場合は右回りに移動して空いている $\mathcal{S}$ 上の軸上以外の点を探す (図 26).

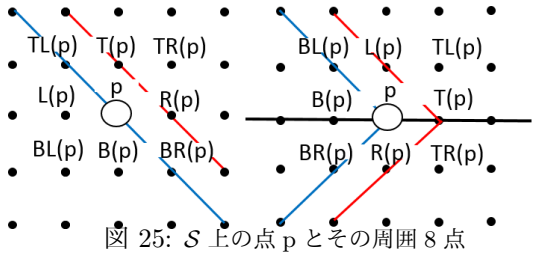


図 25: $\mathcal{S}$ 上の点 p とその周囲 8 点

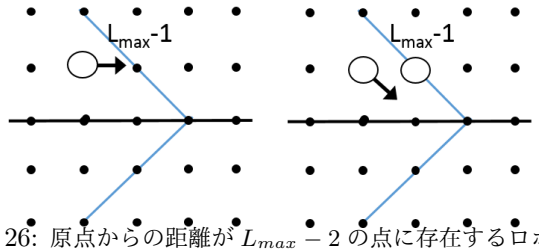


図 26: 原点からの距離が $L_{max}-2$ の点に存在するロボットの動作

$\mathcal{S}$ 上にロボットの存在しない点が存在する場合，その点が軸上の点のとき，原点からの距離が $L_{max}+1$ の点にロボットは存在しないため $A4-2$ より $\mathcal{S}$ 上を一回右回りに移動すると軸上の点に着くロボットは必ず軸上の点に移動できる．$\mathcal{S}$ の軸上の点に存在するロボットは原点からの距離が $L_{max}+1$ 以上の点にロボットが存在しないため，$A4-3$ より $\mathcal{S}$ 上を右回りに移動することはない．また，$\mathcal{L}$ 上のロボットも原点からの距離が $L_{max}+1$ 以上の点にロボットが存在しないため，$A3-3$ より $\mathcal{S}$ 上に移動することはない．そのため，$\mathcal{S}$ 上のロボットの存在しない点の位置が変わり続けることはない．よって原点からの距離が $L_{max}-2$ の点に存在するロボットは，$\mathcal{S}$ 上にロボットの存在しない点が存在する場合，いつかは $\mathcal{S}$ 上に移動することができる．

そのため原点からの距離が $L_{max}-2$ の点に存在するロボットが一台も $\mathcal{S}$ 上に永遠に移動できない場合は $\mathcal{S}$ 上が全てロボットで永遠に埋まっている場合である．$\mathcal{S}$ 上のロボットの $\mathcal{S}$ 以外への移動は，$A4-4$ より $\mathcal{S}$ の軸上の点 p に存在するロボットによる $\mathcal{L}$ の軸上の点 $T(p)$ への移動のみである (図 27).

点 p に存在するロボットが永遠に点 $T(p)$ に移動できない場合は，$A4-4$ より $TL(p)$ または $T(p)$ にロボットが永遠に存在する場合である．しかし，$TL(p)$ は原点からの距離が $L_{max}+1$ の点であり，原点からの距離が $L_{max}+1$ 以上の点にロボットは存在しないため，点 p に存在するロボットが永遠に点 $T(p)$ に移動できない場合は，$T(p)$ にロボットが永遠に存在する場合のみである．なので，$\mathcal{S}$ 上が全てロボットで埋まっていてかつ原点からの距離が $L_{max}+1$ 以上の点にロボットは存在しないとき，$\mathcal{L}$ の軸上の 4 点のうち少なくとも 1 点はロボットが存在しない点となることを示す．

点 $T(p)$ に存在するロボットは，点 $TL(p)$ にロボットが存在することはないので，$A3-3$ の条件を満たすことはない．つまり $\mathcal{L}$ の軸上の点に存在するロボットは $\mathcal{S}$ 上に移動することはなく，$A3-1$ より $\mathcal{L}$ 上を右回りに移動する動作のみ可能である．$\mathcal{L}$ の軸上の点に存在するロボットが $\mathcal{L}$ 上を右回りに移動できない条件は，その移動先の点にロボットが存在するときのみである．つまり，$\mathcal{L}$ の軸上の 4 点に存在するロボット全てがが $\mathcal{L}$ 上を右回りに移動できないとき，$\mathcal{L}$ 上には，$\mathcal{L}$ の軸上の 4 点と，その右回りの $\mathcal{L}$ 上の点の 4 点の合計 8 点にロボットが存在する．しかし，$\mathcal{S}$ 上の点が全てロボットで埋まっているとき，$\mathcal{S}$ 上には $4L_{max}-4$ 台のロボットが存在し，$n-(4L_{max}-4)\leq 7$ より $\mathcal{S}$ 上以外の点には，高々 7 台のみロボットが存在することと矛盾する．よって $\mathcal{L}$ の軸上の点のうち，少なくとも一点はロボットの存在しない点となる．よって $\mathcal{S}$ の軸上の点に存在するロボットの少なくとも一台が $\mathcal{L}$ 上に移動できる．

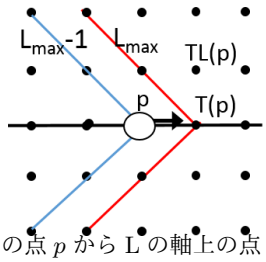


図 27: S の軸上の点 p から L の軸上の点 $T(p)$ への移動．図中の点 $TL(p)$ は点 ep

以上から $\mathcal{S}$ 上が永久に埋まっていることはない．よって，原点からの距離が $L_{max}-2$ の点にいるロボットはいつかは $\mathcal{S}$ 上に移動できる．

□

補題 3.7. 原点からの距離が $L_{max}-3$ 以下の点 *(*原点含む*)* に存在するロボットはいつかは原点からの距離が $L_{max}-2$ の点に移動できる．

証明 ． $A6-1$,$A7-1$ より原点からの距離が $L_{max}-3$ 以下の原点以外の点に存在するロボットは移動先を一つしか持たず，原点からの距離が $L_{max}-4$ 以下の原点以外の点に存在するロボットは，その唯一の移動先の点にロボットが存在しなければ移動可能である．原点からの距離が $L_{max}-3$ の原点以外の点に存在するロボットは $A6-1$ より移動先の原点からの距離が $L_{max}-2$ の点と移動先の点に移動する可能性のある原点からの距離が $L_{max}-2$ の点のどちらかにロボットが存在する場合移動を行わない．補題 3.6 より原点からの距離が $L_{max}-2$ の点に存在するロボットはいつかは $\mathcal{S}$ 上に移動し，補題 3.3,3.5 より $\mathcal{S}$ または $\mathcal{L}$ 上のロボットが原点からの距離が $L_{max}-2$ の点に移動することはないため，原点からの距離が $L_{max}-3$ の原点以外の点に存在するロボットはいつかは原点からの距離が $L_{max}-2$ の点に移動することができる．よって原点からの距離が $L_{max}-4$ 以下の原点以外の点に存在するロボットも同様にいつかは原点からの距離が $L_{max}-2$ の点に移動することができる．

原点に存在するロボットは $A8-1$ より原点からの距離が 1 の点全てにロボットが存在しない場合原点からの距離が 1 の点に移動できる．補題 3.6 とこれまでの結果から原点を除く原点からの距離が $L_{max}-2$ 以下の点に存在するロボットはいつかは $\mathcal{S}$ 上に移動できる．また $L_{max}\geq 3$ より $L_{max}-1\geq 2$ であり，原点からの距離が 1 の点が $\mathcal{L}$ または $\mathcal{S}$ 上の点であることはないため，原点

に存在するロボットはいつかは原点からの距離が 1 の点に移動できる．

よって原点からの距離が $L_{max}-3$ 以下の点 *(*原点含む*)* に存在するロボットはいつかは原点からの距離が $L_{max}-2$ の点に移動できる．

□

補題 3.4,3.5,3.6,3.7 から以下の系が成り立つ．

系 3.2. 原点からの距離が $L_{max}-2$ 以下の点に存在するロボットの台数は減少し増加することはない．

また，系 3.1,3.2 から以下の系が成り立つ．

系 3.3. 全てのロボットがいつかは $\mathcal{L}$ または $\mathcal{S}$ 上の点に移動し，$\mathcal{L}$ または $\mathcal{S}$ 上の点に存在するロボットは $\mathcal{L}$ 上と $\mathcal{S}$ 上以外の点に移動することはない．

補題 3.8. 全てのロボットが $\mathcal{L}$ または $\mathcal{S}$ 上の点に存在するとき，$\mathcal{L}$ 上の点が全てロボットで埋まる．

証明． $A4-1$,$A4-2$ より $\mathcal{S}$ 上の軸上以外の点に存在するロボットは軸上の点に向かって右回りに移動する．$\mathcal{L}$ および $\mathcal{S}$ 上以外の点にロボットは存在しないため，原点からの距離が $L_{max}-2$ の点にロボットは存在せず，移動先の点に存在するロボット以外のロボットに移動を妨げられることはない．$\mathcal{S}$ の軸上の点 p に存在するロボットは点 $T(p)$, 点 $TL(p)$ 両方にロボットが存在する場合，$\mathcal{S}$ 上を右回りに移動する可能性があるが，点 $TL(p)$ は $\mathcal{L}$ または $\mathcal{S}$ 上の点でないため，ロボットは存在せず，この移動を行うことはない．よって $\mathcal{S}$ 上にロボットが存在する場合，少なくとも一台は $\mathcal{S}$ の軸上の点に移動し，また軸上の点から右回りに移動することはない．

また，点 $TL(p)$ にロボットが存在しないため，$A3-3$ より点 $T(p)$ に存在するロボットが点 p に移動することはない，つまり $\mathcal{L}$ 上の点に存在するロボットが $\mathcal{S}$ 上の点に移動することはない．

$\mathcal{L}$ 上の点にまだロボットの存在しない点が存在する場合，$A3-1$,$A3-2$ より $\mathcal{L}$ 上のロボットは $\mathcal{L}$ 上を右回りに移動しようとする．そのため必ず $\mathcal{L}$ の軸上の点にロボットが存在しない状態になる．このとき $\mathcal{S}$ の軸上の点 p に存在するロボットは点 $TL(p)$ にロボットが存在することはないため，$A4-4$ より $\mathcal{L}$ の軸上の点である点 $T(p)$ に移動できる．また，$L_{max}=\lfloor \frac{n}{4} \rfloor$ より，$4L_{max} \leq n$ なので，$\mathcal{L}$ 上の点の数はロボットの総数以下であるため，$\mathcal{L}$ が埋まっていない場合必ず $\mathcal{S}$ 上の点にロボットが存在する．よって $\mathcal{L}$ 上の点が全て埋まるまで $\mathcal{S}$ の軸上の点のロボットが $\mathcal{L}$ の軸上の点に移動する動作が起きる．よって必ず $\mathcal{L}$ 上の点がすべて埋まる．

□

補題 3.9. 全てのロボットが $\mathcal{L}$ または $\mathcal{S}$ 上の点に存在しかつ $\mathcal{L}$ 上の点が全てロボットで埋まっているとき，$\mathcal{L}$ 上のロボットは全て静止する．

証明． 全てのロボットが $\mathcal{L}$ または $\mathcal{S}$ 上の点に存在するため，原点からの距離が $L_{max}+1$ の点にロボットは存在しない．なので $\mathcal{L}$ の軸上の点を点 p とすると点 $L(p)$ にロボットが存在することはないため，ロボットは $A3-3$ の条件を満たすことはない．また $\mathcal{L}$ 上の点が全てロボットで埋まっているため，$\mathcal{L}$ 上の全てのロボットは $\mathcal{L}$ 上を右回りに移動することはできない．つまり $A3-1$,$A3-2$ の条件を満たすことはない．よって全てのロボットが $\mathcal{L}$ または $\mathcal{S}$ 上の点に存在しかつ $\mathcal{L}$ 上の点が全てロボットで埋まっているとき,$\mathcal{L}$ 上のロボットは全て静止する．

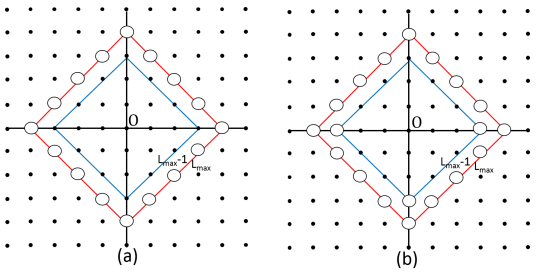


図 28

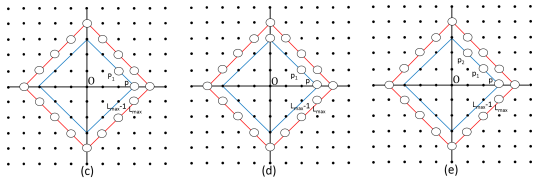


図 29

□

補題 3.10. 全てのロボットが $\mathcal{L}$ または $\mathcal{S}$ 上の点に存在しかつ $\mathcal{L}$ 上の点が全てロボットで埋まっているとき，$\mathcal{S}$ 上のロボットは全て静止する．

証明． $\mathcal{L}$ 上の点が全てロボットで埋まっていることから，$\mathcal{L}$ 上以外には高々三台のみロボットは存在し，全てのロボットが $\mathcal{L}$ または $\mathcal{S}$ 上の点に存在するので $\mathcal{L}$ または $\mathcal{S}$ 上以外の点にロボットは存在しないため，$\mathcal{S}$ 上には高々三台のみロボットは存在する．$\mathcal{S}$ の軸上以外の点に存在するロボットは，$A4-1$,$A4-2$ より軸上の点 p または自身のいる点が軸上以外の点でかつ唯一の移動先の点にロボットが存在する点に到達するまで移動する．そのため，$\mathcal{S}$ 上は以下のいずれかの状態となる (図 28, 29).

(a) ロボットが存在しない

(b) 全てのロボットが異なる軸上の点に存在する

(c) 点 p と，移動先の点が点 p である点 p_1 にロボットが存在する

(d) 軸上の 2 点と，移動先の点が p である点 p_1 にロボットが存在する

(e) 点 p と，移動先の点が p である点 p_1 および点 p_1 が移動先の点である点 p_2 にロボットが存在する

(a) の場合はロボットが $\mathcal{S}$ 上に存在しないため考慮しない．

全てのロボットが $\mathcal{L}$ または $\mathcal{S}$ 上の点に存在するため，点 $TL(p)$ にロボットが存在することはない．よって $A4-4$ の条件を満たすことはない．よって軸上の点に存在するロボットが移動することはない．そのため，唯一の移動可能な点が点 p である点 p_1 や，唯一の移動可能な点が p_1 である点 p_2 に存在するロボットも移動することはな

い．なので (b),(c),(d),(e) どの状態であっても $\mathcal{S}$ 上のロボットが移動することはない．よって全てのロボットが $\mathcal{L}$ または $\mathcal{S}$ 上の点に存在しかつ $\mathcal{L}$ 上の点が全てロボットで埋まっているとき $\mathcal{S}$ 上のロボットは全て静止する．

□

補題 3.9,3.10 から以下の系が成り立つ．

系 3.4. 全てのロボットが L または $\mathcal{S}$ 上に存在しかつ $\mathcal{L}$ 上の点全てにロボットが存在するとき，全てのロボットは静止する．

系 3.3,3.4，補題 3.8 から以下の定理が成り立つ．

定理 3.1. 提案アルゴリズムは以下の仮定の下で格子平面上において，一点包囲問題を解く．

ロボットの半径 *small*．共通座標を持たない．$M(v)$ に存在する他のロボットを観測できる．ロボットの総数を知っている．半同期である． □

4 おわりに

本稿では，グリッド上での small サイズファットロボットの一点包囲について考えた．

今後の課題として，視野範囲をロボットの周囲 8 点のみとした場合，ロボットサイズを *large* とした場合などについて一点包囲問題を解くアルゴリズムを考えることが挙げられる．

参考文献

[1] A.Bandettini,F.Luporini,and G.Viglietta,"A survey on open problems for mobile robots", *ArXiv e-prints*,nov 2011.

[2] G.Prencipe,"On the feasibility of gathering by autonomous mobile robots",*SIROCCO* 2005,*LNCS 3499*,pp.246-.

[3] P.Flocchini, G.Prencipe, N.Santoro, and P.Widmayer, "Gathering of asynchronous robots with limited visibility",*TCS 2005*, vol.337,no.1-3,pp.147-168.

[4] M.Cieliebak, P.Flocchini, G.Prencipe, and N.Santoro, "Solving the robots gathering problem",*LNCS* 2719,pp.1181-,(2003).

[5] G.D'Angelo, G.Stefano, R.Klasing, and A.Navarra, "Gathering of robots on anonymous grids without multiplicity detection",*SIROCCO* 2012 , *LNCS* 7355, pp.327-338.

[6] J.Czyzowicz, L.Gasieniec, and A.Pelc, "Gathering few fat mobile robots in the plane",*TCS*, vol.410,pp.481-499,2009.

[7] A.Cord-Landwehr, A et al,"Collisionless gathering of robots with an Extent",*SOFSEM 2011: Theory and Practice of Computer Science* , *LNCS* 6543, pp.178-189.

[8] S.Gan Chaudhuri et al. "Gathering asynchronous transparent fat robots",*Distributed Computing and Internet Technoroly* , *LNCS* 5966, pp.170-175,, 2010.

[9] K.Bolla, T.Kovacsm, and G.Fazekas, "Gathering of fat robots with limited visibility and without global navigation",*Swarm and Evolutionary Computation*, *LNCS* 7269, pp.30-38, 2012.

[10] 伊藤, 片山, 和田,"共通座標系を有するファットロボットのグリッド上での集合について", 第 143 回アルゴリズム研究会, 研究報告アルゴリズム (AL),2013-AL-143(2),1-8(2013-02-22).

[11] 伊藤, 片山, 和田, "共通座標系を有しないグリッド平面上におけるファットロボットの集合", 研究報告アルゴリズム (AL),2014-AL-148(9),1-7(2014-06-06).

[12] C.Agathangelou, C.Georgiou and M.Mavronicolas, "A Distributed Algorithm for Gathering Many Fat Mobile Robots in the Plane",*ArXiv e-prints*, sep 2012.

[13] 伊藤佳進, 片山喜章, 和田幸一, "グリッド上での Large サイズファットロボットの集合について",*COMP2014-42*,pp.1-8,2015.

[14] 白川遥平, 和田幸一, 片山喜章, "自律分散ファットロボットの様々なグリッド上での集合問題について",*COMP2015-37'*,pp.1-10, 2016

[15] 長尾英剛, 片山喜章, 和田幸一, "三次元グリッド空間における自律分散ロボットの集合の研究",2016 年総合大会,2016

[16] Serafino Cicerone,Gabriele Di Stefano,Alfredo Navarra, "Asynchronous embedded pattern formation without orientation",*DISC 2016*,*LNCS*,vol.9888,pp.85 - 98.

[17] Marcello Mamino, Giovanni Viglietta,"Square Formation by Asynchronous Oblivious Robots",*ArXiv e-prints*,,2016

[18] Défago X, Konagaya A, "Circle Formation for Oblivious Anonymous Mobile Robots with no Common Sense of Orientation",*Proc. 2nd International Annual Workshop on Principles of Mobile Computing*,pp. 97 - 104,2002.

[19] Défago X, Souissi S, "Non Uniform Circle Formation Algorithm for Oblivious Mobile Robots with Convergence Towards Uniformity",*Theoretical Computer Science*, 396(1-3), pp. 97 - 1122,2008.

[20] Suparno Datta, Ayan Dutta, Sruti Gan Chaudhuri, and Krishnendu Mukhopadhyaya,"Circle Formation by Asynchronous Transparent Fat Robots",*Distributed Computing and Internet Technology Lecture Notes in Computer Science Volume 7753*, pp.195-207,2013.

ライトを持つ2台の自律分散ロボットのランデブーについて

奥村 太加志[1]　和田 幸一[2]　片山 喜章[3]
[1] 法政大学大学院理工学研究科応用情報工学専攻
[2] 法政大学理工学部応用情報工学科
[3] 名古屋工業大学大学院工学研究科情報工学専攻

概要：定数ビットの記憶領域(ライト)を持つ，非同期モデルにおける2台の自律分散ロボットによるランデブー問題について考察している．ライトを持たない基本的なロボットモデルにおいては，ランデブー問題はたとえ半同期モデルであっても可解でないことが知られている．しかし，複数の色を表示できるライトを持つことでランデブー問題が可解となる場合がある[5, 11]．自身と他のロボットのライトを観測できる場合(full-light)，非同期モデルで移動性が non-rigid の場合，3色のライトを用いることでランデブー問題を可解にできる．ここで，non-rigid とはロボットが計算した目的地に到達しない可能性があることであり，この場合，少なくとも最小移動距離$\delta > 0$は移動する．計算された目的地に必ず到達する場合を rigid という．また，full-light，non-rigid の半同期モデルでは，2色のライトを用いることでランデブー問題が可解になる[11]．

本稿では，非同期モデルに制限を与えたクラスを提案し，そのクラスにおいては，full-light, non-rigid，2色のライトでランデブー問題を可解になることを示す．また，非同期モデル，full-light，non-rigid で，ロボットが最小移動距離δの値に関する知識を持っている場合，2色のライトでランデブー問題を可解になることを示す．

1. はじめに

自律分散ロボット群とは，分散システムの研究の一つであり，複数のロボットが自律的に計算，移動を行い，全体である問題を解決するシステムである．研究では理論モデルを扱ったものが主である[1-3, 7, 10]．ロボットは平面上で自律的に計算，移動を行う点としてモデル化され，外見によって個体を識別することはできず，同一のアルゴリズムを実行する[4]．ロボットは active と inactive の状態があり，前者では周囲の観測(Look 命令)，行き先の計算(Compute 命令)，移動(Move 命令)を順に行う(LCM サイクル)．Look 命令で得られた情報はサイクル終了時に削除される(無記憶)．また，各ロボットの LCM サイクルの同期の程度から3つのスケジュール FSYNC(全同期:全ロボットの LCM サイクルの動作が共通)，SSYNC(半同期:FSYNC と同様であるが，動作しないロボットを 1 台以上許す)，ASYNC(非同期:全ロボットが独立して動作する)を定義できる．以上の基本的なモデルでは，与えられた問題を可解とすることが困難な場合がある．与えられた問題を可解とするための，必要最低限の能力を明らかにすることは，自律分散ロボット群の分野での重要な課題の一つである．

複数のロボットが任意の初期配置から有

限時間内にあらかじめ決められていない一点に集合する問題を集合問題といい，ロボットの台数が 2 台の集合問題をランデブー問題という．本稿では，ASYNC モデルでランデブー問題を可解とするために必要最低限の能力や仮定について考察している．

集合問題(ランデブー問題)は，基本的なロボットの場合, FSYNC モデルで可解であるが, SSYNC モデルでは非可解であることが知られている[4].

そこで基本的なロボットに，自身の状態を記録できる定数ビットの記憶領域(ライト)を搭載したモデルが提案されている[5]. この状態は Look 命令で観測し，Compute 命令で更新できるものとする．自身のライトのみを観測できるものを internal-light, 相手のライトのみを観測できるものを external-light，双方のライトを観測できるものを full-light と呼ぶ.

また，ロボットの移動性についても考える．Move 命令時に，計算された目的地に確実に移動できるものを rigid という．計算された目的地にたどり着かない場合があり，少なくとも最小移動距離$\delta > 0$は移動するものを non-rigid という．また，移動性が non-rigid であり，ロボットが最小移動距離δに関する知識を持っている場合，non-rigid(+δ)と表す.

full-light の ASYNC モデルの場合，移動性が rigid ならば 2 色のライト，移動性が non-rigid ならば 3 色のライトを用いることでランデブー問題を可解になる．external-light の ASYNC モデルの場合，移動性が rigid ならば 12 色のライト，移動性が non-rigid(+δ)ならば 3 色のライトを用いることでランデブー問題を可解になる(表 1).

本稿では，ASYNC モデルに制限を与え，LCM サイクルにおける Look 命令と Compute 命令が同時刻に行われる LC-atomic モデルを用いて, full-light, non-rigid, 2 色のライトでランデブー問題を可解にするアルゴリズムを示す．互いのライトの色のみを用いて，目的地を計算するアルゴリズムのクラスを L という．ASYNC モデルにおいて full-light，2 色のライトによるランデブー問題に対して，クラス L に属するアルゴリズムは存在しないことが証明されている [11]．しかしながら，本稿で示すアルゴリズムはクラス L に属するものであり，2 色の full-light で実現できるクラス L に属するアルゴリズムに対する ASYNC の十分条件を与えている．また，ASYNC モデル, full-light, non-rigid(+δ), 2 色のライトを用いることで，ランデブー問題を可解にするアルゴリズムを示す.

表 1．ライトを持つロボットによるランデブー問題

scheduler	movement	full-light [11]	external-light [5]	internal-light [5]	no-light [4, 9]
FSYNC	Non-rigid				○
SSYNC	Non-rigid	2	3	?	×
	Rigid		?	6	
	Non-rigid (+δ)		?	3	
ASYNC	Non-rigid	3	?	?	×
	Rigid	2	12	?	
	Non-rigid (+δ)	?	3	?	

斜線は，それよりさらに弱い仮定で可解であることを示している．
?は可解となるアルゴリズムが明らかでないことを示している．

2. ロボットのモデル

システムは，平面空間$\mathbb{R}^2$に存在するロボットの集合$\mathcal{R} = \{r_1, \ldots, r_n\}$から成る．ロボット$r_i$は，ライトと呼ばれる定数ビットの記憶領域$\ell(r_i)$を持つ．

ロボットr_iはそれぞれ，常に自身が原点となる独自の座標系を持っている．それぞれの座標系には向きや単位距離による共通の合意はないものとする．

ロボットには active と inactive の 2 つの状態がある．active の場合，以下の命令サイクル Look-Compute-Move(LCM)サイクルを実行する．

① Look 命令
センサを使用し周囲のロボットを観し，座標を得る．

② Compute 命令
Look 命令で得た観測結果をもとに，移動先の座標を計算する．

③ Move 命令
実際に計算された位置に移動する．このとき，一度の Move 命令で目的地に確実に到達できる場合，その移動性を rigid という．ロボットが一度の Move 命令で目的地にたどり着けず，その場合は少なくとも最小移動距離$\delta > 0$は移動するものとする．この移動性を non-rigid という．また，移動性が non-rigid で，かつロボットが最小移動距離δの値に関する知識を持っている場合，その移動性を non-rigid(+δ)という．

LCM サイクルが終了する度に，Look 命令で得た情報は消去され，次サイクルでは使用できないものとする(無記憶)．

各命令の同期の程度によって，3 種類のスケジュールが考えられている．

- FSYNC(全同期)
  すべてのロボットの LCM サイクルの各命令を行う開始時刻が一致している．
- SSYNC(半同期)
  FSYNC 同様，各動作は同期しているが，サイクルを実行しないロボットの存在を 1 台以上許す．
- ASYNC(非同期)
  すべてのロボットが独立して LCM サイクルを実行していて，各動作が同期していない．

本稿では，ASYNC モデルの制限を与えたクラスについて考えている．

- *LC-atomic*
  毎サイクルにおいて，Look 命令と Compute 命令が同時刻に実行される．すなわち，ロボットは，LCM サイクルにおける他のロボットの Look 命令，Compute 命令間を観測することはできない．
- *Move-atomic*
  毎サイクルにおいて，Move 命令は瞬間的に実行される．すなわちロボットは，移動中の他のロボットを Look 命令で観測することはできない．

ロボットは自身の内部状態を記録できる定数ビットの記憶領域(ライト)を搭載しているものとする．ライトの色は Look 命令で観測でき，Compute 命令で更新できるものとする．ライトの見え方によって，以下の 3

つのモデルが考えられる.

- full-light
  Look 命令時に, 自分と他のロボットの状態を観測できる.
- external-light
  他のロボットの状態は観測できるが, 自分の状態は観測できない.
- internal-light
  自分の状態は観測できるが, 相手の状態は観測できない.

n(≥ 2)台のロボットが有限時間内に, 予め決められていない一点に集合する問題を集合問題といい, n = 2の場合を特別にランデブー問題という.

先行研究

ランデブー問題は FSYNC モデルでは解くことができるが, SSYNC モデルでは解くことができない.

定理 1. [4] ランデブー問題は, SSYNC モデルのロボットでは解くことができない.

ロボットにライトを持たせた場合, ランデブー問題が可解となる条件を以下の定理で示す(表 1).

定理 2. [5, 11]

① full-light, non-rigid, 2 色のライトを用いた SSYNC モデルにおいては, ランデブー問題は可解である.
② external-light, non-rigid, 3 色のライトを用いた SSYNC モデルにおいては, ランデブー問題は可解である.
③ internal-light, rigid, 6 色のライトを用いた SSYNC モデルにおいては, ランデブー問題は可解である.
④ internal-light, non-rigid(+δ), 3 色のライトを用いた SSYNC モデルにおいては, ランデブー問題は可解である.
⑤ full-light, non-rigid, 3 色のライトを用いた ASYNC モデルにおいては, ランデブー問題は可解である.
⑥ full-light, rigid, 2 色のライトを用いた ASYNC モデルにおいては, ランデブー問題は可解である.
⑦ external-light, rigid, 12 色のライトを用いた ASYNC モデルにおいては, ランデブー問題は可解である.
⑧ external-light, non-rigid(+δ) 3 色のライトを用いた ASYNC モデルにおいては, ランデブー問題は可解である.

4. ライトを持つ ASYNC モデルのランデブーアルゴリズムについて

アルゴリズムの正しさの証明についての詳細は[8]に記述されている.

2 台のロボットはライトの初期状態 A から開始する. 互いのライトが A であることを観測した場合, ロボットは互いの位置の中点を目的地とし. ライトの色を B に更新する. 互いのライトが B であることを観測した場合, ライトの色を A に更新する(rigid で, 互いのライトが B であることを観測した場合, 2 台のロボットがすでに集合している). 自身のライトの色が A, 相手のライトの色が B である場合, 目的地を相手の位置とする. 自身のライトの色が B, 相手の

ライトの色が A である場合，目的地を現在の自身の位置とする(この場合，相手のロボットがサイクルを実行することで集合する)．

Algorithm 1 Rendezvous (scheduler, movement, initial-li

Parameters: scheduler, movement-restriction, Initial-light
Assumptions: full-light, two colors (A and B)

```
1:  case me.light of
2:  A:
3:      if other.light =A then
4:          me.light ← B
5:          me.des ← the midpoint of me.position and other.po
6:      else me.des ← other.position
7:  B:
8:      if other.light = A then
9:          me.des ← me.position // stay
10:     else me.light ← A
11: endcase
```

Algorithm 1 はスケジューラ，移動性，ライトの初期状態の 3 つをパラメータとしている．また，ライトは full-light とし，A と B の 2 色を使用する．

定理 3. Rendezvous(ASYNC, rigid, A)は，ランデブー問題は可解である．

LC-atomic ASYNC モデル，non-rigid

アルゴリズムのクラスについて，目的地とライトの色について計算する際，Look 命令で観測したライトの色のみをパラメータとするものを，クラス L と呼ぶ[11]．Algorithm 1 はこのクラス L に属している．また[11]より，non-rigid， 2 色を用いた ASYNC モデルによるランデブーアルゴリズムのうち，クラス L に属するものは存在しない．また，non-rigid， 3 色を用いた，ASYNC モデルによるクラス L のランデブーアルゴリズムは存在するが，LC-atomic ASYNC を用いることで，non-rigid で 2 色のライトでランデブー問題が可解になる．

定理 4. Rendezvous(LC-atomic ASYNC, non-rigid，any)は，ランデブー問題は可解である．すなわち初期状態に関わらず，rigid, 2 色のライトを用いた LC-atomic ASYNC モデルにおいては，クラス L に分類されるアルゴリズムでランデブー問題は可解となる．

ASYNC モデル，non-rigid(+δ)

移動性を non-rigid(+δ)とした場合を考える，ここでは，Rendezvous(ASYNC, non-rigid(+δ)，A)でランデブー問題が可解となる Algorithm 2 を示す．

2 台のロボットの距離を DIST とする．DIST > 2δで，かつ互いのライトの色が B であることを観測した場合，ロボットは相手に向かってδ/2だけ移動を行う．これ以外のライトの色を観測した場合，自身のライトの色を B に更新する．

2δ ≥ DIST ≥ δで，かつ互いのライトが A であることを観測した場合，自身のライトの色を B に更新し，互いの中点を目的地とする．これ以外のライトの色を観測した場合，自身のライトの色を A に更新する．

δ > DISTの場合，上記で示した Algorithm 1 を使用する．この DIST に関する 3 つの場合より，集合することができる．

Algorithm 2 RendezvousWithDelta (ASYNC, Non-Rigid(+δ), A)

Assumptions: full-light, two colors (A and B)

```
1:  case dis(me.position, other.position)(= DIST) of
2:  DIST > 2δ:
3:      if me.light =other.light =B then
4:          me.des ← the point moving by δ/2 from me.position to other.position
5:      else me.light ← B
6:  2δ ≥ DIST ≥ δ:
7:      if me.light = other.light = A then
8:          me.light ← B
9:          me.des ← the midpoint of me.position and other.position
10:     else me.light ← A
11: δ > DIST: //Rendezvous(ASYNC, Rigid, A)
12:     case me.light of
13:     A:
14:         if other.light =A then
15:             me.light ← B
16:             me.des ← the midpoint of me.position and other.position
17:         else me.des ← other.position
18:     B:
19:         if other.light = A then me.des ← me.position // stay
20:         else me.light ← A
21:     endcase
22: endcase
```

定理 5. Rendezvous(ASYNC, non-rigid(+δ), any)は，ランデブー問題は可解である．すなわち，ロボットが最小移動距離δに関する知識を持っている場合，non-rigid，2 色を用いた ASYNC モデルにおいては，ランデブー問題は可解となる．

4. 結論と展望

本稿では，移動性を non-rigid(+δ)と仮定し，ASYNC モデルで最適なライトの色数でランデブー問題を解くアルゴリズムを示した．また，LC-atomic ASYNC モデルにおいて，移動性を non-rigid としたとき，最適なライトの色数でランデブー問題を解く，クラス L に属するアルゴリズムを示した．残された未解決問題として，non-rigid，2 色のライトを用いた ASYNC モデルでランデブー問題が可解であるか[1]，non-rigid，2 色のライトを用いた ASYNC の部分クラスに対して，クラス L に属するランデブーアルゴリズムの開発などが挙げられる．

[1] ごく最近，肯定的に解決された[6]

参考文献

1. N. Agmon and D. Peleg, Fault-tolerant gathering algorithms for autonomous mobile robots, SIAM Journal on Computing, 36, 56-82, 2006.
2. X. Defago, M. Gradinariu Potop-Butucaru, J. Clement, S. Messika, P. Raipin Parvedy: Fault and Byzantine tolerant self-stabilizing mobile robots gathering Feasibility study -. CoRRabs/1602.05546, 2016.
3. B. Degener, B. Kempkes, T. Langner, F. Meyer auf der Heide, P. Pietrzyk, and R. Wattenhofer, A tight runtime bound for synchronous gathering of autonomous robots with limited visibility, In Proceedings of 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA, 139-148, 2011.
4. P Flocchini, G Prencipe, N Santoro, Distributed Computing by Oblivious Mobile Robots, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool, 2012.
5. P.Flocchini , N.Santoro , G.Viglietta , M.Yamashita, Rendezvous with Constant Memory, Theoretical Computer Science, 621, 57-72, 2016.
6. A. Heriban, X. Defago, S. Tixeuil, Optimally gathering two robots, Research Report HAL Id: hal-01575451, UPMC Sorbonne Universites, Aug. 2017.
7. T. Izumi, S. Souissi, Y. Katayama, N. Inuzuka, X. Defago, K. Wada, and M.

Yamashita, The gathering problem for two oblivious robots with unreliable compasses, SIAM Journal on Computing, 41(1):26-46, 2012.
8. T. Okumura, K. Wada and Y. Katayama, Optimal asynchronous rendezvous for mobile robots with lights, Technical Report arXiv: 1707. 04449v1, July 2017.
9. G. Prencipe, Impossibility of gathering by a set of autonomous mobile robots, The oretical Computer Science, 384(2-3):222{231, 2007.
10. I. Suzuki and M. Yamashita, Distributed anonymous mobile robots: Formation of geometric patterns, SIAM Journal on Computing, 28, 1347{1363, 1999.
11. G. Viglietta, Rendezvous of two robots with visible bits, Technical Report arXiv:1211.6039, 2012.

群知能に基づく深層学習アルゴリズムの検討
A Deep Learning Algorithm based on Swarm Intelligence

○ 田村 康将　　Xavier Défago

東京工業大学 情報理工学院 情報工学系
Email: {tamura, defago}@c.titech.ac.jp
Tel.: 03-5734-2772

ハチやアリといった社会性昆虫は，それぞれ個体としての振る舞いは非常に制限的であるものの，多数の個体が形成する群れ全体の振る舞いとして，目的とされる複雑なタスクを実行する．群知能（Swarm Intelligence）は，こうした集団的振る舞いに着目した人工知能（Artificial Intelligence）技術であり，粒子群最適化 [1] や蟻コロニー最適化 [2] といった連続・離散最適化手法，複数ロボットによる協調動作 [3] など，主に確率的・適応的要素を含む分散アルゴリズムならびに分散システムの一方法論として研究が進められている．

本研究は，多岐にわたる群知能研究の中でも特に集団意思決定（Collective Decision Making）の方法論に着目し，これを応用した協調型深層学習（Collaborative Deep Learning）アルゴリズムを議論するものである．集団意思決定は，多数の仮想エージェントあるいはロボットが環境などについて情報を収集し，意思決定アルゴリズムを介して集団としてコンセンサスを得る方法論である．自然界における例としては，ミツバチが環境の探索および Waggle Dance と呼ばれる独特のコミュニケーション手段を用い，新たな営巣地についてコンセンサスをとる行動がよく知られている．集団意思決定は，生命理解ならびに工学的な応用可能性の観点から，群ロボットの制御則として応用する研究も行われている [4]．一方で，協調学習の分野では，特に教師なし学習の一手法である強化学習（Reinforcement Learning）を用いた群ロボットの協調行動について近年数多くの研究が行われている [5]．こうした方法論の多くは，機械学習分野において過学習の抑制を主な目的として用いられてきた集団学習（Ensemble Learning）[6] とは異なり，各ロボットの行動学習時の内部情報を共有・学習に利用する仕組みを構築することで，効率的あるいは適応的な群ロボットの行動生成に焦点を当てている．

本発表は，以上に示した集団意思決定アルゴリズムならびに協調深層学習の方法論を整理し，特にデータ解析などに用いられる教師つき深層学習をターゲットとした協調的深層学習の方法論について，そのキーアイデアを議論する．

謝辞

本研究は JSPS 科研費 若手 B 17K12734 の助成を受けたものである．

参考文献

[1] Kennedy, J., and Eberhart, R. C., “Particle Swarm Optimization”, Proceedings of IEEE International Conference on Neural Networks, pp. 1942-1948, 1995.

[2] Dorigo, M., “Optimization, Learning and Natural Algorithms”, Ph.D. thesis, Politecnico di Milano, Italy, 1992.

[3] Yasuda, T., Nakatani, S., Adachi, A., and Ohkura, K., “Generating Flocking Behaviour of a Real Robotic Swarm that Travels between Two Landmarks”, Proceedings of the First International Symposium on Swarm Behaviour and Bio-Inspired Robotics, pp. 235-238, 2015.

[4] Valentini, G., Hamann, H., and Dorigo, M., “Efficient Decision-Making in a Self-Organizing Robot Swarm: On the Speed Versus Accuracy Trade-off”, Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, pp. 1305-1314, 2015.

[5] Forester, J., Assael, Y., de Freitas, M., and Whiteson, S., “Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks”, arXiv:1602.02672v1[cs.AI], 2016.

[6] Ho, T. K., “Random Decision Forest”, Proceedings of the 3rd International Conference on Document Analysis and Recognition, pp. 278-282, 1995.

Collective learning for swarm robots

Jérôme Bailet, Yasumasa Tamura and Xavier Défago

東京工業大学 情報理工学院

Department of Computer Science, School of Computing
Tokyo Institute of Technology, Tokyo, Japan
Email: {bailet.j, tamura, defago}@coord.c.titech.ac.jp

1. Introduction

Swarm robotics is an active research area in which one takes inspiration from social animals, with the aim to coordinate a large amount of autonomous robots and achieve a common goal [1]. The high redundancy and parallelization in the system brings a great potential for solving real world applications such as a rescue operation during a disaster. Redundancy due to the large population size of the swarm provides robustness. Even if few robots are faulty because of a crash or a program failure, the swarm can still work towards the objectives. Parallelization, also due to the large size of the population, allows to perform tasks involving multiple targets distributed in a vast range in the environment. The sensor coverage of the area is expended and each robot can patrol in an area. Therefore, swarm of robots could be deployed by first responder during a natural disaster, cover the area and explore unreachable paths in a potential dangerous environment for human.

However, a relevant question to point out is how the swarm of robots could operate in an unknown environment where unexpected events could occur. Most studies have been realized in confined laboratory settings with careful design that consist of decomposing the global behavior of the swarm into individual behaviors, then encoding it within the robot controller. It is referred as the “divide and conquer” approach [2], and this design method can be quite complex as it requires developing manually through a trial-and-error process until the resulting collective behavior is obtained.

2. Problem Statement

To remedy the problem, we concentrate on the capability of learning of the swarm robotic system, in a collaborative fashion, so that the swarm adapts over time in order to maximize its utility. A single robot improving its skill to solve a task over time, for instance navigate in rough terrains, corresponds to the learning capability. In collaborative learning strategy, each robot also learns on its own, and in addition each robot learns by exchanging its experience to others individuals. Through knowledge transfer, robots pool their expertise with the aim to improve their ability to solve a task faster and more efficiently. In case we have one robot that increase its abilities at foraging resources, e.g. collecting raw materials, or rescuing the victims in disaster areas; it is highly beneficial that it shares its knowledge with the entire swarm, notably those who are the farthest from it and cannot be reach by direct communication.

3. Machine Learning Methods

To answer the problem, we are focusing on the alternative approach for swarm robotics: the automatic design method [3]. It aims at generating behaviors automatically by optimizing controller over time, and where swarm of robots learns how to solve a task. Automatic design methods can be divided in two sub-fields: reinforcement learning and evolutionary robotics [3, 4].

Reinforcement learning is defined as learning a behavior through interactions with an environment and by receiving positive and negative feedback for taking actions. The goal of the robot is to learn an optimal policy, which is the optimal behavior mapping robot’s states to actions.

Evolutionary Robotics is a method that applies evolutionary computation techniques to robotic systems. It is inspired by the Darwinian models of natural selection and evolution, which consist of representing solutions as genome, and mutating, changing individuals to maximize a fitness function. The fitness function measure how fit a given solution is, that means, it evaluates the performance at solving a task.

The above methods can be both applied into robots of swarm robotics system without the need of collaboration. Each robot learns and optimizes independently its controller to solve a problem. In addition, the automatic methods have been traditionally implemented in swarm robotics in a centralized manner. The performance is evaluated on the swarm-level, for the entire group, instead of the individual-level, and the algorithm coordinates the behavior of all the robots. This approach suffers from scalability problem, and we are focusing on collaborative learning in a distributed way.

4. Collaborative Learning

Among the studies of evolutionary algorithm in swarm robotics systems, a completely decentralized and on-line approach belongs to embodied evolution [5]. In embodied evolution an evolutionary algorithm is executed in a distributed way while the robots are already deployed in the environment. Robots exchange genetic materials when meeting, and selection and variation are performed locally by the robot.

Even though essential approaches and methodologies has been stated, unfortunately, the literature in collaborative learning for swarm robotics appears to be scattered and methods are not properly assessed against a well-established state of the art. We are going to discuss the challenges to be overcome in order to establish proper empirical practice [6].

The following application: “swarm robots for fire disaster search and rescue” is going to be the use-case for the research of collaborative learning in swarm robotics systems. It takes the concrete example of a fire disaster scenario that happened in the city for various reasons: explosion in a factory, fire in a building or because of a natural disaster. Swarm of robots are deployed in the environment to react quickly and largely before the arrival of human rescue teams. We are focusing on the problem of how the swarm of robots could learn and collaborate to rescue people more efficiently. As for the approach, a deeper look into evolutionary robotics and reinforcement learning techniques is undertaken, and by taking advantage of the distributed scheme of the system.

References

[1] Tan, Y., & Zheng, Z. Y. (2013). Research advance in swarm robotics. *Defence Technology*, *9*(1), 18-39.
[2] Trianni, V. (2008). *Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots* (Vol. 108). Springer.
[3] Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1-41.
[4] Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, *11*(3), 387-434.
[5] Watson, R. A., Ficiei, S. G., & Pollack, J. B. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* (Vol. 1, pp. 335-342). IEEE.
[6] Francesca G and Birattari M (2016) Automatic Design of Robot Swarms: Achievements and Challenges. *Front. Robot. AI* 3:29.

A firefly optimization for a connected dominating set in WSNs

Yuta Matsumoto Akihiro Fujiwara
Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502, Japan
Email: n232075y@mail.kyutech.jp, fujiwara@cse.kyutech.ac.jp

***Abstract*—In the sensor network, a set of connected sensors that dominates all other sensors is called the connected dominating set. The minimum connected dominating set is a computationally difficult problem proved to be NP hard, and a number of approximation algorithms have been proposed.**

In the present paper, we propose an approximation algorithm for the minimum connected dominating set using firefly optimization, which is an optimization technique based on behaviors of fireflies. The experimental results show that the proposed algorithm achieves a smaller number of sensors than the existing algorithm.

I. Introduction

The sensor network is a wireless communication network, which is composed of small autonomous sensors. Each sensor can communicate with the other sensors if the sensor is in the communication area. In the sensor network, a set of sensors is called *connected* if each sensor in the set can communicate with any sensor in the set using the other sensors as relays. In addition, a subset of sensors D is called *a dominating set* if every sensor not in D is adjacent to at least one sensor in D. The connected dominating set plays important role for collecting information in the sensor network.

Since the problem of finding the minimum connected dominating set is a computationally hard problem, which is proved to be NP hard [1], a number of approximation algorithms [1], [2], [3], [4] have been proposed for the minimum dominating set. For example, an approximation algorithm using bee colony optimization has been proposed in [4]. The bee colony optimization is an optimization technique based on the habit of honeybees.

