第 12 回
情報科学ワークショップ

The 12th Workshop on
Theoretical Computer Science,
Yamanashi, September 2016
(WTCS2016)

和田 幸一
真鍋 義文
DEFAGO Xavier
首藤 裕一

主担当：法政大学

序言

本論文集は 2016 年 9 月 14 日～16 日，工学院大学富士吉田セミナー校舎（山梨県）にて開催された第 12 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，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．第 1 回は 2005 年 9 月に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 回北名古屋に続き，今回で 12 回目の開催となる． 今回は大阪大学，名古屋工業大学，九州工業大学，奈良先端科学技術大学院大学，広島大学，東京工業大学，工学院大学，法政大学から 34 人の参加者があり，25 件の発表が行われた．今回の開催場所は，山梨県富士吉田市の工学院大学富士吉田セミナー校舎で，富士山のすそ野に位置し夏でも涼しく都会からは隔絶された研究討論に適した場所である．このような研究討論に適した場所で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 年度からは，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プレゼンテー ション賞を授与した．本予稿集に受賞者一覧を掲載している．受賞者らには今後益々の活躍を，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 2017 年度は大阪大学・奈良先端科学技術大学院大学を中心とした大阪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 7 月

和田 幸一
真鍋 義文
Defago Xavier
首藤 裕一

目次

セッション 6：グラフ・ネットワーク

セッション 7：ロボット

9月14日 （13：00～18：00）

自由討論	

9月15日 （8：55～16：55）

8:55				**開会（5min）**	
個体群プロトコル				座長 ： 中村 純哉	
番号	時刻	名前	所属	タイトル	発表時間
A-1	9:00	金鎔煥	名工大	3次元グリッドネットワーク上の自己最適化ルーティングアルゴリズム	ロング
A-2	9:30	清洲 星顕	大阪大	極大距離k-独立集合問題に対する緩自己安定個体群プロトコル	ミドル
A-3	9:50	江口 僚太	名工大	個体群プロトコルの高速な近似計数プロトコル	ミドル
A-4	10:10	片岡 大輝	名工大	緩安定個体群プロトコルの能力について	ショート
Coffee Break (15min)					
分散アルゴリズム				座長 ：金 鎔煥	
番号	時刻	名前	所属	タイトル	発表時間
B-1	10:40	小貫 直之	東工大	分散プロトコル合成とビジュアライゼーション・ツール	ショート
B-2	10:55	五島 剛	大阪大	How to simulate message-passing algorithms by mobile agents with crash faults	ミドル
B-3	11:15	木下 崇央	東工大	IoTのための耐侵入ミドルウェア開発	ショート
Lunch Break (100min)					
アルゴリズム１				座長 ： 大下福仁	
番号	時間	名前	所属	タイトル	発表時間
C-1	13:10	土中 哲秀	九州大	On the maximum weight minimal separator	ロング
C-2	13:40	清水航平	工学院	分割不可能な財のオンライン配分問題	ミドル
C-3	14:00	北村直暉	名工大	一様な分布を持つパチンコ台の釘配置(1)	ミドル
C-4	14:20	川端 祐也	名工大	一様な分布を持つパチンコ台の釘配置(2)	ショート
Coffee Break (15min)					
アルゴリズム２				座長 ： Defago Xavier	
番号	時刻	名前	所属	タイトル	発表時間
D-1	14:50	川邉茂和	工学院	繁殖戦略解析のための考察	ミドル
D-2	15:10	木谷 裕紀	九州大	単貧民における最適戦略と必勝戦略に関する考察	ミドル
D-3	15:30	早川 駿	名工大	最小遮光線長についての検討	ショート
D-4	15:45	石井 大也	名工大	クロストーク回避符号の符号化率と最大クリーク問題の関係	ショート
Coffee Break (15min)					
モバイル				座長 ： 泉泰介	
番号	時刻	名前	所属	タイトル	発表時間
E-1	16:15	柴田将拡	大阪大	リングにおけるメモリ効率の良いモバイルエージェント均一配置アルゴリズム	ミドル
E-2	16:35	土田将司	奈良先端大	認証機能付き白板を用いたビザンチン故障耐性を持つモバイルエージェント集合アルゴリズム	ミドル