In the present paper, we propose an approximation algorithm for the minimum connected dominating set using firefly optimization. The firefly optimization is an optimization technique based on behaviors of fireflies such that flashes of fireflies act as signals to attract other fireflies.

We implement our proposed algorithm and an existing algorithm [4] in simulation environment, and evaluate validity of the proposed algorithm. The experimental result shows the our proposed algorithm achieves a smaller number of sensors than the existing algorithm.

II. Preliminaries

A. Sensor model

The sensor network $G = (V, E)$ is defined by a set of sensors $V = \{s_1, s_2, \cdots, s_n\}$, where n is the number of sensors, and a set of edges E, which denotes a set of communication links between the sensors.

Each sensor s_i has a circular communication area of radius r, and the sensor can communicate only with sensors in the communication area. In case that sensor s_i and sensor s_j are able to communicate each other, a bidirectional communication link $e_{i,j} \in E$ exists between s_i and s_j in the graph G, and the sensors s_i and s_j are called adjacent. We assume that direct communication of messages is possible between adjacent sensors without collision.

B. Connected dominating set

In the sensor network $G = (V, E)$, a subset $D \subseteq V$ is called *connected* if any sensor $s_i \in D$ has a path to all of the other sensors in D. The connectivity of the sensors is needed to communicate data in the sensor network. In addition, a subset of sensors D is called *a dominating set* if every sensor not in D is adjacent to at least one sensor in D.

Therefore, the connected dominating set, which is a subset of sensors that satisfies the above two conditions, is defined as follows.

Definition 1: (Connected dominating set) Given a sensor network $G = (V, E)$, the subset of sensors $D = \{s_{i_1}, s_{i_2}, \ldots, s_{i_m}\}$ $(D \subseteq V)$, which satisfies the following two condition, is called the connected dominating set (CDS) for the sensor network.

1) D is connected.
2) Every sensor not in D is adjacent to at least one sensor in D. □

Although the minimum CDS is known to be NP hard [1], a small number of sensors should be desired for CDS. The CDS is used for constructing virtual backbones for communication in the sensor network, and a fewer number of CDS reduces interference and energy consumption of the sensor network.

Figure 1 shows an example of the connected dominating set. A set of sensors $\{s_1, s_3, s_5\}$ is a dominating set because all of the other sensors are adjacent to one of sensors in the set. In addition, the set of sensors $D = \{s_1, s_2, s_3, s_4, s_5\}$ is connected by including a set of sensors $\{s_2, s_4\}$ as relays. Therefore, D is the connected dominating set of the sensor network.

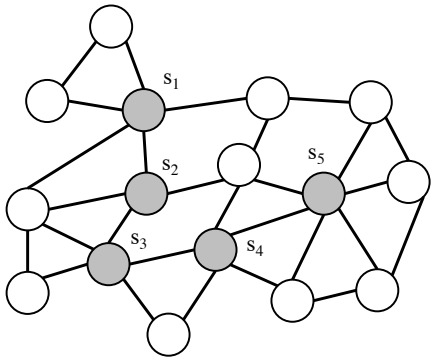


Fig. 1. Connected dominating set

III. An algorithm for the minimal CDS

In this section, we introduce a primitive algorithm for constructing a minimal CDS [5]. The algorithm obtain CDS by repeating deletion of sensors. Let $G = (V, E)$ be an input sensor network.

Algorithm 1: an algorithm for the minimal CDS

Repeat the following two step for each sensor s_i in V until no sensor is deleted in Step 2:

Step 1: Remove s_i and its connected edges from V and E, and create a sensor network $G' = (V', E')$ such that $V' = V - \{s_i\}$ and $E' = E - \{e_{i,k} \mid 1 \leq k \leq n\}$.

Step 2: Check whether sensor network $G' = (V', E')$ is a connected dominating set. If G' is a connected dominating set, set $V = V'$ and $E = E'$.

The obtained sensor network by the above algorithm is apparently minimal because of condition for terminating the repetition.

IV. An optimization algorithm for CDS

A. Firefly optimization

We first explain an outline of firefly optimization. The firefly optimization [6] is an optimization technique based on flashing behavior of fireflies. A firefly flashes to be selected by a partner among the other fireflies. Each firefly select a partner with the strongest intensity of light, and moves to the partner according to reliance on the light.

To formulate the above flashing behavior of fireflies to an optimization technique, the followings are assumed for the fireflies.

- All fireflies are uni-sexual, and each firefly is attracted to all of the other fireflies.
- Attractiveness of a firefly is proportional to intensity of light, and the intensity decreases according to distance between two fireflies.
- The intensity is associated with an objective function of a problem.

In this paper, we assume that $I_{i,j}$, which denotes the intensity of firefly f_j from firefly f_i, is given as follows.

$$I_{i,j} = \frac{I_j}{1 + \gamma r^2}$$

In the above expression, I_j is is an original light intensity of firefly f_j. In addition, γ is a parameter for light attenuation rate, and r is a distance between fireflies f_i and f_j.

In addition to the above, we assume a life span for fireflies. In case that a firefly reaches end of life, the firefly is initialized again with a new solution.

B. Firefly optimization for CDS

We now propose an approximation algorithm for the minimum connected dominating set using firefly optimization. In the algorithm, firefly optimization is used for reducing the number of sensors in CDS. We assume m fireflies, and f_i $(0 \leq i \leq m-1)$ denotes each firefly used for the algorithm.

An outline of the algorithm for CDS using firefly optimization is given below. The algorithm consists of three steps.

Algorithm 2: An algorithm for CDS using firefly optimization

Step 1: For each firefly f_i $(0 \leq i \leq m-1)$, compute an initial CDS using Algorithm 1 in Section III, and then, compute intensity of the obtained CDS, I_i, which is given below.

$$I_i = n - n_{CDS}$$

In the above expression, n is the number of all sensors, and n_{CDS} is the number of sensors in CDS.

Step 2: Repeat the following sub-steps by t times. (t is a given number of trials.)

(2-1): For each firefly that reaches end of life, compute a new CDS using Algorithm 1, and also compute intensity I_i for the computed CDS.

(2-2): For each firefly f_i $(0 \leq i \leq m-1)$, find a firefly f_j with the maximum intensity $I_{i,j}$ $(0 \leq j \leq m-1)$ that satisfies the following condition.

$$I_{i,j} = \max\{I_{i,k} \mid 0 \leq k \leq m-1\}$$

(2-3): For each firefly f_i $(0 \leq i \leq m-1)$, execute the following (2-3-1) and (2-3-2) in case of $I_i < I_{i,j}$.

(2-3-1): Let S_i and S_j be sets of sensors in CDSs with firefly f_i and f_j, respectively. Then, create a CDS such that a set of sensors is $S_i \cup S_j$. (Since S_i and S_j are sets of sensors in CDSs, an union $S_i \cup S_j$ also constructs CDS.)

(2-3-2): Compute a minimal CDS using Algorithm 1 for obtained CDS in (2-3-1), and also compute intensity I_i for the minimal CDS.

Step 3: Choose a CDS with the smallest number of sensors among CDSs in all fireflies.

V. Experimental results

Our proposed algorithm and an existing algorithm [4] are implemented using LEDA 6.3 [7] and C++, and we compare the numbers of sensors in CDSs and execution times between the two algorithms.

We assume the followings for the experiment.

- An input 2D area: 100×100 square area
- The number of sensors: 1000

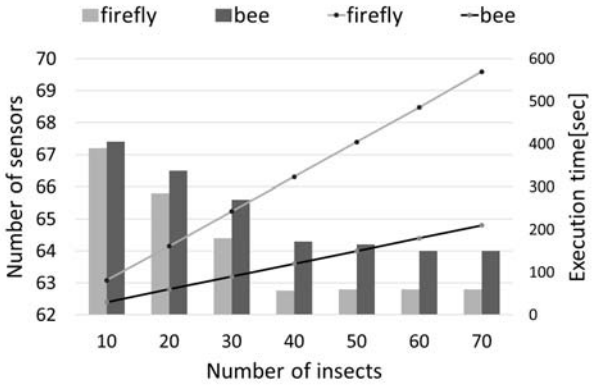


Fig. 2. An experimental result for CDS

- A communication radius of a sensor: 10
- The number of fireflies or bees: 10, 20, 30, 40, 50, 60, 70
- A number of trials t: 300
- A life span of fireflies: 7 to 14

Figure 2 shows a part of our experimental results. The number of sensors in the proposed algorithm is better than the existing algorithm. However, execution times of the proposed algorithm are about three times more than execution times of the existing algorithm. The reason of the execution time is that firefly optimization requires merging two solutions and deleting sensors to obtain the minimal CDS when firefly moves closer to another firefly. Since each bee discards its own solution and copies another solution to its own solution in the existing algorithm, the procedure of the existing algorithm is simpler than the procedure of the proposed algorithm.

VI. Conclusions

In this paper, we proposed an approximation algorithm for the minimum connected dominating set using firefly optimization. We compared our proposed algorithm and an existing algorithm in simulation environment, and showed validity of the proposed algorithm.

As our future work, we are considering speed-up of the proposed algorithm because the execution time of the proposed algorithm with firefly optimization is greater than the existing algorithm. We also consider an optimization algorithm for sensors with battery capacity.

Acknowledgments

This research was partially supported by JSPS KAKENHI, Grand-in-Aid for Scientific Research (C), 24500019.

References

[1] R. Jovanovic, M. Tuba, and D. Simian, "Ant colony optimization applied to minimum weight dominating set problem," in *Proceedings of 12th WSEAS International Conference on AUTOMATIC CONTROL, MODELLING and SIMULATION*, 2010, pp. 322–326.

[2] S. Guha and S. Khuller, "Approximation algorithms for connected dominating sets," *Computer Networks and ISDN Systems*, vol. 20, no. 4, pp. 374–378, 1998.

[3] R. Jovanovic and M. Tuba, "Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem," *Computer Science and Information Systems*, vol. 10, no. 1, pp. 133–149, 2013.

[4] H. Teshima and A. Fujiwara, "Bee colony optimization for minimum connected dominating set on gpgpu," in *3rd International symposium on Applied Engineering and Sciences, (SAES 2015)*, 2015.

[5] S. Kamei and H. Kakugawa, "A self-stabilizing distributed approximation algorithm for the minimum connected dominating set," *International Journal of Foundations of Computer Science*, vol. 21, no. 3, pp. 459–476, 2010.

[6] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[7] K. Mehlhom and S. Naeher, "Leda: A platform for combinatorial and geometric computing," 1999.

2017 年度　情報科学ワークショップ

通信効率のよい全域木構成自己安定アルゴリズムのトポロジ変化に対する出力安定性の実現

Output-stabilization and Communication-efficient for Self-stabilizing Protocols

北口峻行 * Toshiyuki Kitaguchi　角川裕次 † Hirotsugu Kakugawa　増澤利光 ‡ Toshimitsu Masuzawa

1　はじめに

複数の計算機がネットワークを介しながら，協動するシステムが**分散システム**である．近年，分散システムの大規模化が著しいが，大規模分散システムでは一部の計算機の故障やネットワークの形状変化は避けがたく，これらが生じてもシステム全体としてのサービスを継続できる，あるいは，外部から干渉なしに自律的にサービスを復旧できる**適応性**がより重要になってきている [13].

計算機の一時故障やネットワークの形状変化に対する高度な適応性を有する分散アルゴリズムとして，**自己安定アルゴリズム** [2, 13] が注目されている．自己安定アルゴリズムとは，任意の状況から実行を開始しても，いずれ所望の状況（**正当な状況**とよぶ）に到達することを保証する．このため，分散システムに一時故障や形状変化が生じても，外部からの干渉なしに自律的にサービスを復旧できる．

自己安定アルゴリズムはその高度な適応性を実現するため，システムが正当な状況であるかどうかを常に監視する必要がある．このため正当な状況においても，システム状況の監視のために一定のメッセージ交換と処理が必要であり，分散システムの運用コスト増大の一因となっている．[1] はこの問題点に着目し，自己安定アルゴリズムの正当な状況での通信量を削減を目的とした**通信効率**という概念を導入した．これまでに，いくつかの問題について，通信効率のよい自己安定アルゴリズムが提案されている [1, 8, 9]．特に，[9] は，本稿で扱う全域木構成問題に対して，通信効率のよい自己安定アルゴリズムを提案している．このアルゴリズムでは，全域木が構成された正当な状況では，ネットワーク状況の監視のために，各プロセスは全域木の親プロセスとのみ通信を行っている．また，この自己安定アルゴリズムが通信効率に関して最適であることも示している．

自己安定アルゴリズムは，分散システムに大規模な一時故障や形状変化が生じても，正当な状況に自律的に復帰できることを保証している．しかし，大規模な一時故障や形状変化が生じることはごく稀であり，頻繁に生じるのは小規模な一時故障や形状変化と考えられる．そのため，小規模な一時故障や形状変化に対するすぐれた特性を実現することが重要である．例えば，小規模な一時故障や形状変化に対する，自己安定アルゴリズムの効率的な適応性の実現を目的として，**故障封じ込め**という概念が導入され [3, 4]，故障封じ込め自己安定アルゴリズムについて多くの研究が行われている [5, 7, 10, 11, 12]．本稿では，ネットワーク形状の小規模な変化として，複数のリンク消滅を取り上げ，全域木リンク消滅からの復旧において，リンク消滅の影響を軽減することを目指す．具体的には，全域木リンク消滅からの復旧時の**出力安定**を実現することを目指す．

出力安定は，自己安定アルゴリズムの任意の状況からの復旧過程における外部への影響を抑制するために，外部からアクセスできる出力変数に着目し，その変動を抑制することを目的として導入された概念である [6]．自己安定アルゴリズムは任意の状況からの正当な状況への自律的な復旧を保証するが，この復旧過程においては何も保証しない．そのため，復旧過程においてシステム状況が頻繁に変化し，この自己安定アルゴリズムを利用する他の分散システムや外界に大きな影響を与える可能性がある．そのため，自己安定アルゴリズムの出力安定を実現することは，より安定な分散システムを実現するために重要であると考えられている．

本稿では，全域木構成問題を解く，通信効率のよい自己安定アルゴリズムに対し，リンク消滅に対する出力安定を実現する手法を提案する．提案する自己安定アルゴリズムは，増澤らのアルゴリズム [9] に加え，全域木が構成された正当な状況時には，各プ

* 大阪大学 Osaka University of Information Science and Technology
† 大阪大学 The same as the first author
‡ 大阪大学 The same as the first author

ロセスは全域木の親プロセスと通信を行うことに加え、リンクの消滅時にのみ親にリンクの消滅を報告できるものとしている。また，正当な状況で全域木のひとつのリンクが消滅したときには，各プロセスは親プロセスを高々 1 回変更するだけで全域木を再構築することで出力安定を実現している．このときの正当な状況への復帰に要する時間は，増澤らのアルゴリズム [9] と同様に $O(n)$ である．ここで，n は分散システムのプロセス数を表す．

本稿の構成は以下のとおりである．まず，2 節において，分散システムのモデルと自己安定アルゴリズムを定義する．3 節において提案する自己アルゴリズムを示し，4 節において正当性の証明と性能評価を示す．最後に 5 章において本研究をまとめる．

2 諸定義

2.1 システムモデル

分散システム $S = (P, L)$ は，プロセスの集合 P = $\{v_0, v_1, \ldots, v_{n-1}\}$ と双方向通信リンクの集合 L から構成される．各プロセスは，固有の ID を持つと仮定する．プロセス間にリンクが存在する場合，隣接しているといい，プロセス v と隣接するプロセス集合を N_v で表す．

各プロセス v は，1 原子動作として，(1) すべての隣接プロセスの状態を読み出し，(2) それらの状態と v の状態から v の次状態を決定し，(3)v の状態をその次状態に変更する．分散システムの状況は，n 項組 $\sigma = (s_0, s_1, \ldots, s_{n-1})$ で表される．ここで，s_i はプロセス v_i の状態を表す．

プロセス集合 Q に対し，プロセス集合 Q の動作により，状況が σ から σ' に遷移するとき，$\sigma \xrightarrow{Q} \sigma'$ と表す．

スケジュールは，空ではないプロセス集合の無限列 $\zeta = Q_1, Q_2, \ldots$ である．本稿では，スケジュールは弱公平と仮定する．つまり，すべてのプロセスが無限回現れるスケジュールのみを考える．スケジュールはプロセスが動作する順序を指名するものである．本稿では弱公平なスケジュールを考えるが，これはネットワークが非同期式であるが，すべてのプロセスが無限に動作し続ける（状態が変化し続けるとは限らない）ことを意味している．状況 σ_0 とスケジュール $\zeta = Q_1, Q_2, \ldots$ に対し，各 $i (i \geq 1)$ について，$\sigma_{i-1} \xrightarrow{Q_i} \sigma_i$ を満たす状況の無限列 $E = \sigma_0, \sigma_1, \sigma_2, \ldots$ を状況 σ_0 から始まるスケジューラらによる実行とよぶ．また，状況 σ_0 を実行 E の初期状況とよぶ．

時間複雑度を評価するために，実行 E に対し，非同期式ラウンドを導入する．スケジュール $\zeta = Q_1, Q_2, \ldots$ による実行 $E = \sigma_0, \sigma_1, \sigma_2, \ldots$ の最初のラウンドは，$P = \cup_{1 \leq j \leq k} Q_j$ を満たす，最小の接頭部実行 $\sigma_0, \sigma_1, \ldots, \sigma_k$ と定義する．実行 E の二番目以降のラウンドは，実行 $\sigma_k, \sigma_{k+1}, \sigma_{k+2}, \ldots$ に対して再帰的に定義する．

2.2 自己安定アルゴリズム

自己安定アルゴリズムは，任意の状況から実行を開始しても，いずれ所望の状況（正当な状況）に到達して安定する分散アルゴリズムである．本稿で扱う全域木構成問題の正当な状況は，次のように定義する．分散システムの 1 つのプロセスが根として指定される．各プロセス v は隣接プロセスの一つを親として保持するための出力変数 $prnt_v$ を持つ．ただし，根プロセスは親を持たないので $prnt_r = \perp$ とする．また，各プロセスは収束するまでの隣接の状況に応じて，全部で 5 状態の内いずれかの状態を保持している．全プロセスの $prnt_v$ が分散システムのある全域木の親を保持し，かつ全プロセスが状態 $legal$ を保持していれば，その状況は正当な状況である．

2.3 通信効率

各プロセス $v \in Q_i$ のスケジューラ $\zeta = Q_1, Q_2, \ldots$ による実行 $E = \sigma_0, \sigma_1, \sigma_2, \ldots$ に対し，σ_{i-1} においてプロセス v の状態遷移に影響を与える隣接プロセスの集合を $R_v^i(E)$ とし，$R_v(E) = R_v^1(E) \cup R_v^2(E) \cup \ldots$ とする．つまり、v は $R_v^i(E)$ に属さないプロセスの状態を読み取る必要はない。また、$v \notin Q_i$ のとき、$R_v^i(E) = \emptyset$ とする．

Definition 2.1 ($\diamond$-k-通信効率化) すべての実行 E に対し，$\sum_{i=0}^{n-1} |R_{v_i}(E')| \leq k$ となる E の接尾部分実行 E' が存在するとき，プロトコルが $\diamond$-k-通信効率であるといえる．

3 提案アルゴリズム

本節では，提案アルゴリズムを示す．

3.1 プロセスの変数

各プロセスは，以下の変数を持つ．

root:
根プロセスでのみ $true$ となる論理型変数. (各プロセスは自身が根プロセスかどうかを認識しており，値は常に正しい)

prnt:
親プロセスの ID．根プロセスあるいは，親プロセスが未定のプロセスは値 $\perp$ を持つ．

prnt_set:
先祖プロセスの ID の集合．

state:
状態 $legal$，状態 $floating$，状態 $correctT$，状態 pre_legal，状態 $any\_floating$ を持ち、正当な状況では状態 $legal$ を持つ．

consistent:
自身の状態が無矛盾なとき $true$ となる論理型変数.

論理型変数 $root$ は根プロセスは常に $true$，それ以外は常に $false$ を持つ．根以外の各プロセスは隣接プロセスの中から一つを親プロセスとして選び，その ID を変数 $prnt\_set$ に保持する．根プロセスは親プロセスを選ばないので，変数 $prnt$ に値 $\perp$ を保持する．各プロセスは，構成している木において，根からそのプロセスへの経路に現れるプロセスの ID の集合 (ただし，自身は除く) を変数 $prnt$ に保持する．全域木が構成されているときに，全域木のリンクが消滅すると，消滅リンクから全域木に沿って全てのプロセスが状態変化を行い $state$ を状態 $legal, , correctT$ までで管理する，また各プロセスは，自身の変数が無矛盾なら論理型変数 $consistent$ を $true$ にする．

3.2 正当な状況

提案アルゴリズムの正当な状況を定義する．

Definition 3.1 (正当な状況) 各プロセスが以下の条件を満たす状況は正当な状況である．

1. 根プロセス r
   - $root_r = true$
   - $prnt_r = \perp$
   - $prnt\_set_r = \{\phi\}$
   - $state_r = 1$
   - $consistent_r := true$
2. 根以外のプロセス v
   - $root_v = false$
   - $prnt_v \in N_v$
   - $prnt\_set_v = prnt\_set_{prnt_v} \cup \{prnt_v\}$ $\wedge\, v \notin prnt\_set_v$
   - $state_v = 1$
   - $consistent_v := true$

正当な状況では，全域木が構築され，各プロセスの変数 $prnt$ は，構成された全域木における親プロセスの ID が保持されている．定義 4 の正当な状況では，全域木が構成されていることが，親を順にたどっても閉路を構築しないことから証明できる．

3.3 通信効率のよい自己安定全域木構成のアイデア

提案アルゴリズムは，正当な状況に収束できる．任意の状況において，根プロセスに正しく接続された木を $correct\,tree$ とよぶ．$correct\,tree$ に含まれないプロセスが閉路に属していても，変数 $prnt\_set$ から，矛盾を検出できるため，$prnt$ を $\perp$ にする．その後も $correct\,tree$ に含まれないプロセス同士で，繰り返し親を更新することが発生しても，最終的に $correct\,tree$ に接続される．

正当な状況に復帰した後は，根プロセス以外の各プロセスは，親プロセスの変数 $prnt\_set$ を確認することで，構築した全域木を保持できる．よって通信効率もよいとゆえる．

3.4 プロセスの状態説明

提案アルゴリズムでは，任意の状況から正当な状況まで収束する際，また全域木のリンクが消滅した際に、全プロセスがそれぞれの状況に応じ状態変化を行う．それぞれの状態変化について，以下で説明する。

1. 正当な状況
   - 状態 $legal$:
     各プロセスが正当な状況下に置かれた状態
2. 全域木のリンク消滅時
   - 状態 $floating$:
     消滅リンクの子孫で再接続が必要なプロセスの状態，状態 $correctT$ のプロセスに対して再接続を行う．

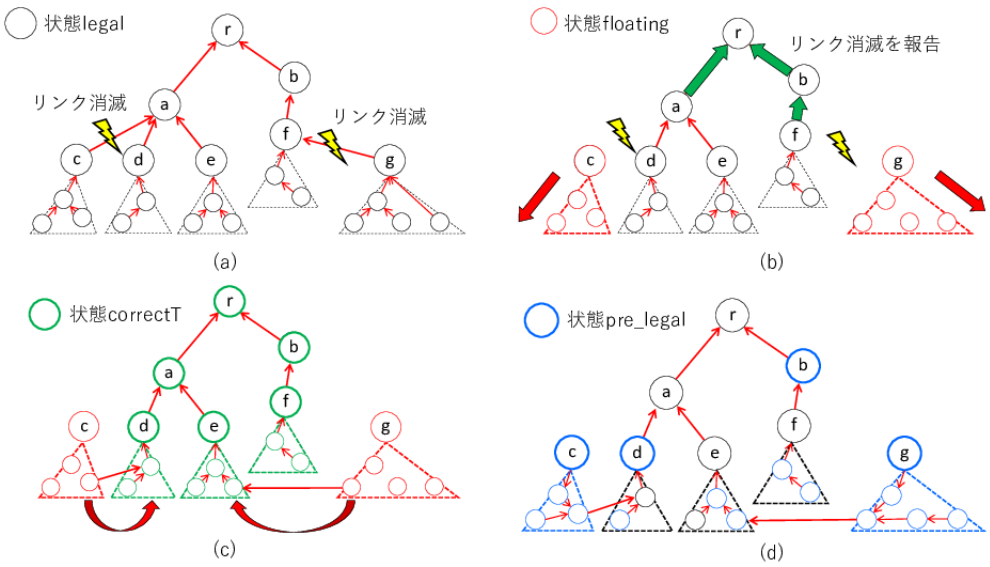


Figure 1: リンク消滅時

- 状態 $correctT$:
  消滅リンクの発生を検知した、$correct\ tree$ に属するノードの状態．隣接に状態 $floating$ がいないことが確認されると状態 $correctT$ に移行する．
- 状態 pre_legal :
  リンク消滅後、状態 $legal$ に戻るための準備．隣接に状態 $correctT$ がいないことが確認されると状態 $legal$ に移行する．

3. 任意の状況からの開始時

- 状態 $any\_floating$:
  $correct\_tree$ に属していない状態．状態変化を行い再接続を行う．

3.5 リンク消滅に対する処理

全域木が構築された正当な状況で，全域木のリンクの消滅が起こった状況を考える (Figure 1(a))．消滅リンクの子孫プロセスは根 r を含む $correct\ tree$ から分断されるため，全域木を再構築するために親の変更が必要になプロセスが存在する．このとき，親を適切に選ばないと，繰り返し親を変更しなければならない場合がある．そこで本アルゴリズムでは，そのような無駄な親の変更が発生しないように，$correct\ tree$ に属することが保証された隣接プロセスのみを親として選択する．

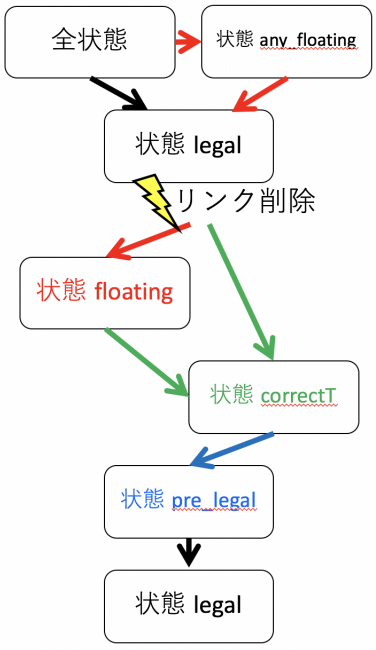


Figure 2: 状態変化

これは以下のように実現する．リンク (a,c)，(f,g) の消滅を子側のプロセス c, g が検知すると，変数 $state_c, state_g$ に $floating$ を格納し，状態変化させ

る．この状態を、c, g の子孫が確認することで状態 $floating$ を伝搬させる．また、リンクの消滅を親側の a, f が検知すると、親に報告を送り、順々に根までリンク消滅を伝搬させる (Figure 1(b)).
根にリンク消滅が伝搬すると，変数 $state_r$ に $correctT$ を格納し，状態変化させる．根の子が状態変化を確認することで状態変化を起こし、伝搬させる．このようにすることで再接続が必要な部分と，根に正しく接続されている部分を状態分けし，判定できるようにしている (Figure 1(c))．c および g の子孫は，$state$ が $correctT$ の隣接プロセスを親として選択することで，無駄な親の変更を回避する．親の候補が，複数存在する場合，根により近いプロセスを親とする．根への距離が等しいプロセスが存在する場合は ID の小さいプロセスを親とする．隣接プロセスに親の候補となるプロセスが存在しない場合は隣接プロセスが状態 $correctT$ に移行するまで，親を $\perp$ に設定し待機する．
状態 $floating$ の全プロセスが，状態 $correctT$ に状態変化すると，隣接に状態 $floating$ がいなくなったプロセスから順に，状態 pre_legal に状態変化を行う．状態 pre_legal を導入することで，リンク消滅前で状態変化がまだ始まっていない状態 $legal$ と状態変化を経た状態 $legal$ かどうかを判定することができるようにしている．また状態 pre_legal も，状態 $correctT$ のプロセスがいなくなったプロセスから状態 $legal$ に状態変化する．

3.6 擬似コード

提案アルゴリズムの疑似コードを Algorithm1(各プロセスが持つ ID，変数の説明，根プロセスの動作), Algorithm2,3(根以外のプロセスの動作) に示す．

4 アルゴリズムの正当性と性能

本節では，アルゴリズムの正当性を証明し，次にいくつかの評価尺度に対して，既存研究と比較する．

4.1 アルゴリズムの正当性

本節ではアルゴリズムの正当性を証明する．アルゴリズムから，次の補題は明らかに成立する．

Lemma 4.1 (閉包性) 正当な状況に到達すると，提案アルゴリズムはその状況から変化しない．

正当な状況にいずれ到達すること（収束性）を示すのに，状況 σ における $correct\,tree\,T_\sigma$ を導入する．

Algorithm 1 変数と根プロセス r の原始動作

```
1: /*プロセス v の変数（常に正しい値を保持）*/
2: ID_v : identifier ;
3: root_v : bool ;
4: state_v : legal or floating or correctT or pre_legal or any_floating ;
5: prnt_v : {⊥} ∪ identifier ;
6: prnt_set_v : ID の集合 ;
7: consistent_v : bool ;
8:
9: /* 根プロセス v(= r) の動作*/
10: prnt_r = ⊥ ;
11: ID_v = φ ;
12: consistent_v : true ;
13: if (bottomup from child) then
14:     state_r = correctT ;
15: if (∀i ∈ N_r | state_i = correctT or pre_legal) then
16:     state_r = pre_legal ;
17: if (∀i ∈ N_r | state_i = pre_legal or legal) then
18:     state_r = legal ;
```

Algorithm 2 根以外のプロセスの動作

```
1: /*根以外のプロセス v の動作*/
2: switch (consistent_v)
3: case true:
4:     if (prnt_v ∉ N_v) ∨ (v ∈ prnt_set_{prnt_v}) ∨ (prnt_set_v ≠ prnt_set_{prnt_v} ∪ prnt_v) then
5:         consistent_v = false ;
6:     if (bottomup from child) then
7:         send bottomup to prnt_v ;
8:     if (state_{prnt_v} = correctT) then
9:         state_v = correctT ;
10:    if (∀i ∈ N_v | state_i = correctT or pre_legal) then
11:        state_v = pre_legal ;
12:    if (∀i ∈ N_v | state_i = pre_legal or legal) then
13:        state_v = legal ;
14: end case
```

Definition 4.1 (*correct tree*) 提案アルゴリズムでの任意の状況を σ としたとき，σ において $correct\,tree\,T_\sigma$ を次のように定義する．

1. T_σ は根プロセス r を含み，$prnt_r = \perp$, $prnt\_set_r = \{\phi\}$, $consistent_r := true$, $state =$

Algorithm 3 根以外のプロセスの動作

15: **case** false:
16: **if** $(prnt_v \notin N_v \cup \{\perp\})$ **then**
17: $prnt_v = \perp$; $prnt\_set_v = \phi$; $state_v = floating$;
18: **else if** $(prnt_v \in N_v)$ **then**
19: **if** $(v \in prnt\_set_v)$ **then**
20: $prnt_v = \perp$;$prnt\_set_v = \phi$;$state_v = any\_floating$;
21: **else**
22: **if** $(prnt\_set_{prnt_v} \neq \phi) \vee (root_{prnt_v} = true)$ **then**
23: $prnt\_set_v \neq prnt\_set_{prnt_v} \cup prnt_v$; $state_v = state_{prnt_v}$; $consistent_v := true$;
24: **else**
25: **if** $state_{prnt_v} = floating$ **then**
26: $prnt_v = \perp$; $prnt\_set_v = \phi$; $state_v = floating$;
27: **else**
28: $prnt_v = \perp$; $prnt\_set_v = \phi$; $state_v = any\_floating$;
29: **else**
30: $prnt\_set_v = \phi$; $state_v = any\_floating$;
31: **if** $(state = floating)$ **then**
32: $N_{cand} = \{\exists u \in N_v \mid state_v = correctT\}$;
33: **if** $(N_{cand} \neq \phi)$ **then**
34: $u = \arg\min_{u \in N_{cand}}(\mid prnt\_set_u \mid \cup ID_u)$;
35: $prnt_v = ID_u$; $prnt\_set_v = prnt\_set_u \cup \{ID_u\}$; $consistent_v := true$; $state_v = correctT$;
36: **if** $(state = any\_floating)$ **then**
37: $N_{cand} = \{\exists u \in N_v \mid state_v = correctT\}$;
38: **if** $(N_{cand} \neq \phi)$ **then**
39: $u = \arg\min_{u \in N_{cand}}(\mid prnt\_set_u \mid \cup ID_u)$;
40: $prnt_v = ID_u$; $prnt\_set_v = prnt\_set_u \cup \{ID_u\}$; $consistent_v := true$; $state_v = state_{prnt_u}$;
41: **end case**
42: **end switch**

$legal$ を満たす．r の変数がこれらの条件を満たさないときは，T_σ を空グラフとする．

2. T_σ に含まれる根プロセス以外のプロセス v は，T_σ に含まれるプロセスの中から変数 $prnt$ を選び，$prnt\_set_v = prnt\_set_{prnt_v} \cup \{prnt_v\}$, $state = regal$
$prnt \in N_v$, $v \notin prnt\_set_v$, $consistent_v := true$
を満たす．

Lemma 4.2 (*correct tree* の安定性) 任意の状況 σ において，プロセス v が，*correct tree* T_σ に含まれるならば，σ 以降 v は状態を変更せず，σ 以降の任意の状況 σ' で v は T_σ に含まれる．

Proof 1 任意のプロセス v の論理変数 $consistent_v$ は，親に隣接しない場合，親の変数 $prnt\_set_{prnt_v}$ の値に矛盾がある場合に，*true* から *false* になる．*correct tree* に含まれるプロセスは，根から変数 $prnt$ と変数 $prnt\_set$ に無矛盾な値を格納し，*correct tree* を構成する．また構成中はリンクの消滅は行われないため $state$ も 1 のままである．よって *correct tree* に含まれるプロセス v の $consistent_v$ が *false* になることはない．

Lemma 4.3 (収束性) 任意の状況から実行を開始しても，提案アルゴリズムはいずれ正当な状況に収束する．

Proof 2 *correct tree* がいずれ全域木になることを証明すればいい．
背理法を使用し，*correct tree* が決してネットワークの全プロセスを含むことがないと仮定する．これは補題 *2* より，ある状況以降，全域木でない *correct tree* が変化しないことを意味する．この *correct tree* を T とし，T に含まれるプロセスの集合を P_T，T に含まれないプロセスの集合を $P_{\overline{T}}$ とする．

Table 1: 解析結果

評価尺度＼対象アルゴリズム	既存研究 [9]	提案アルゴリズム
収束ラウンド数	$O(n)$	$O(n)$
リンク消滅からの復帰ラウンド数	$O(n)$	$O(n)$
リンク消滅からの親の変更回数	$O(n^2)$	$O(n)$

これ以上変化のない *correct tree* T が構築された状況を σ とする．σ の後の任意の次状況 σ' を考え，σ' での $P_{\overline{T}}$ に含まれる全プロセスのうち空集合でない $prnt\_set$ の要素数 $|prnt\_set|$ が最も少ないものを $|prnt\_set|^{min}_{\sigma'}$ とする．

各プロセス $v \in P_{\overline{T}}$ は，根プロセスではないため，$prnt_v$ に隣接プロセスを選択するか，$\perp$ を設定する．また，$prnt\_set_v = \phi$ が成立する．$P_{\overline{T}}$ のプロセスは，*correct tree* に含まれないので，$prnt_v$ に隣接プロセス u を設定するときは，$u \in P_{\overline{T}}$ が成り立つ．このとき，$prnt\_set_v$ も更新し，$|prnt\_set_v| = |prnt\_set_u| + 1$ が成立する．このことから，$prnt\_set$ が空でないプロセスが存在する限り，各ラウンドで $|prnt\_set|^{min}_{\sigma}$ は少なくとも 1 増加する．従って，各プロセス $v \in P_{\overline{T}}$ で $prnt_set = \phi$ とならなければ，n ラウンド以内に $v \in prnt\_set_v$ となり，$prnt\_set_v = \phi$ となる．根以外のプロセスは，$prnt$ が空なら親として選択されないので，n ラウンド以内にすべての $v \in P_{\overline{T}}$ で $prnt\_set_v = \phi$ となる．このとき，P_T に隣接プロセスを持つ $P_{\overline{T}}$ のプロセス v が存在し，v は P_T のプロセスを親として選択し，*correct tree* に含まれる．これは，*correct tree* が T から変化しないことに矛盾する．

Theorem 4.1 提案アルゴリズムは，全域木構成問題に対する，◇-$(n-1)$ 通信効率でかつ，◇-*1* 安定の自己安定アルゴリズムである．

Proof 3 補題 *1* および *3* までより，提案アルゴリズムは，自己安定アルゴリズムである．また，根プロセスは *read action* を実行せず，正当な状況では根以外のプロセスは，親のみと通信を行う．従って，◇-$(n-1)$ 通信効率でかつ，◇-*1* 安定である．

4.2 アルゴリズムの性能評価

以下の評価尺度を用いて，提案アルゴリズムの性能評価を行う．

- 収束ラウンド数
  任意の初期状況から実行を開始したときに，正当な状況に到達するまでに要する最悪時のラウンド数
- リンク消滅からの復帰ラウンド数
  正当な状況で，全域木のリンクが消滅したときに，再び正当な状況に到達するまでに要する最悪時のラウンド数
- リンク消滅からの親の変更回数
  正当な状況で，全域木のリンクが消滅したときに，再び正当な状況に到達するまでに変数 $prnt$ の変更の総数．

[9] のアルゴリズムと提案アルゴリズムそれぞれの各評価尺度での評価結果を，表 1 にまとめる．

4.2.1 収束ラウンド数

任意の状況から正当な状況に復帰するまでの収束ラウンド数を解析する．
状況 σ で，*correct tree* に含まれないプロセスの中で一番小さい先祖集合の要素数を $|prnt\_set|^{min}_{\sigma}$ とする．*correct tree* に含まれないプロセスは，根プロセスと違い，必ず親を選択を選択する必要があるため，閉路を構成する．そのため，$|prnt\_set|^{min}_{\sigma}$ の値はインクリメントされ続け，開始から $2n$ ラウンド以内で $|prnt\_set|^{min}_{\sigma}$ が *correct tree* に含まれない全プロセス数を超えるため，各プロセスが *correct tree* に接続され始める．そこから構成に n ラウンドかかるので，収束するまでに $O(n)$ ラウンドで正当な状況までかかる．

4.2.2 リンク消滅からの復帰ラウンド数と親の変更回数

正当な状況において，全域木のリンクの消滅が起こった際に，再び正当な状況に復帰するのに必要なラウンド数と親の変更回数について解析を行う．

- 提案アルゴリズム
  全域木のリンク消滅時，そのリンクの子孫は状態遷移を行い，変数 $state = floating$ にする．また，リンク消滅を根プロセスに伝搬させ，$correct\_tree$ に属するプロセスの状態を $state = correctT$ にする．状態 $floating$ のプロセスは，隣接の状態 $correctT$ に対して接続し，なければ親を変更することなく隣接プロセスが $correct\,tree$ に含まれるのを待つ．

  部分木の全プロセスに状態遷移が伝搬するのに最大 $n-1$ ラウンドかかる．1 ラウンドで少なくとも 1 つのプロセスが新たに $correct\,tree$ に含まれる．従って，リンク消滅からの復帰ラウンド数は $O(n)$ である．次に親の変更回数について考える．リンク消滅のため $correct\,tree$ に含まれないプロセスは，$prnt$ を一度 $\perp$ に変更することはあるが，隣接プロセスに設定するときは，必ず $correct\,tree$ に含まれる．補題 2 より，$correct\,tree$ のプロセスは親を変更することはないため，リンク消滅からの親の変更回数は $O(n)$ である．

  Theorem 4.2 提案アルゴリズムのリンクからの復帰ラウンド数，親の変更回数はともに $O(n)$ である．

5 まとめ

本稿では，全域木構成が行われた $\diamond$-$(n-1)$ 通信効率化の通信効率のよい自己安定アルゴリズムを紹介し，そのアルゴリズムが所望される状況の際，全域木のリンクが消滅した場合に無駄な親の交換が起こらないような新しい改良を加えたアルゴリズムを提案した．また提案したアルゴリズムが所望する状況まで正しく収束する正当性と，親の交換回数の削減を証明した．

References

[1] S. Devismes, T. Masuzawa, and S. Tixeuil: “Communication efficiency in self-stabilizing silent protocols,” Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS), pp. 474–481, 2009.

[2] S. Dolev: “Self-stabilization,” MIT Press, 2000.

[3] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju; “Fault-containing self-stabilizing algorithms,” Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 45–54, 1996.

[4] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju; “Fault-containing self-stabilizing distributed protocols,” Distributed Computing, Vol. 20, Issue 1, pp 53–73, 2007.

[5] S. Ghosh and x. He, “Fault-containing self-stabilization using priority scheduling,”

[6] S. Kutten and T. Masuzawa: “Output stability versus time till output,” Proceedings of the 21st International Symposium on Distributed Computing (DISC), pp.343–357, 2007.

[7] S. Kutten and B. Patt-Shamir: “Stabilizing time adaptive protocols,”

[8] S. Kutten and D. Zinenko: “Low communication self-stabilization through randomization,” Proceedings of the 24th International Symposium on Distributed Computing (DISC), pp.465–479 (2010).

[9] T. Masuzawa, T. Izumi, Y. Katayama and K. Wada.: “Brief announcement: Communication efficient self-stabilizing protocols for spanning tree construction,” Proceedings of the 13th International COnference on Principles of Distributed Systems (OPODIS), pp. 219–224, 2009.

[10] Y. Yamauchi, S. Kamei, F. Ooshita, Y. Katayama, H. Kakugawa, and T. Masuzawa: “Timer-based composition of fault-containing self-stabilizing protocols,” Information Sciences, Vol. 180, No. 10, pp. 1802–1816, 2010.

[11] Y. Yamauchi, S. Kamei, F. Ooshita, Y. Katayama, H. Kakugawa, and T. Masuzawa: “Hierarchical composition of self-stabilizing protocols preserving the fault-containment property,” IEICE Transactions, Vol. 92-D, No. 3, pp.451–459, 2009.

[12] Y. Yamauchi, T. Masuzawa, and D. Bein: “Preserving the fault-containment of ring protocols executed on trees,” The Compututer Journal, Vol. 52, No. 4, pp.483–498, 2009.

[13] 増澤利光, 山下雅史: T. Masuzawa, T. Izumi, Y. Katayama and K. Wada.: “適応的分散アルゴリズム,” 共立出版, 2010.

2017 年度情報科学ワークショップ

リンクの追加・削除が連続的に生じる動的ネットワークにおけるGrundyノード彩色アルゴリズム

Grundy node coloring on the dynamic network where link addition/deletion occurs continuously

井本　宗一郎 *　　角川　裕次 *　　増澤　利光 *
Soichiro Imoto　　Hirotsugu Kakugawa　　Toshimitu Masuzawa

Abstract

モバイル・アドホック・ネットワークなどの動的ネットワークが普及しつつある．それにともない動的ネットワーク上で動作する分散システムの研究が注目を集めている. 本報告ではリンクの追加・削除が連続的に生じることによりトポロジーが変化する動的ネットワークにおいて,Grundy ノード彩色を実現する分散アルゴリズムを提案するこの分散アルゴリズムは, 初期状況から高々 n ステップ (n はノード数) で隣接ノードの色が異なるように, すべてのノードを彩色する．さらに，トポロジ変化が生じても，各ラウンドの終了時には，彩色された隣接ノードが必ず異なる色を持つことを保証する．さらに，トポロジ変化が $4\Delta - 1$ ラウンドの間生じなければ, Grundy 彩色を実現する．ここで Δ はノードの最大次数を表す.Grundy 彩色は色数の局所最小性を満たすノード彩色であり，少ない色数でノード彩色を実現できる．

1　はじめに

1.1　本研究の背景と成果

動的ネットワークとは, ノードの参加・離脱, リンクの追加･削除などによってネットワークの構造が変化するネットワークのことである. 無線接続されたスマートフォンやセンサなどのモバイル端末を含むモバイル・アドホック・ネットワーク (MANET) などの普及が進み，現在のネットワークはその多くを動的ネットワークとみなすことができる．

分散システムとは，ネットワークを通じて多数の計算機（以下，ノード）が互いに通信することにより協調動作するシステムのことである．またその分散システム上で動作するアルゴリズムを**分散アルゴリズム**という．これまでに, 静的ネットワークを対象とした分散アルゴリズムの研究は，さまざまな問題に対して数多く行われている（文献 [1],[2]）．それらのほとんどは，分散アルゴリズム実行中にトポロジ変化が生じうる動的ネットワーク上では正しく動作しない．そのため, 動的ネットワーク上で動作する分散アルゴリズムに対する必要性が近年, 高まっている．動的ネットワークにおいては，分散アルゴリズム実行中に生じるトポロジ変化に対応する（どのようなトポロジ変化が生じても問題の解を得る）必要があるため，分散アルゴリズムには新たな手法が要求される．また，分散アルゴリズムが対象とする問題についても，ネットワークのトポロジ変化を考慮した，新たな問題が提示されている．

*大阪大学大学院情報科学研究科,Graduate School of Information Science and Technology, Osaka University

これまでに，動的ネットワークのモデルもいくつか提案されている（文献 [3],[4],[5]）．最も代表的で一般的なモデルは TVG (Time-Varying Graph) と呼ばれるものである．このモデルでは，各時刻におけるトポロジを指定することで動的ネットワークを定義している．また，離散的な時刻を導入し，TGV をグラフの系列として表すことも多い．本稿では，この離散的な時刻を導入した，同期式動的ネットワークを対象とする．

ノード彩色とは, 隣接するノードが同じ色にならないように全ノードを彩色することである. ノード彩色はスケジューリングや局所排他的制御, 無線ネットワークにおけるノードへの周波数割り当てなどで応用される. 最小の色数でノード彩色を行う問題は NP-困難であることが知られており，なるべく少ない色でノード彩色を行うために，局所探索法などの手法が利用されている．

Grundy ノード彩色とは，なるべく少ない色でノード彩色を実現するための一つの方法である．Grundy ノード彩色では，色に全順序関係があることを想定し，各ノードにおいて, そのノードよりも小さいすべての色がそのノードの隣接ノードに割り当てられていることを満たすノード彩色である．つまり，Grundy ノード彩色は，色数の局所最小性を満たすノード彩色であり，少ない色数のノード彩色を期待できる．

既存研究として文献 [6] で Hedetniemi らは最大次数 Δ の静的なネットワークに対して Grundy 彩色を高々 n ステップで色数 $\Delta + 1$ で彩色を行う自己安定アルゴリズムを提案している. 自己安定アルゴリズムとは，どのような初期状況から実行を開始しても，いずれ問題の解を得て安定するという特長を持つ分散アルゴリズムである．自己安定アルゴリズムは，動的ネットワークにおいてどのようなトポロジ変化が生じても，十分に長い間, 新たなトポロジ変化が生じなければ，解を得て安定する．この特長から，自己安定アルゴリズムは動的ネットワークに適する分散アルゴリズムとして期待されている．しかし，本稿が対象とするような，トポロジ変化が連続して発生する動的ネットワークでは，自己安定アルゴリズムがいずれ解を求めて安定することは保証できない．また，十分に長い間, トポロジ変化が生じない場合は，自己安定アルゴリズムがいずれ解を求めて安定することは保証できるが，安定するまでの振る舞いについては何も保証できない．

本稿では, リンクの追加・削除が連続的に生じる動的ネットワークにおける Grundy ノード彩色に対して，以下の特長をもつ分散アルゴリズムを提案する．ただし，同期式ネットワークを仮定し，トポロジ変化としては高々 1 つのリンクの追加または削除だけが同時に生じるものと仮定している．

(1) 初期状況から高々n ステップですべてのノードが彩色される．
(2) トポロジ変化にかかわらず各ラウンド終了時には，彩色されたノードが同じ色のノードと隣接することはない．
(3) 十分に長い間，トポロジ変化が生じなければ，Grundy ノード彩色が達成される．
特長 (2) は，トポロジ変化が生じても，隣接ノードで色の衝突がないという安全な状況にすばやく復帰できることを意味している．また，特長 (3) は，安全な状況を保ちながら，徐々に色数の削減を行うことを意味している．

2 諸定義

2.1 動的ネットワーク

本稿では，通信リンク（以下，単にリンクとよぶ）で相互接続されたノード（計算機）からなるネットワークを考える．リンクは相異なるノードを接続し，ノード u, v がリンクで接続されているとき，ノード u, v は双方向全二重通信が可能である．V をノード集合，E をリンク集合とし，ネットワークを $G=(V,E)$，ノード数を $n=|V|$，リンク数を $m=|E|$ とする．ノード u, v を接続するリンクを (u, v)（または (v,u)）と表し，リンク (u, v) が存在するとき，ノード u と v は隣接しているという．またノード v の隣接ノードの集合を $N(v)$ とする．ネットワーク $G=(V,E)$ はノードを頂点，リンクを辺とする単純無向グラフ（自己ループも多重辺もないグラフ）と見なせるので，グラフに関する用語をネットワークに対しても用いる．本稿では連結ネットワークのみを扱う．

本稿では，ラウンドとよぶ時間間隔で同期している同期式ネットワークを考える．また，リンクの追加・削除が生じる動的ネットワークを扱う．リンクの追加・削除はラウンド間で発生し，ラウンド内では発生しないものとする．従って，ラウンド $t(t\geq 0)$ のネットワークを $G_t=(V_t,E_t)$ と表し，動的ネットワークをネットワーク系列 $G_0,G_1,G_2,\ldots$ として表す．ただし，2 つのラウンド間に発生するのは，高々 1 つのリンクの追加または削除のみと仮定する．つまり，各 $t(t\geq 0)$ について

$$|(E_{t+1}-E_t)\cup(E_t-E_{t+1})|\leq 1 \qquad (1)$$

が成り立つ．またノードの参加と離脱は考えないため，各 $t(t\geq 0)$ について

$$V_t=V \qquad (2)$$

とする．

2.2 計算モデル

ノードはメッセージを送受信できる状態機械としてモデル化する．各ノード u は，全順序集合から選ばれた大域的に一意な識別子 ID_u を持つ．各ノード u は各ラウンド $t(t\geq 0)$ で以下の動作を順に行う．
(1) ネットワーク G_t での隣接ノードへのメッセージ送信．
複数の隣接ノードにメッセージを送信できる．
(2) ネットワーク G_t での隣接ノードからのメッセージ受信．
ラウンド t に隣接ノードから u に送信されたメッセージをすべて受信する．
(3) 受信したメッセージに基づき，現状態を次状態に更新．

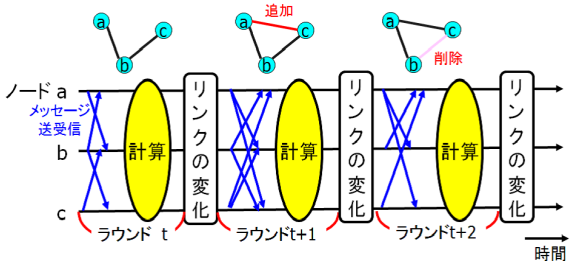


Figure 1: ラウンド動作

Figure??では，3 つのノード a,b,c で構成されるネットワークにおけるラウンド $t,t+1,t+2$ の動作例を示す．ネットワークトポロジの変化はこの図のように，ラウンドの間で生じる．この図では例として，ラウンド t と $t+1$ の間でリンク (a,c) が追加され，ラウンド $t+1$ と $t+2$ の間でリンク (b,c) が削除されている．もちろんリンクの変化ではリンクの追加も削除も生じないこともありえる．

2.3 ノード彩色

以下にノード彩色問題を定義する．各ノード v は，$color(v)$ という色を格納する出力変数を所持している．$color(v)$ は $\{\perp,1,2,\ldots,2\Delta\}$ から選んだ一つの値をノード v の色として格納する．ここで $\perp$ は未彩色であることを示すために導入する特別な記号である．ラウンド t においてノード v に与えられた色を $color_t(v)\in\{\perp,1,2,...,2\Delta\}$ と表現する．ここで，n はそのネットワークのノード数とする．ノード彩色とは隣接ノードと異なるように各ノードに $\perp$ 以外の色を割り当てることである．つまりノード彩色されたネットワークは以下の条件を満たす．

$$\forall v, color(v)\in 1,2,\ldots,2\Delta\wedge(\nexists u\in N(v), color(u)=color(v)) \quad (3)$$

またノードの初期状態として任意のノード v に対して $color(v)=\perp$ が与えられているとする．

未彩色なノードを含むネットワークでの彩色が以下の条件を満たすとき，彩色が無矛盾であるという．

$$\forall v,\forall t, color_t(v)\neq\perp\Rightarrow\nexists u\in N(v)(color_t(u)=color_t(v)) \quad (4)$$

2.4 Grundy ノード彩色

ノード彩色されたネットワーク（未彩色のノードを含まない）において，以下が成り立つとき Grundy 彩色されているという．

$$\forall v, color(v)=k(\neq\perp)\Rightarrow(\forall k'<k,\exists u\in N(v), color(u)=k') \quad (5)$$

つまり，Grundy 彩色されているならば，ノード v は隣接ノードと同じとならないような最小の色で彩色されている．また，ある色 $h(\neq\perp)$ について，h 以下の色で彩色されたすべてのノードで上の条件が成立するとき，色 h 以下について Grundy 彩色されているという．

3 提案手法

Grundy ノード彩色問題に対して，提案アルゴリズムを以下に示す．

3.1 アルゴリズムの概要

形状変化が生じない静的ネットワークであれば，未彩色ノードを順に，隣接ノードに彩色されていない最小の色で彩色すればGrundyノード彩色を実現できる．しかし，動的ネットワークでは,(Grundy）彩色された状況でも，リンクの追加・削除によって以下の問題が生じる．
(a) 同じ色で彩色されたノード間にリンクが追加されると彩色に矛盾が生じる．
(b) リンクが削除されると，彩色の矛盾は生じないがGrundy彩色ではなくなる．

そこで本稿では，次の3つの条件を満たす分散アルゴリズムを提案する．
(1) ラウンド終了時には，彩色は常に無矛盾である．つまり，ラウンド終了時には，$\perp$ 以外で彩色されたノードは同じ色の隣接ノードを持たない．
(2) すべてのノードはいずれ彩色される．つまり，各ノード v の出力変数は，ある時点以降 $\perp$ 以外の色を格納する．ただし，ネットワークの形状変化が継続的に発生するとノード彩色が変化し続けることもある．
(3) リンクの追加・削除が十分に長い間生じなければ，Grundy彩色を実現し彩色が変化しない．

提案アルゴリズムでは，各ラウンドにおいて，すべての隣接ノード間で (ID, 現在の色，次の候補色) の3項組を交換する．以降では，現在の色を現色，次の候補色を次色とよぶ．ここで，次色は現色より小さい色で，隣接ノードに使われていない最小の色とする．ただし，そのような色が存在しない場合は，次色を $\perp$ とする．そして，すべての隣接ノードから受信したこの3項組に基づいて，そのラウンドで色を変更するかどうか，また，変更する場合にはどの色に変更するかを決定する．その詳細は次節以降で説明する．

3.2 各ノードの定数，変数

各ノード v は，以下の定数，変数を持つ．

- ID_v: 自身の ID を格納する定数.
- $color_v$: 自身の現色を格納する変数．初期値は $\perp$.
- $nextc_v$: 自身の次色を格納する変数．初期値は $\perp$.
- MSG_v: そのラウンドで受信したメッセージ（上記の3項組）の集合.

3.3 各ノードの動作

ここでは, 各ラウンドにおける各ノードの動作を示す. すべてのノードは同じアルゴリズムを実行する.

1. 3項組（$ID_v, color_v, nextc_v$）をすべての隣接ノードに送信する.
2. すべての隣接ノード u から3項組 $(ID_u, color_u, nextc_u)$ を受信し，変数 MSG_v に格納する.
3. $color_v = \perp$ ならば，初期彩色法（後述）に従って $color_v$ を更新する．6. へ.
4. 変数 MSG_v に保持されたメッセージ集合に $color_v = color_u$ となる隣接ノード u が存在する場合，色衝突解消法（後述）に従って $color_v$ を変更する．6. へ.
5. $col = min\{c \mid c \neq color_u \ for\ any\ u \in N(v)\} < color_v$ なる色 col が存在する場合，色詰め法（後述）に従って，$color_v$ を変更する.
6. $col = min\{c \mid c \neq color_u \ for\ any\ u \in N(v)\} < color_v$ （上記 3 ,4 または 5 で $color_v$ が変更された場合，変更後の $color_v$) なる色 col が存在する場合，$nextc_v = col$ とする．このような色 col が存在しない場合 $nextc_v = \perp$ とする.

上記のステップ 3 は，未彩色のノードを初めて彩色するための動作である．隣接するノードが同じ色で彩色されないように，後述の初期彩色法に従って彩色する．ただし，隣接ノードと同時に同じ色に彩色されることを避ける必要があり，ノード v が現ラウンドで彩色されるとは限らない.

ステップ 4 は，ノード v とその隣接ノード u が同じ現色を持つ（色衝突とよぶ）場合の動作である．このような色衝突は現ラウンド開始時にリンク (v, u) が追加されたときのみに生じる．提案アルゴリズムでは，各ラウンド終了時にノード彩色が実現している（色衝突がない）ことを保証するために，後述の色解消法によってただちに色衝突を解消する．ただし，この色衝突解消法の結果はGrundyノード彩色になっているとは限らない.

ステップ 5 はノード v がGrundyノード彩色の条件を満たさない際の動作である．このような状況は現ラウンド開始時にノード v に接続するリンクが削除された場合，あるいは，前ラウンドで v もしくは v のある隣接ノードが現色を変更した場合にのみ生じる．提案アルゴリズムでは，Grundyノード彩色を実現するため，後述の色詰め法により現色を変更する．ただし，隣接ノードと同時に現色を同じ色に変更することを避ける必要があり，ノード v が現ラウンドでGrundyノード彩色の条件を満たすように現色を変更できるとは限らない.
隣接ノードが同時に現色を変更した結果, これらのノード間で色衝突が発生することを避けるために, 各ステップにおいて以下のルールを適用する.

初期彩色法

- 隣接ノードに自分よりもIDが小さく色 $\perp$ をもつノードがある（$\exists u \in N(v) \wedge color_u = \perp, ID_u < ID_v$）場合,$color_v$ は変更しない.

以下, $\nexists u \in N(v), color_u = \perp \wedge ID_u < ID_v$ を満たす場合の彩色法を記述する.

- 次色がどの隣接ノードの現色または次色とも異なる場合は $color_v = nextc_v$ とする.
- 次色が隣接ノード u の現色と同じ（$nextc_v = color_u \neq \perp$）場合
  - リンクの追加によって色衝突が発生したなら $color_v = nextc_v$
  - リンクの追加以外によって色衝突が発生したなら $color_v = max\{c \mid c \notin \{color_w, nextc_w\}\ for\ any\ w \in N(v)\}$ とする.

この初期彩色法によって, 未彩色ノードのうち,ID 最小のノードは必ず彩色される. したがって,n ラウンドですべてのノードが彩色される.

色衝突解消法

- 次色をもたない（$nextc_v = \perp$）場合
  - 色衝突している隣接ノード u が次色をもつ（$nextc_u \neq \perp$）なら 3. へ.
  - ノード u が次色をもたないとき $ID_v < ID_u$ なら 4. へ.$ID_u < ID_v$ なら 3. へ.
  - リンクの追加によって色 $\perp$ をもつノード u の次色と衝突した場合（$color_v = nextc_u$）は,4. へ.
- 次色をもつ（$nextc_v \neq \perp$）場合
  - 色衝突している隣接ノード u が次色をもたない（$nextc_u = \perp$）なら 1. へ.
  - ノード u が次色をもち, かつ, 次色が同じ（$nextc_v = nextc_u$）とき $ID_v < ID_u$ なら 1. へ.$ID_u < ID_v$ なら 3. へ.
  - ノード u が次色をもち, かつ, 次色が異なる（$nextc_v \neq nextc_u$）とき $ID_v < ID_u$ なら 1. へ.$ID_u < ID_v$ なら 2. へ.

1. 色詰め法（後述）に従って $color_v = nextc_v$ が可能なら行う. 不可能なら 4. へ.
2. 色詰め法（後述）に従って $color_v = nextc_v$ が可能なら行う. 不可能なら 3. へ.
3. ノード u が $color_u$ を変更し色衝突を解消するため, なにも変更しない.
4. $color_v = min\{c \mid c \neq color_w \vee nextc_w \; for \; any \; w \in N(v)\}$ とする.

色詰め法

- 次色が同じ隣接ノードが存在しない（$\nexists u \in N(v), nextc_v = nextc_u$）なら 2．へ.
- 次色が同じ隣接ノードが存在する（$\exists u \in N(v), nextc_v = nextc_u$）なら 2．へ.

1. 次色が同じどの隣接ノードよりも現色がそれ以下（$\forall u, color_v \leq color_u (u \in N(v) \wedge nextc_v = nextc_u)$ （ただし $\perp < 1$ とする)）なら 2. へ. そうでないなら 3．へ.
2. $color_v = nextc_v$
3. 隣接ノードが $netc_v$ と同じ色になる可能性があるため変更しない.

4 正当性の証明

以下の 3 つの定理に関して証明を行う. ただし，以下では，ラウンド t 終了時のノード v の出力変数 $color_v$ の値を $color_t(v)$ と表す.

定理 1 任意のノードに対して, ラウンド終了時にはリンクでつながっている隣接ノードと色が異なるように割り当てられている

- $\forall v, \forall t, color(v) \neq \perp \Rightarrow \nexists u \in N(v)[color_t(v) = color_t(u)]$

定理 2 ラウンド開始時に未彩色のノードがあれば, そのラウンドで少なくとも一つのノードが彩色される.

- $1 \leq t \leq n, \; |\{v| \; color_t(v) \neq \perp\}| \geq min\{n|\,, |\{v| \, color_{t-1}(v) \neq \perp\}| + 1\}$

定理 3 全ノードが彩色されているなら, リンクの追加・削除が最後に生じてから $4\Delta - 1$ ラウンド経てば,Grundy ノード彩色が実現している

4.1 定理 1 の証明

任意のラウンドにおいて, ラウンド終了時には隣接ノードが異なる色で彩色されていることを証明する.

まず, ノードの色変更によって色衝突が発生することがないことを示す. ノードが色を変える方法は 2 通りある.1 つは次色以外の色（次色が $\perp$ の場合も含む）に変更するもの. もう 1 つは次色に変更するものである.

前者の変更は 2 パターンある.1 つは新しいリンクの追加により隣接ノードと色が競合した際に, どちらか 1 方がその隣接ノードの現色と次色以外の最小の色に変更する場合. もう 1 つは未彩色のノードがリンクの追加によらず隣接ノードの現色と色衝突したときに MSG に含まれない最大の色になる時である（初期彩色法）.$\perp$ が現色のときに色を変更するのは現色が $\perp$ であるすべての隣接ノードの ID が自分よりも大きいときのみなので, この変更によって隣接ノードと色が衝突することはない.

後者のように次色に変更するのは隣接ノードの現色と, 次色が自分の次色と異なる場合のみである.

従って, このときも色変更によって色衝突が発生することはない. したがって隣接ノードの色が等しくなるのは同じ現色を持つノード間にリンクが追加された場合のみである. このとき, 必ず一方のノードが色変更を行うが, 色変更によって色衝突は発生しないので, 任意のラウンド終了時には隣接ノードは異なる色で彩色されている． □

4.2 定理 2 の証明

ノードが彩色されると（$color_v \neq \perp$ になると）, それ以降, 未彩色（$color_v \neq \perp$） になることはない. したがって, ラウンド n 終了時までにすべてのノードが彩色されることを示すには，未彩色ノード（現色が $\perp$ のノード）がある限り，各ラウンドで少なくとも 1 つの未彩色ノードが彩色されることを示せば十分である.

未彩色ノードを彩色できないケースは, 自分よりも ID が小さい未彩色の隣接ノードを待つ場合のみである. したがって未彩色ノードの内，ID が最小のノードは必ず彩色される. □

4.3 定理 3 の証明

全ノードの彩色が終了しているなら, リンクの追加・削除が最後に生じてから $2k-1$ ラウンド経てば, 色 k 以下のすべてのノードが Grundy ノード彩色の条件を満たしていることを証明する. これにより, 最大色が 2Δ であることから定理 3 が証明される.

??節よりリンクの追加・削除が行われても 1 ラウンドで必ず彩色は完了する. 従って，ノード彩色がされている（色の衝突がない）状況から $2k-2$ ラウンドの間，リンクの追加・削除が発生していないと考えてよい.

リンクの追加によってあるノード v の色が i から他の色に変更されたとする. これによって色詰めしなければならないのは i よりも大きい色をもつ v の隣接ノードであり,v が色の変更をしてから 2 ラウンド後に色を変更する. また $N(v)$ に属するあるノード u の色変更によって，さらにその隣接ノードで Grundy ノード彩色条件が破綻し，色詰めしなければならない場合がある．しかし，この場合も，色を変更するノードの現色は u の元の色よりも大きい色を持つノードのみである. リンクの削除が行われた場合は Grundy ノード彩色の条件を満たさないノードが高々 1 つ発生するだけである.

以上のことより, 彩色が完了されているなら, リンクの追加・削除が最後に生じてから $2k-1$ ラウンド経てば色 k 以下に関して色詰めされた状態である. □

5 おわりに

本稿では,Grundy ノード彩色に対する分散アルゴリズムを提案した．この分散アルゴリズムでは，彩色されていないリンクの追加･削除が連続的に生じる動的ネットワークに対して各ノードが隣接ノードの色をうまく考慮し変更することで Grundy ノード彩色を実現した．この分散アルゴリズムは，以下の長所を持つ.

1. すべてのノードは n ラウンド以内に彩色される
2. 各ラウンド終了時，どの隣接ノードも異なる色を持つ
   (リンク追加による色の衝突を 1 ラウンドで解消)
3. トポロジ変化が $4\Delta - 1$ ラウンドの間なければ,Grundy ノード彩色を実現

本稿では，同期式動的ネットワークにおいて，高々 1 つのリンクの追加または削除だけが同時に発生すると仮定した．しかし，提案アルゴリズムは，互いに独立したリンク（端点を共有しないリンク）であれば，複数リンクの追加や削除が同時に発生しても，同じ特長を持ちながら Grundy ノード彩色を実現できることを示せる．今後の課題は，互いに独立とは限らない複数リンクの追加または削除が同時に発生する場合にも対応できる分散アルゴリズムを開発することである.

謝辞

本研究は JSPS 科研費 JP26280022, JP16K00018, JP17K19977 の助成を受けたものです.

References

[1] Aspenes, J. and Shah, G. : *Skip graphs.* In 14th Symposium on Distributed Algorithms (SODA),Baltimore, MD, pp. 384393, 2003.

[2] Gallager, R. G., Humblet, P. A., Spira, P. M. : *A distributed algorithm for minimum-weight spanning trees.* ACM Transactions on Programming Languages and Systems 5 (1), 6677, 1983.

[3] Nicola,Santoro. : *Time to Change: On Distributed Computing in Dynamic Networks.* OPDIS, 2015.

[4] Fabian Kuhn. Rotem Oshman : *Dynamic Networks: Models and Algorithms.* ACM SIGACT News vol.42, no1, 2011.

[5] Giuseppe, Di, Luna1, Roberto, Baldoni. : *Non Trivial Computations in Anonymous Dynamic Networks.* OPDIS, 2015.

[6] Hedetniemi,S.T., Jacobs, D.P. and Srimani. P.K. : *Linear time Self-stabilizing colorings*. In-formation Processing Letters 87，pp.251-255，2006.

二人単貧民の必勝判定アルゴリズム

九州大学　　*　木谷　裕紀KIYA Hironori
名古屋大学　　小野　廣隆ONO Hirotaka

概 要

近年 AI 研究の対象になっている不完全情報ゲーム「大貧民」を完全情報ゲーム化したゲーム「単貧民」を定義し, グラフ理論の視点から解析を行った. 二人で行う単貧民の場合入力されたデータがソートされていれば, 線形時間で必勝戦略保持者が判定できることを示した.

1　はじめに

大貧民はトランプカードゲームの中でも全国的に認知度, 人気が高い遊びであり, 大富豪とも呼ばれる. このゲームは不完全情報多人数ゲームであり, 一般には最適戦略が存在しない, あるいは求めることが困難である. 近年, 将棋やオセロなどの完全情報ゲームに関する研究と共に, 大貧民のような不完全情報ゲームの研究も進んでいる. 2006 年度から毎年コンピューター大貧民大会が電気通信大学で開催され,「強い」大貧民 AI の開発を競う場となっている. 2015 年, プロ棋士に対する勝利宣言がだされた将棋コンテスト同様, 日々コンピューター大貧民 AI の実力も向上しているがまだまだ成長の余地がある.

このような大貧民研究の一環として電気通信大学の西野は単貧民というゲームを定義した [1]. 単貧民は大貧民に

- 特殊ルールが一切存在しない,
- 1 枚出しのみを認める,
- 手札は公開で行われる,

という制約を課したものであり, 大貧民を単純化し, かつ完全情報多人数ゲーム化したものと言える.

本研究では 2 人で行う単貧民について扱う. 二人単貧民は完全情報型の 2 人ゲームとなるため, いずれかのプレイヤーに必勝戦略が常に存在する. しかし, 将棋や囲碁などを考えると明らかなように, 完全情報型の 2 人ゲームにおける必勝戦略の存在性とそれを具体的に知ることには大きな隔たりがある. これに対し著者らはこれまで二人単貧民において最初に配られた手札が同一である場合等, 単純なケースにおいての具体的な必勝戦略を与える研究を行ってきた [2].

本研究では二人単貧民において, 与えられた手札の組からの必勝プレイヤー判定とその具体的な戦略を札の枚数 n に対してソート済みの手札であれば $\mathcal{O}(n)$ 時間で求めることができることを示す.

2　問題

2.1　単貧民のルール

まず二人単貧民を以下のようにモデル化する: 各プレイヤーの**手札集合**を $\{1, 2, \ldots, n\}$ の部分集合の形で与える (簡単のためそれぞれの手札集合は単純集合として説明するが, 多重集合であっても良い. また, 両プレイヤーが同じ値の札を持っていてもよい). プレイヤーに配布された札 (**手札**と呼ぶ) の札数は必ずしも等しくなくてよい. 札の番号は強さを表し, 大きいほど強いものとする. この設定の下, 以下の形でゲームを進める.

- 先手後手を決め, **先手プレイヤー**, **後手プレイヤー**の順に交代で, 手札から**場**に 1 枚ずつ札を出していく.
- 場は最初, 空である (0 の札がおかれていると考える).
- 順番のプレイヤーは, 手札の中から場の札の値よりも大きい値の札を 1 枚出すことができる. 出した札はそれまで出ていた札の上に置かれる (場に出ている札は今出した札に代わる). このとき, 順番はもう一人のプレイヤーに移る.
- 順番のプレイヤーは, 手札を出さずに順番をもう一人のプレイヤーに譲ることができる (**パス**する, という). このとき場に出ている札は空になる (改めて, 0 の札がおかれる).
- いずれかのプレイヤーの手札がなくなった時点で終了であり, このとき手札を 0 枚にしたほうが勝ちである.

ゲームを通して順番は交互に移るが, それぞれの手札を出すタイミングを**手番**, ある時点で順番が来ているプレイヤーを**手番プレイヤー**, もう一人のプレイヤーを**相手プレイヤー**と呼ぶ. また場が空の状態からいずれかのプレイヤーがパスをするまでを**巡**と呼ぶ.

以下ではこのゲームの必勝判定を考える.

2.2 諸定義

本研究ではいくつかの変数を新たに導入することによって線形時間で本必勝判定ができることを示した. そこでまずその諸変数を定義する. ゲームが進むことは手番が進むことを意味するので, 以下では手番を $t = 0, 1, 2, \ldots$ で参照する. A を先手プレイヤー, B を後手プレイヤーとする. つまり, A, B はそれぞれ $t = 0$ における手番プレイヤー, 相手プレイヤーである. プレイヤー A, B は互いに対称な関係にあるので, プレイヤーを表す変数 X に対して, 一方を X としたとき, 他方を $\bar{X}$ で参照する. すなわち, $X = A$ のとき $\bar{X} = B$ であり, $X = B$ のとき $\bar{X} = A$ である. 手番 t における X, $\bar{X}$ の手札を $X[t] \subseteq \{1, 2, \ldots, n\}$, $\bar{X}[t] \subseteq \{1, 2, \ldots, n\}$ で表す.

任意の手番 t に対して, 以下のような二部グラフを定義する:

$$G_X[t] = (X[t], \bar{X}[t], E_X[t]),$$

ただし,

$$E_X[t] = \{\{i, j\} \mid i \in X[t], j \in \bar{X}[t], i > j\}.$$

つまり, $G_X[t]$ は, プレイヤー X と $\bar{X}$ の手札の各札を点としたグラフであり, プレイヤー X の各手札から, その札が勝てるプレイヤー $\bar{X}$ の手札へ辺を引く形で定義されている.

このように定義したグラフ $G_X[t]$ においてマッチング辺は, プレイヤー $\bar{X}$ が出した札に対してプレイヤー X が札を出す関係を表しており, ゲームが進むことはそれぞれのグラフにおいてマッチング辺が抜かれていく状況を表している. 本研究で提案する判定法ではこれを利用するため, $G_X[t]$ の最大マッチングのサイズ $\mu_X[t]$ を定義する. この他に, 大貧民では比較的弱い, しかし使いようによっては順当に消費できる札を有効に切ることが重要になる. このような性質の札を定義するためいくつかの記号を導入する:

$$X^+[t] = \{v \in X[t] \mid d_{G_X[t]}(v) > 0\}.$$

ただし, $d_G(v)$ はグラフ G における点 v の次数を表す. これを用いて以下を定義する.

$$\begin{aligned} d^*(X^+[t]) &= \min\{d_{G_X[t]}(v) \mid v \in X^+[t]\}, \\ \min X^+[t] &= \{v \in X^+[t] \mid d_{G_X[t]}(v) = d^*(X^+[t])\}. \end{aligned}$$

以上を用いて次の値を定義する:

$$\nu_X[t] = d^*(X^+[t]) - |\min X^+[t]|.$$

つまり, $\nu_X[t]$ は $G_{X[t]} = \{V, E_a\}$ において孤立点でない頂点 $(X^+[t])$ のうち最も次数が小さいノードの次数 $(d^*(X^+[t]))$ からそのような頂点の数 $(|\min X^+[t]|)$

を引いたものを表す. また, マッチングがないときつまり $\mu_X[t]$=0 のとき $\nu_X[t] > 0$ とする. これらの値はいずれもそれぞれの手札がソートされていれば貪欲的な方法を用いることによって $\mathcal{O}(n)$ 時間で計算ができる. (二部マッチングアルゴリズム等を用いる必要はない). 本研究ではこれらの値を用いた必勝プレイヤー判定アルゴリズムを設計する.

2.3 主結果

定理 1. ある巡の開始時 t の手番プレイヤーを X とする. X は以下のいずれかを満たすとき, またそのときに限り, 必勝プレイヤーである.

- $\mu_X[t] > \mu_{\bar{X}}[t]$,
- $\mu_X[t] \geq \mu_{\bar{X}}[t]$, $\nu_X[t] > 0$,
- $\mu_X[t] \geq \mu_{\bar{X}}[t]$, $\nu_{\bar{X}}[t] = 0$,
- $\mu_X[t] + 1 = \mu_{\bar{X}}[t]$, $\nu_X[t] > 0$, $\nu_{\bar{X}}[t] = 0$.

この定理は，巡に関する帰納法により示すことができる. この定理から次が直ちに言える.

系 1. $\mu_A[0], \mu_B[0], \nu_A[0], \nu_B[0]$ を計算することにより, ソート後 $\mathcal{O}(n)$ 時間で二人単貧民の必勝プレイヤーとその必勝戦略を求めることができる.

表 1 ではこの勝利判定の条件をまとめている.

3 証明

以下ではこれを示すための補題を順に示す. まず, 基本的な補題を示す.

補題 1. 手番 t における手番プレイヤーを X とし, また $u < v$ とする. 場の札が v であるとき X 必勝→場の札が u であるとき X 必勝が成立. 場の札が u であるとき $\bar{X}$ 必勝→場の札が v であるとき $\bar{X}$ 必勝が成立

証明. 場の札が v であるとき X は戦略 y をとることによって X 必勝とする. このとき場の札が u であっても $u < v$ より先手は戦略 y をとることができる. 従って X 必勝同様に場の札が u であるとき $\bar{X}$ が戦略 z をとることによって $\bar{X}$ 必勝とするこのとき場の札が v であっても $u < v$ より先手は戦略 y をとることができる. 従って $\bar{X}$ 必勝. □

また、この補題より以下が直ちに成立

系 2. 場の札が v であるとき X 必勝→場が空であるとき X 必勝が成立. 場が空であるとき $\bar{X}$ 必勝→場の札が v であるとき $\bar{X}$ 必勝が成立

また以下の補題が成立

補題 2. $\mu_X[t] = 0$ であるとき, $\max(X[t]) \leq \min(\bar{X}[t])$.

証明. $\max(X[t]) > \min(\bar{X}[t])$ であるとき, グラフ $G_X[t]$ には辺が存在するため, $\mu_X[t] = 0$ とはなりえない. すなわち, $\mu_X[t] = 0$ ならば $\max(X[t]) \leq \min(\bar{X}[t])$ である. □

補題 3. ある空場時の手番 t における手番プレイヤーを X とする. このとき (i) $\mu_X[t] = |X[t]|$ なら X 必勝. また, (ii) $\mu_{\bar{X}}[t] = |\bar{X}[t]|$ かつ, $\nu_{\bar{X}}[t] \geq 1$ ならば $\bar{X}$ 必勝.

証明. $\mu_X[t]$, $\mu_{\bar{X}}[t]$ に関する帰納法により示す. $\mu_X[t] = 1$ のとき $\mu_X[t] = |X|$ とする. このとき手番プレイヤー X は 1 枚の手札を出すと X の手札がなくなるため, X 必勝, (i) が成立する. また $\mu_{\bar{X}}[t] = 1$ のとき, $\mu_{\bar{X}}[t] = \bar{X}[t]| = 1$ かつ $\nu_{\bar{X}}[t] \geq 1$ とする. すなわち $X[t] = \{w_X[t]\}$ である. このとき $|X[t]| \geq |N(w_X[t])| = \nu_X[t] + 1 \geq 2$ である. つまり, 先手プレイヤー X が $N(w_X[t])$ にある札を出したとしても X は手札を $|N(w_X[t])| - 1 \geq 1$ 枚以上の札をもっている，これに対し, $\bar{X}$ は 次の手順で $w_X[t]$ を出して手札をなくすことができるため, $\bar{X}$ の勝ちとなり, (ii) が成立する.

次に, ある $k > 0$ が存在して, $\mu_X \leq k$ ならば (i) が, $\mu_{\bar{X}} \leq k$ ならば (ii) が成立するとする. このとき,

	$\mu_X[t] > \mu_{\bar{X}}[t]$	$\mu_X[t] = \mu_{\bar{X}}[t]$	$\mu_X[t]+1 = \mu_{\bar{X}}[t]$	$\mu_X[t]+1 = \mu_{\bar{X}}[t]$
$\nu_X[t], \nu_{\bar{X}}[t] = 0$	X 必勝	X 必勝	$\bar{X}$ 必勝	$\bar{X}$ 必勝
$\nu_X[t] = 0, \nu_{\bar{X}}[t] > 0,$	X 必勝	$\bar{X}$ 必勝	$\bar{X}$ 必勝	$\bar{X}$ 必勝
$\nu_X[t] > 0, \nu_{\bar{X}}[t] = 0,$	X 必勝	X 必勝	X 必勝	$\bar{X}$ 必勝
$\nu_X[t], \nu_{\bar{X}}[t] > 0$	X 必勝	X 必勝	$\bar{X}$ 必勝	$\bar{X}$ 必勝

表 1: 勝者判定表

$\mu_X = k{+}1$ のとき, (i) が成立することを，$\mu_{\bar{X}} = k{+}1$ のとき, (ii) が成立することを示す.

まず前者から示す. $\mu_X[t] = |X[t]| = k+1$ のとき, 手番 t で X は最も弱い札 $(w_X[t])$ を出すことにする. このとき, $w_X[t]$ がなくなり $w_X[t]$ とマッチしていた $\bar{X}$ の頂点の存在から, 手番 $t+1$ では $\mu_X[t+1] = \mu_X[t]-1 = |X[t]]-1 = |X[t+1]| = k$, $\nu_X[t+1] \geq 1$ となる. このとき, $t+1$ の手番プレイヤーは $\bar{X}$ であるため, (ii) の条件から $\bar{X} = X$ 必勝となり, $k+1$ においては (i) が成立する.

後者は, $\mu_{\bar{X}}[t] = |\bar{X}[t]| = k+1$ のとき, 手番 t で相手プレイヤー X が出す札が $G_{\bar{X}}[t]$ の上詰めマッチング $M_{\bar{X}}[t]$ に現れない札であった場合, パスを選択することで上詰めマッチングは変化しない. よって引き続き, $\mu_{\bar{X}}[t+1] = |\bar{X}[t+1]| = \mu_{\bar{X}}[t] = |\bar{X}[t]|$ となる. 手番 t で相手プレイヤー X が出す札が $G_{\bar{X}}[t]$ の上詰めマッチングに現れている札である場合, $t+1$ 巡においてマッチングしている手を出すことによって t $+2$ 巡において場に札がある状態で $\mu_{\bar{X}}[t+2] = |\bar{X}[t+2]| = \mu_{\bar{X}}[t]-1 = |\bar{X}[t]|-1 = k$ かつ, $\nu_{\bar{X}}[t] \geq 1$ が成立している.

手番 t で相手プレイヤー X が出す札が $M_{\bar{X}}(w_{\bar{X}})[t]$ に現れる札である場合 $(w_X)[t]$ を出すことによって $\nu_{\bar{X}} \geq 1$ であるため, t+2 巡において, $G_{\bar{X}}[t+2]$ でも $\mu_{\bar{X}}[t+2] = |\bar{X}[t+2]| = \mu_{\bar{X}}[t]-1 = |\bar{X}[t]|-1 = k$ かつ, $\nu_{\bar{X}}[t] \geq 1$ が成立するため, 手番 $t+1$ の手番プレイヤー $\bar{X}[t]$ の必勝となり, (ii) が成立する. 以上より, 帰納法により任意の $\mu_X[t]$, $\mu_{\bar{X}}[t]$ に対して (i), (ii) が成立する. □

補題 4. ある空場時の手番 t における手番プレイヤーを X とする. このとき, (i) $\mu_X[t] > \mu_{\bar{X}}[t]$ ならば X 必勝. (ii) $\mu_X[t] < \mu_{\bar{X}}[t]-1$ ならば $\bar{X}$ 必勝.

証明. $\mu_X[t], \mu_{\bar{X}}[t]$ に関する帰納法を用いて示す. まず, $\mu_X[t] = 1$ のとき, (i) が, $\mu_{\bar{X}}[t] = 2$ のとき, (ii) が成立することをしめす. 前者に関しては, $\mu_X[t] = 1$ のとき $\mu_X[t] > \mu_{\bar{X}}[t]$ であるとすると, $\mu_{\bar{X}}[t] = 0$. すなわち, (手札が減る一方であることを考えると)$\bar{X}$ はこれ以降一枚も手札を出すことができないため, X 必勝である. 後者に関しては, $\mu_X[t] < \mu_{\bar{X}}[t]-1$ のとき, すなわち, $\mu_X[t] = 0, \mu_{\bar{X}}[t] = 2$ のとき, 補題 2 から X の手札は 2 枚以上あるにもかかわらずそのすべてが $\bar{X}$ の手札に対して勝つことができない札なので $\bar{X}$ 必勝である.

次に, ある $k > 0$ が存在して, $\mu_X \leq k$ ならば (i) が, $\mu_{\bar{X}} - 1 \leq k$ ならば (ii) が成立するとする. このとき, $\mu_X = k+1$ のとき, (i) が成立することを，$\mu_{\bar{X}} = k+1$ のとき, (ii) が成立することを示す.

まず前者から示す. $\mu_X[t] = |X[t]| = k+1$ のとき, $\mu_X[t] = |X[t]|$ のときは補題 1 より X 必勝. したがって以下では手番 t で X が $w_X[t]$ より弱い札を保有しているときを考える. 手番 t において X は $\mu_X[t]{+}1$ 番目に強い札を出す. このとき, X のマッチング構造は変わらないため $\mu_X[t] = \mu_X[t+1]$ である. ここで $\bar{X}$ がパスを選択した場合パスを選択しなくなるか $\mu_X[t{+}h] = |X[t{+}h]|$ となるまでこの操作を繰り返す. $\mu_X[t+h] = |X[t+h]|$ となれば X 必勝なのでパスを選択しないケースを考えるがこのとき $\bar{X}$ が出した札が上詰めマッチング $M_X[t]$ に現れない札であった場合パスを選択することで t+2 巡において $\bar{X}$ が空場で手番になるが $\mu_X[t+2] = \mu_X[t] > \mu_{\bar{X}}[t+2]+1$ となり仮定より X 必勝 $\bar{X}$ が出した札が上詰めマッチング

$M_X[t]$ に現れた札であった場合その度にマッチングしている札を出すことによって $M_X[t+2m+1]$（m は正の整数）巡において $\bar{X}$ はパスを選択するか上詰めマッチング $M_X[t]$ に現れない札を出すかどちらかであるが $\bar{X}$ がパスを選択した場合は $\mu_X[t+2m+2] = \mu_X[t] - m > \mu_{\bar{X}}[t] - m = \mu_{\bar{X}}[t+2m+2]$ で空場で X の手番になるので仮定より (i) は成立. 上詰めマッチング $M_X[t]$ に現れない札を出した場合もそれにパスすることによって $\mu_X[t+2m+3] = \mu_X[t] - m > \mu_{\bar{X}}[t] - m = \mu_{\bar{X}}[t+2m+3] + 1$ で $\bar{X}$ の手番となるので仮定より (i) は成立.

後者は, 手番 t において X が上詰めマッチング $M_{\bar{X}}[t]$ に現れる札を出した場合そのマッチングしている札を t +1 巡に出すことによってその札が相手にとってマッチングしていない札であれば $\mu_X[t+2] < \mu_{\bar{X}}[t+2]$ で $\bar{X}$ の手番になるので $\bar{X}$ 必勝マッチングしている札であれば $\mu_X[t] - 1 = \mu_X[t+2] < \mu_{\bar{X}}[t+2] - 1$ で場に札がある状態なので $\bar{X}$ 必勝したがって上詰めマッチング $M_{\bar{X}}[t]$ に現れる札を出した場合 (ii) が成立.

上詰めマッチング $M_{\bar{X}}[t]$ に現れない札を出した場合パスを選択することによって t +1 巡において手番はかわらず $\mu_{\bar{X}} = k+1$ の状態になるがこれは有限界しか起こらない. したがって (ii) は成立. 以上より, 帰納法により任意の $\mu_X[t], \mu_{\bar{X}}[t]$ に対して (i), (ii) が成立する. □

補題 5. ある空場時の手番 t における手番プレイヤーを X とする. このとき, (i) $\mu_X = \mu_{\bar{X}}$ 且つ $\nu_a, \nu_b = 0$ ならば X 必勝 (ii)$\mu_X + 1 = \mu_{\bar{X}}$ 且つ $\nu_a, \nu_b = 0$ ならば $\bar{X}$ 必勝

証明. $\mu_X[t], \mu_{\bar{X}}[t]$ に関する帰納法を用いて示す.
まず, $\mu_X[t] = 1$ のとき, (i) が, $\mu_{\bar{X}}[t] = 2$ のとき, (ii) が成立することをしめす. （$\mu_{\bar{X}}[t] = 1$ のときは $\nu > 0$ となるので (ii) は成立しない. ）前者に関しては, $\mu_X[t] = 1$ のとき $\mu_X[t] = \mu_{\bar{X}}[t]$ であるとすると, $\mu_{\bar{X}}[t] = 1$. 且つ $\nu_a, \nu_b = 0$ すなわち, (手札が減る一方であることを考えると)$\bar{X}$ はこれ以降 X の一枚を除いて一回も手札を出すことができないため, X はその札つまり一番弱い札を最後に出すことによって必勝である. 後者に関しては, $\mu_X[t] = \mu_{\bar{X}}[t]$ のとき, すなわち, $\mu_X[t] = 0, \mu_{\bar{X}}[t] = 2$ のとき, X の手札は 2 枚 $\bar{X}$ にマッチングされている手札を持つ. また $\bar{X}$ の手札のうち一枚しかマッチングしている手がない. X はマッチングされているいずれかの手を出したときに $\bar{X}$ は大きいほうから札を出すことによって全ての手札を出し切ることができるので $\bar{X}$ 必勝

次に, ある $k > 0$ が存在して, $\mu_X \leq k$ ならば (i) が, $\mu_{\bar{X}} \leq k$ ならば (ii) が成立するとする. このとき, $\mu_X = k+1$ のとき, (i) が成立することを, $\mu_{\bar{X}} = k+1$ のとき, (ii) が成立することを示す.

まず前者から示す. $\mu_X[t] = |X[t]| = k+1$ のとき, $\mu_X[t] = |X[t]|$ のときは補題 1 より X 必勝. したがって以下では手番 t で X が $w_X[t]$ より弱い札を保有しているときを考える. 手番 t において X は $\mu_X[t]+1$ 番目に強い札を出す. このとき, X のマッチング構造は変わらないため $\mu_X[t] = \mu_X[t+1]$ である. ここで $\bar{X}$ がパスを選択した場合パスを選択しなくなるか $\mu_X[t+h] = |X[t+h]|$ となるまでこの操作を繰り返す. $\mu_X[t+h] = |X[t+h]|$ となれば X 必勝なのでパスを選択しないケースを考えるがこのとき $\bar{X}$ が出した札が上詰めマッチング $M_X[t]$ に現れない札であった場合パスを選択することで t+2 巡において $\bar{X}$ が空場で手番になるが $\mu_X[t+2] = \mu_X[t] = \mu_{\bar{X}}[t+2]+1$ となり仮定より X 必勝 $\bar{X}$ が出した札が上詰めマッチング $M_X[t]$ に現れた札であった場合その度にマッチングしている札を出すことによって $M_X[t+2m+1]$（m は正の整数）巡において $\bar{X}$ はパスを選択するか上詰めマッチング $M_X[t]$ に現れない札を出すかどちらかであるが $\bar{X}$ がパスを選択した場合は $\mu_X[t+2m+2] = \mu_X[t] - m > \mu_{\bar{X}}[t] - m = \mu_{\bar{X}}[t+2m+2]$ で空場で X の手番になるので仮定より (i) は成立. 上詰めマッチング $M_X[t]$ に現れない札を出した場合もそれにパスすることによって $\mu_X[t+2m+3] = \mu_X[t] - m = \mu_{\bar{X}}[t] - m = \mu_{\bar{X}}[t+2m+3] + 1$
で $\bar{X}$ の手番となるので仮定より (i) は成立.

後者は, 手番 t において X が上詰めマッチング $M_{\bar{X}}[t]$ に現れる札を出した場合そのマッチングしている札を t +1 巡に出すことによってその札が相手

にとってマッチングしていない (=出せない) 札であれば $\mu_X[t+2]=\mu_{\bar{X}}[t+2]$ で $\bar{X}$ の手番になるので $\bar{X}$ 必勝マッチングしている札であれば $\mu_X[t]-1=\mu_X[t+2]=\mu_{\bar{X}}[t+2]-1$ で場に札がある状態なので系 2より $\bar{X}$ 必勝したがって上詰めマッチング $M_{\bar{X}}[t]$ に現れる札を出した場合 (ii) が成立.

上詰めマッチング $M_{\bar{X}}[t]$ に現れない札を出した場合パスを選択することによって t +1 巡において手番はかわらず $\mu_{\bar{X}}=k+1$ の状態になるがこれは有限界しか起こらない. したがって (ii) は成立. 以上より, 帰納法により任意の $\mu_X[t]$, $\mu_{\bar{X}}[t]$ に対して (i), (ii) が成立する. □

補題 6. ある空場時の手番 t における手番プレイヤーを X とする. このとき, (i) $\mu_X=\mu_{\bar{X}}$ 且つ $\nu_a,\nu_b>1$ ならば X 必勝 (ii)$\mu_X+1=\mu_{\bar{X}}$ 且つ $\nu_a,\nu_b>1$ ならば $\bar{X}$ 必勝

証明. $\mu_X[t],\mu_{\bar{X}}[t]$ に関する帰納法を用いて示す.
まず, $\mu_X[t]\leq 1$ のとき, (i) が, $\mu_{\bar{X}}[t]\leq 2$ のとき, (ii) が成立することをしめす. （$\mu_{\bar{X}}[t]=1$ のときは $\nu>0$ となるので (ii) は成立しない. ）前者に関しては, $\mu_X[t]=0$ のとき $\mu_X[t]=\mu_{\bar{X}}[t]$ であるとすると, $\mu_{\bar{X}}[t]=0$. したがって $\bar{X}$ はこれ以降 X の札に対して手札を出すことができないので X 必勝

$\mu_X[t]=1$ のとき $\mu_X[t]=\mu_{\bar{X}}[t]$ であるとすると, $\mu_{\bar{X}}[t]=1$. 且つ $\nu_a,\nu_b>0$ (手札が減る一方であることを考えると)$\bar{X}$ はこれ以降 X の札に対して一回だけ手札を出すことができるが, X は一番弱い札から出すことによってどのタイミングで $\bar{X}$ が札を出してもその後後手の手札の中には二枚以上先手の一番強い札よりも弱い札が存在する. したがって X 必勝である. 後者に関しては, $\mu_X[t]+1=\mu_{\bar{X}}[t]$ のとき, すなわち, $\mu_X[t]=0,\mu_{\bar{X}}[t]=1$ のとき, X の手札は 2 枚以上 $\bar{X}$ にマッチングされている手札を持つ. また $\bar{X}$ の手札のうち一枚も X がマッチングしている手がない. X はマッチングされているいずれかの手を出したときに $\bar{X}$ は大きいほうから札を出すことによって全ての手札を出し切ることができるので $\bar{X}$ 必勝

次に, ある $k>0$ が存在して, $\mu_X\leq k$ ならば (i) が, $\mu_{\bar{X}}\leq k$ ならば (ii) が成立するとする. このとき, $\mu_X=k+1$ のとき, (i) が成立することを, $\mu_{\bar{X}}=k+1$ のとき, (ii) が成立することを示す.

まず前者から示す. $\mu_X[t]=|X[t]|=k+1$ のとき, $\mu_X[t]=|X[t]|$ のときは補題 1 より X 必勝. したがって以下では手番 t で X が $w_X[t]$ より弱い札を保有しているときを考える. 手番 t において X は $\mu_X[t]$+1 番目に強い札を出す. このとき, X のマッチング構造は変わらないため $\mu_X[t]=\mu_X[t+1]$ である. ここで $\bar{X}$ がパスを選択した場合パスを選択しなくなるか $\mu_X[t+h]=|X[t+h]|$ となるまでこの操作を繰り返す. $\mu_X[t+h]=|X[t+h]|$ となれば X 必勝なのでパスを選択しないケースを考えるがこのとき $\bar{X}$ が出した札が上詰めマッチング $M_X[t]$ に現れない札であった場合パスを選択することで t+2 巡において $\bar{X}$ が空場で手番になるが $\mu_X[t+2]=\mu_X[t]=\mu_{\bar{X}}[t+2]+1$ となり仮定より X 必勝 $\bar{X}$ が出した札が上詰めマッチング $M_X[t]$ に現れた札であった場合その度にマッチングしている札を出すことによって $M_X[t+2m+1]$（m は正の整数）巡において $\bar{X}$ はパスを選択するか上詰めマッチング $M_X[t]$ に現れない札を出すかどちらかであるが $\bar{X}$ がパスを選択した場合は $\mu_X[t+2m+2]=\mu_X[t]-m>\mu_{\bar{X}}[t]-m=\mu_{\bar{X}}[t+2m+2]$ で空場で X の手番になるので仮定より (i) は成立. 上詰めマッチング $M_X[t]$ に現れない札を出した場合もそれにパスすることによって $\mu_X[t+2m+3]=\mu_X[t]-m=\mu_{\bar{X}}[t]-m=\mu_{\bar{X}}[t+2m+3]+1$

で $\bar{X}$ の手番となるので仮定より (i) は成立.

後者は, 手番 t において X が上詰めマッチング $M_{\bar{X}}[t]$ に現れる札を出した場合そのマッチングしている札を t +1 巡に出すことによってその札が相手にとってマッチングしていない (=出せない) 札であれば $\mu_X[t+2]=\mu_{\bar{X}}[t+2]$ で $\bar{X}$ の手番になるので $\bar{X}$ 必勝マッチングしている札であれば $\mu_X[t]-1=\mu_X[t+2]=\mu_{\bar{X}}[t+2]-1$ で場に札がある状態なので系 2より $\bar{X}$ 必勝したがって上詰めマッチング $M_{\bar{X}}[t]$ に現れる札を出した場合 (ii) が成立.

上詰めマッチング $M_{\bar{X}}[t]$ に現れない札を出した場合パスを選択することによって t +1 巡において手番はかわらず $\mu_{\bar{X}}=k+1$ の状態になるがこれは有

限界しか起こらない．したがって (ii) は成立．以上より, 帰納法により任意の $\mu_X[t]$, $\mu_{\bar{X}}[t]$ に対して (i), (ii) が成立する． □

補題 7. ある空場時の手番 t における手番プレイヤーを X とする．このとき, (i)$\mu_X[t]+1 \geq \mu_{\bar{X}}[t]$ 且つ $\nu_X[t] > 0, \nu_{\bar{X}}[t] = 0$ ならば X 必勝 (ii)$\mu_X[t] \leq \mu_{\bar{X}}[t]$ 且つ $\nu_X[t] = 0, \nu_{\bar{X}}[t] > 0$ ならば $\bar{X}$ 必勝

証明. $\mu_X[t], \mu_{\bar{X}}[t]$ に関する帰納法を用いて示す．
まず, $\mu_X[t] = 0$ のとき, (i) が, $\mu_{\bar{X}}[t] = 1$ のとき, (ii) が成立することをしめす．（$\mu_{\bar{X}}[t] = 0$ のときは $\nu_X > 0$ となるので (ii) は成立しない．）前者に関しては, $\mu_X[t] = 0$ のとき $\mu_X[t] = \mu_{\bar{X}}[t]$ であるとすると, $\mu_{\bar{X}}[t] = 1$. 且つ $\nu_a, \nu_b = 0$ すなわち, (手札が減る一方であることを考えると)$\bar{X}$ はこれ以降 X の一枚を除いて一回も手札を出すことができないため, X はその札つまり一番弱い札を最後に出すことによって必勝である．後者に関しては, $\mu_X[t] = \mu_{\bar{X}}[t]$ のとき, すなわち, $\mu_X[t] = 1, \mu_{\bar{X}}[t] = 1$ のとき, X の手札は 2 枚以上 $\bar{X}$ にマッチングされている手札を持つ．また $\bar{X}$ の手札のうち一枚しかマッチングしている手がない．X はマッチングされているいずれかの手を出したときに $\bar{X}$ は大きいほうから札を出すことによって全ての手札を出し切ることができるので $\bar{X}$ 必勝

次に, ある $k > 0$ が存在して, $\mu_X \leq k$ ならば (i) が, $\mu_{\bar{X}} \leq k$ ならば (ii) が成立するとする．このとき, $\mu_X = k+1$ のとき, (i) が成立することを，$\mu_{\bar{X}} = k+1$ のとき, (ii) が成立することを示す.

まず前者から示す．$\mu_X[t] = |X[t]| = k+1$ のとき, $\mu_X[t] = |X[t]|$ のときは補題 1 より X 必勝．したがって以下では手番 t で X が $w_X[t]$ より弱い札を保有しているときを考える．手番 t において X は $\mu_X[t]+1$ 番目に強い札を出す．このとき, X のマッチング構造は変わらないため $\mu_X[t] = \mu_X[t+1]$ である．ここで $\bar{X}$ がパスを選択した場合パスを選択しなくなるか $\mu_X[t+h] = |X[t+h]|$ となるまでこの操作を繰り返す．$\mu_X[t+h] = |X[t+h]|$ となれば X 必勝なのでパスを選択しないケースを考えるがこのとき $\bar{X}$ が出した札が上詰めマッチング $M_X[t]$ に現れない札であった場合パスを選択することで t+2 巡において $\bar{X}$ が空場で手番になるが $\mu_X[t+2] = \mu_X[t] = \mu_{\bar{X}}[t+2]$ となり仮定より X 必勝 $\bar{X}$ が出した札が上詰めマッチング $M_X[t]$ に現れた札であった場合その度にマッチングしている札を出すことによって $M_X[t+2m+1]$（m は正の整数）巡において $\bar{X}$ はパスを選択するか上詰めマッチング $M_X[t]$ に現れない札を出すかどちらかであるが $\bar{X}$ がパスを選択した場合は $\mu_X[t+2m+2]+1 = \mu_X[t] - m + 1 = \mu_{\bar{X}}[t] - m = \mu_{\bar{X}}[t+2m+2]$ で空場で X の手番になるので仮定より (i) は成立．上詰めマッチング $M_X[t]$ に現れない札を出した場合もそれにパスすることによって $\mu_X[t+2m+3]+1 = \mu_X[t] - m + 1 = \mu_{\bar{X}}[t] - m + 1 = \mu_{\bar{X}}[t+2m+3]$
で $\bar{X}$ の手番となるので仮定より (i) は成立.

後者は, 手番 t において X が上詰めマッチング $M_{\bar{X}}[t]$ に現れる札を出した場合そのマッチングしている札を t +1 巡に出すことによってその札が相手にとってマッチングしていない (=出せない) 札であれば $\mu_X[t+2] = \mu_{\bar{X}}[t+2]$ で $\bar{X}$ の手番になるので $\bar{X}$ 必勝マッチングしている札であれば $\mu_X[t] - 1 = \mu_X[t+2] = \mu_{\bar{X}}[t+2] - 1$ で場に札がある状態なので系 2 より $\bar{X}$ 必勝したがって上詰めマッチング $M_{\bar{X}}[t]$ に現れる札を出した場合 (ii) が成立.

上詰めマッチング $M_{\bar{X}}[t]$ に現れない札を出した場合パスを選択することによって t +1 巡において手番はかわらず $\mu_{\bar{X}} = k+1$ の状態になるがこれは有限界しか起こらない．したがって (ii) は成立．以上より, 帰納法により任意の $\mu_X[t]$, $\mu_{\bar{X}}[t]$ に対して (i), (ii) が成立する． □

これらの補題より定理を導くことが可能

4 まとめ

本研究では, 二人単貧民の必勝プレイヤー判定とその必勝戦略導出を扱った．単貧民自体は大貧民解析のための足がかりとして定義された簡易版であり, また真の困難性は多人数版にあるが, そういった, より一般化した設定でのアルゴリズム設計に, 本研究

で提案したアルゴリズムがサブルーチンとして有効に機能することを期待している.

参考文献

[1] 西野順二: 単貧民における多人数完全情報展開型ゲームの考察 (“An analysis on TANHINMIN game”), ゲームプログラミングワークショップ 2007 論文集, pp. 66 – 73, 2007.

[2] 木谷裕紀, 小野廣隆: 単貧民における必勝戦略と必勝判定問題に関する考察, 火の国情報シンポジウム 2016 論文集, 3B-3, 2016.

コード配色の変更を認めるマスターマインドの推測回数に関する考察

迫田 賢宜

Abstract. Mastermind is a board game played by two players: code-maker and code-breaker. The code-maker just sets a secret code at the beginning of the game, and the code-breaker makes a "guess" and receives the score of the guess one by one, until the guess matches the secret code. The most popular setting uses 4 pins of 6 colors for a secret code, and it is known that the code-breaker can identify any secret code via 5 guesses. In this paper, we consider an extended variant of mastermind, where the code-maker can change the color of a pin in the secret code at any timing. An obvious upper bound on the number of guesses to identify the changed secret code is 10. We investigate the number of guesses to identify the changed secret code.

1 はじめに

マスターマインドとはボードゲームの一種であり, 2 人のプレイヤーが出題者と解答者に分かれて行う. 最も一般的なマスターマインドは, 出題者は 6 色あるピンから 4 本を選び, 解答者に見えないように並べる, というものである. なお, このピンの配置をコードと呼び, コードに使用するピンの色の重複は認める. 解答者は, 出題者の作成したコードを推測し, コードを明らかにすることがこのゲームの目的である. 出題者は, 解答者の推測に対して次のルールで返答をする.

コードと推測を照らし合わせて

- 位置も色も正しいピンがあれば, 黒いピンを立てる.
- 色は正しいが位置が違うピンがあれば, 白いピンを立てる.

以下, 本論文ではコードに使用されるピンと返答に使用されるピンとを区別するために, 返答に使われる黒いピンを "ヒット", 白いピンを "ブロー" と呼ぶ. 2 人のプレイヤーは推測と返答を繰り返し, 解答者が 4 ヒットの返答を得る, つまりコードが判明した時点でゲームは終了する (図 1). ただし, 図 1 において返答にある黒丸がヒット, 白丸がブロー, そして斜線入りの丸はヒットもブローも得られていない状態を表しており, 推測に用いられたピンと返答のヒットやブローなどの位置は無関係である.

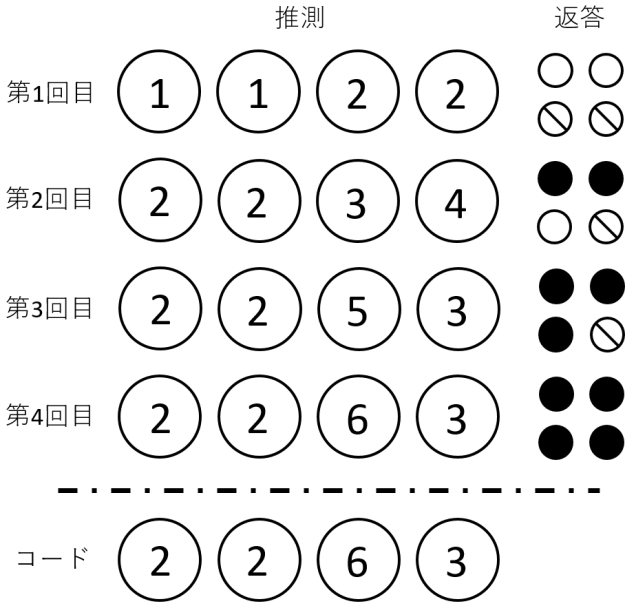


図 1　ゲーム進行の例

マスターマインドは, ボードゲームとして 1970 年代にイギリスで発売され, その後アメリカや日本でも発売された. ボードゲームとして特別な道具を使わずとも, ピンの色と数字を対応させることで数当てゲームとして楽しむことができる.

6 色 4 ピンで行うマスターマインドに関して, D. E. Knuth は, 解答者は高々 5 回の推測回数でコードを判

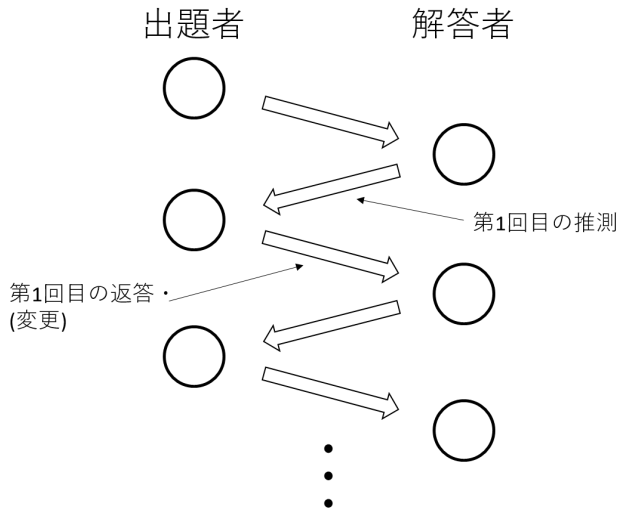


図 2　アクションの定義

別できることを示している [1]. また, 出題者が 1 度だけ返答に嘘をついてもよい拡張ルールを設けたマスターマインドに対する推測回数の上界を研究した論文や [2], 解答者が直前の推測と返答しか記憶できないという制限を設けたマスターマインドの研究などがある [3].

2　1 ピン変更可のマスターマインド

2.1　ルールの定義

本論文で取り扱う, 拡張版マスターマインドでのルールを定義する. 一般的なマスターマインドのルールに加え, 出題者が自分で作成したコードのピンの色を 1 つだけ, ゲームの途中, 任意のタイミングで変更してもよい, というルールを設ける. このとき, 解答者が第 i 回目に行った推測の直後の返答, もしくはコード配色の変更を第 i 回目の返答 (変更) とする. この様子を表したものが図 2 である.

2.2　自明な上界

まず, このゲームにおける推測回数の自明な上界を示すために, 出題者に対して, コード配色の変更時期に制限をかけたゲームを考える. 第 i 回目に出題者は必ずコード配色を変えるとする. このとき, 解答者は i+1 回目の推測から Knuth のアルゴリズムを用いることで, 高々 $i+5$ 回の推測でコードを判別することができる.

このことから, 本論文における拡張版マスターマインドの自明な上界が得られる. 出題者が, いつコードの配色を変えるかわからない場合, 解答者は 1 回目の推測から Knuth のアルゴリズムを用いてゲームを行う. 5 回以内の推測でゲームが終わらなければ, 出題者が 1 4 回目の返答の後にコードの配色を変更したことがわかる. したがって, 解答者は 6 回目の推測から新たに Knuth のアルゴリズムを用いゲームをすることで, 高々 10 回の推測でコードを判別することができる. このことから以下の定理が導かれる.

定理 1. 1 ピン変更可のマスターマインドでは, 高々 10 回の推測回数でコードを判別することができる.

3　推測回数の下界

本節では, 1 ピン変更可のマスターマインドの推測回数の下界, すなわちどのようなコード, あるいはピン変更に対しても, コード判別のためにはこれだけの回数は推測する必要がある, という回数について考察する.

定理 2. 1 ピン変更可のマスターマインドで, 確実にコードを判別するには少なくとも 9 回の推測が必要である.

この定理を証明するために, いくつかの補題を示す.

補題 1. 1 ピン変更可のマスターマインドで, 確実にコードを判別するには少なくとも 8 回の推測が必要である.

証明.
解答者が, Knuth のアルゴリズムに基づいてゲームを進行するとする. 出題者が 4 回目にコード配色を変えるとき, 解答者が 5 回目の推測で得られる返答は 3 ヒット 0 ブローである. このとき, この後 2 回の推測ではコードを判別することができないことを示す.

解答者が, 直前の返答で 3 ヒット 0 ブローが得ら

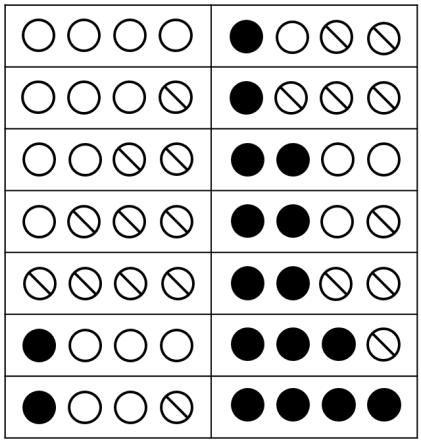


図 3　返答の種類

れたとする. つまりコードは, 5 回目の推測に使われた数のうち 1 つを他の数字に変えたものである. このコードの候補数は $5 \times 4 = 20$ 通りある. これに対して, 出題者の返答は, 3 ヒット 1 ブローという返答がありえないことに注意すると, 図 3 のように 14 通りのパターンに分けられる. 2 回でコードを判別するためには, 2 回目であらゆるコードの候補に対して 4 ヒット 0 ブローを得なければならない. しかし解答者は, 残りの 20 通りのコード候補数に対して, 1 回目の推測で高々 14 通りの返答にしか分類できない. つまり, 必ず同じ返答に分類される候補が存在する. よって 2 回の推測ではコードを判別することはできない.

解答者がコードを見つけるのに 6 回以上の推測を必要するとする. 解答者が 6 回の推測でコードを判別する手順を持つならば, 出題者は 5 回目の返答の後にコード配色を変更するという戦略をとると仮定すると, 上と同様の議論により, 6 回目の返答では 3 ヒット 0 ブローの返答が得られ, ここから 1 回の推測でコードを見つけることができない. 解答者がコードを判別するのに 7 回以上の推測を要するのであれば, 出題者は 6 回目の返答の後にコード配色を変更することで, 7 回以内でのコードの判別を防ぐことができる.

よってこのゲームにおいて, 解答者は 7 回の推測ではコードを判別することができない. つまり, コードの判別には, 少なくとも 8 回の推測は必要であることがわかる. □

補題 2. 任意のアルゴリズムは出題者がピンを変更しない場合でも, ぞろ目, あるいは 4 種の数を使うコードを正しく判別するのに 5 回の推測回数を要する.

証明.

1 回目の推測に 1234, 1123, 1122, 1112, 1111 のコードを用いた場合全てにおいて, 5 回の推測が必要であることを示す.

まず, 以下の 6 色 4 ピンの一般的なマスターマインドに,「出題者はぞろ目, もしくは 4 種の数を使うコードしか使用できない」という拡張ルールを設けた, 制限版マスターマインドを考える. コードになり得る候補はぞろ目が 6 個, 4 種の数を使うコードが順列を考えた 360 個の合計 366 個である.

- 1234 のパターン

  返答が 0 ヒット 2 ブローであったときの候補に着目する. この時点でコードがぞろ目である可能性はなく, 84 個の候補が含まれる. 2 回目の推測として解答者が 2156 と推測をする. このとき, 返答にヒットとブローの数が合計で 4 となるパターンは 4 ヒット 0 ブロー, 2 ヒット 2 ブロー, 1 ヒット 3 ブロー, 0 ヒット 4 ブローの 4 パターン存在し, この状況でそのいずれかに分類されるコードの候補数は 14 個ある. それぞれの返答に属する候補数は, 4 ヒット 0 ブローが 1 つ, 2 ヒット 2 ブローが 5 つ, 1 ヒット 3 ブローが 4 つ, 0 ヒット 4 ブローが 4 つである. 2 ヒット 2 ブローに含まれる候補は 5 つ存在するので, 鳩の巣原理より, 3 回目の推測後に少なくとも 2 つのコードが同じ返答の中に含まれる. これはその後の 4 回目の推測では判別することができない.
- 1123 のパターン

  0 ヒット 1 ブローのときに着目する. 2 回目の推測として 2456 と推測をすると, 1 ヒット 3 ブローに属するものが 7 つ, 0 ヒット 4 ブローに属するものが 6 つであり, 上の議論と同様, 鳩の巣原理より 4 回の推測では判別できない.

- 1122 のパターン
  0 ヒット 0 ブローのときに着目する. 2 回目の推測として 3456 と推測をすると, 2 ヒット 2 ブローに属するものが 6 つ, 1 ヒット 3 ブローに属するものが 7 つ, 0 ヒット 4 ブローに属するものが 10 個となり, 4 回の推測では判別できない.
- 1112 のパターン
  0 ヒット 1 ブローのときに着目する. 2 回目の推測として 2345 と推測をすると, 2 ヒット 2 ブローに属するものが 5 つ, 1 ヒット 3 ブロー, 0 ヒット 4 ブローに属するものが 6 つずつとなり, これも 4 回の推測では判別できない.
- 1111 のパターン
  この場合も, 0 ヒット 1 ブローのときに着目する. 2 回目の推測として 1345 を推測すると, 2 ヒット 2 ブローに属するものが 6 つ, 1 ヒット 3 ブローに属するものが 8 つ, 0 ヒット 4 ブローに属するものが 9 つとなり, これも 4 回の推測では判別できない.
  以上のことから, この制限版マスターマインドにおいて解答者はコードを判別するのに 5 回の推測を要する. つまり, 制限を設けない 6 色 4 ピンのマスターマインドにおいても解答者はコードを判別するのに 5 回の推測は必要である. □

補題 3. ぞろ目, あるいは 4 種の数を使うコードで推測をし, それに対し 3 ヒット 0 ブローの返答が得られるとき, 解答者はコードの判別にさらに 4 回の推測を必要とする.

証明.
ぞろ目の推測を 1111 とする. この推測に対して 3 ヒット 0 ブローの返答がきた場合, 変更された箇所には 1 が使われていないことがわかる. この状態から変更後のコードを見つけることを考える. 一般性を失わずに, 解答者の次の推測を 2345, 2234, 2233, 2223, 2222 とする. どの推測に対する返答を考えたときでも, 返答には高々 1 つのヒットもしくはブローしか立たない. よって返答は 0 ヒット 0 ブローであるか, そうでないかの 2 つの場合に分けられる.

- 推測に対して 0 ヒット 0 ブロー
  2345, 2234 に対する場合, 変更後のコードに用いられた数字が高々 2 通りに絞られる. このときは残り 2 回の推測で, 正しいコードに使われた数字を判別できる. 他方で, 2233, 2223, 2222 に対する返答だったとき, 少なくとも使われている可能性のある候補が 3 つある. これは残り 2 回の推測では使われている数字を判別することができない.
- 推測に対して 1 ヒット 0 ブローもしくは 0 ヒット 1 ブロー
  上記とは逆に, この場合 2345, 2234 のときに少なくともコードに使用されている数字の候補が 3 つとなり, 残り 2 回の推測で特定することができない. 2233, 2223, 2222 のとき, 使われている色の候補は高々 1 つであり, この後 2 回の推測でコードに使われている色は特定できる.
  以上のことより, ぞろ目の推測に対する返答が 3 ヒット 0 ブローであった場合, コードの判別にはさらに 4 回以上の推測を要する.
  4 種の数を使うコードの場合も同様の議論で, 4 回以上の推測が必要であることがわかる.
  よって補題が示された. □

補題 4. 出題者が以下の戦略をとることにより、解答者はコードを判別するのに 9 回の推測回数を必要とする.

- ぞろ目、あるいは 4 種の数字を使うコードのうちから 1 つ選ぶ
- 解答アルゴリズムの 4 回目の推測後にコードの変更を行う

証明.
補題 2 より, 出題者がこのコードの選び方をすることで, コードの正しい判別に 5 回の推測回数を必要とす

る. そして出題者が, 解答者の 4 回目の推測後にコードを変更することで補題 3 により, 解答者は変更後のコードの判別にさらに 4 回の推測を要する. つまり, 解答者は合計で少なくとも 9 回の推測を必要とする. □

4 9 回の推測で解けるか

4.1 3 ヒットが得られたあとの推測回数

この拡張版マスターマインドにおいて, 解答者が Knuth のアルゴリズムに基づいてゲームを進行するとすると, 出題者が 4 回目までにコードの変更を行わなかった場合, 5 回目の推測で必ずコードが判別する. よって出題者は 1 4 回目のどこかでコードを変更すると仮定する. このように仮定をおいた場合でも, 出題者のコード変更の仕方によっては, 返答によって得られるコードの候補数が 0 となることがあるが, このときは解答者が次の推測から Knuth のアルゴリズムを始めればよく, 9 回以内の推測でコードを判別することができる.

ここで, 出題者が 4 回目の推測の後にコードを変更した場合を考える. コードのうち変更できるのは 1 つのピンのみなので, 5 回目の推測に対する返答は 3 ヒットであることがわかる.

5 回目の推測で, 解答者が 3 ヒットの返答を得られたとする. この後, 4 回の推測でコードを見つけることができることを示す. ここで, 5 回目の推測に用いられた推測を, 1234, 1123, 1122, 1112, 1111 とする. それぞれの場合に対して, どのようなコードであっても 4 回の推測でコードを判別できることを示せばよい. ちなみに, 各場合それぞれに 20 通りのコードの候補がある.

- 1234 の場合

  まず, 1 回目に 1122 の推測をする. すると 20 通りのコードの候補は 2 ブロー (1), 1 ブロー (4), 1 ヒット 2 ブロー (2), 1 ヒット 1 ブロー (6), 1 ヒット (4), 2 ヒット 1 ブロー (2), 2 ヒット (1) の 7 パターンの返答に分類される. なお () 内の数値はそれぞれの返答後に残るコードの候補数を表している. しらみつぶしに推測を行うと, n 回の推測で n 個の候補からコードを判別できることを考え, ここでは 1 ブロー, 1 ヒット 2 ブロー, 1 ヒットについて残り 3 回の推測でコードを判別することを示す.

  まず, 1 ブローと 1 ヒットの返答が得られた場合は, どちらにおいても 3344, 1 ヒット 1 ブローの返答が得られた場合, 1256 と推測する. 得られる各返答に対する残りの候補数は全て 2 つとなり, これはあと 2 回の推測でコードを判別することができる.

  以上のことから, 1234 の推測に対して 3 ヒットの返事が得られた場合, その後 4 回の推測でコードを判別することができる.

- 1123 の場合

  2233 と推測する. このとき, 上と同様に 7 通りの返答に分けられる. この中で候補数が 4 以上になるのは, 1 ブロー (4), 1 ヒット 1 ブロー (6), 1 ヒット (4) の場合である. 1 ヒット 1 ブローであれば, 4455, それ以外の 2 つのときは 1114 と推測するとどの場合も候補数が 2 以下となり, あと 2 回の推測でコードを判別することができる.

- 1122 の場合

  5162 と推測する. このとき候補が 4 以上残るのは, 1 ヒット 1 ブロー (6), 1 ヒット 2 ブロー (4), 2 ヒット (6) の場合である. 1 ヒット 1 ブローであれば 2323, 1 ヒット 2 ブローなら, 1525, 2 ヒットならば 3132 と推測することで, どの場合でも候補数が 2 以下となる.

- 1112 の場合

  1342 と推測する. 1 ヒット 1 ブロー (5), 2 ヒット (6) のときのみ候補が 4 以上残る. 1 ヒット 1 ブローのとき 5634, 2 ヒットのときには 1512 と推測することで 4 回でコードを判別することができる.

- 1111 の場合

1234 と推測する. このとき候補が 4 以上残るのは, 1 ヒット 1 ブロー (6), 1 ヒット (6) の場合である. それぞれ次の推測で 5651, 3321 と推測すればよい.

これらのことから, 直前の推測で 3 ヒットの返答が得られた場合には, 上のように推測を行うことで, その後 4 回の推測があればあらゆるコードが判別できることが示された.

4.2 ゲーム木の枝刈り

Knuth のアルゴリズムをもとに, 1 回目, 2 回目, 3 回目と各推測の後にコードを変更させたときのゲームの過程を列挙するプログラムを作成した. 変化前のコード数が $6^4 = 1296$ 通りあり, あるコードから変化させうるコードの種類が $5 \times 4 = 20$ 通り, 変化させるタイミングが 1 回目, 2 回目, 3 回目と 3 回あることを考えると, パターン数は, $1296 \times 20 \times 3 = 77760$ 通りと計算できる. なお, 変化させうるタイミングは 4 回目もあるが, その場合は 4.1 節よりその後 4 回の推測でコードを判別することができるため, 計算には含まれていない. これらの列挙されたパターンの中には, 途中でコードの候補数が 0 になり, コード変更が解答者に見破られてしまうもの, 重複するものや 5 回目の推測後に 3 ヒット 0 ブローの返答が得られ, その後 4 回の推測でコードを判別できるものなどが含まれている. よってこれらの候補をパターン総数から取り除いていくと, 細かくチェックしないといけないパターンは 16 通りにまで絞られる.

これらのパターンをチェックし, いずれの場合も残り 4 回の推測で全てのコードを判別できることがわかった. よって本論文における拡張版マスターマインドは 9 回の推測回数で解くことができる.

5 おわりに

本論文では, コードの変更を一度だけ認めたマスターマインドにおいて, 8 回の推測ではコードを判別することができないこと, そして 9 回の推測回数でコードを判別できることを示した. 解答者がコードのピンをゲーム中にスワップすることを許したもの, ピンの変更とスワップの両方を許すなどの拡張も考えられる.

参考文献

[1] D. E. Knuth: The Computer As Master Mind, Journal of Recreational Mathematics, 9(1), pp. 1–6 (1976).

[2] L. T. Huang, S. T. Chen, S. S. Lin: Exact-Bound Analyzes and Optimal Strategies for Mastermind with a Lie, Advances in Computer Games, Lecture Notes in Computer Science 4250, pp. 195–209 (2005).

[3] J. Stuckman, G. Q. Zhang: Mastermind is NP-Complete, arxiv:cs/0512049, (2005).

An asynchronous P system using branch and bound technique for SAT

Yuki Jimen Akihiro Fujiwara
Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502, Japan
Email: n232035y@mail.kyutech.jp, fujiwara@cse.kyutech.ac.jp

***Abstract*—Membrane computing, which is a computational model based on cell activity, has considerable attention as one of new paradigms of computations. In the general membrane computing, computationally hard problems have been solved in a polynomial number of steps using an exponential number of membranes. However, reduction of the number of membranes must be considered to make P system more realistic model.**

In the paper, we propose an asynchronous P system using branch and bound, which is a well known optimization technique, to reduce the number of membranes. The proposed P system solves the satisfiability problem (SAT) with n variables and m clauses, and works in $O(m2^n)$ sequential steps or $O(mn)$ parallel steps. The number of membranes used in the proposed P system is evaluated using a computational simulation. The experimental result shows the effectiveness of the proposed P system.

I. Introduction

A number of next-generation computing paradigms have been considered due to limitation of silicon-based computational hardware. As an example of the computing paradigms, natural computing, which works using natural materials for computation, has considerable attention. A membrane computing, which is a computational model inspired by the structures and behaviors of living cells, is a representative of the natural computing.

A basic feature of the membrane computing was introduced in [1] as a P system. The P system consists mainly of membranes and objects. A membrane is a computing cell, in which independent computation is executed, and may contain objects and other membranes. Each object evolves according to evolution rules associated with a membrane in which the object is contained.

The P system and most variants have been proved to be universal [2], and several P systems have been proposed for solving computationally hard problems [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. In addition, asynchronous parallelism has been considered on the P system. The asynchronous parallelism means that all objects may react on rules with different speed, and evolution rules are applied to objects independently. A number of asynchronous P systems have been proposed for the computationally hard problems in [14], [15], [16], [17], [18]. For example, an asynchronous P system has been proposed for solving the satisfiability problem [15].

However, the computationally hard problems have been solved in polynomial numbers of steps using exponential numbers of membranes on the above P systems. Since the number of membranes is limited in case of using real living cells, reduction of the number of membranes must be considered to make P system more realistic model.

In the paper, we propose an asynchronous P system using branch and bound, which is a well known optimization technique, to reduce the number of membranes. The proposed P system solves the satisfiability problem with n variables and m clauses, and works in $O(m2^n)$ sequential steps or $O(mn)$ parallel steps.

In addition to the above, the number of membranes used in the proposed P system is evaluated using a computational simulation. The experimental result shows the effectiveness of the proposed P system using branch and bound.

II. Preliminaries

A. Computational model for membrane computing

Several models have been proposed for membrane computing. We briefly introduce a basic model of the P system in this subsection.

The P system consists mainly of membranes and objects. A membrane is a computing cell, in which independent computation is executed, and may contain objects and other membranes. In other words, the membranes form nested structures. In the present paper, each membrane is denoted by using a pair of square brackets, and the number on the right-hand side of each right-hand bracket denotes the label of the corresponding membrane. An object in the P system is a memory cell, in which each data is stored, and can divide, dissolve, and pass through membranes. In the present paper, each object is denoted by finite strings over a given alphabet, and is contained in one of the membranes.

For example, $[[a]_2[b]_3]_1$ denotes membrane structure that consists of three membranes. The membrane labeled 1 contains two membranes labeled 2 and 3, and the two membranes contain objects a and b, respectively.

Computation of P systems is executed according to evolution rules, which are defined as rewriting rules for membranes and objects. All objects and membranes are transformed in parallel according to applicable evolution rules. If no evolution rule is applicable for objects, the system ceases computation.

We now formally define a P system and the sets used in the system as follows.

$$\Pi = (O, \mu, \omega_1, \omega_2, \cdots, \omega_m, R_1, R_2, \cdots, R_m)$$

O: O is the set of all objects used in the system.
μ: μ is membrane structure that consists of m membranes. Each membrane in the structure is labeled with an integer.
ω_i: Each ω_i is a set of objects initially contained only in the membrane labeled i.
R_i: Each R_i is a set of evolution rules that are applicable to objects in the membrane labeled i.

In the present paper, we assume that input objects are given from the outside region into the outermost membrane, and computation is started by applying evolution rules. We also assume that output objects are sent from the outermost membrane to the outside region.

In membrane computing, several types of rules are proposed. In the present paper, we consider five basic rules of the following forms.

(1) Object evolution rule: $[\ a\ ]_h \rightarrow [\ b\ ]_h$
where h is a label of the membrane and $a, b \in O$. Using the rule, an object a evolves into another object b. (We omit the brackets in each evolution rule such as $a \rightarrow b$ for cases that a corresponding membrane is obvious.)

(2) Send-in communication rule: $a[\ ]_h \rightarrow [\ b\ ]_h$
where h is a label of the membrane, and $a, b \in O$. Using the rule, an object a is sent into the membrane, and can evolve into another object b.

(3) Send-out communication rule: $[\ a\ ]_h \rightarrow [\ ]_h b$
where h is a label of the membrane, and $a, b \in O$. Using the rule, an object a is sent out of the membrane, and can evolve into another object b.

(4) Dissolution rule: $[\ a\ ]_h \rightarrow b$
where h is a label of the membrane, and $a, b \in O$. Using the rule, the membrane, which contains object a, is dissolved, and the object can evolve into another object b. (The outermost membrane cannot be dissolved.)

(5) Division rule: $[\ a\ ]_h \rightarrow [\ b\ ]_h [\ c\ ]_h$
where h is a label of the membrane, and $a, b \in O$. Using the rule, the membrane, which contains object a, is divided into two membranes that contain objects b and c respectively.

We assume that each of the above rules is applied in a constant number of biological steps. In the following sections, we consider the number of steps executed in a P system as the complexity of the P system.

B. Maximal parallelism and asynchronous parallelism

In the standard model in membrane computing, which is a P system with maximal parallelism, all of the above rules are applied in a non-deterministic maximally parallel manner. In one step of computation of the P system, each object is evolved according to one of applicable rules. (In case there are several possibilities, one of the applicable rules is non-deterministically chosen.) All objects, for which no rules applicable, remain unchanged to the next step. In other words, all applicable rules are applied in parallel in each step of computation.

On the other hand, evolution rules are applied in a fully asynchronous manner on the asynchronous P system, and any number of applicable evolution rules is applied in each step of computation. In the other words, the asynchronous P system can be executed sequentially, and also can be executed in the maximal parallel manner.

The reason why we assume the asynchronous parallelism in this paper is based on the fact that every living cell acts independently and asynchronously. Since the standard P system ignores the asynchronous feature of living cells, the asynchronous P system is a more realistic computation model for cell activities.

We now show an example for difference between the P system with maximal parallelism and the asynchronous P system. We define P system Π and the sets used in the system as follows.

$$\Pi = (O, \mu, \omega_1, R_1)$$
$$O = \{a, b, c, d, e\}, \mu = [\ ]_1, \omega_1 = \phi,$$
$$R_1 = \{a \rightarrow b, bc \rightarrow d, c \rightarrow e, ae \rightarrow f\}$$

We consider computations of the P system Π. Let us assume that input objects ac are given into the membrane from the outside region. On the standard P system, all applicable evolution rules, which are $a \rightarrow b$ and $c \rightarrow e$, are applied in parallel, and the objects ac are evolved into be.

On the other hand, five computations given below can be considered on the asynchronous P system according to the order of application of the evolution rules.

$$ac \rightarrow be$$
$$ac \rightarrow ae \rightarrow be$$
$$ac \rightarrow ae \rightarrow f$$
$$ac \rightarrow bc \rightarrow be$$
$$ac \rightarrow bc \rightarrow d$$

Therefore, a number of executions are possible in the asynchronous P system, and the evolution rules in the standard P system, which assumes the maximal parallel manner, may not work in the asynchronous parallel manner.

In the asynchronous P system, all evolution rules can be applied completely in parallel, which is the same as the conventional P system, or all evolution rules can be applied sequentially. We define the number of steps executed in the asynchronous P system in the maximal parallel manner as the number of parallel steps. We also define the number of steps in the case that the applicable evolution rules are applied sequentially as the number of sequential steps. The numbers of parallel and sequential steps indicate the best and worst case complexities for the asynchronous P system. In addition, the proposed asynchronous P system must be guaranteed to output a correct solution in any asynchronous execution.

III. A P SYSTEM USING BRANCH AND BOUND FOR SAT

In this section, we present an asynchronous P system using branch and bound for the satisfiability problem. We first explain an input and an output of the problem for the system, and then, show an outline and details of the P system with an example. Finally, we discuss complexity of the P system, and shows effectiveness of the proposed P system using experimental simulation.

A. Input and output

The satisfiability problem (SAT) is a well-known problem that determines if there exists a truth assignment for a given Boolean formula. We assume that an input formula of SAT is given in the conjunctive normal form (CNF) with n Boolean variables and m clauses. We also assume that an output of SAT is one of two values, "TRUE" and "FALSE". The output is "TRUE" if there exists a truth assignment for satisfying the formula, otherwise, the value is "FALSE".

The following is an example of an input formula with 2 variables and 3 clauses. Since a truth assignment, $x_1 = 1, x_2 = 0$, satisfies the input formula, an output of SAT is "TRUE".

$$(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2)$$

The above input is given by the following set of objects O_L in the P system.

$$O_L = \{\langle X_{i,j}, V\rangle \mid 1 \le i \le n, 1 \le j \le m, V \in \{0, 1, N\}\}$$

Each object $\langle X_{i,j}, V\rangle$ denotes a Boolean value of variable x_i in the j-th clause. In addition, V is set to N if neither x_i nor $\bar{x}_i$ is in the j-th clause.

For example, the following set of objects denotes the above input formula

$$\begin{aligned} O_L \;=\; & \{\langle X_{1,1}, 1\rangle, \langle X_{2,1}, 0\rangle, \langle X_{1,2}, 1\rangle, \langle X_{2,2}, 1\rangle \\ & \langle X_{1,3}, 0\rangle, \langle X_{2,3}, N\rangle\} \end{aligned}$$

We assume that input set O_L is given from the outside region into the outer membrane.

The output of the P system is one of two objects, $\langle TRUE\rangle$ and $\langle FALSE\rangle$. The object $\langle TRUE\rangle$ is sent out from the outer membrane to the outside region if the input formula is satisfiable, otherwise, the object $\langle FALSE\rangle$ is sent out to the outside region.

B. Branch and bound for the SAT

Branch and bound is a well known computing paradigm for optimization problem. For computing SAT, all assignments are enumerated for n Boolean variables, and 2^n solutions must be created for the exhaustive enumeration. However, a number of assignments can be discarded if the assignment cannot produce "TRUE".

Figure 1 shows a search tree and illustration of the above idea. Let $(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_2)$ be an input formula. Then, the last clause cannot be satisfied in case of $x_2 = 1$, and assignments for x_1 can be ignored.

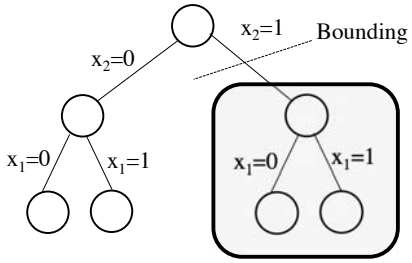


Fig. 1. An example of branch and bound for SAT

We now explain an overview of the asynchronous P system using branch and bound for computing SAT. The membrane structure used in the computation is $[\ [\ ]_2]_1$. We call the membranes labeled 1 and 2 *outer* and *inner* membranes, respectively.

The computation of the P system mainly consists of the following three steps.

Step 1: Move all input objects in the outer membrane into the inner membrane.

Step 2: In each inner membrane, repeat the following (2-1) and (2-2) until "TRUE" or "FALSE" is outputted.

(2-1) Create a truth assignment for a variable by dividing the inner membrane.

(2-2) Check satisfiability for a truth assignment in each divided membrane. If all the clauses are satisfied, an object denoting "TRUE" is outputted to the outer membrane. On the other hand, if one of the clauses cannot be satisfied by the truth assignment, an object denoting "FALSE" is outputted to the outer membrane.

Step 3: Send out a final result, "TRUE" or "FALSE" from the outer membrane.

C. Details of the P system

We now explain details of each step of the computation. In Step 1, all input objects in the outer membrane are moved into the inner membrane. Since the P system used in the paper is asynchronous, we cannot move the input objects in parallel, and input objects are moved one by one applying following two sets of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{1,1} \;=\; & \{\langle X_{1,1}, V\rangle[\ ]_2 \;\to [\langle M_{2.1}\rangle\langle X_{2,1}, V\rangle]_2 \mid V \in \{0, 1, N\}\} \\ & \cup\{\langle M_{i,j}, k\rangle\langle X_{i,j}, V\rangle[\ ]_2 \;\to [\langle M_{i+1,j}, 0\rangle\langle X_{i,j}, V\rangle]_2 \\ & \quad \mid 1 \le i \le n, 1 \le j \le m, 0 \le k \le n-1, V \in \{0, 1\}\} \\ & \cup\{\langle M_{i,j}, k\rangle\langle X_{i,j}, N\rangle[\ ]_2 \;\to [\langle M_{i+1,j}, k+1\rangle\langle X_{i,j}, N\rangle]_2 \\ & \quad \mid 1 \le i \le n, 1 \le j \le m, 1 \le k \le n\} \\ & \cup\{\langle M_{n+1,j}, k\rangle \;\to\; \langle M_{1,j+1}, 0\rangle\langle F_j, n-k, 0\rangle \\ & \quad \mid 1 \le j \le m, 0 \le k \le n-1\} \\ & \cup\{\langle M_{1,m+1}\rangle[\ ]_2 \;\to [\langle M_{1,m+2}\rangle]_2\} \end{aligned}$$

(Evolution rules for the inner membrane)

$$\begin{aligned} R_{2,1} &= \{[\langle M_{i,j},k\rangle]_2 \rightarrow [\ ]_2\langle M_{i,j},k\rangle \\ &\quad \mid 2\le i\le n, 1\le j\le m\} \\ &\cup\{[\langle M_{1,m+2},k\rangle]_2 \rightarrow [\langle S_1\rangle]_2[\langle m\rangle]_2 \\ &\quad \mid 0\le k\le n-1\} \end{aligned}$$

In the above evolution rules, object $\langle X_{1,1},V\rangle$ is moved from the outer membrane to the inner membrane, and then, object $\langle M_{2,1},0\rangle$ is created and moved to the inner membrane. (Object $\langle M_{i,j},k\rangle$ moves object $\langle X_{i,j},V\rangle$ from the outer membrane to the inner membrane.)

In addition, object $\langle F_j,k,f\rangle$ is created for the j-th clause. In the object, k denotes the order of division for the clause, and $f\in\{0,1\}$ is a flag that is set to 1 if j-th clause is satisfied. The object $\langle F_j,k,f\rangle$ is also moved to the inner membrane.

After all input objects are moved to the inner membrane, object $\langle M_{1,m+1},0\rangle$ is created and moved to the inner membrane. Then, object $\langle S_1\rangle$ and $\langle m\rangle$ are created in the inner membrane. Object $\langle m\rangle$ denotes the the number of unsatisfied clauses, and object $\langle S_1\rangle$ triggers Step2.

In Step 2, object $\langle S_i\rangle$ is used as a trigger for membrane division, and object $\langle A_i,0\rangle$ and $\langle A_i,1\rangle$ are used for assignments such that $x_i=0$ and $x_i=1$, respectively. In addition, object $\langle C_{i,j},i\rangle$ is used for checking if assignment for i-th literal satisfies j-th clause.

In (2-1), a truth assignment for a variable is created by dividing the inner membrane. The sub-step is executed by applying the following set of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{2,2,1} &= \{[\langle S_i\rangle]_2 \rightarrow [\langle A_i,0\rangle\langle C_{i,1},i\rangle]_2[\langle A_i,1\rangle\langle C_{i,1},i\rangle]_2 \\ &\quad \mid 1\le i\le n\} \end{aligned}$$

In (2-2), satisfiability for a truth assignment in each divided membrane is checked. The sub-step is executed by applying the following sets of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{2,2,2} &= R_{2,2,2,1}\cup R_{2,2,2,2}\cup R_{2,2,2,3} \\ R_{2,2,2,1} &= \{\langle C_{i,j},i\rangle\langle X_{i,j},V\rangle\langle A_i,V'\rangle\langle F_j,k,f\rangle \\ &\quad \rightarrow \langle C_{i,j+1},i\rangle\langle X_{i,j},V\rangle\langle A_i,V'\rangle\langle F_j,k,f\rangle \\ &\quad \mid 1\le i\le n, 1\le j\le m, i\ne k, f\in\{0,1\}, \\ &\quad V\in\{0,1,N\}, V'\in\{0,1\}, V\ne V'\} \\ &\cup\{\langle C_{i,j},i\rangle\langle X_{i,j},V\rangle\langle A_i,V'\rangle\langle F_j,i,0\rangle \\ &\quad \rightarrow []_2\langle FALSE,2^{n-i}\rangle \\ &\quad \mid 1\le i\le n, 1\le j\le m, V\in\{0,1,N\}, \\ &\quad V'\in\{0,1\}, V\ne V'\} \\ &\cup\{\langle C_{i,j},i\rangle\langle X_{i,j},V\rangle\langle A_i,V\rangle\langle F_j,k,0\rangle\langle l\rangle \\ &\quad \rightarrow \langle C_{i,j+1},i\rangle\langle X_{i,j},V\rangle\langle A_i,V\rangle\langle F_j,k,1\rangle\langle l-1\rangle \\ &\quad \mid 1\le i\le n, 1\le j\le m, 1\le l\le m, \\ &\quad 1\le k\le n, V\in\{0,1\}\} \\ &\cup\{\langle C_{i,j},i\rangle\langle X_{i,j},V\rangle\langle A_i,V\rangle\langle F_j,k,1\rangle \\ &\quad \rightarrow \langle C_{i,j+1},i\rangle\langle X_{i,j},V\rangle\langle A_i,V\rangle\langle F_j,k,1\rangle \\ &\quad \mid 1\le i\le n, 1\le j\le m, 0\le k\le n-1, \\ &\quad i\ne k, V\in\{0,1\}\} \\ R_{2,2,2,2} &= \{\langle C_{i,m+1},i\rangle\langle l\rangle \rightarrow \langle S_{i+1}\rangle\langle l\rangle \mid 1\le i\le n, \\ &\quad 1\le j\le m, 1\le l\le m\} \\ &\cup\{\langle C_{i,m+1},i\rangle\langle 0\rangle \rightarrow []_2\langle TRUE\rangle \mid 1\le i\le n\} \\ R_{2,2,2,3} &= \{\langle S_{n+1}\rangle \rightarrow []_2\langle FALSE,1\rangle\} \end{aligned}$$

In the above evolution rules, satisfiability of each clause is checked in order of $X_{i,1},X_{i,2},\cdots X_{i,m}$, and the number of satisfied clauses is computed using evolution rules in $R_{2,2,2,1}$. If all clauses are satisfied, $\langle TRUE\rangle$ is outputted to the outer membrane, and membrane division is terminated. On the other hand, if a clause cannot be satisfied by a truth assignment in the divided membrane, object $\langle FALSE,2^{n-i}\rangle$ is outputted to the outer membrane, and membrane division is also terminated.

Otherwise, if truth assignments are created for all variables, object $\langle S_{n+1}\rangle$, which indicates that all checks for variables are completed, is created, and object $\langle FALSE,1\rangle$ is outputted to the outer membrane. If the check is not completed for all variables, object $\langle C_{i,m+1},i\rangle$ is changed into object $\langle S_{i+1}\rangle$, and Step 2 is repeated by executing evolution rules in $R_{2,2,2,1}$.

In Step 3, a final result is sent out from the outer membrane. The Step 3 is executed applying the following set of evolution rules.

(Evolution rules for the outer membrane)

$$\begin{aligned} R_{1,3} &= \{\{[\langle TRUE\rangle]_1 \rightarrow []_1\langle TRUE\rangle\} \\ &\cup\{\langle FALSE,2^p\rangle\langle FALSE,2^p\rangle \rightarrow \langle FALSE,2^{p+1}\rangle \\ &\quad \mid 0\le p\le n-1\} \\ &\cup\{[\langle FALSE,2^n\rangle]_1 \rightarrow []_1\langle FALSE\rangle\}\} \end{aligned}$$

If object $\langle TRUE\rangle$ is in the outer membrane, there is a assignment that satisfies the input formula, and the object is sent out from the membrane immediately applying $[\langle TRUE\rangle]_1 \rightarrow [\ ]_1\langle TRUE\rangle$. On the other hand, 2^n objects $\langle FALSE,1\rangle$ are sent out from the divided inner membranes to the outer membrane in case that the formula cannot be satisfied. Since we assume the P system is asynchronous, the sum of the objects must be counted asynchronously.

We now summarize the asynchronous P system $\Pi_{\mathrm{BB-SAT}}$ for SAT as follows.

$$\Pi_{\mathrm{BB-SAT}} = (O,\mu,\omega_1,\omega_2,R_1,R_2)$$

- $O = O_L\cup\{\langle TRUE\rangle,\langle FALSE\rangle\}$

  $O_L = \{\langle X_{i,j},V\rangle | 1\le i\le n, 1\le j\le m, V\in\{0,1,N\}\}$
- $\mu = [\ [\ ]_2\ ]_1$
- $\omega_1=\omega_2=\phi$
- $R_1 = R_{1,1}\cup R_{1,3}$, $R_2 = R_{2,1}\cup R_{2,2,1}\cup R_{2,2,2}$

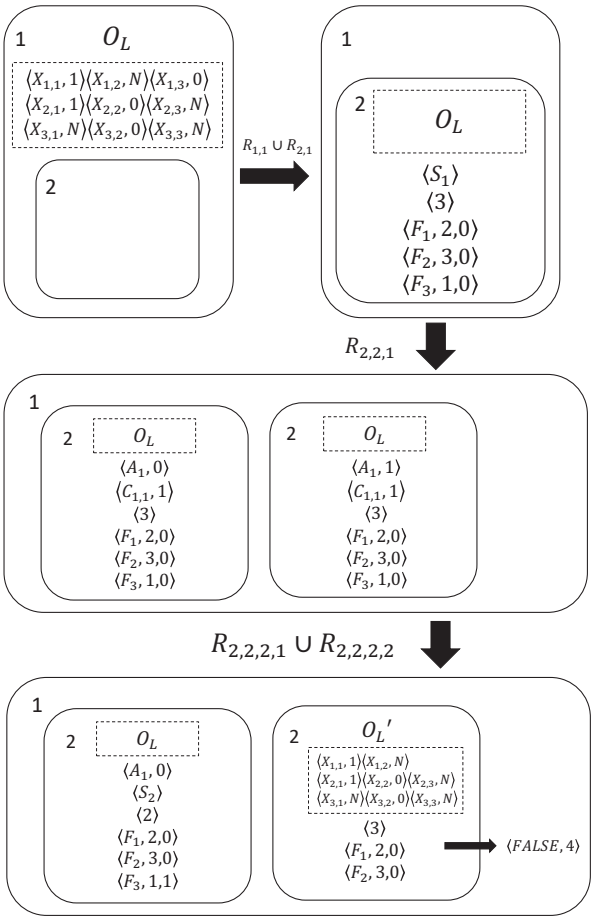


Fig. 2. An example of execution of Π_{BB-SAT} (Step 1 and Step 2)

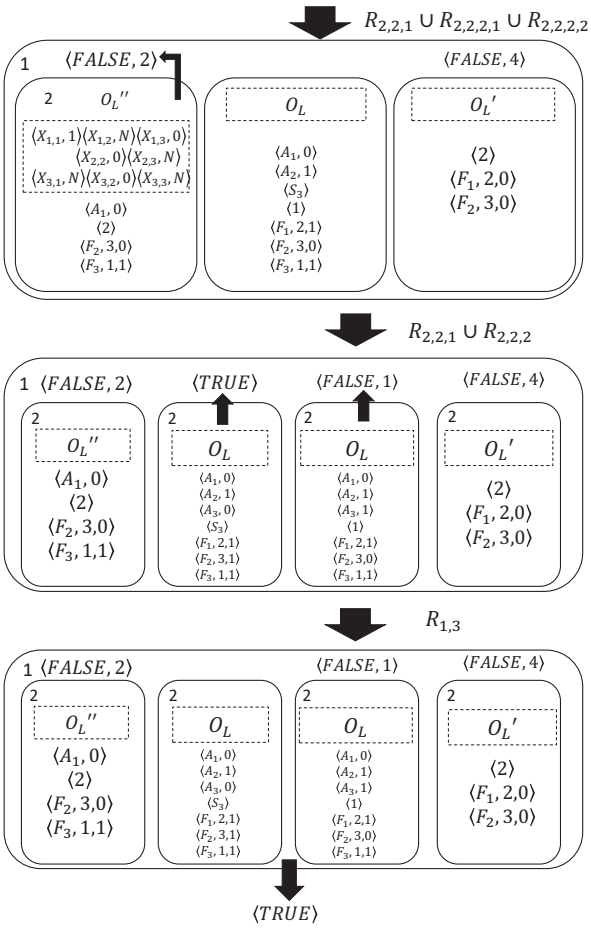


Fig. 3. An example of execution of Π_{BB-SAT} (Step 2 and Step 3)

D. An example of P system

Figure 2 and Figure 3 illustrate an example execution of the proposed P system Π_{BB-SAT}. An input formula for the example is $(x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1)$, and $n = 3$, $m = 3$.

Figure 2 illustrates an execution of Step 1 and Step 2. A set of objects O_L is given from the outside region into the outer membrane. By applying evolution rules $R_{1,1}$ and $R_{2,1}$, objects in O_L and other objects are moved to the inner membrane.

Next, by applying the evolution rule in $R_{2,2}$, $\langle A_1, 0\rangle$ and $\langle A_1, 1\rangle$, which indicate a truth assignment for the first variable x_1, are created. Then, satisfiability for a truth assignment, $x_1 = 0$ or $x_1 = 1$, is checked by applying the evolution rule in $R_{2,3}$. Since the last clause cannot be satisfied in case of $x_1 = 1$, object $\langle FALSE, 4\rangle$ is sent out to the outer membrane, and membrane division is terminated for the membrane.

Figure3 illustrates an execution of Step 2 and Step 3. The evolution rules of Step 2 are applied to the inner membranes repeatedly until inner membrane outputs all solutions to the outer membrane. Then, the final solution object $\langle TRUE\rangle$ is outputted from the outer membrane to the outside region.

E. Complexity of the P system

We now consider the complexity of asynchronous P system Π_{BB-SAT}. Step 1 can be executed in $O(mn)$ parallel step and $O(mn)$ sequential step using $O(mn)$ kinds of objects and $O(mn)$ kinds of evolution rules. Step 2 can be executed in $O(mn)$ parallel step $O(m2^n)$ sequential step using $O(mn^2)$ kinds of objects and $O(m^2n^2)$ kinds of evolution rules. Step 3 can be executed in $O(n)$ parallel step, $O(n)$ sequential step using $O(n)$ kinds of objects and $O(n)$ kinds of evolution rules.

Therefore, we obtain the following theorem for the asynchronous P system Π_{BB-SAT}.

Theorem 1: The asynchronous P system Π_{BB-SAT}, which computes the satisfiability problem works in $O(m2^n)$ sequential steps or $O(mn)$ parallel steps by using $O(mn^2)$ types of objects and evolution rules of size $O(m^2n^2)$. □

F. Experimental simulation

We develop an original simulator for asynchronous P systems using C language, and compare the number of membranes used in executions of an existing P system for SAT [15] and our proposed P system.

Figure 4 shows that the number of membranes on the existing P system increases exponentially to the number of

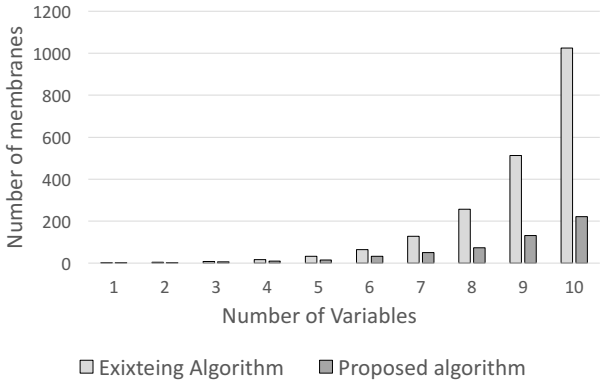


Fig. 4. An experimental result for the number of membranes

variable n. Although the number of membrane on the proposed P system also increases in proportion to n, the number of membrane is smaller by at most 75 percent in comparison to the existing P system.

IV. Conclusions

In this paper, we proposed an asynchronous P system, which solves the satisfiability problem, using the branch and bound. We showed that the proposed P system reduces the number of membranes than the existing P system.

In our future research, we are considering reduction of the number of objects and evolution rules used in the proposed P system.

Acknowledgments

This research was partially supported by JSPS KAKENHI, Grand-in-Aid for Scientific Research (C), 24500019.

References

[1] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.
[2] ——, *Introduction to Membrane Computing*. Springer, 2006.
[3] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and F. J. Romero-Campero, "A uniform solution to SAT using membrane creation," *Theoretical Computer Science*, vol. 371, no. 1-2, pp. 54–61, 2007.
[4] V. Manca, "DNA and membrane algorithms for SAT," *Fundamenta Informaticae*, vol. 49, no. 1-3, pp. 205–221, 2002.
[5] L. Q. Pan and A. Alhazov, "Solving HPP and SAT by P systems with active membranes and separation rules," *Acta Informatica*, vol. 43, no. 2, pp. 131–145, 2006.
[6] G. Păun, "P systems with active membranes: Attacking NP-complete problems," *Journal of Automata, Languages and Combinatorics*, vol. 6, no. 1, pp. 75–90, 2001.
[7] M. J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini, "A polynomial complexity class in P systems using membrane division," *Journal of Automata, Languages and Combinatorics*, vol. 11, no. 4, pp. 423–434, 2003.
[8] C. Zandron, C. Ferretti, and G. Mauri, "Solving NP-complete problems using P systems with active membranes," in *Unconventional Models of Computation*, 2000, pp. 289–301.
[9] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez, "A fast P system for finding a balanced 2-partition," *Soft Computing*, vol. 9, no. 9, pp. 673–678, 2005.
[10] M. J. Pérez-Jiménez and A. Riscos-Núñez, "A linear-time solution to the knapsack problem using P systems with active membranes," *Membrane Computing*, vol. 2933, pp. 250–268, 2004.
[11] ——, "Solving the subset-sum problem by P systems with active membranes," *New Generation Computing*, vol. 23, no. 4, pp. 339–356, 2005.
[12] M. J. Pérez-Jiménez and F. Romero-Campero, "Solving the BIN PACKING problem by recognizer P systems with active membranes," in *The Second Brainstorming Week on Membrane Computing*, 2004, pp. 414–430.
[13] A. Leporati, C. Zandron, and G. Mauri, "Solving the factorization problem with P systems," *Progress in Natural Science*, vol. 17, no. 4, pp. 471–478, 2007.
[14] R. Freund, "Asynchronous P systems and P systems working in the sequential mode," in *International workshop on Membrane Computing*, 2005, pp. 36–62.
[15] H. Tagawa and A. Fujiwara, "Solving SAT and Hamiltonian cycle problem using asynchronous p systems," *IEICE Transactions on Information and Systems (Special section on Foundations of Computer Science)*, vol. E95-D, no. 3, 2012.
[16] T. Murakawa and A. Fujiwara, "Arithmetic operations and factorization using asynchronous P systems," *International Journal of Networking and Computing*, vol. 2, no. 2, pp. 217–233, 2012.
[17] K. Tanaka and A. Fujiwara, "Asynchronous P systems for hard graph problems," *International Journal of Networking and Computing*, vol. 4, no. 1, pp. 2–22, 2014.
[18] J. Imatomi and A. Fujiwara, "An asynchronous P system for MAX-SAT," in *8th International Workshop on Parallel and Distributed Algorithms and Applications*, 2016, pp. 572–578.

ブロックチェーンにおけるPoWの代替となるアルゴリズムのサーベイ

木下 崇央　　田村 康将　　Xavier Defago
東京工業大学情報理工学院
kinoshita.t@coord.c.titech.ac.jp

概要

電子仮想通貨であるビットコイン（Bitcoin）やイーサリアム（Etherium）では, 全てのトランザクション（取引記録）で構成される台帳情報を複数ノードに分散して保存・同期をするためにブロックチェーン（分散型台帳技術）を用いている. ブロックチェーンでは複数のトランザクションからなる「ブロック」を定義し, これを時系列でつなぐことによって台帳情報を保管する. もし全ての正常なノードが同じブロックチェーンを保有している場合, システムが正常に動作しているといえる.

ブロックチェーンではコンセンサスアルゴリズムを用いることでシステム内のデータの一貫性を確保している. 多くのブロックチェーンではコンセンサスアルゴリズムの一つであるProof of Work（PoW）[1] が使用されており, これは仕事量によってデータの一貫性の保証をする. PoW にはブロックの改竄が極めて困難であるという利点がある. その一方で, 報酬が多いことを前提としたシステムで仕事量に対して十分な報酬が見込めない場合は正常に動作するインセンティブがないこと, さらに有益ではない計算を総当たり的に行うことによるコストの浪費が欠点である. こうした欠点を補うためのコンセンサスアルゴリズムとして, 主に Proof of Stake(PoS)[2],Proof of Elapsed Time(PoET)[3],Proof of Importance(PoI)[4],Proof of Activity(PoA)[5] が提案されている.PoS は保有資産の大きさによって,PoET は経過時間によって,PoI はノードが関与したトランザクションの数によってデータの一貫性の保証を行うアルゴリズムである. PoA は PoS を用いて PoW を拡張したアルゴリズムである.

本サーベイでは, ブロックチェーンに用いられているコンセンサスアルゴリズムの変遷を示したうえで, それぞれのコンセンサスアルゴリズムの目的, 利点, 欠点をまとめる. そして, 自律分散ロボット群において, ロボット間の全ての通信を複数ロボットに分散して保存・同期するのにブロックチェーンを用いる場合, 適しているコンセンサスアルゴリズムの検討を行う.

参考文献

[1] Markus Jakobsson, Ari Juels: Proofs of Work and Bread Pudding Protocols. Communications and Multimedia Security 1999: 258-272
[2] QuantumMachanic: Proof of Stake. Available from https://bitcointalk.org/index.php?topic=27787.0.
[3] Intel:Proof of elapsed time(poet). Available from http://intelledger.github.io/
[4] NEM: NEM Technical Reference, Version 1.0 .2015
[5] Iddo Bentov, Charles Lee, Alex Mizrahi, Meni Rosenfeld: Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake. IACR Cryptology ePrint Archive 2014: 452

マルチツリー型 P2P ビデオストリーミングにおいて配信を実現するための最短ホップ数

庄司 拓矢
広島大学 大学院工学研究科

藤田 聡
広島大学 大学院工学研究科

概要

ピア・ツー・ピア (P2P) ビデオストリーミングシステムにおいて，ビデオストリームはいくつかのストライプと呼ばれるサブストリームに分割され，それぞれ異なるスパニングツリーを介してソースピアから視聴ピアに配信される．このときソース以外のピアも自身のアップロード帯域を用いてビデオストリームの転送を行うため，システム全体のアップロード容量を増やすことができる．ビデオストリームを b 個のストライプに分割して，ソースを含むすべてのピアは b 個のストライプを同時に他のピアへ転送可能であるとする．本稿では，ビデオストリーミングサービスの最大遅延の観点から，n 台の視聴ピアすべてにおいて，b 個のストライプの配信を実現するときの b 個のスパニングツリーにおける最大の深さに着目する．具体的には，任意の b と n の組み合わせにおいて，スパニングツリーの最大の深さに対するタイトな下界を導出する．

1 はじめに

ピア・ツー・ピア (P2P) ネットワークにおけるビデオストリーミングは近年注目を集めており，様々な研究が行われている [1] [2] [3] [4] [5] [6]. P2P ネットワーク上でソースピアがビデオストリームを配信する際に，ソース以外のピアもストリーミングデータを拡散させるために自身のアップロード帯域を使用する．すなわち，各ピアは受信したストリーミングデータを他のピアに転送することができるため，ソースピアに対する転送負荷を低減させることができる．P2P ビデオストリーミングシステムにおけるビデオストリームの配信に関しては様々なネットワーク構造が存在する．ツリー型 P2P システムでは，ソースピアを根とした木構造のオーバーレイを構築して各ピアへの配信を行う．具体的には，ビデオストリームをいくつかのストライプに分割して，各ストライプに対して異なるスパニングツリーを用いて転送することで，システム全体のアップロード容量を増やすことができる．

以下では，マルチツリー型 P2P ビデオストリーミングシステムについて考える．ここでは，ソースピア s がビデオストリームを n 台の視聴ピアに配信する場合を考える．ソースを含む $n+1$ 台のピアは完全結合されており，各ピアのアップロード帯域を用いて任意のピア間で双方向への直接通信が可能であるとする．ソースピアはビデオストリームを b 個のストライプに分割し，それらのストライプは異なる b 個のスパニングツリーを介してソースピア s から n 台の視聴ピアに配信される．簡単のため，各リンクの容量と各ピアのダウンロード帯域は無限とする一方で，各ピアのアップロード帯域は有限で，今回は定数とする．

本稿では，ソースピアを含む各ピアのアップロード帯域が b であると仮定したとき，ソースピア s から n 台の視聴ピアに対して b 個のストライプの配信を実現するときの b 個のスパニングツリーにおける最大の深さに着目する．このスパニングツリーにおける最大の深さは最大ホップ数を意味するため，ビデオストリーミングサービスの遅延を考慮して，スパニングツリーの深さは小さいほうが望ましいとされる．本稿の目的は，すべての視聴ピアがすべてのストライプを最短ホップ数で受信することである．ゆえに，任意の b と n の組み合わせに対して，各 b 個のスパニングツリーにおける最大の深さのタイトな下界を導出する．なお本結果は，b が偶数であるときのみの議論を行うことにする．また，n 台のすべてのピアが b 個のストライプの受信を必要とし，ストライプの受信を必要とせずに視聴ピアへの転送の手助けをするヘルパーは存在しないものと仮定しているため，b が偶数の場合において，すべての視聴ピアに対して配信を可能とするための最短ホップ数のタイトな下界を真に示すことになる．

2 関連研究

マルチツリー型 P2P ビデオストリーミングシステムについては近年様々な研究が行われている．SplitStream [1] では，メディアサーバはストリームをいくつかのストライプに分割し，それぞれ異なるスパニングツリーを介して各ストライプを配信する．各ピアはすべてのストライプを受信するためにすべてのスパニングツリーに参加するが，このとき，1 つのスパニングツリーに対しては内部ピアとして参加し，残りのスパニングツリーに対しては葉ピアとして参加する．この方法で，すべてのピアが効率的に自身のアップロード帯域を利用することができるため，ストライプの転送負荷を分散することができる．これらの実験的評価は [7] でも行われている．

Zhao ら [8] は 2 ホップ one-view MPVC において，任意の視聴状況で保証される最大ビデオレートの下限を導出した．各ピアはビデオ会議に参加しているユーザを選択することができ，そのユーザによって配信されたストリームのみを視聴することができる．各ストリームはそれぞれ連続モデル上でいくつかのストライプに分割され，マルチツリー型 P2P システムで配信される．

安藤ら [9] は Zhao らの結果 [8] を離散モデル上に拡張し，ソースを含む各ピアのアップロード帯域が b で均一な P2P ビデオストリーミングシステムにおいて，ソースから 2 ホップ以内で n 台の視聴ピアに a 個のストライプを配信するためのアップロード帯域のタイトな下界を導出した．具体的には，安藤らが得た結果は，$a \leq n$ のとき，$b \geq \frac{n}{\lfloor b/a \rfloor} - 1$ を満たすときかつそのときに限りソースから 2 ホップで配信可能であるというものである．また，$a > n$ の場合に関しても，$a \leq n$ のときとは別に，すべてのストライプを配信するため

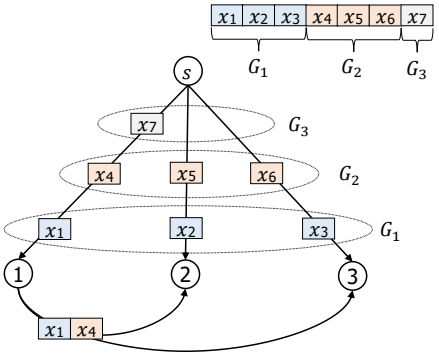


図 1　$b > n$ のとき，ストライプ x_1, x_4 における 2 ホップでの配信の例 ($n = 3$，$b = 7$).

のアップロード帯域のタイトな下界を導出した．しかしながら，2 ホップでの配信のみに着目しているため，3 ホップ以上の場合においては考慮されていない．本稿では，上述の安藤らの結果を 3 ホップ以上の場合に拡張して，任意の b と n に対してすべてのストライプをすべての視聴ピアに対して配信を可能とするための最短ホップ数のタイトな下界を導出する．

3　2 ホップでの配信を実現する場合の一般化

安藤らの結果 [9] に基づいて，以下では任意の b と n に対して，ソースから 2 ホップでの配信を実現するためのアップロード帯域のタイトな下界における一般解を導出する．安藤らが得た結果では，$a \leq n$ のとき，a 個のストライプを 2 ホップで配信可能とするためのアップロード帯域に関する必要十分条件は

$$b \geq \frac{n}{\lfloor b/a \rfloor} - 1$$

である．具体的には，上述の式はソースからストライプを受信した視聴ピアが 2 ホップ目に他の視聴ピアにそのストライプを転送する際の拡散力に関する式である．ゆえに，右辺に現れるアップロード帯域 b はソースピアのアップロード帯域を示し，左辺に現れるアップロード帯域 b は視聴ピアのアップロード帯域を意味する．以下ではソースから b 個のストライプの転送に着目する．つまり，$a = b$ である．換言すると，$b \leq n$ のとき，b 個のストライプを 2 ホップで配信可能とするためのアップロード帯域に関する必要十分条件は

$$b \geq n - 1 \tag{1}$$

となる．このときのアップロード帯域 b は視聴ピアのアップロード帯域である．

$b > n$ のとき，図 1 に示すように，b 個のストライプに対してそれらのストライプを n 個ずつにグループ分けする．得られたグループを $G_1, G_2, \cdots, G_{\lfloor b/n \rfloor}, G_{\lfloor b/n \rfloor+1}$ とする．ここで，$G_{\lfloor b/n \rfloor+1}$ だけはサイズが 0 以上 n 未満であることに注意する．$G_{\lfloor b/n \rfloor+1}$ のサイズを $\tilde{b}$ とすると，

$$\tilde{b} = b - \lfloor \frac{b}{n} \rfloor \times n$$

である．$G_{\lfloor b/n \rfloor+1}$ 以外のグループに含まれているストライプの配信については，以下のような手順で実現される: 1) ソースが n 台の視聴ピアに向けて自身のアップロード帯域を n だけ使ってストライプを送信する; 2) それらのストライプを受け取った各視聴ピアが, 自身のアップロード帯域を $n - 1$ だけ使って転送する．また $\lfloor b/n \rfloor \times n \leq b$ であるため，$G_{\lfloor b/n \rfloor+1}$ 以外のグループに含まれているすべてのストライプの配信を上述の手順でおこなったとしても，そこで使われているアップロード帯域は，いずれのピアにおいても b 以下で抑えられている．各ピアのアップロード帯域に関して，具体的には，ソースは $G_{\lfloor b/n \rfloor+1}$ のサイズ分の余剰アップロード帯域がある．また，視聴ピアは $G_{\lfloor b/n \rfloor+1}$ のサイズ分に加え，$\lfloor b/n \rfloor$ だけの余剰アップロード帯域がある．ソースと視聴ピアの余剰アップロード帯域はそれぞれ $\tilde{b}$ と $\tilde{b} + \lfloor b/n \rfloor$ となる．ゆえに，ソースから 2 ホップで b 個のストライプを配信するためには，これらの余剰アップロード帯域を用いて $\tilde{b}$ 個のストライプを転送する必要がある．換言すると，$b > n$ のとき，b 個のストライプを n 台の視聴ピアにソースから 2 ホップで配信する問題は，各ピアが余剰アップロード帯域を使って n 台の視聴ピアに $\tilde{b}$ 個のストライプをソースから 2 ホップで配信する問題に帰着される．$\tilde{b} = 0$ のとき，自明的にソースから 2 ホップでの配信は可能となるため，以下では，$\tilde{b} \neq 0$ のときに焦点を当てる．

視聴ピアの余剰アップロード帯域を $b_{R(n)}$ とすると，

$$b_{R(n)} = \tilde{b} + \lfloor \frac{b}{n} \rfloor = b - \lfloor \frac{b}{n} \rfloor \times (n-1)$$

である．$b > n$ のとき，式 (1) から，ソースから 2 ホップで n 台の視聴ピアに b 個のストライプを配信するためのアップロード帯域に関する必要十分条件は，

$$\begin{aligned} b_{R(n)} &\geq n - 1 \\ b_{R(n)} - (n-1) &\geq 0 \end{aligned}$$

となる．議論を簡潔にするため，$f(n) \stackrel{\text{def}}{=} b_{R(n)} - (n-1)$ と定義すると，以下の補題が成り立つ．

補題 1. すべてのピアがアップロード帯域 b を利用できると仮定する．$b > n$ かつ $\tilde{b} \neq 0$ のとき，$f(n) \geq 0$ を満たすときかつそのときに限り，n 台の視聴ピアに b 個のストライプをソースから 2 ホップで配信可能である．

$b \leq n$ のとき，$\tilde{b} = b$ であるため，補題 1 は $b \leq n$ のときも含む．ゆえに，以下のような定理が成り立つ．

定理 1. すべてのピアがアップロード帯域 b を利用できると仮定する．$\tilde{b} = 0$ のとき，n 台の視聴ピアに b 個のストライプをソースから 2 ホップで常に配信可能である．$\tilde{b} \neq 0$ のとき，$f(n) \geq 0$ を満たすときかつそのときに限り，n 台の視聴ピアに b 個のストライプをソースから 2 ホップで配信可能である．

図 1 と図 2 では，3 台の視聴ピアに対して 7 個のストライプを配信する例を示している．図 1 では，6 個のストライプ x_1，x_2，$\cdots$，x_6 を 2 つのグループ G_1，G_2 に分けている．これらのストライプに関してはソースから 2 ホップで配信することができる．図 2 では，残りのグループ G_3 におけるストライプ x_7 の配信の例を示している．このとき，$\tilde{b} = 1$，$b_{R(n)} = 3$ と計算されるため，定理 1 より，ソースから 2 ホップで配信することができる．

ここからは，$n - 1$ 台の視聴ピアが自身のアップロード帯域を利用することができ，残りの 1 台の視聴ピアはアップロード帯域を利用できない場合について考える．これまで

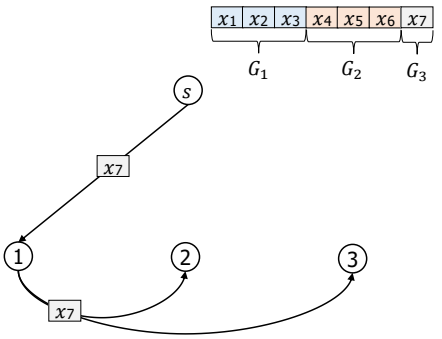


図 2　$b > n$ のとき，ストライプ x_7 における 2 ホップでの配信の例 ($n = 3, b = 7, \tilde{b} = 1, b_{R(n)} = 3$).

の議論をもとに，b 個のストライプに対してそれらのストライプを $n-1$ 個ずつにグループ分けする．得られたグループを $G_1, G_2, \cdots, G_{\lfloor b/(n-1) \rfloor}, G_{\lfloor b/(n-1) \rfloor+1}$ とする．ここで，$G_{\lfloor b/(n-1) \rfloor+1}$ だけはサイズが 0 以上 $n-1$ 未満であることに注意する．$G_{\lfloor b/(n-1) \rfloor+1}$ のサイズを $\tilde{b}'$ とすると，

$$\tilde{b}' = b - \lfloor \frac{b}{n-1} \rfloor \times (n-1)$$

である．各ピアは自身のアップロード帯域を用いて，$G_{\lfloor b/(n-1) \rfloor+1}$ を除く各グループの異なるストライプを転送することができる．このとき，ソースピアと視聴ピアの余剰アップロード帯域は同じであることに注意する．視聴ピアの余剰アップロード帯域を $b'_{R(n)}$ とすると，

$$b'_{R(n)} = \tilde{b}' = \ b - \lfloor \frac{b}{n-1} \rfloor \times (n-1)$$

となる．$\tilde{b}' = 0$ のとき，自明的にソースから 2 ホップでの配信は可能となる．$\tilde{b}' \neq 0$ のとき，式 (1) から，n 台の視聴ピアに b 個のストライプをソースから 2 ホップで配信するためのアップロード帯域に関する必要十分条件は，

$$\begin{aligned} b'_{R(n)} &\geq \ n-1 \\ b'_{R(n)} &- \ (n-1) \geq 0 \end{aligned}$$

となる．議論を簡潔にするため，$g(n) \stackrel{\text{def}}{=} b'_{R(n)} - (n-1)$ と定義すると，以下の補題が成り立つ．

補題 2. ソースピアと $n-1$ 台の視聴ピアがアップロード帯域 b を利用できると仮定する．$b > n$ かつ $\tilde{b}' \neq 0$ のとき，$g(n) \geq 0$ を満たすときかつそのときに限り，n 台の視聴ピアに b 個のストライプをソースから 2 ホップで配信可能である．

補題 2 は $b \leq n$ のときも含むため，以下の定理が成り立つ．

定理 2. ソースピアと $n-1$ 台の視聴ピアがアップロード帯域 b を利用できると仮定する．$\tilde{b}' = 0$ のとき，n 台の視聴ピアに b 個のストライプをソースから 2 ホップで常に配信可能である．$\tilde{b}' \neq 0$ のとき，$g(n) \geq 0$ を満たすときかつそのときに限り，n 台の視聴ピアに b 個のストライプをソースから 2 ホップで配信可能である．

4　ストライプを配信可能なタイトな下界

本章では，b が偶数の場合において，n 台の視聴ピアに b 個のストライプを配信するときにかかる最大ホップ数のタイトな下界を導出する．初めに $b \leq n$ の場合（4.1 章-4.4 章）を考え，次に $b > n$ の場合（4.5 章）を考える．

4.1　ベースとなる考え

深さ k の各 b 個のスパニングツリーのすべてに参加できる視聴ピアの最大数を B とすると，$B = \sum_{i=0}^{k-1} b^i$ である．また，各スパニングツリーにおいて必要となる内部ピアの数を $\hat{n}$ とすると，$\hat{n} = \sum_{i=0}^{k-2} b^i$ である．このとき，以下の補題が成り立つ．

補題 3. $n = B$ のとき, n 台の視聴ピアに対して b 個のストライプを k ホップで配信可能である．

証明. $n = B$ のとき, あるストライプをすべての視聴ピアに k ホップで配信するためには，$\hat{n}$ 台の内部ピアが自身のアップロード帯域を使い切って b 台の視聴ピアにストライプを転送する必要がある．ここで，

$$\frac{B}{\hat{n}} = \frac{\hat{n}b+1}{\hat{n}} = b + \frac{1}{\hat{n}} > b$$

であるため，そのようなサイズ $\hat{n}$ のピア群を B 個のピア群の中に重なりなく b 個とることができる．よって題意が成立する．□

また，次のような場合にも，b 個のストライプを n 台の視聴ピアに k ホップで配信することができる．

補題 4. $1 \leq p < \hat{n}$ とする．このとき $n = B - pb$ ならば，n 台の視聴ピアに対して b 個のストライプを k ホップで配信可能である．

証明. $n = B$ のとき，$\hat{n}$ 台の内部ピアは自身のアップロード帯域をすべて使い切る必要がある．しかしながら，$n = B - pb$ のとき，$\hat{n}$ 台の内部ピアのうちのいくつかのピアは自身のアップロード帯域を用いて転送を行う必要がない．ゆえに，これらの内部ピア数を p とすると，配信可能なストライプ数は，

$$\frac{B-pb}{\hat{n}-p} = \frac{(\hat{n}b+1)-pb}{\hat{n}-p} = b + \frac{1}{\hat{n}-p} > b \qquad (2)$$

となり，サイズ $\hat{n}-p$ のピア群を $B-pb$ 個のピア群の中に重なりなく b 個とることができる．ゆえに題意が成立する．□

補題 3 より，$n = \hat{n}$ のとき，n 台の視聴ピアに対して b 個のストライプを $k-1$ ホップで配信可能である．また，補題 4 より，$n = \hat{n}$ のとき，

$$\begin{aligned} \hat{n} &= \ B - pb \\ pb &= \ B - \hat{n} \\ pb &= \ b^{k-1} \\ p &= \ b^{k-2} \end{aligned}$$

となる．ゆえに，以下の定理が成り立つ．

定理 3. $0 \leq p < b^{k-2}$ とする．このとき $n = B - pb$ ならば，n 台の視聴ピアに対して b 個のストライプの配信を可能とするための最短ホップ数は k である．

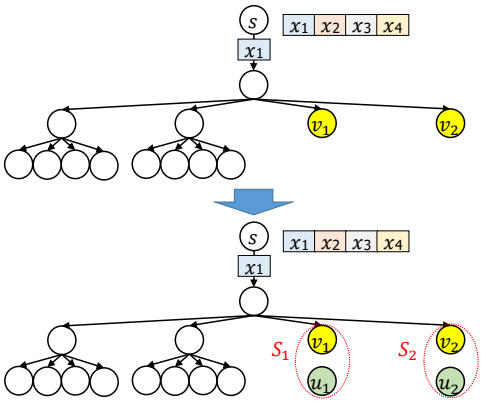


図 3　ストライプ x_1 のスパニングツリーに対する q^* 台のピアの割り当て ($n=15, b=4, B=21, \hat{n}=5, p^*=2, q^*=2, k=3$).

4.2　$B-(p+1)b<n<B-pb$ のとき

以下では，$B-(p+1)b<n<B-pb$ の範囲において，配信可能なストライプ数の上限がどのように変化するのかについて考える．ある $0\le p<b^{k-2}$ と $1\le q<b$ に関して，$n=B-pb-q$ とする．これは，$\hat{n}$ 台の内部ピアのうち，p 台のピアが自身のアップロード帯域を用いて転送を行う必要がなく，残りの $\hat{n}-p$ 台の内部ピアのうちのひとつは，余剰アップロード帯域 q を使い切らずに残す．このとき，配信可能なストライプ数は，

$$\frac{B-pb-q}{\hat{n}-p}=\frac{(\hat{n}b+1)-pb-q}{\hat{n}-p}=b-\frac{q-1}{\hat{n}-p}$$

となる．$q=1$ のとき，n 台の視聴ピアに対して b 個のストライプを k ホップで配信可能となる．ゆえに，以下の補題が成り立つ．

補題 5. $0\le p<b^{k-2}$ かつ $1\le q<b$ とする．このとき $n=B-pb-1$ ならば，n 台の視聴ピアに対して b 個のストライプを k ホップで配信可能である．

4.3　配信可能なストライプ数の上限の改良

以下では，$n=B-pb-q$ かつ $2\le q<b$ のとき，n 台の視聴ピアが受信可能なストライプ数の上限を改善する．ここで，$n=B-pb-q$ を $n=B-p^*b+q^*$ として，$1\le p^*\le b^{k-2}$ と $1\le q^*<b$ に関する表現に変更する．補題 5 より，$n=B-pb-q$ のとき，$q=1$ ならば常に b 個のストライプを k ホップで配信可能であるため，一般性を失うことなく $1\le q^*<b-1$ とする．

n 台の視聴ピアの集合を S として，その中に p^* 個の部分集合 $S_1, S_2, \cdots, S_{p^*}$ を形成する．n 台の視聴ピアが受信可能なストライプ数の上限を改善するための基本的なアイデアは以下のような手順に従う: 1) まず初めに $n=B-p^*b$ において，各ストライプに対する b 個のスパニングツリーを構築する; 2) 各スパニングツリーにおいて，自身のアップロード帯域を用いて転送を行う必要がない p^* 台の内部ピアをすべて同じピアに固定する; 3) 固定した p^* 台のピアを中心として，p^* 個の部分集合 $S_1, S_2, \cdots, S_{p^*}$ を形成する; 4) 残りの q^* 台のピアを加える際は，p^* 個の部分集合に均等に分割する．図 3 は，15 台の視聴ピアに対して 4 個のストライプを配信するときの例を示している．図 3 の上図に示すように，まず $13(=B-p^*b)$ 台のピアに関するスパニングツリーを構築する．これらのピアに関しては，$3(=k)$ ホップで $4(=b)$ 個のストライプを配信可能である．各スパニングツリーにおいて，自身のアップロード帯域を使用しない $2(=p^*)$ 台の内部ピア v_1 と v_2 は固定される．式 (2) より，1 台の視聴ピアは自身のアップロード帯域を使用しないことに注意して，このピアをピア v_1 の位置におく．換言すると，S_1 内のすべてのピアは自身のアップロード帯域を使用できるのに対して，もう一方の部分集合 S_2 では 1 台のピアが自身のアップロード帯域を使用できない．次に，内部ピア v_1, v_2 を中心として 2 つの部分集合 S_1, S_2 を形成する．最後に，図 3 の下図に示すように，残りの $2(=q^*)$ 台の視聴ピアを $2(=p^*)$ 個の部分集合に均等に分割する．一般的に，各部分集合内の要素の最大数は $1+\lceil\frac{q^*}{p^*}\rceil$ であり，要素の最小数は $1+\lfloor\frac{q^*}{p^*}\rfloor$ である．

各ストライプはすべての p^* 個の部分集合に $k-1$ ホップでたどり着くことができる．各部分集合内の視聴ピアは，同じ集合内のピア間でのみストライプを転送することで，すべてのストライプを受信するようにする．これは，各部分集合内のすべてのピアが集合内のピア間のみの転送で 1 ホップですべてのストライプを受信可能であれば，k ホップで b 個のストライプを n 台の視聴ピアに配信可能であることを意味する．換言すると，この問題は $k-2$ ホップ後に各部分集合内のすべてのピアに 2 ホップで b 個のストライプを配信する問題に帰着される．以下では，$k-2$ ホップ後における p^* 個の部分集合内での配信に焦点を当てる．

4.3.1　$p^*\ge q^*$ のとき

$p^*\ge q^*$ のとき，各部分集合内の要素数は高々 2 個であるため，$k-2$ ホップ後に各部分集合内においてソースから 2 ホップでの配信は以下のような手順に従って実現することができる: 1) 各部分集合において，あるピアは $k-1$ ホップで b 個のスパニングツリーからすべてのストライプを受信する; 2) 同じ集合内にピアが存在する際には，ストライプを受信したピアはもう一方のピアに対してすべてのストライプを転送する．ゆえに，以下の補題が成り立つ．

補題 6. $n=B-p^*b+q^*$ のとき，$p^*\ge q^*$ ならば，n 台の視聴ピアに対して b 個のストライプを k ホップで配信可能である．

4.3.2　$p^*<q^*$ のとき

$p^*<q^*$ のとき，p^* 個の部分集合 $S_1, S_2, \cdots, S_{p^*}$ を以下のように S_A, S_B, S_C の 3 種類に分類することができる: 1) S_A では，すべてのピアが自身のアップロード帯域を使用できるものとし，$|S_A|=1+\lceil\frac{q^*}{p^*}\rceil$ である; 2) S_B では，1 台のピアのみが自身のアップロード帯域を使用できないものとし，$|S_B|=1+\lfloor\frac{q^*}{p^*}\rfloor$ である; 3) S_C では，1 台のピアのみが自身のアップロード帯域を使用できないものとし，$|S_C|=1+\lceil\frac{q^*}{p^*}\rceil$ である．議論を簡潔にするため，$|S_A|, |S_B|, |S_C|$ をそれぞれ n_A, n_B, n_C とおく．以下では，定理 1 と定理 2 より，b 個のストライプを n 台の視聴ピアに k ホップで配信可能であるか否かを検証する．

$p^*=1$ のとき，考慮する必要があるのは S_A のみであり，S_A 内のすべてのピアは自身のアップロード帯域を利用可能である．このとき $n_A=1+q^*$ である．したがって，以下の補題が成り立つ．

補題 7. $p^*=1$ のとき，$n=1+q^*$ における定理 1 を満たすときかつそのときに限り，n 台の視聴ピアに対して b 個のストライプを k ホップで配信可能である．

図 4 は $b=4, n=19$ のときのスパニングツリーの一例を

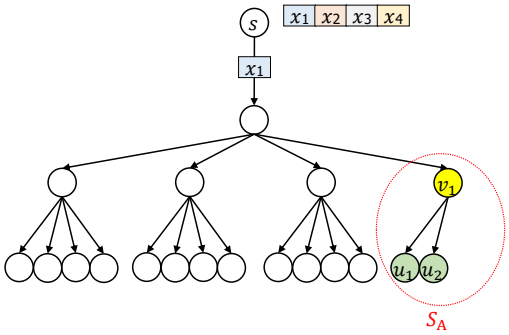


図 4　$p^* < q^*$ のときのストライプ x_1 に対するスパニングツリーにおける q^* 台のピアの割り当ての一例 ($n = 19$, $b = 4$, $B = 21$, $\hat{n} = 5$, $p^* = 1$, $q^* = 2$).

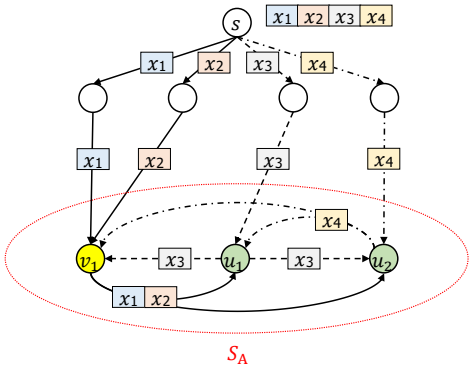


図 5　S_1 内の 3 台のピアに対するストライプの転送の一例 ($n = 19$, $b = 4$, $B = 21$, $\hat{n} = 5$, $p^* = 1$, $q^* = 2$).

示している．図のように，$p^* = 1$　かつ $p^* < q^*$ のとき，各スパニングツリーにおいて集合 S_A を形成することができる．このとき，ソースから 1 ホップ後に S_A 内のピアが 2 ホップで 4 個のストライプを受信可能であるならば，ソースから 3 ホップですべての視聴ピアにすべてのストライプを配信可能となる．図 5 に示すように，S_A 内の 3 台のピア v_1, u_1, u_2 は自身のアップロード帯域を利用可能であるため，ソースから 1 ホップ後に 2 ホップで S_A 内のすべてのピアに 4 個のストライプをすべて転送可能である．この例ではソースから 3 ホップですべての視聴ピアにすべてのストライプを配信可能である．

ここからは $p^* \geq 2$ のときを考える．q を p で割ったときの余りを R とすると，$R = q^* - \lfloor q^*/p^* \rfloor \times p^*$ である．$p^* \geq 2$ かつ $R \leq 1$ のとき，S_A と S_B の 2 種類の集合を考慮する必要がある．S_A 内のすべてのピアは自身のアップロード帯域を利用可能であるのに対して，S_B 内では，n_B 台のうちの 1 台のピアのみが自身のアップロード帯域を使用できない．これらを踏まえて，以下の補題が成り立つ．

補題 8. $R \leq 1$ のとき，$n = n_A$ における定理 1 かつ $n = n_B$ における定理 2 を満たすときかつそのときに限り，n 台の視聴ピアに対して b 個のストライプを k ホップで配信可能である．

$R \geq 2$ のとき，S_A, S_B, S_C の 3 種類の集合を考慮する必要がある．S_A 内のすべてのピアは自身のアップロード帯域を利用可能であるのに対して，S_B 内では，n_B 台のうちの 1 台のピアのみが自身のアップロード帯域を使用できず，S_C 内では，n_C 台のうちの 1 台のピアのみが自身のアップロード帯域を使用できない．これらを踏まえて，以下の補題が成り立つ．

補題 9. $R \geq 2$ のとき，$n = n_A$ における定理 1 かつ $n = n_B$ における定理 2 かつ $n = n_C$ における定理 2 を満たすときかつそのときに限り，n 台の視聴ピアに対して b 個のストライプを k ホップで配信可能である．

4.4　配信可能なホップ数のタイトな下界

任意の b と n に対して，定理 3 と補題 5 - 9 のいずれかを満たすならば，n 台の視聴ピアに対して b 個のストライプを配信可能な最大ホップ数のタイトな下界は k となり，そうでなければ，配信可能とするために各スパニングツリーにおける最大の深さを k から k^* に増やす必要がある．以下では，すべての視聴ピアに b 個のストライプを k ホップで配信できない場合，すべてのストライプが配信可能となる各スパニングツリーにおける最大ホップ数のタイトな下界 $k^*(> k)$ を導出する．なお，ここでは b が偶数のときのみについて考える．

基本的なアイデアは，各ピアがあるストライプの転送に対して用いるアップロード帯域を減らすことによって，各スパニングツリーにおける最大の深さを増加させる．このとき，各ピアがあるストライプの転送に対して用いるアップロード帯域を b^* とする．ここでは，一般性を失うことなく $b^* < b$ であるとする．以下では，視聴ピア数 n における表現を $n = B - p^*b + q^*$ から $n = B^* - p^*b^* + q^*$ に変更する．ただし，$B^* = \sum_{i=0}^{k^*-1} b^{*i}$ である．b^* の値が大きくなればなるほど，各スパニングツリーにおける最大の深さ k^* は小さくなるため，これがすべてのストライプを配信可能とするホップ数のタイトな下界につながることに注意する．

b が偶数のとき，ある 1 つのスパニングツリーに対して，$\lfloor b/b^* \rfloor$ 個のストライプを n 台の視聴ピアに同時に転送することができる．ゆえに，b^* 個のスパニングツリーにおいて，

$$\lfloor \frac{b}{b^*} \rfloor \times b^* = b$$

を満たすならば，n 台の視聴ピアに b 個のストライプを k^* ホップで配信可能となる．b の正の約数の集合を D_b とする．上記の式から以下の補題が成り立つ．

補題 10. $n = B^* - p^*b^* + q^*$ に関して，$b^* \in D_b$ と仮定する．このとき，n 台の視聴ピアに対して b^* 個のストライプを k^* ホップで配信可能であるならば，n 台の視聴ピアに対して b 個のストライプを k^* ホップで配信可能となる．

さらに，b が偶数のとき，n 台の視聴ピアに対して b^* 個のストライプを配信可能とする b^* の値が D_b において最大であるならば，各スパニングツリーにおける最大の深さ k^* がホップ数のタイトな下界となる．ゆえに，以下の定理が成り立つ．

定理 4. $b \leq n$ かつ $n = B^* - p^*b^* + q^*$ において，n 台の視聴ピアに対して b 個のストライプを k ホップで配信不可能であると仮定する．b が偶数のとき，k^* ホップで n 台の視聴ピアに b 個のストライプを配信可能とする $b^*(\in D_b)$ の値が最大かつそのときに限り，n 台の視聴ピアに対して b 個のス

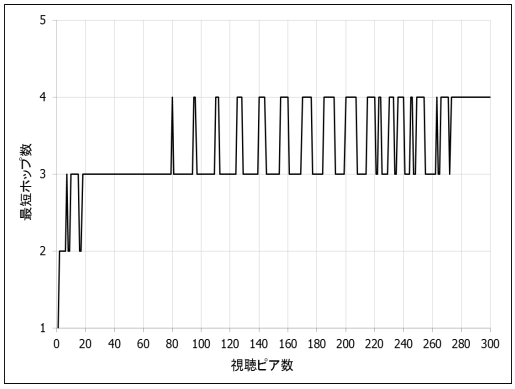


図 6　$b = 16$ のときの n の変化に伴う最短ホップ数の推移.

トライプを配信可能とする最短ホップ数は k^* である.

4.5　$b > n$ のとき

以下では，$b > n$ のとき，n 台の視聴ピアに対して b 個のストライプを配信可能にするための最短ホップ数を求める．まず，定理 1 より，n 台の視聴ピアに対して b 個のストライプをソースから 2 ホップで配信可能か否かを検証する．もし，ソースから 2 ホップでの配信が不可能であるならば，4.4 章と同様の手順で最短ホップ数を求める．ここで，b^* のとり得る範囲は $b \leq n$ の場合とは異なることに注意する．つまり，ホップ数を k 以上に増加させるために，とり得る範囲を $b^* < b$ から $b^* < n - 1$ に変更する必要がある．

5　おわりに

本稿では，マルチスパニングツリーを介した P2P ビデオストリーミングシステムを考える．この P2P システムでは，ソースピア s を含む $n + 1$ 台のピアのアップロード帯域は b で均一であるとし，ソースピア s が n 台の視聴ピアに対して b 個のストライプを配信する．我々はすべての視聴ピアに対して b 個のストライプを配信可能にするための b 個のスパニングツリーにおける最大の深さに着目する．具体的には，b が偶数の場合において，任意の b と n の組み合わせに対して，各スパニングツリーにおける最大の深さのタイトな下界を導出した．図 6 では，$b = 16$ のときの n の変化に伴う最短ホップ数の推移を示している．図より，ピア数の変化に伴い，配信可能な最短ホップ数も激しく変化していることがわかる．今後の課題としては，

- b が奇数のとき，任意の b と n の組み合わせにおける最短ホップ数を導出すること．
- 配信可能なストライプ数を b 個から任意の a 個 $(a \leq b)$ に変更して，同様の議論を行うこと．
- クラウドアシスト型 P2P システムを用いて，ソースピアのアップロード帯域を仮想的に増加させることによって，同様の議論を行い，ピア数の変化に伴う最短ホップ数の変動を抑制させること．
- 配信を行うソースピアが複数存在する場合について，同様の議論を行うこと．

などが挙げられる．

参考文献

[1] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh. “SplitStream: High-Bandwidth Multicast in Cooperative Environments.” In *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP)*, 2003, pages 298–313.

[2] M. Castro, P. Druschel, A.-M Kermarrec and A. Rowstron. “SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure.” *IEEE J.Sel. A. Commun.*, 20(8): 1489–1499, 2006.

[3] G. Bianchi, N. B. Melazzi, L. Bracciale, F. Lo Piccolo, and S. Salsano, “OPSS: An Overlay Peer-to-Peer Streaming Simulator for Large-Scale Networks,” *IEEE Trans. on Parallel and Distributed Systems*, 21(6): 857-871, 2010.

[4] Y. Guo, C. Liang, and Y. Liu, “AQCS: Adaptive Queue-Based Chunk Scheduling for P2P Live Streaming,” *Proc. IFIP Networking 2008, Singapore*, 2008, pages 433-444.

[5] H. Shen, Y. Lin and J. Li, “A Social-Network-Aided Efficient Peer-to-Peer Live Streaming System.” *IEEE/ACM Trans. on Networking*, 23(3): 987-1000, 2015.

[6] C. Zhao, J. Zhao, and X. Lin, “Capacity of P2P On-Demand Streaming With Simple, Robust, and Decentralized Control.” *IEEE/ACM Trans. on Networking*, 24(5): 2607-2620, 2016.

[7] G. Dan, V. Fodor and I. Chatzidrossos. “On the Performance of Multiple-Tree-Based Peer-to-Peer Live Streaming.” In *Proc. of the 26th IEEE Int’l Conf. on Computer Communications (INFOCOM)*, 2007, pages 2556–2560.

[8] Y. Zhao, Y. Liu, C. Chen and J. Zhang. “Enabling P2P One-View Multiparty Video Conferencing.” *IEEE Trans. on Parallel and Distributed Systems*, 25(1): 73–82, 2014.

[9] H. Ando and S. Fujita. “Tight Bounds for Two-Hop Delivery in Homogeneous P2P Video Streaming Systems.” In *Proc. 4th International Symposium on Computing and Networking (CANDAR)*, 2016, pages 1–8.

WebRTC を用いたツインビュー型 P2P ビデオ会議システム

広島大学大学院工学研究科　木村友彦
広島大学大学院工学研究院　　藤田聡

概要

WebRTC を利用した P2P 型ビデオ会議システムを P2P ネットワーク上に実現する方法を提案する．提案システムではシステムのスケーラビリティを確保するため，各ユーザが一度に視聴できるビューの数を 2 つに制限している．ユーザが視聴するユーザを指定するビューと，現在発言しているユーザに自動的に切り替わるビューとを使うことで，各ユーザは自分が注目するユーザの映像を確認しつつ，最新の発言についても注目することができる．

1. はじめに

安定したインターネット回線の普及と通信環境の改善によって，ネットワークを介して会議を行うビデオ会議が広く行われるようになった．物理的に離れた場所にいても会議を行うことができるビデオ会議は，会議のみならず遠隔教育や遠隔医療，モニタリングなどの遠隔コミュニケーションにも用いられている．ビデオ会議システムには，専用ハードウェアとソフトウェアで構成されたテレビ会議システムと，汎用 PC にソフトウェアをインストールして利用する Web 会議システムとがある．

また近年 WebRTC という Web 技術を用いたサービスが増えている．WebRTC はブラウザでリアルタイムコミュニケーションを行うための API・プロトコル群である．WebRTC を使うことでプラグインを導入することなく，映像や音声データをはじめとする様々なデータのやりとりを行うことができる．WebRTC の利用例として，Web サイトにアクセスするだけでビデオチャットを利用できるシステムや，専用ソフトウェアをインストールせずとも動画やファイルなどのコンテンツを共有することができるシステムがある．

本研究では Web 会議システムのうち，専用ソフトウェアを使用せず，また専用サーバも利用しない P2P 型のシステムを構築する．実装するシステム(提案システム)では参加者すべてがカメラから取得した映像をもっており，各参加者は同時に発言しないものとする．

2. 関連研究

Web 会議システムには，ストリームをサーバ経由で配信する方法と，サーバを経由せず参加者同士で配信する方法とがある．サーバ経由で配信する場合は比較的安定したサービスを提供することができるが，高負荷に耐えることができるサーバを構築して運用する必要がある．一方でサーバを経由しない場合は，高価なサーバを用意する必要はないが，サービスが各端末の能力に依存するため安定したサービスを提供することが難しい．サーバを経由する手法として MCU や SFU を使う手法が提案されている[1]．またサーバを経由しない手法が談らの研究[2]，信家らの研究[3]で提案されている．

サーバを経由する手法では，各ピアがストリームを中央サーバに送信し，中央サーバがストリームを各ピアへ配信する．MCU(Multipoint Control Unit)ではサーバがストリームにコーディックの変換や音量・画質の調整などの編集処理を行い，

各ユーザに配信する．また SFU(Selective Forwarding Unit)では，編集処理を行わずそのまま配信する．サーバを経由する場合，処理の負荷やネットワークトラフィックの集中がサーバで起こるため，十分な性能のサーバを構築・管理する必要がある．またサーバはすべての映像を視聴できるためプライバシーの問題もある．

サーバを経由しない手法では各ユーザがストリームを交換する．しかしすべての映像を得る場合，ピア同士の接続が完全結合となり，非常に負担が高い．そこで，ストリームをリレーして多くのピアにストリームを届ける手法が，談らの研究や信家らの研究で提案されている．これらの手法ではピア同士で木構造の配信経路を構築し，親から子へストリームをリレーして配信を行う．談らの研究では深さ 3, 子の数 3 の場合が考察されており，また信家らの研究では深さをb, 子の数をB_iと一般化して考察されている．

3. 提案システム

3.1. 概要

提案システムは専用ソフトウェアをインストールする必要なく Web ブラウザのみで利用でき，専用サーバを利用しない P2P 型のシステムとして実装している．実装の詳細は 3.3 節で述べる．

また提案システムでは一度に視聴できるビューの数を 2 つに制限している．これは参加者が多人数である場合でもネットワーク帯域や端末のリソースを際限なく利用しないためである．すべてのユーザが映像を相互に送受信する場合，トラフィック量は参加人数の 2 乗に比例して増えることとなり，ネットワークトラフィックが飽和する可能性がある．また多くのユーザの映像を持っていても，ユーザはすべてのビューを同時に見ることはできないため，端末のメモリや CPU リソースを無駄にしてしまうこととなる．

そのため，ビューの数を制限することによって，端末のメモリや CPU リソースを節約することができる．しかし視聴できるビューが 2 つであることで，すべてのユーザの映像を見ることができないため，今誰がどんな発言をしているかが一見してわからない．そこで提案システムではビューを 2 種類にわけ，それぞれメインビューとサブビューとする．メインビューにはユーザが指定したユーザの映像が表示される．ユーザは自由なタイミングでビューに表示するユーザを切り替えることができる．一方でサブビューには一番最近に発言を開始したユーザの映像が表示される．ビューはシステムによって自動的に切り替わる．発言の検知方法については 3.2 節で述べる．このような 2 種類のビューを利用することで，各ユーザは自分が注目するユーザの映像を確認しつつ，最新の発言についても注目することができる．

3.2. 発言の検知方法

発言開始は発言者側で検知する．システムはマイクから一定時間毎に音量を取得し，その音量が事前に設定した閾値を超えた場合を「発声」とする．閾値を設定するのは風や吐息など，発言と関係ないノイズを防ぐためである．また「発声」していない状態で発声した場合は，すぐに発言開始とし，各参加者に通知するとともにストリームを送る．また一定時間「発声」がなかった場合は発言終了とし，参加者に通知する．閾値は事前に発言中の音量を測定した結果をもとに設定した．マイクロフォンに向かって話しかけ，話し始めと話し終わりにボタンを押し，その動作を記録する．話し始めから話し終わりまでの時間は話しているということなので，話している最中の音量の最小値を発声の閾値とした．

3.3. システムの実装

提案システムは専用ソフトウェアをインストールする必要なく Web ブラウザのみで利用できる

ように Web 標準技術を利用して実装している．映像やデータの送受信には WebRTC を，音声の解析には Web Audio API を用いている．開発を容易にするため，NTT コミュニケーションズ社による WebRTC の既存ライブラリである SkyWay を利用した．SkyWay を利用することで，各ピアに割り振ったユニークな ID によってピア同士の接続・切断操作を容易に行えるほか，通信中の様々なイベントに対してのハンドリングができる．また NAT トラバーサルに利用する STUN サーバや，P2P 通信ができない場合に利用する TURN サーバ，ピア同士の接続情報を交換するシグナリングサーバとしての機能も提供されている．SkyWay は 2017 年 7 月現在，無料で利用することができる．

提案システムでは，ピア同士の P2P 通信に SkyWay を利用した WebRTC による通信チャネルを利用する．WebRTC の通信チャネルには，映像や音声を送受信するメディアチャネルと，任意のデータをバイナリで送受信するためのデータチャネルとがある．データチャネルは，ストリームの送信要求や発言開始の通知など，各ピアが互いに情報を伝え合うために用いる．またメディアチャネルは，実際のカメラの映像を送受信するために用いる．

スムーズなビューの切り替えのために，事前にいくつかのストリームを受け取っておく．受け取ったストリームは画面に描画させずに蓄えておき，そのユーザの映像を要求されたときに描画する．これをキャッシュと呼ぶこととする．キャッシュを用いることにより，ストリームを要求して実際に送信するというやり取りが省略され，すばやいビューの切り替えが可能になる．

発言の検知は Web Audio API の AnalyzerNode を用いた．オーディオコンテキスト内でストリームと AnalyzerNode を接続し，一定時間毎に分析する．取得した時間領域の音量を平均した値と閾値を比較し，発火かどうかを発火履歴に追加する．発火履歴はキュー型のデータ構造で固定長とする．キュー内に発火が一つも格納されていない場合は，一定時間発火していなかったことになるので，発言が終了したことがわかる．

3.4. システムのインタフェース

提案システムにアクセスすると，システムに自動的に参加する．カメラアクセスを許可すると MyVideo に自分のカメラの映像が映る．また現在参加中のピアが一覧で表示され，クリックするとメインビューがそのピアの映像に切り替わる．また発言を始めたピアがいればサブビューがそのピアの映像に切り替わる．自分が発言中の場合は，スピーカーのアイコンが表示されるため，自分が発言中と認識されているかがわかる．またチャットフィールドを使うことで，参加者全体とテキストチャットを行うこともできる．

提案システムを実際に使用している画面を図 1,2 に示す．Windows10 の Google Chrome でシステムを利用している様子が図 1 で，Android の Google Chrome でシステムを利用している様子が図 2 である．

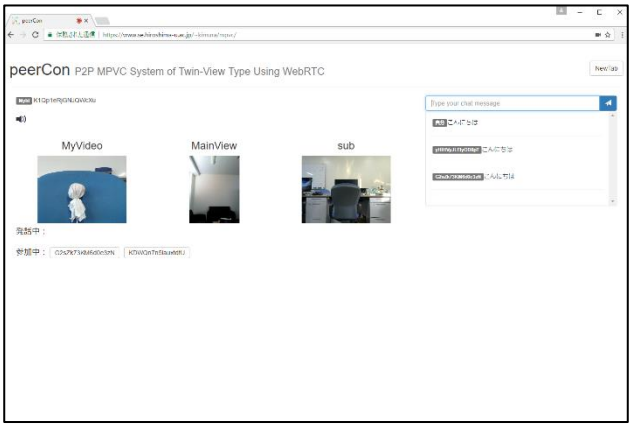


図 1 提案システムに Windows10 の Google Chrome でアクセスした様子

図 2 提案システムに Android の Google Chrome でアクセスした様子

4. 評価

提案システムの性能を評価するため，ストリームの遅延を測定する実験とビューの切り替えにかかる時間を測定する実験とを行った．それぞれ 4.1 節， 4.2 節で詳しく述べる．評価の基準として 300ms という基準を設定した．これは人間の視覚や聴覚の反応速度が，反応時間 - 脳科学辞典[4]によると 150～300ms，体力科学[5]によると 130～180ms であったためである．

4.1. ストリームの遅延測定実験

ストリームが送信されてから実際に相手に届くまでの時間をストリームの遅延時間として評価する．各ピアは時計を同期させた 2 台の PC 上でタブとして存在し，通信を行う．送信ピアは UNIX 時間をアニメーションのように映像ストリームとして配信し，受信ピアは受け取ったストリームを再生し，現在の時刻とともにスクリーンショットとして保存する．ストリームの内容は送信側の時計で，再生されるときとの時差がストリームの遅延時間として計測できる．キャッシュピア数を変化させ，ミリ秒単位で遅延の平均を計測した．

結果は図 3 のようになった．キャッシュピア数が増えても，遅延は 50ms を超えず，ほぼ一定の遅延を保っている．つまりキャッシュしているピアの数によって再生されるストリームの遅延は変化せず, 300ms に比べると小さなものでしかない．

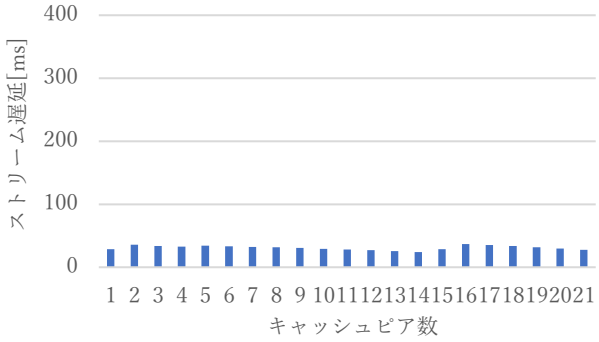


図 3 ストリームの遅延時間

4.2. ビューの切り替え時間測定実験

ストリームを受信してから実際にビューが切り替わるまでの時間を測定し，評価する．キャッシュがなく，ビューの切り替えの度にストリームを要求する場合，キャッシュを使ってすぐに切り替える場合，保持しているキャッシュにないピアのストリームを要求して再生する場合，の 3 つの場合について実験を行う．ピアには配信ピア S1, S2, 参加ピア C_n，観測ピア E の 4 種類があり，それぞれ以下のような働きをする．

・配信ピア 1(S1)：E にストリーム 1 を提供する
・配信ピア 2(S2)；E にストリーム 2 を提供する
・参加ピア n(C_n)：S1, S2 以外のピア
・観測ピア(E)　：S1, S2, C_1～C_n からストリームを受け取り再生する．また時間の測定を行う

再生するストリームは S1 から S2 に切り替わり，ストリームの受信から実際にビューが切り替わるまでにかかる時間をミリ秒単位で計測する．

結果は図 4 のようになった．

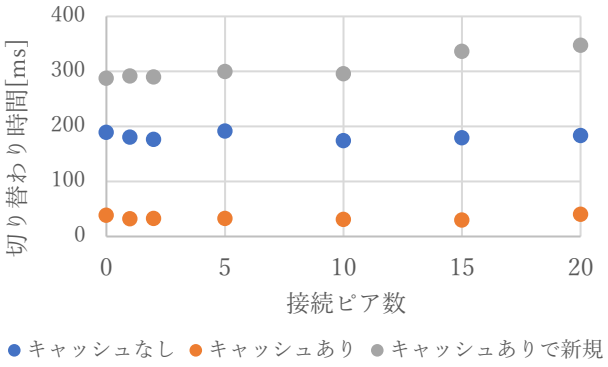


図 4 ビューの切り替わり時間

接続ピア数によらず，それぞれの場合について切り替えにかかる時間は一定であった．しかしどの場合でも同様にストリームを受信したタイミングでビューを切り替えているにもかかわらず，切り替わりにかかる時間が異なった．キャッシュがある場合は，50ms 以下で切り替えることができるが，ない場合は 200ms 程度かかる．またキャッシュを持っていても，指定されたストリームがない場合は余計に時間がかかった．この結果はキャッシュによる効果を示しているほかに，できるだけ指定されるユーザのストリームを予測して保持することの重要性を示している．将来的に使われるユーザのストリームを優先してキャッシュできる仕組みについては今後の課題とする．

5. おわりに

本研究では WebRTC を利用した P2P 型ビデオ会議システムを P2P ネットワーク上に実現する方法を提案した．提案システムではネットワーク帯域などのシステムリソースを節約しシステムのスケーラビリティを確保するため，各ユーザが一度に視聴できるビューの数を 2 種類に制限した．ユーザが視聴するユーザを指定するビューと，現在発言しているユーザに自動的に切り替わるビューとを使うことで，各ユーザは自分が注目するユーザの映像を確認しつつ，最新の発言についても注目することができるシステムを実装した．

今後の課題として，キャッシュ方式の最適化や人気のユーザへの負担の集中を解決することが挙げられる．

参考文献

[1] Tarjei Klinge Husøy, Topology in WebRTC Services, 2015

[2] 談エン，新城靖，李咏，佐藤聡，中井央： 分散型 Web ブラウザにおける遠隔会議システムの実装， 研究報告システムソフトウェアとオペレーティング・システム（OS）, 2015.

[3] 信家悠司： WebRTC を用いた分散型ビデオ会議システムの構築と評価， 広島大学卒業論文, 2016.

[4] 新美亮輔, 横澤一彦：反応時間 - 脳科学辞典, https://bsd.neuroinf.jp/wiki/%E5%8F%8D%E5%BF%9C%E6%99%82%E9%96%93

[5] 越川裕正：新しい反応時間測定装置について, 体力科学, Vol.7, No.2, P88-93, 1958.

BitTorrent 型並列ダウンロードシステムにおける ニューラルネットワークを用いた効率的な利得の獲得方法

内藤 世啓
広島大学大学院工学研究科

藤田 聡
広島大学大学院工学研究院

概要

BitTorrent 型並列ダウンロードシステムにおいて, システム内の各ピアが自律分散的に自身の利得を最大化するような行動を獲得する方法を提案する. 提案手法では, まずシミュレーションによって, ピア自身の利得を最大化するような行動をモンテカルロ法によって探索する. そして, シミュレーションから得られた大量のデータサンプルをニューラルネットワークによって学習し, ピアが観測した隣接ピアとのファイルのやり取りの履歴から, 最適な行動を導き出せるような関数を獲得する. 提案手法の性能はシミュレーションによって評価される.

1 はじめに

近年, P2P(Peer-to-Peer) 技術の進化により, ネットワークを介して大容量のデータをやり取りすることも多くなってきている.

この P2P 技術を用いたファイル共有システムのひとつに BitTorrent[1], [2] の並列ダウンロードシステムがある. BitTorrent では, ファイルをピースと呼ばれる固定長の単位に分割し, ピア間でピースのやり取りを行っている. また, 完全なファイルを持っているピアをシーダ, ファイルが完成していないピアをリーチャと呼ぶ. シーダはピースをダウンロードする必要がないため, アップロードのみを行い他のピアがファイルを完成させることに貢献する. リーチャは, ピースのアップロード・ダウンロードを行い自身のファイルの完成を目指す.

このシステムでは, リーチャが他のリーチャへアップロードを促すための仕組みとして, しっぺ返し戦略 (tit-for-tat) と, ランダムにアップロード先を選択する楽観的アンチョークと呼ばれるピア選択アルゴリズムが採用されている. しっぺ返し戦略とは, 一定期間のタイムスロットを設けて, 直前のタイムスロットで多くダウンロードした隣接リーチャへアップロードを行うというものである. しかし, しっぺ返し戦略では短期間の履歴からアップロード先を選択するため, ピア自身にとって有益な隣接ピアとの関係が簡単に途切れてしまい, 有益でないピアをアップロード先として選択してしまうことがある. ピアは本来自身の欲しいピースをできるだけ多くの隣接ピアから得たいはずであるが, しっぺ返し戦略では短期的にピースを獲得できるだけで, 長期的なピースの獲得方法としては不十分である.

本稿では, 隣接ピアとの長期的な過去の履歴から「できるだけ少ないアップロード量で, より早くシーダになる」という条件を満たす最適な行動を導き出せるような関数を作成する方法について提案する.

2 システムモデル

本稿では, ピアの集合を $\mathbf{P} = \{p_1, p_2, \ldots, p_N\}$ として, BitTorrent 型並列ダウンロードシステムを単純化したシステムモデルを用いる. このとき, 全てのピアの帯域は均一であり, オーバーレイネットワーク上のリンクによって結ばれているものとする. アップロードルールは次の通りである.

- 毎時刻, 各ピアは隣接リーチャの中からアップロード先を高々 1 人決定する.
- 各リーチャは他のピアからアップロード先として選択された回数が x 回を超えると十分なアップロードを得られたものとして, シーダとなる.
- シーダとなったピアは, システム内に残りダウンロードは行わず, 他の隣接リーチャの中からランダムにアップロード先を選択する.

ここでは, ある時刻における隣接ピアからのダウンロードの様子を $N-1$ 個の要素から成るビットベクトルで表現し, ダウンロードがあった場合は 1, なかった場合は 0 とする. 隣接ピアへのアップロードも同様である.

また, アップロード先を選択するためのアルゴリズムとして, しっぺ返し戦略とランダム戦略を用意した. 具体的には, 次の通りである.

しっぺ返し戦略

時刻 t で隣接リーチャからダウンロードを行っていない場合, $t+1$ では誰にもアップロードしない.
時刻 t で隣接リーチャからダウンロードを行った場合, $t+1$ では t でダウンロードを受けた隣接リーチャに対してアップロードを行う. ただし, 時刻 t で 2 人以上のリーチャからダウンロードを行っていた場合, その中からランダムに 1 人をアップロード先として選択する.

ランダム戦略
誰にもアップロードしない, または隣接リーチャの誰かにアップロードするという行動の中から, ランダムに行動を選択する.

このようなシステムモデルを用いて, ピアの帯域幅の広狭や, 所持しているファイルの種類の多寡ではなく, 隣接ピアの戦略の違いのみに着目して最適な行動を決定する.

また単純化したシステムモデルを用いる意図としては, 後に述べるニューラルネットワークの学習を比較的容易に行える環境を作り, 段階的に研究を進めていくためである.

3 提案手法

3.1 概要

提案手法では, (1) シミュレーションを行い, モンテカルロ木探索によって最適な行動を探索・決定し, (2) シミュレーションによって得られた隣接ピアとのアップロード・ダウンロード履歴から, 最適な行動を出力するような関数をニューラルネットワークで学習することによって獲得する.

3.2 最適な行動の探索

本節では, 第 2 章で述べたシステムモデルを用いてシミュレーションを行い, 最適な行動を探索する方法について述べる. シミュレーションでは, ピア集合 $\mathbf{P}$ の中から 1 人を学習者とする. 学習者以外の各ピアはしっぺ返し戦略とランダム戦略のいずれかをランダムに選択し, その戦略に従って毎時刻行動を決定する. 学習者は任意の時刻 t_p になると, その時に選択できる行動ごとに分岐し, 最終的に学習者がシーダになるまでランダムシミュレーション (プレイアウト) を繰り返す. 具体的な学習者の行動は以下のように行う.

1. 時刻 $t < t_p$ の時, 学習者はランダム戦略に従って行動を決定する.
2. 時刻 $t = t_p$ の時, 学習者はその時に選択できる行動ごとに分岐し, プレイアウトを開始する.
3. 時刻 $t > t_p$ の時, 学習者はランダム戦略に従って行動を決定する. 学習者がシーダになるとプレイアウトを終了する.

全てのプレイアウトが終了すると, 学習者が t_p でとった行動が, 最終的に学習者がシーダになるまでに費やした時間とアップロード回数へにどのような影響があるかを検証し, t_p において最適である行動を決定する.

具体的には, 各行動ごとに学習者がシーダになるまでの平均アップロード回数 $mean\_upload$ と, 学習者がシーダになるまでの平均時間 $mean\_time$ を計算し,

$$\alpha \cdot mean\_upload + (1-\alpha) \cdot mean\_time$$
$$(0 \leq \alpha \leq 1)$$

が最も小さい行動を最適な行動とする.

3.3 ニューラルネットワーク

3.3.1 ニューラルネットワークによる学習

シミュレーションを何度も行うことによって, 大量の訓練サンプルを生成し, フィードフォワード型ニューラルネットワークによる学習を行う.

参照する過去の履歴数を h とすると, ニューラルネットワークに与える入力データは $t_p - h$ から $t_p - 1$ の間に学習者が観測した全ての隣接ピアとのアップロード履歴とダウンロード履歴となり, ベクトルの要素数は $2h(N-1)$ となる. ラベルは, 最適な行動を 1, その他の行動を 0 とした N 個の要素からなる one-hot ベクトルで表される.

これらの教師データから学習するニューラルネットワークの出力は, 行動ごとに「その行動が最適である確率」である. ニューラルネットワークは, 学習者が観測した過去の履歴から, ラベルに近い出力を出せるように学習をしていく.

3.3.2 データ拡張

ニューラルネットワークでの学習を行うための大きな課題として, 大量の訓練サンプルを作成するには時間がかかるため, 学習に必要な十分なサンプル数の確保が難しいことが挙げられる. これを解決するために 1 つの訓練サンプルから, 学習者が観測した各隣接ピアのデータをランダムに入れ替えて, 複数の訓練サンプルを作成する. このことによって, 1 つのサンプルから $(N-1)!$ 通りのパターンを生成することができる. もちろん, 1 つのサンプルから生成されたパターンは似ているため, オリジナルのサンプルが多ければ多いほうがよいが, この方法によってサンプル数の問題を軽減することができる.

4 数値評価

数値評価では, シミュレーションを行い, 学習済みのニューラルネットワークで行動を決定する場合と, 他の戦略の比較を行う. ピア数 $N = 10$, プレイアウト開始時刻 t_p は 51 から 90 の範囲でランダムに選択し, ピアがシーダになるダウンロード回数 $x = 100$ として, プレイアウトのシミュレーションを行った. そして, そこで得られた 30000 個のサンプルを 3.3.2 項で述べた方法でデータ拡張を行い, ニューラルネットワークでの学習を行った. 実装は Google が提供する機械学習ライブラリ TensorFlow[3] を用いて行った. ニューラルネットワークの構造は図 1 に示す.

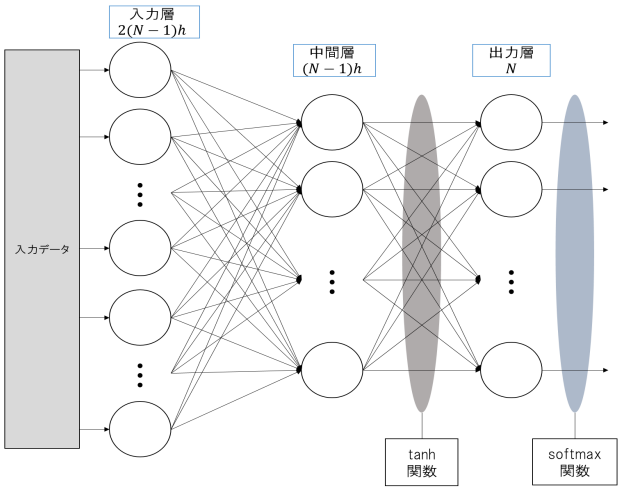


図 1　ニューラルネットワークの構造

評価のためのシミュレーションは以下の流れで行う.

1. ピアの中から 1 人を観測者とする. 観測者以外のピアはそれぞれしっぺ返し戦略とランダム戦略のいずれかをランダムに選択し, その戦略に従って毎時刻行動を決定する.
2. 時刻 $t \leq 50$ の時, 観測者はランダム戦略で行動を決定する. これはニューラルネットワークを用いるときに, 観測者の行動の履歴が入力データとして必要なためである.
3. 時刻 $t = 51$ の時, 観測者はランダム戦略, しっぺ返し戦略, 提案手法のニューラルネットワークによるピア選択法 (以降, nn) に分岐する. この時全てのピアは $t \leq 50$ に得たダウンロードを 0 にリセットして, この時点から 100 回のダウンロードを得られたらシーダになるものとする.

シミュレーションによって, 観測者がランダム戦略, しっぺ返し戦略, nn で行動を決定した場合のシーダになるまでに費やした時間や, アップロード回数などを比較する. また, nn でも参照する過去の履歴数 h やパラメータ α を変化させて比較していく.

4.1　参照する過去の履歴数 h の比較

本節では, ランダム戦略, しっぺ返し戦略, 提案手法で参照する過去の履歴数 $h = 10, 20, 30, 40, 50$ の場合を比較する. またパラメータ $\alpha = 0.5$ として実験を行った.

図 2 は, 観測者がシーダになるまでのアップロード回数と時間の結果である. 提案手法では, ランダム戦略と比べると少ないアップロード回数で, 早くシーダになることができている. またしっぺ返し戦略と比べると, アップロード回数は多くなっているものの, シーダになるまでの時間は大幅に小さくなっていることが分かる.

図 3 は観測者がアップロードする確率とランダム戦略をとっている隣接ピアをアップロード先として選択する確率の結果である. 提案手法はほぼ毎時刻アップロードしているが, 観測者にとって利得がないランダム戦略をとっている隣接ピアに対しては, アップロードを行わず, しっぺ返し戦略をとっている隣接ピアを判別してアップロードしていることが分かる.

また提案手法の参照する履歴数 h で比較すると, 参照する履歴数が多いほど, ランダム戦略をとっているピアにアップロードすることが少なくなっている.

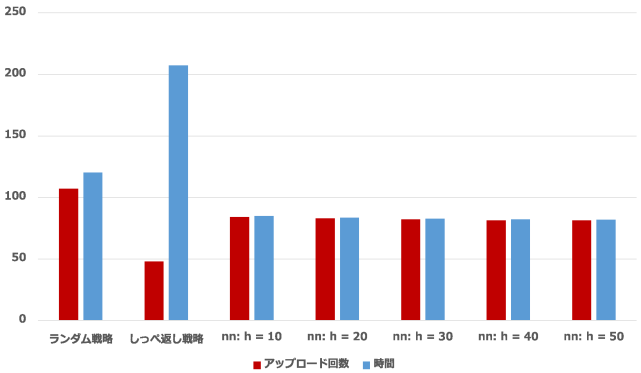


図 2　シーダになるまでのアップロード回数と時間 (h の比較)

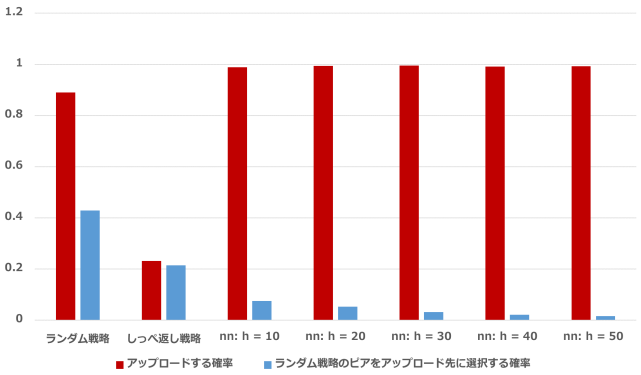


図 3　アップロードする確率とランダム戦略をとっている隣接ピアをアップロード先として選択する確率 (h の比較)

4.2　パラメータ α の比較

本節では, ランダム戦略, しっぺ返し戦略, 提案手法で参照する過去の履歴数 $\alpha = 0, 0.25, 0.5, 0.75, 1$ の場合を比較する. $\alpha = 0$ の時は, アップロード量を気にせず, 貪欲にダウンロードを獲得し, 早くシーダになろうとするような行動を, $\alpha = 1$ の時は, 自身のアップロード回数を抑えようとする行動を学習させる. また参照する履歴数 $h = 50$ として実験を行った.

図 2 は, 観測者がシーダになるまでのアップロード回数と時間の結果である. 提案手法とランダム戦略と比べると, $\alpha = 1$ の場合以外では, アップロード回数とシーダになるまでの時間は小さくなっていることが分かる. また, $\alpha = 1$ の場合としっぺ返し戦略と比べると, 提案手法のほうが良い結果を示している.

図 3 は観測者がアップロードする確率とランダム戦略

をとっている隣接ピアをアップロード先として選択する確率の結果である．提案手法では，いずれの場合も，観測者にとって利得のないランダム戦略をとっているピアに対しては，アップロードを行わず，しっぺ返し戦略をとっている隣接ピアを判別してアップロードしていることが分かる．

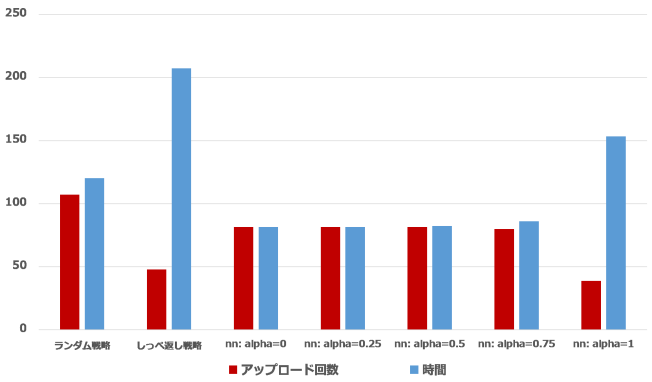


図 4　シーダになるまでのアップロード回数と時間 (α の比較)

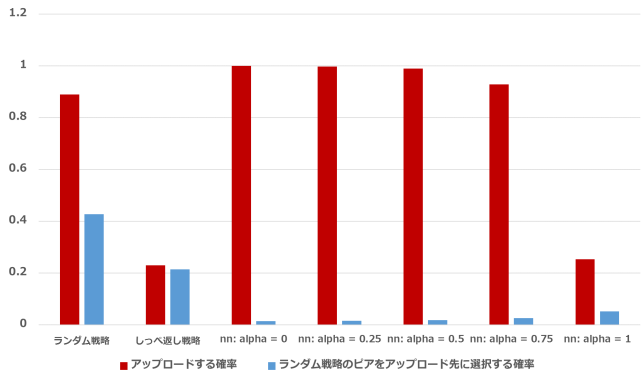


図 5　アップロードする確率とランダム戦略をとっている隣接ピアをアップロード先として選択する確率 (α の比較)

5　おわりに

本研究では，シミュレーションによって，ピア自身の利得を最大化するような行動をモンテカルロ法によって探索し，シミュレーションから得られた大量のデータサンプルをニューラルネットワークによって学習し，ピアが観測した隣接ピアとのファイルのやり取りの履歴から，最適な行動を導き出せるような関数を獲得する方法について提案した．提案手法は，従来のランダム戦略やしっぺ返し戦略と比べて，隣接ピアの戦略を判別し効率的に利得を得ることができていることを示した．

今後の課題として，より多くの種類の戦略をとる隣接ピアがいる場合や，ピアの帯域が均一でない場合などにも対応できるような汎化能力の高い関数の作成などがある．

参考文献

[1] BitTorrent, "http://www.bittorrent.com"
[2] B. Cohen, "Incentives Build Robustness in BitTorrent", *Proc. Workshop Economics of Peer-to-Peer Systems(P2PEcon)*, 2003
[3] TensorFlow, "http://www.tensorflow.org"

分散アルゴリズムの実環境での実験のためのロボット開発

亀山 聖太 † 田村 康将 ‡ Défago Xavier‡

† 東京工業大学工学部情報工学科
‡ 東京工業大学 情報理工学院 情報工学系
E-mail:kameyama.s@coord.c.titech.ac.jp

分散ロボット (群ロボット) に用いる分散アルゴリズムを研究する際は，コンピュータシミュレーションによる動作確認が一般的な方法である．しかしながら，現実世界には離散的要素がなく，ロボットの故障やホイールの公差による移動誤差が生じるなど，シミュレータと異なる点（Reality gap）が存在する．こうした課題に対し，現実のロボットを用いたアルゴリズムの動作実験が可能ならば，シミュレータ上での動きとの比較ができ，考慮すべきパラメーターの見直しなどの課題点を見つけることが容易となる．研究に用いられるロボットとしては，例えば Kilobot[1] や e-puck[2] などが挙げられる．特に Kilobot は群ロボットのアルゴリズム研究のために作られている．これらは専用のシミュレータ [3] やシミュレーションモデル [2],[4] が公開されており，そのまま同じプログラムを実機に載せて動かす事が可能なので，シミュレータ上との動作の比較はしやすい．一方で，これらのロボットにはメモリが数 kB しかない等のハードウェア的な制約が存在し，周囲の環境に応じてプログラムを動的に変更するなど，柔軟ではあるが複雑な行動規則を実装することは困難であるといった課題も存在する．本研究では，RaspberryPi と Arduino を用いた実験用分散ロボットについて考える．RaspberryPi は上述した一般的な研究向けロボットと比較してより大きな計算資源とメモリを持ち，Arduino はメモリは小さいがリアルタイム性を保ってロボットを動作させる事が可能である．この点に着目して，ロボットの柔軟な動作を可能とする組み合わせ制御の方法を考える．

参考文献

[1] Rubenstein, Michael, Christian Ahler, and Radhika Nagpal. ”Kilobot: A low cost scalable robot system for collective behaviors.” Robotics and Automation (ICRA), 2012 IEEE International Conference on. IEEE, 2012.

[2] Mondada, Francesco, et al. ”The e-puck, a robot designed for education in engineering.” Proceedings of the 9th conference on autonomous robot systems and competitions. Vol. 1. No. LIS-CONF-2009-004. IPCB: Instituto Polit é cnico de Castelo Branco, 2009.

[3] Jansson, Fredrik, et al. ”Kilombo: a Kilobot simulator to enable effective research in swarm robotics.” arXiv preprint arXiv:1511.04285 (2015).

[4] CYBERBOTICS Ltd., ”Webots documentation: Using the e-puck robot”, `https://www.cyberbotics.com/doc/guide/using-the-e-puck-robot`, (参照 2017-08-27)

未知環境における動的なアンカーを使った移動ロボット群の協調探索

奥村圭祐, 田村康将, Xavier Dèfago
東京工業大学 工学部情報工学科
Email : okumura.k@coord.c.titech.ac.jp

Collaborative Exploration of Wanderer Robots with the Dynamic Anchor in the Unknown Environment

Keisuke Okumura, Yasumasa Tamura, Xavier Dèfago
Tokyo Institute of Technology, Department of Computer Science

Exploration とは, 未知環境下で自律的に移動するロボット群が必要な情報を収集して, その環境の地図を作成することであり [1], マルチロボットシステムの主要な問題の一つとして多くの研究が行われている. この問題は, モバイルロボットがタスクを実行するために環境の地図が必須であることを背景としており, 災害時の救助活動 [2] や惑星探索 [3] など応用先が幅広いことから注目を浴びている. 単体のロボットと比較してロボット群で Exploration を行う利点としては, 探索時間の削減, タスク実行の確実性などが挙げられる [4][5]. ロボット群による未知環境の探索では, 各ロボットに未探索の領域を効率的に割り当て, 全体の探索時間を最小限に抑える行動計画の設計が重要である. これは各ロボットが探索する領域の重なりを削減することで実現されるが, 現実社会で活用していくためには, ロボットの通信範囲が限定的である場合を考慮すべきである [6]. さらに, ロバスト性やフォールトトレランス性の観点から, 中央集権的な手法ではなく, 分散的な手法を用いることが好ましい. このような前提のもと, Exploration の効率化に取り組むにあたって, 環境についての情報の取得するスピードと, その情報を共有するスピードのトレードオフを考える必要がある. あるロボットが他のロボットから離れた地点に移動すると, そのロボットは環境について多くの新しい情報を取得することが可能だが, 取得した情報を他のロボットに共有することには時間がかかる. 他のロボットから近い地点にいると, 他のロボットから新しい情報が共有されやすいものの, そのロボット自身が環境について新しい情報を得ることは困難となる. この問題を解決するために, Anchor-Wanderer モデルを導入し, 各モバイルロボット (Wanderer) が情報共有を集中的に行う地点 (Anchor) を設置する. 本研究では, 効率的にロボット群が協調探索を行う Anchor-Wanderer モデルと, 取得した情報に基づいて動的に Anchor を決定するアルゴリズムの開発に取り組む.

参考文献

[1] Julià M., Gil, A., & Reinoso, O. (2012). A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. Autonomous Robots, 33(4), 427-444.

[2] Nevatia, Y., Stoyanov, T., Rathnam, R., Pfingsthorn, M., Markov, S., Ambrus, R., & Birk, A. (2008, September). Augmented autonomy: Improving human-robot team performance in urban search and rescue. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on (pp. 2103-2108). IEEE.

[3] Apostolopoulos, D. S., Pedersen, L., Shamah, B. N., Shillcutt, K., Wagner, M. D., & Whittaker, W. L. (2001). Robotic antarctic meteorite search: Outcomes. In Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on (Vol. 4, pp. 4174-4179). IEEE.

[4] Burgard, W., Moors, M., Stachniss, C., & Schneider, F. E. (2005). Coordinated multi-robot exploration. IEEE Transactions on robotics, 21(3), 376-386.

[5] Yan, Z., Jouandeau, N., & Cherif, A. A. (2013). A survey and analysis of multi-robot coordination. International Journal of Advanced Robotic Systems, 10(12), 399.

[6] Sheng, W., Yang, Q., Tan, J., & Xi, N. (2006). Distributed multi-robot coordination in area exploration. Robotics and Autonomous Systems, 54(12), 945-955.

多倍長演算ライブラリ MPIR 互換の CUDA ライブラリの実装について

藤田崚太，藤本典幸，澤田幸一郎，紫垣賢人，川口大輔 †1

1. はじめに

32 ビットや 64 ビットを超えるような，レジスタに収まらないほど大きな値の数を扱う場合に，多倍長数が用いられることがある．多倍長数は計算機のメモリ容量の範囲内で任意に精度を決めることができる．この任意精度の性質により，多倍長演算には精度の高いデータの演算を行えることの他にも，オーバーフローを考慮せずにプログラムを書けるという利点がある．その一方で，多倍長演算は処理が重く，固定精度整数や浮動小数点数を用いた場合と比較して演算に時間がかかるという欠点もある．多倍長演算，特に四則演算や平方根といった基本的かつ使用頻度の高い演算を高速化できれば，それ自体の高速化だけでなく高精度のデータを用いた問題全体の解決の高速化にもつながるため，多倍長演算の高速化は重要な問題である．

性能の良い多倍長演算ライブラリのひとつに GMP[1] があり，GMP の互換ライブラリとして MPIR ライブラリ [2] がある．GMP や MPIR では多倍長整数や多倍長実数の仮数の絶対値を limb の配列に格納することで計算機の 1 語では表すことのできない大きな値を表現している．limb の 0 番目の要素の LSB，最後の要素の MSB が多倍長整数や多倍長実数の仮数の絶対値の LSB，MSB となるように各要素に値が格納されている．

また，演算装置のひとつに GPU がある．GPU を用いて数値計算を行う GPU プログラミングでは，GPU の数百〜数千個あるコアによる並列処理で様々な計算処理を高速化することが可能である [3]．例えば，配列全体に対して何らかの操作を行う場合，CPU 上で実行されるプログラムでは 1 要素ずつ逐次的に処理を行うことになる．しかし GPU プログラミングを用いれば同時に配列の複数の要素に対してアクセスすることができるので，実行時間を短縮可能である．以上の多倍長数のデータ構造と GPU プログラミングが並列処理を得意とする点から，多倍長演算を GPU 上に実装することが高速化に有効なのではないかと考えた．

本研究では，MPIR とインターフェース互換性を持つ GPU 向けの多倍長演算ライブラリを実装し，MPIR とそのライブラリの性能比較を円周率計算によって行い，高速化の可能性を検証する．

本論文の構成は以下の通りである．第 2 章で GPU と，NVIDIA 社の GPU を対象とした GPGPU 向けプログラミング環境である CUDA[4] の説明を行う．第 3 章で既存の多倍長演算ライブラリ MPIR についての説明を行った後，第 4 章で自作した多倍長演算ライブラリの基本構造について示す．第 5 章で評価実験について述べ，第 6 章でまとめを行う．

2. GPU と CUDA

2.1. GPU について

GPU は Graphics Processing Unit の略称で画像処理を担当する主要な部品の一つである．CPU は逐次処理用に最適化された少数個のコアで構成されており，CPU はコア数が数個〜数十個程度であるのに対して，GPU は複数のタスクに同時に対応できるよう設計された，より小さく，より効率的な数百〜数千個ものコアで構成されている．一つ一つのプロセッサの構造は単純なためその機能は CPU に比べて限定されたものであるが，大量のデータを複数のプロセッサで並列処理することが可能である [5]．GPU の高い処理性能を画像処理以外の汎目的計算に応用する技術のことを GPGPU と言う．GPGPU アプリケーションの開発環境のひとつに CUDA がある．

2.2. CUDA について

CUDA とは Compute Unified Device Archtecture の略称で，NVIDIA 社が提供する GPU コンピューティング向けの統合開発環境である．プログラム記述，コンパイラ，ライブラリ，デバッガなどから構成されている．プログラム言語は C 言語と C++を拡張した CUDA C/C++という言語で，GPU に処理させる部分以外は C 言語や C++と全く同じように記述できる．

CUDA によって操作できる NVIDIA 社製の GPU の構造について簡単に説明する．GPU チップの内部にはストリーミング・マルチプロセッサ（SM）が多数入っており，それぞれが GPU のメモリであるグローバルメモリに接続されている．さらに SM の内部には CUDA コアという最小単位の演算処理ユニットが多数搭載されている [6]．SM の内部にも複数のメモリ領域が存在する．CUDA で GPU に処理をさせるときはカーネル関数という関数を呼び出すことで，その関数の記述内容が GPU 上で実行される．

2.3. CUDA によるマルチスレッドプログラム

カーネル関数を呼び出す際に，実行するブロック数とスレッド数を指定する必要がある．スレッドとはプログラムの実行の最小単位のことで，CUDA では全てのスレッドが同じプログラムを実行する．複数のスレッドがプログラムを実行し，それぞれのスレッドが異なるメモリ領域にアクセスするなどして並列処理を実現している．ブロックとはスレッドをまとめたものである．ブロック内では，スレッドは 3 次元的に管理され，同一ブロック内のスレッドは全て同じ SM に割り当てられる．また，1 ブロック当たりのスレッド数は最大で 1024 という制限がある．そのブロックをさらにまとめたものをグリッドと呼び，ブロックのようにグリッドは複数のブロックを 3 次元的に管理する．このように，CUDA のスレッドは階層構造になっている．各スレッド，ブロックにはインデックスが割り当てられ，カーネル関数内で組み込み変数を使って参照可能で，スレッドインデックスは threadIdx，ブロックインデックスは blockIdx で参照できる．グリッド内のブロックの数は組み込み変数の gridDim，ブロック内のスレッドの数は組み込み変数の blockDim で参照できる．これらを用いて全スレッドを管理する．

プログラムを実行するスレッドは 32 個ごとにまとめられ，Warp という単位で管理される．スレッドは Warp ごとに動作するので，実際の並列処理の最小単位は Warp である．そのため，実行するスレッド数は 32 の倍数にするのが望ましいとされる．

2.4. GPU のメモリ

GPU には数種類のメモリ領域が存在する．ここでは使用頻度の高いレジスタ，グローバルメモリ，シェアードメモリについて説明する．レジスタは，GPU のチップ上に実装されており，容量は小さいが最も高速にアクセスすることが可能で，1 クロックで値の読み書きができる．スレッドが宣言した変数はこのレジスタに記憶される．スレッドごとに独立しており共有はできない．グローバルメモリは GPU チップの外に置かれており，全ての SM と接続されているため全ス

†1 大阪府立大学 大学院工学研究科 電気・情報系専攻 知能情報工学分野

レッドがグローバルメモリに記憶されたデータを共有できる．メモリ容量は大きいが，レジスタやシェアードメモリと比較して 100 倍以上アクセスが低速である．CPU 側のメモリとデータの受け渡しをするときは通常グローバルメモリに読み書きされる．シェアードメモリは SM 内にあるメモリ領域であり，同一 SM 内の CUDA コアはシェアードメモリに記憶されたデータを共有することが可能である．そのため同一ブロック内のスレッドはシェアードメモリに記憶されたデータを共有できる．容量が小さめであるが，CUDA コアに近い位置に存在するため高速にアクセスが可能で，シェアードメモリの有効活用は CUDA プログラミングにおいて良い性能を出すための重要な要素となる．

3. MPIR ライブラリ

MPIR は Multiple Precision Integers and Rationals の略称で，GMP の互換ライブラリである．GMP は Windows を公式にはサポートしていないが，MPIR は Windows をサポートしている．

MPIR ライブラリは C 言語，C++で使えるが，C++で用いる場合は多倍長数クラスを用いることでより簡潔な記述で多倍長数の演算や入出力を行える．クラスを使った場合，多倍長数クラスから生成されるオブジェクトに多倍長数が格納される．演算の種類によっては，その演算を行うためのサブルーチンを呼び出さなくても，変数の演算を行う場合と同じように記述するだけで多倍長数同士の演算ができる．多倍長数同士の減算を行う場合を例にすると，多倍長数オブジェクト a，b，c があり，a から b を引いた結果を c に代入したいとき， C 言語の場合は減算のメソッドを呼び出し，引数に a，b，c を指定する必要があるが，C++では

```
c = a - b
```

と記述するだけで減算を行える．多倍長数の入出力をする場合も，C++の入出力ストリームである cin や cout が使用できる．

4. MPIR 互換ライブラリ

MPIR にある多倍長浮動小数点数クラス mpf_class に互換のクラス my_mpf_class を実装した．my_mpf_class は，符号，仮数の絶対値，指数の 3 つの要素で多倍長実数を表現している．その内の仮数の絶対値は固定長の unsigned 配列で表現しており，この配列の要素数を変えることで表現する実数の有効桁数を任意に設定できる．仮数は必ず正規化される．my_mpf_class における正規化はビット単位ではなく，unsigned 配列の最上位の要素（32 ビット）が非ゼロ（$1 \sim 2^{32}-1$ までの可能性がある）となるように，右シフトあるいは左シフトで指数を調整することでなされる．現在の my_mpf_class がサポートしている演算は，加減乗除算と平方根，絶対値，比較演算のみである．int や double，mpz_class, mpf_class などの値からのコンストラクトと，mpf_class への型変換，および double 型の近似値を返すメソッドもサポートしている．my_mpf_class および後述の cuda_mpf_class の入出力は mpf_class とのコンストラクトや型変換を利用する形で実装している．

さらに，my_mpf_class と同じ機能を持ち，演算など一部の操作が GPU 上で動作するクラス cuda_mpf_class を実装した．cuda_mpf_class は CPU 版である my_mpf_class や MPIR とプログラミングインターフェース互換性を持つ．そのため，多倍長実数の演算を CPU 上で行うか，CUDA によって GPU 上で行うかをクラス名の記述を変えるだけで切り替えられ，プログラマは GPU 版を特別な考慮や GPU プログラミングの知識なしに使用できる．例えば，my_mpf_class を用いるプログラムのソースコードの，my_mpf_class と書かれている部分を cuda_mpf_class に書き換えるだけで GPU 版に切り替えられる．また，基本的に GPU 上で演算を行うためにはあらかじめ対象のデータを CPU から GPU に転送する必要があり，演算結果を出力するときも GPU から CPU にデータを転送する必要があるが，cuda_mpf_class では GPU と CPU の間のデータ転送は自動化され，プログラマには隠蔽される．多倍長数のデータはコンストラクタあるいは代入演算で GPU の VRAM 上に生成され，基本的には VRAM 上に常駐し，すべての多倍長演算は CPU-GPU 間のデータ転送無しに GPU 上で実行される．演算結果の CPU への転送も，データを出力するときなどの必要な場合に，mpf_class への型キャストを記述するだけで内部で自動的に行われる．こうすることで，時間のかかるデータ転送処理を最低限の回数に抑え，GPU の性能を引き出すことができる．

5. 評価実験

5.1. 実験の設定

自作ライブラリの性能評価を行う実験として以下のように設定した．MPIR にある多倍長浮動小数点数クラス mpf_class，自作ライブラリの CPU 版クラス my_mpf_class，CUDA で実装した GPU 版クラス cuda_mpf_class の 3 つで，円周率計算を行った．求める円周率の精度を 10 進数 1000 桁から，現在対応している最大の精度である 9600 桁の範囲で変化させ，計算にかかった時間を計測した．設定した精度ごとに 10 回ずつ試行を行い，その平均値を平均値を最終的な性能とした．また，精度を 1000 桁に設定したときの実行時間を基準にして，各精度の実行時間の増加率も求めた．GPU 版においては，どの処理に時間がかかっているか NVIDIA Visual Profiler を用いて解析した．なお，今回の実験では円周率計算のアルゴリズムにガウス＝ルジャンドルのアルゴリズムを改良したもの [7] を用いた．

5.2. 実験環境

実験を行った環境を以下に示す．

OS	Windows Server 2012 R2
CPU	Intel Core i7-6700T（2.80 GHz）
GPU	NVIDIA GeForce GTX 1080
CUDA のバージョン	CUDA7.5
MPIR のバージョン	2.7.2

5.3. 自作ライブラリの性能

まず，円周率の精度を変化させたときの実行時間の比較を挙げる（図 1）．全ての場合において MPIR の mpf_class が最も実行時間が短いという結果になった．自作ライブラリの CPU 版と GPU 版を比較すると，精度が低いときは GPU 版自作ライブラリの方が高速だが，精度が約 10 進数 6500 桁を超えると GPU 版の方が実行時間が短くなり，精度が高いほど GPU 版が有効であることがわかる．

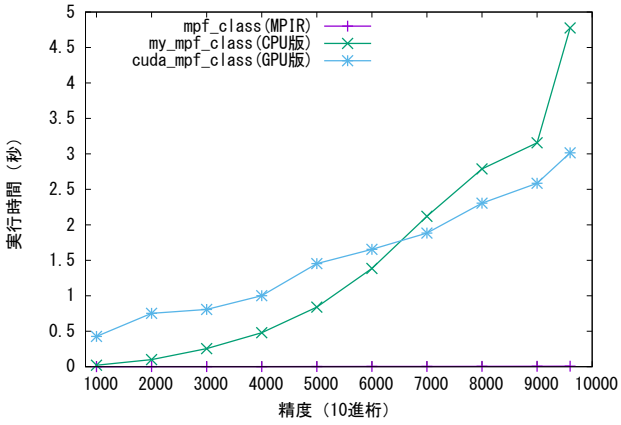


図 1: 各クラスの実行時間の比較

次に，精度が 1000 桁のときを基準とした実行時間の増加率の比較を挙げる（図 2）．GPU 版が最も増加率が低く，9600 桁のとき約 7 倍という結果になった．このとき MPIR の mpf_class が約 40 倍となっており，精度をさらに高く設定

すればいずれは GPU 版が mpf_class よりも高速に円周率を求められることが期待できる．

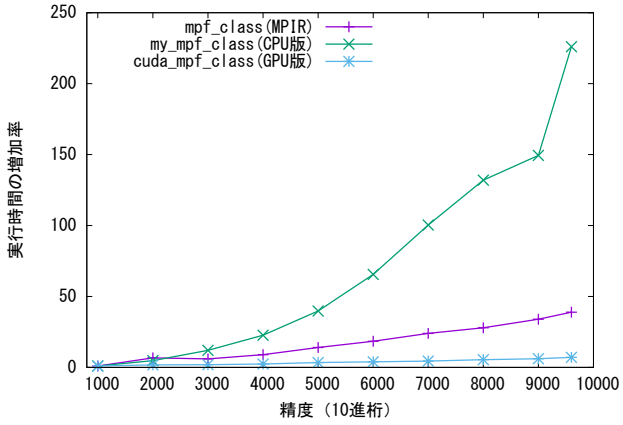


図 2: 実行時間の増加率の比較

最後に，GPU 版をプロファイラで解析した結果を図 3 に示す．解析時のみ実行環境が異なっている．以下に解析をした時の環境を示す．

OS Windows7 Professional 64bit
CPU Intel Core i5-3450（3.10 GHz）
GPU NVIDIA GeForce GTX 980

図 3 から，実行時間の大半が乗算で使われるカーネル関数 multiple_3to2_gpu で占められており，ボトルネックになっていることがわかる．

6. まとめ

本論文では GPU に対応する多倍長実数の演算ライブラリを実装し，円周率を既存の多倍長演算ライブラリ MPIR と自作ライブラリを用いて計算した場合のそれぞれの性能を比較，検証し，現在対応している桁数の範囲では MPIR に性能が劣るが，さらに多い桁数に対応できれば GPU 版が最も高速に動作する可能性があることを示した．

今後の課題として以下のことが挙げられる．一つ目は 9600 桁を超える精度への対応である．精度が高いほど GPU 版が有利になるという結果からも，この問題は最重要課題であるといえる．二つ目は，ボトルネックである乗算部分の高速化である．この問題を改善できれば，GPU 版が有利な精度の範囲が広がる．三つ目は，仮数の正規化の性質上，演算精度が安定せず悪くなるので，精度がどの程度悪くなるかの評価，あるいはビット単位での正規化である．

参考文献

[1] T. Granlund and the GMP Development Team. *GNU MP 6.1.2*, December 2016. `https://gmplib.org/`.

[2] T. Granlund, the GMP Development Team, W. Hart, and the MPIR Team. *MPIR 2.7.2*, November 2015. `http://mpir.org/downloads.html`.

[3] J. Cheng, M. Grossman, T. McKercher, (株式会社クイープ訳). CUDA C プロフェッショナル プログラミング (impress top gear). インプレス, 2015.

[4] NVIDIA. *CUDA C Programing Guide*, June 2017. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.

[5] 岡安 優. G-DEP GPU コンピューティングおよび CUDA について. `http://http://www.gdep.jp/page/view/248`.

[6] 青木 尊之, 額田 彰. はじめての CUDA プログラミング. 工学社, 2009.

[7] D. Takahashi. Parallel implementation of multiple-precision arithmetic and 2,576,980,370,000 decimal digits of π calculation. *Parallel Computing*, 36:439–448, 2010.

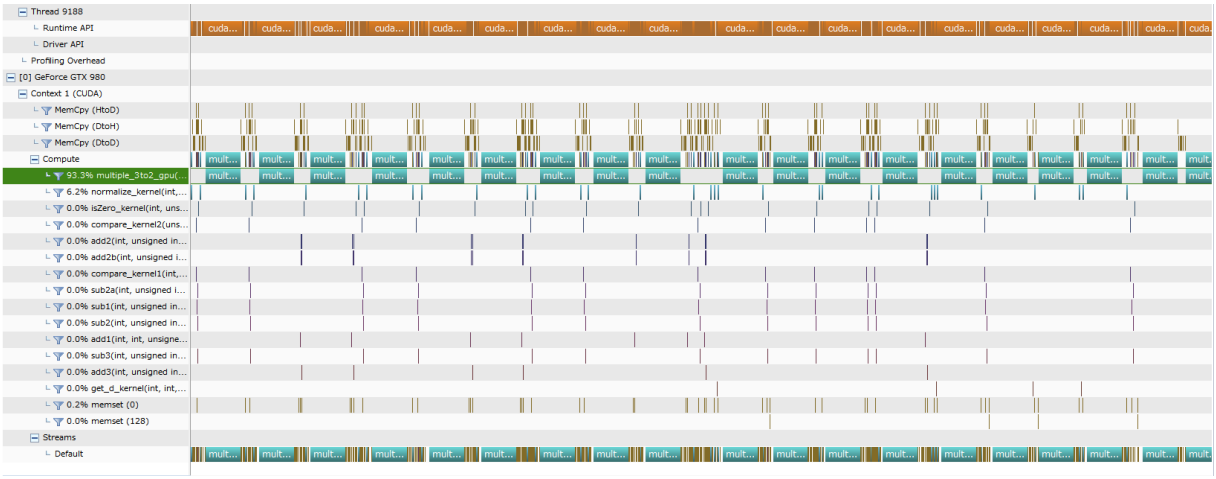


図 3: GPU 版の解析結果

CUDA を用いた多倍長乗算の実装について

澤田 幸一郎，藤本典幸 †1
和田 幸一 †2

1. はじめに

計算機にはワード長が定められており，数値計算においてその範囲を超える数値をそのまま扱うことはできない．しかし，円周率 π や $\sqrt{2}$ といった数学定数の研究ではより精密な値を計算し数値列に周期的な性質が含まれていないか調べるため，数百万桁に上るような膨大な数値を扱う必要がある．また，公開鍵暗号の方式の 1 つである RSA 暗号では 2 つの素数の積を利用している．これは効率的な素因数分解の解法が今なお見つかっておらず，素数の桁数を十分大きくすれば積から 2 つの素数を求めることが計算時間的にほぼ不可能だからである．こうした任意精度の数値計算が必要な場合，多倍長整数という表現方法が採用される．多倍長整数はメモリ容量以外の制約を受けないため非常に大きな数値でも表現できるが，ワード長に収まる場合の数値演算と比較して処理時間が遅くなるというデメリットが生じる．

一方，四則演算はあらゆる数値計算において基本となる演算であり，これを高速化できれば様々なアルゴリズム，計算がその恩恵を受ける．単純な n ビット乗算アルゴリズムは計算量が $O(n^2)$ であるので $O(n)$ の加減算と比較して計算コストが高く，処理時間も遅くなりやすい．そのため，FFT[8] や Strassen のアルゴリズム [9]，Toom-Cook 法 [10] といった高速な乗算アルゴリズムが提案されている．

多倍長演算の高速化に関する研究の中には GPU の並列演算を利用した研究がいくつか見られる．文献 [1] では多倍長整数の行列積の計算においてモジュラー算法と中国人剰余定理に基づき GPU と CPU で処理を分割する手法が提案されており，文献 [2] では多倍長精度演算ライブラリである GNU MP[11] の代替として GPU コード中で使用できる多倍長精度浮動小数点演算ライブラリを実装したと報告している．また，文献 [3] では多倍長整数乗算について積表という新たな手法が，文献 [4] では Warp シャッフルを利用した多倍長乗算の実装方法が提案されている．

本論文では文献 [7] の桁上げ保存型加算器を用いた Wallace 木乗算器をソフトウェアで実装した提案手法で多倍長整数同士の乗算を実装し，CPU での実装と CUDA[5], [6] を用いた GPU での実装，そして CPU の多倍長整数演算ライブラリとして提供されている MPIR ライブラリの乗算プログラムの 3 つを比較し，性能を検証した．

本論文の構成は以下のとおりである．2 章では GPU と GPU 向けの統合開発環境である CUDA について述べる．3 章では多倍長整数と桁上げ保存型加算器・Wallace 木乗算器について述べる．4 章では提案手法について述べる．5 章では評価実験とその結果および考察について述べ，6 章でまとめを行う．

2. GPU と CUDA

2.1. GPU の概要

GPU(Graphics Processing Unit) は従来画像処理に特化した演算装置であり，ディスプレイへの描画などを担当していた．しかし近年の技術発展に伴う GPU の演算性能の向上と，インターネットの普及によって大量のデータを高速に処理する必要が生じ，その方法として並列処理が一般的となったことにより画像処理以外の作業にも GPU が活用されている．一般に CPU のコア数は数十個ほどだが GPU は数百から数千個のコアを持ち，さらにそのコア数を上回る数百万ものスレッドを起動して大量のデータを並列に処理できるという特徴がある．

2.2. GPGPU

CPU はある程度のサイズのデータに対して複雑で逐次的な処理を高速に実行できるという特徴がある．しかし，逆に依存関係のない大量のデータに対して単純な処理を逐次実行する場合，CPU の性能を発揮しづらい．そこで，そうした処理の並列実行が得意な GPU を利用することを GPGPU(General-Purpose computing on GPU) と呼ぶ．画像処理以外の演算に GPU を活用することで，従来の CPU のみでの処理よりも高速な処理を実現する狙いがある（図 1）．

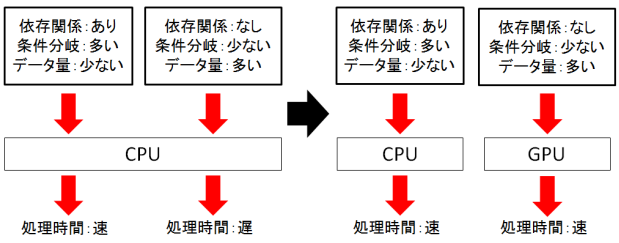


図 1: GPGPU のイメージ

2.3. CUDA の概要

CUDA(Compute Unified Device Architecture) は nVIDIA 社が開発・提供している GPU 向けの統合開発環境である．C 言語をベースとして GPU の並列演算に対応した拡張言語であり，C 言語の知識があれば比較的容易に扱えるというメリットがある．CUDA は nVIDIA 社製の GPU のみ対応しているので，ここで nVIDIA 社の GPU のアーキテクチャについて簡潔に述べる．GPU 内部では演算ユニットが複数のストリーミングマルチプロセッサ（SM）に分割されている．各 SM は GPU の VRAM と接続されており，SM ごとに異なる命令が実行できる．それぞれの SM には数百の CUDA コア（プロセッサ）や後述するシェアードメモリ，キャッシュ，レジスタ，ロード・ストアユニットといった命令ユニットが含まれており，1 つの SM だけでも数千ものスレッドを起動できる．

2.4. スレッドの構成

CUDA では複数スレッドから成るブロック，複数ブロックから成るグリッドという 2 階層構造を用いている．ブロックはスレッドの 3 次元配列として，グリッドはブロックの 3 次元配列として構成されており，プログラマは *dim*3 型変数という CUDA 独特の変数を使って各次元のスレッド・ブロック数指定できる．指定されたスレッドはそれぞれ一意なインデックスを持ち，並列に命令を実行する．ブロックとグリッドをどのように指定するかで GPU のパフォーマンスが大きく変わるため，プログラマはそのプログラムにとって最適な指定を見つける必要がある．

†1 大阪府立大学 大学院工学研究科 電気・情報系専攻 知能情報工学分野
†2 法政大学 理工学部 応用情報工学科

2.5. CUDA のメモリモデル

CUDA はプログラマ自身が明示的にデータの配置・移動を制御できるプログラマブルなメモリを数多く提供しており，これによって C 言語などと比べてより最適化されたプログラムの実装を可能としている．図 2 は CUDA のメモリ階層の一部を表している．

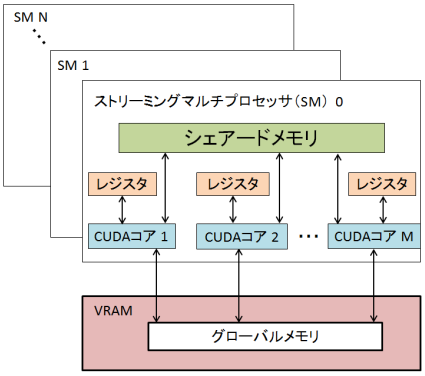


図 2: CUDA におけるメモリ階層の一部

- レジスタ
  レジスタは GPU が扱えるメモリのうち最も高速にアクセス可能なメモリである．各スレッドに対してプライベートであり，1 スレッド当たりが使用できる個数も決められている．
- シェアードメモリ
  シェアードメモリはグローバルメモリと比べて高速にアクセスが可能であり，同一ブロック内の全スレッドが共有可能なメモリとなっている．よって，ブロック内のスレッドが複数回同じデータを参照する場合はシェアードメモリが有効である．
- グローバルメモリ
  GPU が扱えるメモリのうち最も容量が大きくアクセスが遅いメモリである．CPU から GPU へデータの転送が行われるとき，この領域にメモリが確保される．すべてのスレッドがアクセス可できる．

2.6. カーネル関数

CUDA を使って GPU の並列演算を実行するときは，カーネルと呼ばれる関数を CPU から起動する必要がある．書式は

$$void\ kernel\_name <<< grid, block >>> (arguments)$$

となっており，返り値は常に void 型である．$grid, block$ でブロック・グリッド数を指定でき，$arguments$ に必要な引数を追加する．カーネル関数によって起動されたスレッドはブロック単位で SM に割り当てられるが，命令を実行する際はブロック内のスレッドを 32 個ずつに分割して実行している．この 32 個のスレッドのグループをウォープ（Warp）と呼び，必要なデータがそろったウォープから順次命令を実行する．そのため，ブロック・グリッド数を指定する際は 32 の倍数を指定すると無駄なスレッドを発生させることなく並列実行できる．

2.7. 遅延の隠蔽

CUDA は大量のスレッドを起動して大量のデータを並列に処理することが可能である．しかしその演算性能に比べてメモリアクセスが遅く，必要なデータが CUDA コアに届くまで時間がかかる．これをメモリアクセスの遅延と呼び，アイドル状態のスレッドが出てしまうので GPU の性能が発揮できなくなる．そこで，起動するブロック・グリッド数を変更して各 SM が実行可能なウォープの数を増やし，遅延が発生した場合ほかのウォープを実行させることで遅延を隠蔽できる．実行可能なウォープの数はカーネル関数起動時のブロック・グリッド数の指定のほかレジスタやシェアードメモリといった GPU のリソースによって決定される．そのため，レジスタやシェアードメモリを無駄なく利用する必要がある．

3. 多倍長整数と桁上げ保存型加算器および Wallace 木乗算器

3.1. 多倍長整数の概要

計算機のワード長 w 以上の数値を扱う方法として，2^w などを基数とする配列で表現する方法がある．図 3 は 2^{100} を 64 ビット整数型の配列で表現した多倍長整数を表している．図 3 の右側を下位桁，左側を上位桁として基数は 2^{64} としている．こうすることで，メモリ容量を超えないのであればどれだけ大きな数値でも扱うことができるようになる．ただし，多倍長整数による表現はメモリ容量が多くなるうえ繰り上がり・繰り下がり処理も実装する必要があり計算が遅くなってしまうという欠点がある．

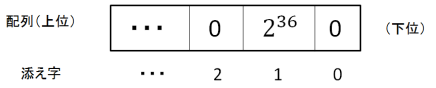


図 3: 64 ビット整数型配列の多倍長整数による 2^{100} の表現

3.2. 桁上げ保存型加算器と Wallace 木乗算器

桁上げ保存型加算器 [7] を利用すると，多倍長整数の加算の繰り上がり処理による条件分岐の回数を減らすことができる．まず初めに桁上げ保存型加算器によって 3 つの多倍長整数 x, y, z の和を 2 つの多倍長整数 u, v の和に変換するアルゴリズムを以下に示す．ただし，n は x, y, z のビット数とする．

Algorithm1: Carry-save adder

入力：$x = (x_0, x_1, \ldots, x_{n-1}), y = (y_0, y_1, \ldots, y_{n-1}),$
$z = (z_0, z_1, \ldots, z_{n-1})$
出力：$u + v = x + y + z$ を満たす u, v

1. $unsigned\ \ u_n = 0, v_{n+1} = 0;$
2. **for** $i = 0$ **to** $n - 1$ **do**
3. $\quad u_i = x_i \oplus y_i \oplus z_i;$
4. $\quad v_{i+1} = (x_i \vee y_i) \wedge (y_i \vee z_i) \wedge (z_i \vee x_i);$
5. **end for**

桁上げ保存型加算器は下位ビットからの桁上げを無視した加算結果 u と下位ビットからの桁上げを表す v をそれぞれ求めることで，3 つの多倍長整数 x, y, z の和を 2 つの多倍長整数 u, v の和へと変換する．この変換によって x, y, z の和を計算する計算量は $O(n)$ のまま桁上げ処理の条件分岐の回数が減らすことができる．

一方，多倍長整数の乗算の単純な実装では部分積の加算において，3.1 節で述べたように条件分岐を含んだ桁上げ処理が発生する．これは桁数に比例して回数が増えるため，処理時間が遅くなる一因となる．この問題を解決する手段として，桁上げ保存型加算器を用いて部分積の加算を変換し高速に計算を実行する Wallace 木乗算器 [7] がある．Wallace 木乗算器は乗算回路の一種であり，ハードウェアでの実装に利用される．今回はこの Wallace 木乗算器をソフトウェアで実装した．図 4 のように配列 A の各要素と配列 B との部分積について 3 つの多倍長整数の和を 2 つの多倍長整数の和に変換するという操作を繰り返し，図 5 のように 2 つの多倍長整数の和しかの残らなくなったら多倍長整数の加算の単純な実装でこれらの加算の計算を行う．これによって部分積の和を求める際に繰り上がり処理から生じる条件分岐の回数を減らすことができる．

以下に CPU 向けの実装を示す．ただし，8 行目の usll は unsigned long long 型を表している．

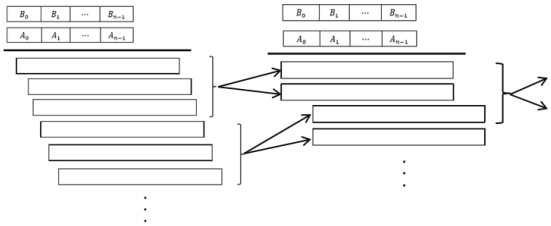


図 4: 3 つの多倍長整数の和を 2 つの多倍長整数の和に変換

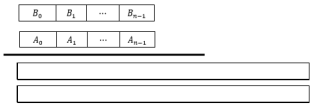


図 5: 図 4 の処理の結果 2 つの多倍長整数の和のみ残った状態

Algorithm2: Multiple precision multiplication with Carry-save adder for CPU(A, B, n, C)

入力：要素数 n の unsigned 型配列 A, B,
　　要素数 $2n$ の unsigned 型配列 C
出力：$C = A \times B$

1. *static unsigned* $D[2][n][2n+1]$;
2. *int* $s = 0, n_2 = n$;
3. **for** $i = 0$ **to** $n-1$ **do**
4. 　**for** $j = 0$ **to** i **do**
5. 　　$D_{sij} = 0$;
6. 　**end for**
7. 　**for** $j = 0$ **to** $n-1$ **do**
8. 　　*usll* $tmp = (usll)A_i \times (usll)B_j$;
9. 　　$D_{si(i+j)} += (unsigned)tmp$;
10. 　　$D_{si(i+j+1)} = (unsigned)(tmp \gg 32)$
　　$+(D_{si(i+j)} < (unsigned)tmp)$;
11. 　**end for**
12. 　**for** $j = j+1$ **to** $2n-i$ **do**
13. 　　$D_{si(i+j)} = 0$;
14. 　**end for**
15. **end for**
16. **while** $n > 2$ **do**
17. 　**for** $i = 0, j = 0$
　　to $i = n_2 - 3$ **step** $i+=3, j+=2$ **do**
18. 　　unsigned $tmp = 0$;
19. 　　**for** $k = 0$ **to** $2n-1$ **do**
20. 　　$D_{(1-s)jk} = D_{sik} \hat{} D_{s(i+1)k} \hat{} D_{s(i+2)k}$;
21. 　　$D_{(1-s)(j+1)k} = tmp \gg 31$;
22. 　　$tmp = (D_{sik} \& D_{s(i+1)k}) | (D_{s(i+1)k} \& D_{s(i+2)k})$
　　$| (D_{s(i+2)k} \& D_{sik})$;
23. 　　$D_{(1-s)(j+1)k} |= (tmp \ll 1)$;
24. 　　**end for**
25. 　**end for**
26. 　**for** $i = i$ **to** $n_2 - 1 \quad i++, j++$ **do**
27. 　　**for** $k = 0$ **to** $2n-1$ **do**
28. 　　$D_{(1-s)jk} = D_{sik}$;
29. 　　**end for**
30. 　**end for**
31. 　$s = 1 - s, n_2 -= n_2/3$;
32. **end while**
33. $C = sum(D_{s0}, D_{s1})$

1 行目の 3 次元配列 D は多倍長整数 A, B の部分積を格納するダブルバッファである．3〜15 行目で部分積を D に格納し，16〜32 行目では多倍長整数に対応した Carry-save adder を実行して 3 つの部分積を 2 つの多倍長整数に変換している．このとき，変換した多倍長整数は while ループが繰り返されるたびに D[0][i] と D[1][j] を交互に行き来する．多倍長整数が 2 つしか残らなくなるまでこの変換を続け，最後に 2 つの多倍長整数の単純な加算を行って C に格納している．

4. 提案手法

今回の提案手法は 3.2 節で述べた Wallace 木乗算器に基づいた CPU 向けの多倍長乗算プログラムをナイーブに実装したものである．以下にプログラムの流れに沿って CUDA を用いた GPU 向けの実装について説明する．ただし，今回の実装では乗数・被乗数の配列の要素数が等しいと仮定し，その要素数を起動するスレッド数として指定している．

4.1. 部分積の計算および格納

部分積の計算は乗数の 1 要素と被乗数の各要素との積を並列に計算して繰り上がりの処理を行うという操作を繰り返している．図 4.1 は乗数・被乗数の配列の要素数が 4 のときの操作のイメージである．配列 *tmp* は unsigned long long 型で積の結果を一時的に格納するための物であり，レジスタから領域を確保した．tid はスレッドのインデックスを表している．

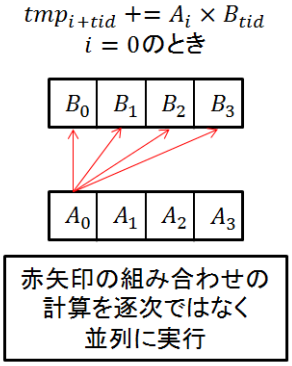


図 6: 要素数が 4 の時の乗数 A の 1 要素 A_0 と被乗数 B の各要素の部分積の計算のイメージ

4.2. 3 つの多倍長整数の和を 2 つの多倍長整数の和に変換

乗数・被乗数の要素数 n が等しいので，積の要素数はその 2 倍になる．そのため，バッファは要素数 $2n$ 個ごとに 1 つの和を格納している．スレッドは要素数個しか起動していないため，バッファ内の $2n$ 個ごとの配列について先頭と n 個分ずれた場所の 2 箇所から n 個の要素に対して並列に処理を行っている．

4.3. 変換の結果残った 2 つの多倍長整数の合計を計算

3 つの多倍長整数を 2 つに変換する操作を繰り返すと最終的に 2 つの多倍長整数が残る．この 2 つの和を求めるときはスレッドを 1 つだけ使用し，CPU と同じように逐次的に処理を行う．

ただし，この方法での実装は 1 スレッドブロックしか使わず，1024 スレッドまでしか呼べないうえ配列の要素数も 1024 までと制限されてしまっている．そこで，先ほど述べた 2 つの操作に対してスレッドの同期が必要な部分で切り離して別々のカーネル関数として定義し，複数のカーネル関数から成るプログラムも用意した．これによって複数のスレッドブロックで実行が可能となり並列性が向上するメリットがあるが，実装の都合上同じ計算を異なるカーネル関数で繰り返す必要があり計算量が増えてしまうというデメリットもある．

表 1: CPU と GPU の実行環境

サーバ		名称	クロック
CPU サーバ		Xeon E3-1220V5	3.0GHz
GPU サーバ	CPU	Intel Core i7-6700T	2.8GHz
	GPU	GeForce GTX 1080	1.6GHz

表 2: 1 スレッドブロックのみ起動する GPU 実装と CPU の逐次実装，MPIR の乗算演算子の実行時間の計測結果（単位は秒）

配列の要素数	MPIR	CPU	GPU
128	1.35×10^{-5}	9.30×10^{-6}	0.4806017
256	3.98×10^{-5}	8.25×10^{-5}	0.486304526
512	0.000112592	9.07×10^{-5}	0.4731121
1024	0.000244656	0.000479681	0.481087263

5. 評価実験・結果

今回の実験における実験環境は表 1 のとおりである．Wallace 木乗算器に基づいた多倍長乗算を CPU で実装した逐次的なプログラムと MPIR の乗算演算子を CPU サーバで，GPU を用いた提案手法のプログラムを GPU サーバで実行し処理時間を計測した．

各サーバの OS は Windows Server 2012 R2，コンパイラは Visual Studio 2012 Ultimate，CUDA のバージョンは CUDA v7.5 である．

時間の計測について，CUDA はプログラマーが明示的に GPU の VRAM 上にメモリを確保したのち CPU から GPU へ必要なデータを転送し，処理が終わった後に GPU から CPU に結果を書き戻す必要がある．今回実装した提案手法では初めに CPU から GPU へデータを 1 度転送したあと GPU 上で大量の多倍長整数の計算を行い，すべての計算が終わってから 1 度だけ CPU に結果を書き戻すという使い方を想定している．そのため，カーネル関数が起動されるたびに CPU と GPU でデータのやりとりが必要になるというようなことはなく，転送時間を無視しても問題ないので計測時間にはデータ転送の時間は含めず，多倍長乗算の計算時間のみを計測した．また，今回比較対象として用いた CPU 用の多倍長演算ライブラリ MPIR の乗算演算子は，配列の要素数が $n > 32$ である場合 k-way Toom-Cook 法（$k \geq 3$）や FFT といったより高速なアルゴリズムに自動的に切り替わるようプログラムされている．

表 2 は乗数・被乗数の配列の要素数 NUM を 128 から 1024 まで 2 倍ずつ増やした時の 1 スレッドブロックしか起動できない GPU 実装を用いた場合の実行時間の変化を表している．3 つある凡例のうち，CPU は Wallace 木乗算器に基づいて多倍長乗算を CPU で実装した逐次的なプログラム，GPU は提案手法のプログラム，MPIR は MPIR ライブラリの乗算演算子を表している．いずれの場合も GPU 実装の実行時間が最も遅い結果となった．GPU 実装が 1 スレッドブロックしか起動できないために GPU の性能を引き出せていないことが大きく影響していると思われる．

表 3 は乗数・被乗数の配列の要素数 NUM= 2000〜11000，1 ブロック当たりのスレッド数を 256 で固定して複数のスレッドブロックを起動したときの実行時間の変化を表している．3 つある凡例は表 2 と同様である．さきほどの 1 スレッドブロックしか起動できない場合と比べると並列性は確保できたものの，実行時間は改善されたとは言えない結果となった．並列性を確保したことによるメリットよりもプログラムが実行する計算の量が増えてしまったことによるデメリットが大きくなってしまった可能性がある．

表 3: 複数のスレッドブロックを起動する GPU 実装と CPU の逐次実装，MPIR の乗算演算子の実行時間の計測結果（単位は秒）

配列の要素数	MPIR	CPU	GPU
2000	0.000537156	0.000529322	0.51272615
4000	0.001521951	0.001511155	0.510047158
6000	0.00267839	0.002659055	0.52124945
8000	0.003480215	0.003490043	0.518306737
10000	0.003331582	0.003248075	0.5149249
11000	0.003873951	0.003782187	0.510929053

6. おわりに

本論文では Wallace 木乗算器に基づいた多倍長整数乗算を GPU を用いて実装し，CPU 実装の逐次プログラムおよび CPU の多倍長演算ライブラリである MPIR の乗算プログラムと比較した．その結果，いずれの場合も GPU 実装の性能が最も低いという結果となった．

今後の課題としてはプログラムの計算量を増やさずに複数スレッドブロックを起動して並列性を確保できるようにプログラムを改善すること，シェアードメモリを効果的に利用して実行時間を改善することが挙げられる．

参考文献

[1] 田中雄大，村尾雄一. *GPU* を用いた多倍長整数演算法の設計. 情報処理学会研究報告, Vol.2010-HPC-124 No.2, 2010.

[2] 中山空星，高橋大介. *GPU* 上における多倍長精度浮動小数点演算の実装. 情報処理学会研究報告, Vol.2011-ARC-197 No.25, Vol.2011-HPC-132 No.25, 2011.

[3] 北野晃司，藤本典幸. *GPU* による多倍長整数乗算の高速化手法の提案とその評価. 先進的計算基盤システムシンポジウム, SACSIS2012, 2012.

[4] Takumi Honda, Yasuaki Ito, Koji Nakano. *A Warp-synchronous Implementation for Multiple-length Multiplication on the GPU*. 2015 Third International Symposium on Computing and Networking, 2015.

[5] J. Cheng, M. Grossman, T. McKercher. 株式会社クイープ訳，*CUDA C* プロフェッショナル プログラミング. インプレス, 2015.

[6] NVIDIA. *CUDA Prgramming Guide Version 8.0*. https://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest（邦訳：浅野哲夫, 岩野和夫, 梅尾博司, 山下雅史, 和田幸一 共訳）. アルゴリズムイントロダクション 第 *3* 巻 精選トピックス. 近代科学社, 1995.

[8] J. W. Cooley, J. W. Tukey, *Math. of Comput. 19*, p.297, 1965.

[9] V. Strassen, *Gaussian Elimination is not Optimal*, Numer. Math. 13, 1969, pp.354-356.

[10] A. L. Toom, *The Complexity of a Scheme of Functional Elemtns Realizing the Multiplication of Integers*, Soviet Math. Doklady, No.3, 1963, pp.714-716.

[11] Free Software Foundation, *The GNU Multiple Precision Arithmetic Library*, https://gmplib.org.

CUDA を用いた多倍長加減算の実装について

紫垣 賢人，藤本典幸 †1

1. はじめに

整数の加減算は最も基本的で重要な演算の一つであり，様々なアルゴリズムに用いられている．そのため，加減算を高速に処理できれば，それ自体の高速化だけではなく，加減算をサブルーチンとして用いる多くのアルゴリズムの高速化につながるため加減算の高速化は重要な問題である．

ただし，CPU において計算を行う場合，CPU の仕様によって一度に扱える整数の範囲が決まってしまうため，32 ビットや 64 ビットを超えた巨大な整数の計算を直接には行えない．例えば，32 ビット CPU では符号なし整数で $0 \sim 2^{32}$-1 の範囲の整数しか 1 度に扱えない．これは円周率の計算や暗号のような特定の処理を行うためには小さすぎる．

そこで，何らかの方法によって巨大な整数を CPU 上に表現し，演算することが必要となる．そのための方法が多倍長整数である．多倍長整数とは一度に扱える整数の桁数をより多く指定できる整数のことであり，1 倍長整数の配列で表現される．現在ではそのような多倍長整数の演算を行うためのライブラリとして GMP [1] が開発されている．文献 [2] で Bellard は，Intel Core i7 PC 上で GMP を用いて約 131 日で約 2.7 兆桁までの π を計算したとされている．しかし，文献 [3] で妥当な時間内にさらに多くの桁を計算するには，並列計算機において並列に処理することが必要と述べられている．

また，GPU(Graphic Processing Unit) による計算加速が近年注目されている．GPU は本来画像処理に用いられるプロセッサであるが，汎用計算に用いることも可能であり，CPU 単体ですべての計算を行うよりも高速に演算可能である．このことから，多倍長演算は多数のプロセッサを持ち同時に複数の処理が可能である GPU による高速化が期待できる．また，Intel 社の 64 ビット CPU 向けにアセンブリ言語を用いて高度に最適化された GMP の派生ライブラリとして，GMP と互換のある MPIR [4] が開発されている．MPIR は高度に最適化されているため，より高速に処理できると考え，本研究で比較対象として使用した．

本研究では，文献 [3] で提案されている並列多倍長加減算アルゴリズムを GPU 向けに実装し，MPIR ライブラリを用いた多倍長加減算と性能比較を行い，高速化の可能性を検証する．提案手法の実装は MPIR との性能比較のため，MPIR ライブラリを用いて多倍長整数の入出力を行い，MPIR がメモリに読み込んだ多倍長整数を GPU において加減算を行うようにした．

以降の論文の構成は次の通りである．第 2 章で GPU と NVIDIA 社の GPU を対象とした GPGPU 向けプログラミング環境である CUDA [5] の説明を行う．第 3 章で GMP とその派生ライブラリである MPIR の説明を行う．第 4 章で多倍長遂次加減算について，文献 [3] で提案されている並列多倍長加算アルゴリズムとそれを基に作成した多倍長減算アルゴリズムの説明をした後，第 5 章でそれらのアルゴリズムを用いた提案手法を示す．第 6 章で評価実験について述べ，第 7 章でまとめと今後の課題について述べる．

†1 大阪府立大学 大学院工学研究科 電気・情報系専攻 知能情報工学分野

2. GPU と CUDA

2.1. GPU

GPU はディスプレイに出力するための画像をすばやく生成できる特殊なグラフィックスプロセッサとして開発されたが，現在では，超高速な処理が必要で，広範囲の計算処理を高速化するために GPU が CPU に追加されるケースが増えてきている．そのため，多くのデスクトップシステム，演算クラスタ，さらには世界最大級のスーパーコンピュータの多くに GPU が搭載されている．GPU は電力消費を抑えた上で，大量の演算コアを並列に動作させることで高速化を可能にしている．しかし，単に GPU に処理を任せればよいというわけではなく，並列処理に向いたアルゴリズムを考案する必要がある．データ依存性は並列化の妨げとなる主な要因の一つであり，これを分析することが重要である．

GPU アーキテクチャと CPU アーキテクチャはメニーコアおよびマルチコアとして分類されるが，GPU コアと CPU コアはまったく異なるものである．CPU コアは一度に実行するスレッドを 1 つか 2 つに制限することで，遅延を最小限に抑えるように設計されている．それに対して，GPU コアは大量のスレッドを並行かつ同時に実行することで，スループットを最大化するように設計されている．そのため，CPU は，問題のデータサイズが小さく，制御ロジックが複雑で，並列性が低い場合に適していて，GPU は，大量のデータを処理し，データの並列性が圧倒的に高い問題に適している．

現在のシステムは，GPU 上で動作するアプリケーションソフトウェアを作成するためのはるかに便利な手段が提供されており，それが CUDA である．

2.2. CUDA

CUDA は NVIDIA 社が提供する GPU 向けの統合開発環境であり，C 言語と C++言語を拡張した CUDA C/C++という言語により GPU プログラミング向けの機能を記述できる．ここで，CUDA によって操作できる NVIDIA 社製の GPU の構造について説明する．内部に複数のストリーミングマルチプロセッサ (SM) をもち，それぞれが GPU のメモリであるデバイスメモリに接続されている．一つの SM には，それ自体に命令ユニット，演算器，レジスタ，キャッシュが含まれているため，従来の CPU 的な観点で見れば，SM 自体は「プロセッサ・コア」と見立てることもできる．ただし，命令発行等の仕組みを含めて動作形態は大きく異なる．また，CUDA の目的は，直接 GPU 上で「データ並列」型の計算を行えるようにすることで,「データ並列」型とは多くの演算器で異なるデータ要素の計算を同時に行うことである．

C での並列プログラミングと CUDA C での並列プログラミングの主な違いは，メモリモデルや実行モデルといった CUDA のアーキテクチャ要素にプログラマが直接アクセスできることである．これによって，超並列 GPU 環境をより細かく制御することが可能になる．

一般的な CUDA プログラムの構造は，主に以下の 5 つのステップで構成される．

1. GPU メモリを確保する．
2. CPU メモリからデータを GPU メモリにコピーする．
3. CUDA カーネルを呼び出し，プログラムに必要な計算を実行する．
4. GPU メモリからデータを CPU メモリにコピーする．
5. GPU メモリを解放する．

このステップで鍵を握っているのはステップ 3 のカーネルである．カーネルとは GPU デバイスで実行されるコードのことで，

KernelName<<<Blocks，Threads>>>(Arguments)

という形式で呼び出すことができる．Blocks で 3 次元までのブロック数を指定し，Threads で 3 次元までのスレッド数を指定し，Arguments で任意の数の引数を指定する．カーネルが呼び出されると，指定された数のスレッドが呼び出されたカーネルを並列に実行するようになっている．

2.3. CUDA のメモリモデル

メモリへのアクセスとメモリの管理は，どのプログラミング言語においても重要な部分である．CUDA のメモリモデルではいくつかのプログラマブルメモリが提供されている．その中でも本研究に使用したグローバルメモリについて説明する．グローバルメモリは，GPU において最も大きく，最も遅延が高く，最もよく使用されるメモリである．確保されたグローバルメモリはアプリケーションが終了するまで有効であり，すべてのカーネルのすべてのスレッドからアクセスできる．その他のプログラマブルメモリに関しては文献 [5] を参照されたい．

図 1 は，CUDA のメモリモデルの階層を示している．

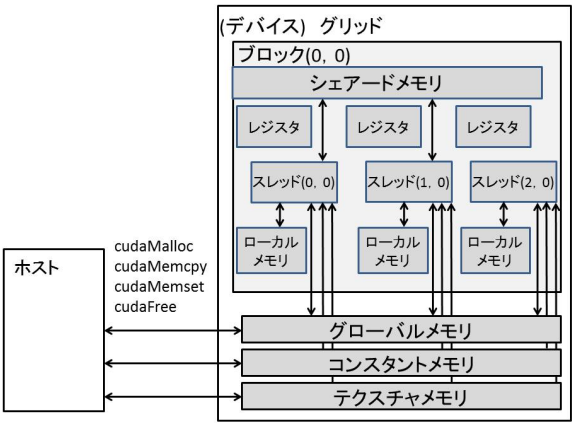


図 1: GPU のメモリ構造

2.4. CUDA によるマルチスレッドプログラム

並列なプログラム実行の最小単位としてスレッドが存在する．各スレッドにスレッド番号が与えられ，基本的に同じプログラムを実行するが，その番号によって異なるメモリ領域にアクセスする．スレッドはブロックと呼ばれる 3 次元のスレッド集合によってまとめられ，ブロックはグリッドと呼ばれる 3 次元のブロック集合によってまとめられる．これらの変数は dim3 型の変数で一括で管理できる．2 次元や 1 次元の dim3 型変数を宣言する場合は第 2，第 3 変数を省略することで表現できる．この時，同じブロックに存在するスレッドはすべて同じ SM 内で実行され，シェアードメモリの確保もブロック単位で行われる．また，異なるブロックのスレッドとデータの交換を行う場合はデバイスメモリを通じて行う．

3. GMP と MPIR

3.1. GMP

GMP とは GNU プロジェクトから提供されている多倍長演算ライブラリの一つで，正式名称は GNU Multiple Precision Arithmetic Library という．多倍長精度の符号付き整数だけではなく，有理数，浮動小数点数も扱うことができ，それらを対象とする算術関数を豊富に持っているため，多くのシステムが内部で GMP を利用している．事実上，動作中のハードウェアが持つメモリ容量以外には精度は制限されない．また，様々な関数がありそれらが一貫したインタフェースで提供されている．

主な対象アプリケーションは，暗号アプリケーションと研究，インターネットセキュリティアプリケーション，計算代数研究などである．

3.2. MPIR

MPIR とは Intel 社の 64 ビット CPU 向けにアセンブリ言語を用いて高度に最適化された GMP の派生ライブラリのことで，正式名称は Multiple Precision Integers and Rationals という．GMP と同様に多倍長精度の整数，有理数，および浮動小数点数のために C 言語で書かれたライブラリである．基本的な C 型で直接サポートされているものよりも高い精度が必要なすべてのアプリケーションに対して，可能な限り高速な算術演算を提供することを目的としている．

MPIR はオペランドのサイズに基づいてアルゴリズムを選択し，オーバーヘッドをできる限り最小限に抑えることによって，最適な演算を行えるように設計されている．また，MPIR の速度は洗練されたアルゴリズムを使用したり，多くの異なる CPU のための最も一般的な内部ループ用に慎重に最適化されたアセンブリ言語を組み込んだり，一般的なスピードを重視することにより最適化されている．

MPIR を用いて多倍長加算を行う C++プログラムの例を以下に示す．このプログラムでは 5 行目で MPIR 関数を用いるためのヘッダーファイルをインクルードし，7 行目で多倍長整数のビット数を定義している．10 行目で多倍長整数として用いる配列を定義して，12，13 行目でオブジェクト x または y が内部に保持する mpz_t 構造体へのポインタを得る．17 行目で乱数ルーチンの初期化を行い，21 行目で乱数の種を設定している (標準 C ライブラリの srand() に相当する)．23，24 行目で多倍長整数 x,y に乱数を生成し，26 行目で多倍長加算を行う．

減算の場合は，26 行目の + を - に変えるだけでよい．

```
1 : #include<iostream>
2 :
3 : using namespace std;
4 :
5 : #include<mpirxx.h>
6 :
7 : #define n 12800000
8 :
9 : int main(){
10:    mpz_class x, y, z;
11:
12:    const mpz_ptr mpzpx = x.get_mpz_t();
13:    const mpz_ptr mpzpy = y.get_mpz_t();
14:
15:    gmp_randstate_t state;
16:
17:    gmp_randinit_default(state);
18:
19:    mpz_class seed = 1234;
20:
21:    gmp_randseed(state, seed.get_mpz_t());
22:
23:    mpz_urandomb(mpzpx, state, n);
24:    mpz_urandomb(mpzpy, state, n);
25:
26:    z = x + y;
27:
28:    return 0;
29: }
```

4. 多倍長加減算のアルゴリズム

4.1. 多倍長加算

m 語の多倍長整数の加算の計算量は O(m) と並列処理するには小さいうえに，加算は繰り上がりの伝播を伴うため，逐次性が強い．このため加算の高速な並列化は容易ではない．

例として，多倍長逐次加算の疑似コードを Algorithm 1 に示す．ここで，carry は繰り上がりを格納するための変数である．

Algorithm 1: 多倍長逐次加算アルゴリズム

入力: 要素数 m の unsigned 型配列 x で表現された n ビット多倍長整数 X,
要素数 m の unsigned 型配列 y で表現された n ビット多倍長整数 Y

出力: 要素数 m+1 の unsigned 型配列 z で表現された n+1 ビット多倍長整数 X+Y

```
1: unsigned carry = 0
2: for i = 0 to n do
3:     z[i] = y[i] + carry
4:     carry = (z[i] < y[i])
5:     z[i] += x[i]
6:     carry += (z[i] < x[i])
7: z[i] = carry
8: carry = carry
```

Algorithm 1 を並列化可能にしたものが Algorithm 2 である．繰り上がりの伝播なしの多倍長加算は 1 行目で行われる．一方，2 行目では，1 行目の加算で z[i]<x[i] であれば繰り上がりが生じているので，c[i] に 1 が格納され，z[i]>x[i] であれば繰り上がりが生じていないので，c[i] に 0 が格納される．4 行目以降で繰り上がりの伝播がなくなるまで，計算が行われる．本研究では並列計算機向けに提案されている Algorithm 2 を GPU 向けに実装した．

Algorithm 2: 並列多倍長加算アルゴリズム [3]

入力: 要素数 m の unsigned 型配列 x で表現された n ビット多倍長整数 X,
要素数 m の unsigned 型配列 y で表現された n ビット多倍長整数 Y

出力: 要素数 m+1 の unsigned 型配列 z で表現された n+1 ビット多倍長整数 X+Y

```
1: z[i]=x[i]+y[i] を全ての i に対して並列に計算する
2: ステップ 1 の加算で繰り上がりが生じる場合は c[i] に 1 を，生じない場合は c[i] に 0 を格納する処理を全ての i に対して並列に行う
3: Max=0
4: 以下を繰り返す
     (1) a[i] = z[i] + c[i-1] を全ての i に対して並列に計算する
     (2) (1) の加算で繰り上がりが生じた場合は c[i] に 1 を，生じない場合は c[i] に 0 を格納する処理を全ての i に対して並列に行う
     (3) if(c[i]=1)Max=1 を全ての i に対して並列に行う
     (4) Max = 0 ならばループ終了
     (5) Max=0
     (6) z[i] に a[i] をコピーすることを全ての i に対して並列に行い，(1) に戻る
```

4.2. 繰り上がりの伝播が最悪の場合

繰り上がりの伝播が最悪の場合とは，図 2 のように x[i] と y[i] が与えられているときのことで，このような場合，全ての繰り上がりがなくなるまで繰り上がり c[i] が繰り返し伝播されてしまうため大きな時間を要してしまう．

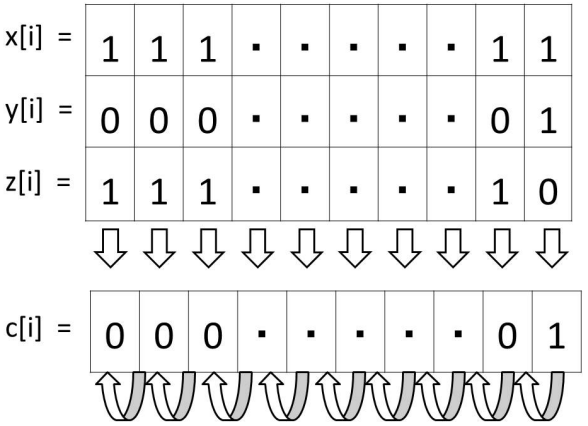


図 2: 繰り上がりの伝播が最悪のときの加算の様子

4.3. 多倍長減算

加算と同様に，m 語の多倍長整数の減算の計算量は O(m) と並列処理するには小さいうえに，減算は繰り下がりの伝播を伴うため，逐次性が強い．このため減算の高速な並列化は容易ではない．

例として，多倍長減算の疑似コードを Algorithm 3 に示す．ここで，borrow は繰り下がりを格納するための変数である．

Algorithm 3: 多倍長逐次減算アルゴリズム

入力: 要素数 m の unsigned 型配列 x で表現された n ビット多倍長整数 X,
要素数 m の unsigned 型配列 y で表現された n ビット多倍長整数 Y,
ただし，X≥Y

出力: 要素数 m の unsigned 型配列 z で表現された n ビット多倍長整数 X-Y

```
1: unsigned borrow = 0
2: for i = 0 to n-1 do
3:     unsigned b=(x[i] < (unsigned long long) y[i] + borrow)
4:     z[i] = x[i] - y[i] - borrow
5:     borrow = b
6: z[i] = x[i] - y[i] - borrow
```

並列多倍長加算アルゴリズムを基に作成した多倍長減算アルゴリズムが Algorithm 4 である．まず，x と y の値を比較して，x の方が大きければそのまま計算を行い，x の方が小さければ，x と y の値を入れ替えて計算を行う．3 行目では，2 行目の減算で z[i]>x[i] であれば繰り下がりが生じているので，b[i] に 1 が格納され，z[i]≤x[i] であれば繰り下がりが生じていないので b[i] に 0 が格納される．5 行目で繰り下がりの伝播がなくなるまで，計算が行われる．

Algorithm 4: 並列多倍長減算アルゴリズム [3]

入力: 要素数 m の unsigned 型配列 x で表現された n ビット多倍長整数 X,
要素数 m の unsigned 型配列 y で表現された n ビット多倍長整数 Y,
ただし，X≥Y

出力: 要素数 m の unsigned 型配列 z で表現された n ビット多倍長整数 X-Y

1: z[i] = x[i] - y[i] を全ての i に対して並列に計算する
2: ステップ 1 の減算で繰り下がりが生じる場合は b[i] に 1 を，生じない場合には b[i] に 0 を格納する処理を全ての i に対して並列に行う
3: Max = 0
4: 以下を繰り返す
   (1) a[i] = z[i] - b[i-1]
   (2) (1) の減算で繰り下がりが生じる場合は b[i] に 1 を，生じない場合には b[i] に 0 を格納する処理を全ての i に対して並列に行う
   (3) if(c[i]=1)Max=1 を全ての i に対して並列に行う
   (4) Max = 0 ならばループ終了
   (5) Max = 0
   (6) z[i] に a[i] をコピーすることを全ての i に対して並列に行い，(1) に戻る

4.4. 繰り下がりの伝播が最悪の場合

繰り下がりの伝播が最悪の場合とは，図 3 のように x[i] と y[i] が与えられているときのことで，このような場合，全ての繰り下がりがなくなるまで繰り下がり b[i] が繰り返し伝播されてしまうため大きな時間を要してしまう．

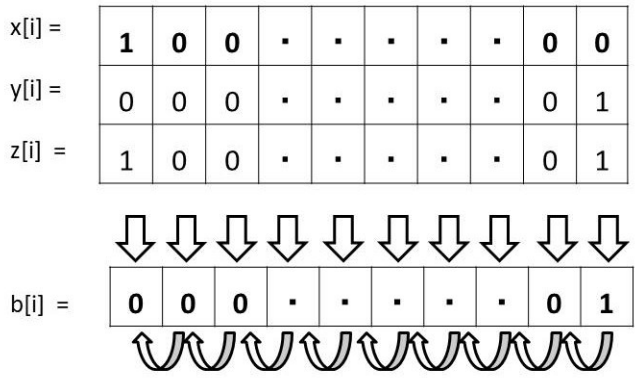


図 3: 繰り下がりの伝播が最悪のときの減算の様子

5. 提案手法

Algorithm 2 と Algorithm 4 の i を GPU の各ブロックのスレッドに対応させ，様々な多倍長整数のビット数に対して，ブロック数を ⌈⌈ ビット数/32⌉/256⌉，スレッド数を 256 として GPU で多倍長加減算を行う．

Algorithm 5 は Algorithm 2 を GPU で実装するために書き換えたものである．Algorithm 6 は Algorithm 4 を GPU で実装するために書き換えたものである．

Algorithm 5: 提案手法 (加算)

入力: 要素数 m の unsigned 型配列 x で表現された n ビット多倍長整数 X,
要素数 m の unsigned 型配列 y で表現された n ビット多倍長整数 Y

出力: 要素数 m+1 の unsigned 型配列 z で表現された n+1 ビット多倍長整数 X+Y

1: 要素数 m の unsigned 型配列 d_x,d_y,d_z,d_c,d_a を GPU メモリに確保する
2: CPU 側の x と y を GPU 側の d_x と d_y にそれぞれコピーする
3: GPU によって d_z[i]=d_x[i]+d_y[i] を全ての i に対して並列に計算する
4: ステップ 3 の加算で繰り上がりが生じる場合は d_c[i] に 1 を，生じない場合は d_c[i] に 0 を格納する処理を GPU によって全ての i に対して並列に行う
5: GPU 上の d_Max に 0 を代入する
6: 以下を繰り返す
   (1) GPU によって d_a[i] = d_z[i] + d_c[i-1] を全ての i に対して並列に計算する
   (2) (1) の加算で繰り上がりが生じた場合は d_c[i] に 1 を，生じない場合は d_c[i] に 0 を格納する処理を GPU によって全ての i に対して並列に行う
   (3) if(d_c[i]=1)d_Max=1 を GPU によって全ての i に対して並列に行う
   (4) GPU 側の d_Max を CPU 側の Max にコピーする
   (5) Max = 0 ならばループ終了
   (6) d_Max=0
   (7) d_z[i] に d_a[i] をコピーすることを GPU によって全ての i に対して並列に行い，(1) に戻る
7: cudaDeviceSynchronize(); で同期をとる
8: d_x,d_y,d_z,d_c,d_a を GPU メモリから解放する

Algorithm 6: 提案手法 (減算)

入力: 要素数 m の unsigned 型配列 x で表現された n ビット多倍長整数 X,
要素数 m の unsigned 型配列 y で表現された n ビット多倍長整数 Y,
ただし，X≥Y

出力: 要素数 m の unsigned 型配列 z で表現された n ビット多倍長整数 X-Y

1: 要素数 m の unsigned 型配列 d_x,d_y,d_z,d_b,d_a を GPU メモリに確保する
2: CPU 側の x と y を GPU 側の d_x と d_y にそれぞれコピーする
3: GPU によって d_z[i]=d_x[i]-d_y[i] を全ての i に対して並列に計算する
4: ステップ 3 の減算で繰り下がりが生じる場合は d_b[i] に 1 を，生じない場合は d_b[i] に 0 を格納する処理を GPU によって全ての i に対して並列に行う
5: GPU 上の d_Max に 0 を代入する
6: 以下を繰り返す
  (1) GPU によって d_a[i] = d_z[i] - d_b[i-1] を全ての i に対して並列に計算する
  (2) (1) の減算で繰り下がりが生じた場合は d_b[i] に 1 を，生じない場合は d_b[i] に 0 を格納する処理を GPU によって全ての i に対して並列に行う
  (3) if(d_b[i]=1)d_Max=1 を GPU によって全ての i に対して並列に行う
  (4) GPU 側の d_Max を CPU 側の Max にコピーする
  (5) Max = 0 ならばループ終了
  (6) d_Max=0
  (7) d_z[i] に d_a[i] をコピーすることを GPU によって全ての i に対して並列に行い，(1) に戻る
7: cudaDeviceSynchronize(); で同期をとる
8: d_x,d_y,d_z,d_b,d_a を GPU メモリから解放する

6. 評価実験

6.1. 実験設定

次の 6 つのアルゴリズムの処理時間の比較を行った．

1. CPU で多倍長逐次加算を行う
2. 提案手法を用いて GPU で多倍長加算を行う
3. MPIR のライブラリを用いて CPU で多倍長加算を行う
4. CPU で多倍長逐次減算を行う
5. 提案手法を用いて GPU で多倍長減算を行う
6. MPIR のライブラリを用いて CPU で多倍長減算を行う

この実験を多倍長整数がランダムに生成される場合と繰り上がりおよび繰り下がりの伝播が最悪の場合で行った．

ランダムに生成される場合は，多倍長整数のビット数を変化させていき，それぞれのビット数に対して 10 回の試行を行い，それによって算出された性能を以下の図にまとめた．

図 4：アルゴリズムの 1 と 2 を比較したもの
図 5：アルゴリズムの 2 と 3 を比較したもの
図 6：アルゴリズムの 4 と 5 を比較したもの
図 7：アルゴリズムの 5 と 6 を比較したもの

これらの図から最大でどれだけの違いが生じたか検証した．

繰り上がりの伝播が最悪の場合は，ビット数を 128000, 1280000, 12800000, 128000000 の場合のみで実験を行い，表 1 にまとめた．繰り下がりの伝播が最悪の場合も同様に，ビット数を 128000, 1280000, 12800000, 128000000 の場合のみで実験を行い，表 2 にまとめた．

6.2. 実験環境

実験を行った環境を以下に示す．

CPU Intel Core i7-6700T(2.8GHz)
GPU GeForce GTX 1080
CUDA コア 2560 コア
VRAM 8GB
OS Windows Server 2012 R2
CUDA のバージョン CUDA7.5
コンパイラ Visual Studio 2012

6.3. 提案手法の性能

図 4 および図 6 は多倍長整数をランダムに生成した場合の提案手法と逐次アルゴリズムの比較実験の結果をまとめたものであり，図 5 および図 7 は多倍長整数をランダムに生成した場合の提案手法と MPIR の実験結果をまとめたものである．それぞれ横軸を多倍長整数のビット数，縦軸を逐次アルゴリズムおよび MPIR の処理時間/提案手法の処理時間としている．

図 4 から加算する数のビット数が 12800000 を超えたあたりから逐次アルゴリズムよりも提案手法の方が速くなり，最大で約 5.5 倍速くなっていることがわかり，図 5 から加算する数のビット数が 12800000 を超えたあたりから MPIR よりも提案手法の方が速くなり，最大で約 4 倍高速になっていることがわかった．

また，図 6 から減算する数のビット数が 12800000 を超えたあたりから逐次アルゴリズムよりも提案手法の方が速くなり，最大で約 12.5 倍になっていることがわかった．図 7 から減算する数のビット数が 12800000 を超えたあたりから MPIR よりも提案手法の方が速くなり，最大で約 9 倍高速になっていることがわかった．

表 1 は繰り上がりの伝播が最悪の場合についての実験結果をまとめたものである．この表から明らかに逐次アルゴリズムと MPIR よりも提案手法の方が遅くなっていることがわかった．また，表 2 は繰り下がりの伝播が最悪の場合についての実験結果をまとめたもので，この表から明らかに逐次アルゴリズムと MPIR よりも提案手法の方が遅くなっていることがわかった．しかし，加算も減算もこのような最悪の場合はほとんど現れないため，提案手法により多くの場合で多倍長加減算は高速化可能であるといえる．

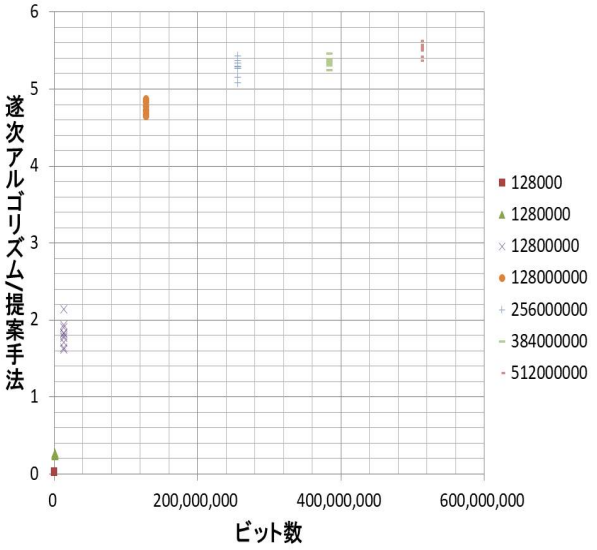


図 4: 提案手法と逐次アルゴリズムの性能比較 (加算)

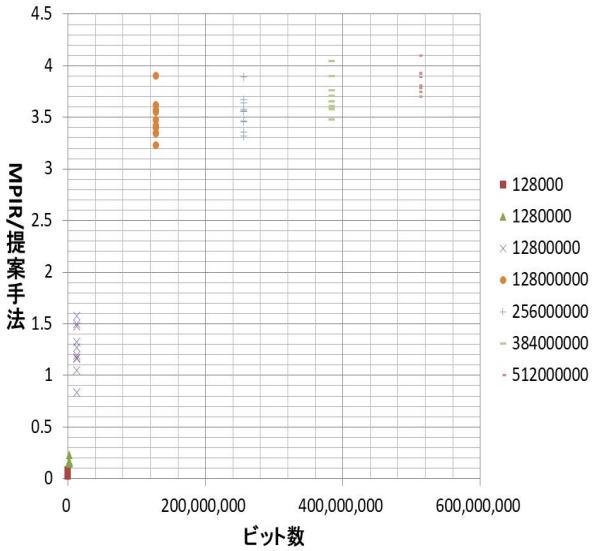


図 5: 提案手法と MPIR の性能比較 (加算)

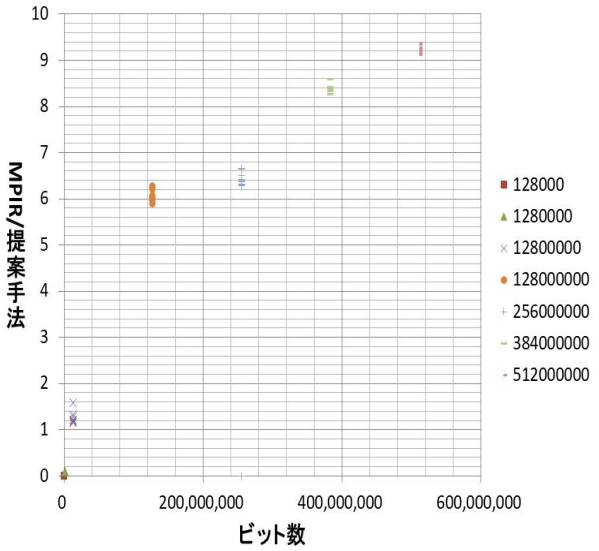


図 7: 提案手法と MPIR の性能比較 (減算)

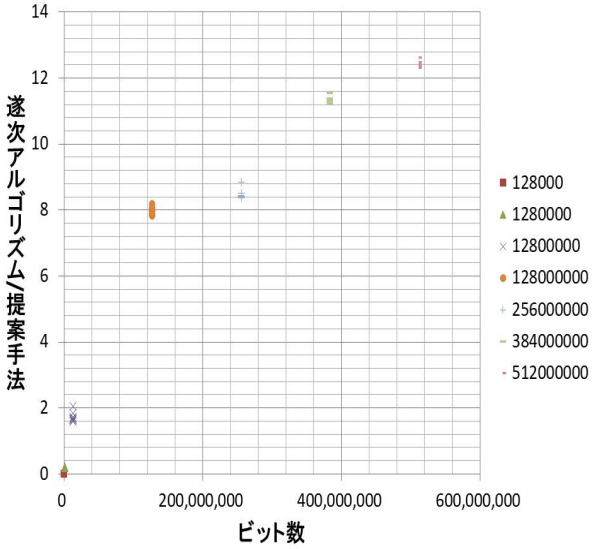


図 6: 提案手法と遂次アルゴリズムの性能比較 (減算)

表 1: 提案手法と遂次アルゴリズム，MPIR の性能比較 (繰り上がりの伝播が最悪の場合)

ビット数	遂次アルゴリズムの処理時間(s)	MPIRの処理時間(s)	提案手法の処理時間(s)
128000	0.00000893	0.00000641	0.139
1280000	0.0000923	0.0000465	1.551
12800000	0.00110	0.000915	28.515
128000000	0.0102	0.00483	1718.580

表 2: 提案手法と遂次アルゴリズム，MPIR の性能比較 (繰り下がりの伝播が最悪の場合)

ビット数	遂次アルゴリズムの処理時間(s)	MPIRの処理時間(s)	提案手法の処理時間(s)
128000	0.00000777	0.00000286	0.493
1280000	0.0000812	0.0000179	4.639
12800000	0.000913	0.000517	67.419
128000000	0.00852	0.00443	2819.052

7. おわりに

本論文では，高速化が難しいと言われている多倍長加減算を提案手法によって実装し，逐次アルゴリズムおよび MPIR との性能比較を行い，並列化は有効であるか，高速化できるかを検証した．その結果，提案手法により並列化は有効であり，高速に計算できる場合があること，繰り上がりの伝播および繰り下がりの伝播が最悪の場合では提案手法は効果的ではないことを示した．しかし，入力多倍長整数の確率分布が一様分布であると仮定すると，最悪の場合が起こることはビット数の大きさからしても確率的にほとんどないため，提案手法によって多くの場合，多倍長加減算は高速化可能であると考えられる．

今後の課題は本研究では 512000000 までのビット数しか計算を行うことができなかったため，より大きなビット数でも計算できるようにすること，また，提案手法をより高速に処理する方法を検証すること，他のアルゴリズムに応用してそのアルゴリズムの高速化の可能性を検証することである．

参考文献

[1] Free Software Foundation. *The GNU Multiple Precision Arithmetic Library*. `https://gmplib.org/`, 1991.

[2] F.Bellard. *Computation of 2700 billion decimal digits of pi using a desktop computer*. 2010.

[3] D.Takahashi. Parallel Implementation of Multiple-Precision Arithmetic and 2,576,980,370,000 Decimal Digits of π Calculation. *Parallel Comput.*, 36(8):439–448, 2010.

[4] W. Hart. *MPIR:Multiple Precision Integers and Rationals*. `http://mpir.org/links.html`, 2015.

[5] J. Cheng, M. Grossman, T. McKercher. CUDA C プロフェッショナル プログラミング. インプレス, 株式会社クイープ訳, 2015.

CUDA を用いた多倍長除算と多倍長平方根演算の実装について

川口大輔, 藤田崚太, 藤本典幸 †1

1. はじめに

CPU が直接扱える数の範囲には限りがあり，言語によってはその制限が残っていることが多い．C 言語を例にとると，int 型の変数は通常 CPU のレジスタのサイズとなるため，32bit の CPU では $2^{32} = 4294967296$ 種類の数しか表現することができないことが多い．この制限を超過するような数を多倍長数と呼び，多倍長数を扱うためには限られた桁数の数のみによる任意の桁数の数の計算，つまり多倍長演算を行うプログラムを作成する必要がある．本研究では，Goldschmidt 法 [1] を用いた多倍長除算及び多倍長平方根演算の実装を，NVIDIA 社が提供する GPU 向けの並列コンピューティングアーキテクチャである CUDA(Compute Unified Device Architecture)[2] によって行い，CPU 上で多倍長演算を行うために最適化されたライブラリである MPIR(Multiple Precision Integers and Rationals)[3] と比較し，高速化の可能性を検証する．

2. Goldschmidt 法

本章では，本研究で並列化の対象とした除算，及び平方根演算のアルゴリズムを述べる．

2.1. Goldschmidt 法による除算

a_0/b_0 を求める場合，

$$\frac{a_0}{b_0} = \frac{a_0 Y_0 ... Y_{i-1}}{b_0 Y_0 ... Y_{i-1}} = \frac{a_i}{b_i} = \frac{a_i Y_i}{b_i Y_i} = \frac{a_{i+1}}{b_{i+1}} \quad i = 1, 2, ... \quad (1)$$

が成立する．もしも b_n の値が 1 に十分近づくならば，a_n が求める値となる．そのような実数列 $Y_0, Y_1, ...$ は次のように求められる．$3/4 \leq Y_0 b_0 < 3/2$ を満たす $1/b_0$ の近似値を Y_0 とし，$1/b_0$ の相対誤差を e とする時 ($e = \frac{1/b_0 - Y_0}{1/b_0} = 1 - b_0 Y_0$)，

$$e^{2^{i+1}} = (e^{2^i})(e^{2^i}) \quad (2)$$

$$\prod_{k=0}^{i+1} Y_k = \prod_{k=0}^{i} Y_k + \prod_{k=0}^{i} Y_k \cdot (e^{2^i}) \quad (3)$$

と定義できる．この更新を繰り返すことで相対誤差 e^{2^i} の $1/b_0$ の値が得られる．この値に a_0 を乗じれば a_0/b_0 の値が得られる．

2.2. Goldschmidt 法による平方根演算

$\sqrt{b_0}$ を求める場合，

$$b_n = b_0 Y_0^2 Y_1^2 ... Y_{n-1}^2 \quad (4)$$

において，b_n の値が 1 に十分近づく時，

$$y_n = Y_0 Y_1 ... Y_{n-1} = 1/\sqrt{b_0} \quad (5)$$

$$g_n = b_0 Y_0 Y_1 ... Y_{n-1} = \sqrt{b_0} \quad (6)$$

となる．そのような実数列 $Y_0, Y_1, ...$ を考えると，$1/2 < b_0 Y_0^2 < 3/2$ を満たす $1/\sqrt{b_0}$ の近似値 Y_0 に対して，$y_0 = Y_0$, $g_0 = b_0 y_0$, $h_0 = y_0/2$ として

$$r_{i-1} = 0.5 - g_{i-1} h_{i-1} \quad (7)$$

$$g_i = g_{i-1} + g_{i-1} r_{i-1} \quad (8)$$

$$h_i = h_{i-1} + h_{i-1} r_{i-1} \quad (9)$$

の更新を繰り返すことで $\sqrt{b_0}$ の値が g_n に得られることがわかる．

†1 大阪府立大学 大学院工学研究科 電気・情報系専攻 知能情報工学分野

3. CUDA を用いた Goldschmidt 法の実装

我々の研究グループで実装された多倍長浮動小数点数クラス”cuda_mpf_class”[4] による乗算 [5]，及び加減算 [6] を組み合わせることで GPU 環境で実行できる多倍長除算，及び多倍長平方根演算を実装した．そのソースコードを図 1，及び図 2 に示す．GPU 上で演算を行う場合，CPU から GPU へデータを転送する必要があるが，この”cuda_mpf_class” はオブジェクト生成時のみデータの転送が自動的に行われる実装となっている．この特徴により，ソースコードに示した do-while 文の最中に CPU と GPU 間で多倍長数を表すデータのデータ転送が行われることはなく，全て GPU 上で繰り返し計算が行われるため，効率的な演算が可能である．

4. 実験手法と結果

CUDA で上記の Goldschmidt 法を実装した場合，それを CPU 用に実装した場合，MPIR の除算演算子と sqrt 関数を用いて演算を行った場合の演算速度の比較を行う．10 進数 $Digits$ 桁の多倍長整数 a，b に対して，$50 \leq Digits \leq 300$ の範囲で $Digits$ の値を変動させ，各 $Digits$ に対して 20 回演算を行いその実験結果をプロットした散布図を図 3，及び図 4 に示し，平均計算時間をまとめたものを表 1，表 2 に示す．なお，CPU 用に実装したプログラムと MPIR で実装したプログラムの実験環境は”3.0GHz Xeon E3-1220V5”であり，CUDA で実装したプログラムの実験環境は，CPU は”Intel Core i7-6700T(2.8GHz)”，GPU は”GeForce GTX 1080 （2560 コア，VRAM8GB）” である．

5. 考察

実験結果より，いずれの場合も CUDA による実装は MPIR による実装と比較して数百倍から数千倍処理速度が遅いことが分かる．今回の実装では，初期値 Y_0(除算の場合は $1/b_0$ の近似値，平方根演算の場合は $1/\sqrt{b}$ の近似値を示す) を求める際に double 型変数を用いたことにより 10 進数 300 桁までの多倍長数にしか対応しておらず GPU の性能を十分に引き出せていないため，このような結果になったのだと考えられる．オーバーフローを生じさせずに初期値 Y_0 を求める手法で再度実装し，より巨大な多倍長数での実験を行うことが今後の課題である．

参考文献

[1] P. Markstein, Software Division and Square Root Using Goldschmidt’s Algorithms, CiteSeerX 10.1.1.85.9648，2004

[2] NVIDIA Corporation, CUDA Zone, https://developer.nvidia.com/cuda-zone，2017

[3] B. Gladman, W. Hart and J. Moxham, et al., MPIR: Multiple Precision Integers and Rationals, http://mpir.org/，2015

[4] 藤田崚太，藤本典幸，澤田幸一郎，紫垣賢人，川口大輔，多倍長演算ライブラリ MPIR 互換の CUDA ライブラリの実装について，第 13 回情報科学ワークショップ，2017

[5] 澤田幸一郎，藤本典幸，和田幸一，CUDA を用いた多倍長乗算の実装について，第 13 回情報科学ワークショップ，2017

[6] 紫垣賢人，藤本典幸，CUDA を用いた多倍長加減算の実装について，第 13 回情報科学ワークショップ，2017

```
cuda_mpf_class operator /
(const cuda_mpf_class& a,
 const cuda_mpf_class& b)
{
 if (b.sign == 0)
   throw std::out_of_range("division by zero");
 if (a.sign == 0)
  return 0;

 cuda_mpf_class
  y = 1 / b.get_d(), e = 1 - b * y;
 cuda_mpf_class
  epsilon(2,-nap_int::default_prec);

 do {
  y += y * e;
  e *= e;
 } while (e > epsilon);

  return a * y;
}
```

図 1: 多倍長除算のソースコード

```
cuda_mpf_class sqrt
(const cuda_mpf_class& b0)
{
 if (b0.sign < 0)
  throw std::out_of_range
  ("sqrt for a negative value");

 cuda_mpf_class
  y = 1 / sqrt(b0.get_d()), g, h, r, g0;
 cuda_mpf_class
  epsilon(2,-nap_int::default_prec);

 g = b0 * y;
 h = y / 2;

 do {
  g0 = g;
  r = 0.5 - g * h;
  g += g * r;
  h += h * r;
 } while(abs( g - g0) > epsilon);

 return g;
}
```

図 2: 多倍長平方根演算のソースコード

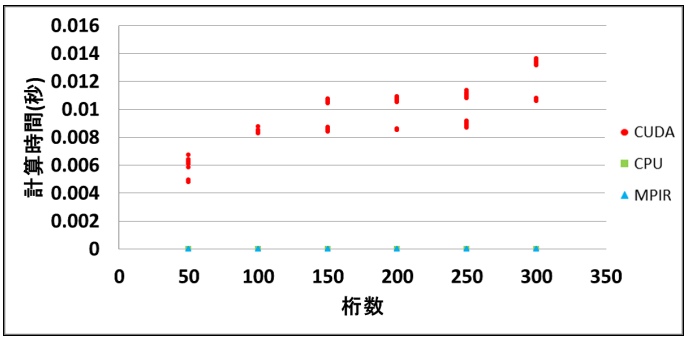


図 3: 多倍長整数 a と b の桁数を変動させた場合の a/b の計算時間 (秒)

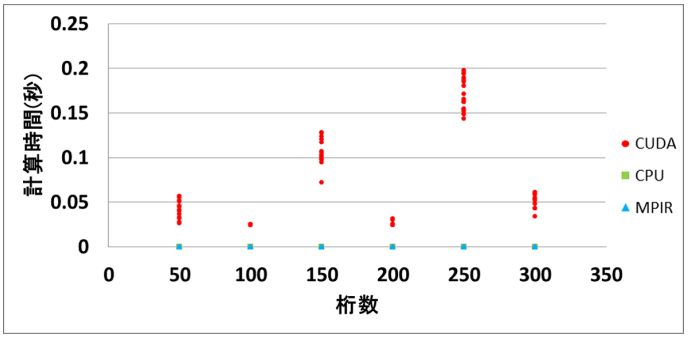


図 4: 多倍長整数 b の桁数を変動させた場合の $\sqrt{b}$ の計算時間 (秒)

表 1: 多倍長整数 a と b の桁数を変動させた場合の a/b の平均計算時間 (秒)

Digits	CUDA	CPU	MPIR
50	4.86550×10^{-3}	3.54986×10^{-6}	3.24266×10^{-7}
100	6.64827×10^{-3}	2.79893×10^{-6}	5.12000×10^{-7}
150	9.37594×10^{-3}	4.28373×10^{-6}	6.82666×10^{-7}
200	8.67085×10^{-3}	3.48160×10^{-6}	9.38666×10^{-7}
250	9.79568×10^{-3}	6.02453×10^{-6}	1.14347×10^{-6}
300	1.11313×10^{-2}	3.53280×10^{-6}	1.58720×10^{-6}

表 2: 多倍長整数 b の桁数を変動させた場合の $\sqrt{b}$ の平均計算時間 (秒)

Digits	CUDA	CPU	MPIR
50	3.63659×10^{-2}	1.90805×10^{-5}	5.11999×10^{-7}
100	1.96853×10^{-2}	6.86079×10^{-6}	1.02400×10^{-6}
150	1.08312×10^{-1}	5.39477×10^{-5}	1.29707×10^{-6}
200	2.48594×10^{-2}	4.43733×10^{-6}	1.41653×10^{-6}
250	1.69199×10^{-1}	7.43936×10^{-5}	1.72373×10^{-6}
300	4.65398×10^{-2}	1.67253×10^{-5}	2.25280×10^{-6}