9月16日 （9：00～12：30）

グラフ・ネットワーク				座長 ： 土中 哲秀	
番号	時間	名前	所属	タイトル	発表時間
F-1	9:00	中野浩嗣	広島大	グリッドグラフについての考察	ロング
F-2	9:30	吉町 優	工学院	MANET向けの通信性質を考慮した公平性ルーティング	ミドル
F-3	9:50	渡部 連太郎	大阪大	無線センサネットワークにおけるベースステーションからの無線波情報を利用した負荷分散動的ルーティングア	ミドル
F-4	10:10	筒井 黎	九工大	カエルのサテライト行動に基づく連結支配集合の構築	ショート
Coffee Break (15min)					
ロボット				座長 ： 柴田将拡	
番号	時間	名前	所属	タイトル	発表時間
G-1	10:40	竹中 将成	名工大	単位円グラフの最小支配集合問題について	ショート
G-2	10:55	奥村太加志	法政大	状態を持つ２台の自律分散ロボットの計算能力について	ミドル
G-3	11:15	白川遥平	法政大	自律分散ファットロボットに対する様々なグリッド上における汎用集合アルゴリズムについて	ミドル
G-4	11:35	小玉 悠人	九工大	自律分散ロボット群における捕獲アルゴリズム	ショート
12:00				**表彰式・閉会（30min）**	

受賞者一覧

優秀研究賞（2 件，発表順）

- 北村直樹，川端裕也，泉泰介（名工大），
  “一様な分布を持つパチンコ台の釘配置”（C-3,C-4)
- 土田将司，大下福仁，井上美智子（奈良先端大学院大学），
  “認証機能付き白板を用いたビザンチン故障耐性を持つモバイルエージェント集合アルゴリズム”
  (E-2)

優秀プレゼンテーション賞（3 件，発表順）

- 江口僚太（名工大），
  ”個体群プロトコルの高速な近似係数プロトコル” (A-3)
- 五島剛（大阪大学），
  ”メッセージ通信型分散アルゴリズムの移動エージェントによる耐故障シミュレーション” (B-2)
- 吉町優（工学院大学），
  ”MANET 向けの通信性質を考慮した公平性ルーティングプロトコル” (F-2)

A Self-optimizing Routing Algorithm in a 3-dimensional Virtual Grid Network

Yonghwan KIM
Graduate School of Engineering
Nagoya Institute of Technology
Aichi, Japan, 466–8555
Email: kim@nitech.ac.jp

Yoshiaki KATAYAMA
Graduate School of Engineering
Nagoya Institute of Technology
Aichi, Japan, 466–8555
Email: katayama@nitech.ac.jp

***Abstract*—In this paper, we present a self-optimizing routing algorithm using local information only, in a three-dimensional (3D) virtual grid network. A virtual grid network is a well-known network model for its ease of designing algorithms and saving energy consumption. We consider a 3D virtual grid network which is obtained by virtually dividing a network into a set of unit cubes called *cells*. There is one specific node named a *router* at each cell, and each router is connected with the routers at adjacent cells. This implies that each router can communicate with 6 routers.**

We suppose one special node (named a *source* node) and one moving node (named a *destination* node) in a 3D virtual grid networks. We consider maintenance of an inter-cell communication path to a destination node from a source node. We propose an optimizing protocol in a 3D virtual grid network, which can transform an arbitrary given path (from a source node to a destination node) to the optimal (shortest) path using only local information (6 hops: 3 hops each back and forward along the routing path) of each router.

I. Introduction

Recently, wireless networks, such as MANETs (Mobile Ad-hoc NETworks)[2], [3] or WSNs (Wireless Sensor Networks)[4], become popular and important in the distributed systems. In the typical wireless networks, nodes are deployed on a two-dimensional plane, and each node can directly communicate only with nodes within its communication range. If the destination node (the node receives the message) is outside of the communication range of the source node (the node sends the message), the message should be relayed to the destination node. The topology of wireless networks can be changed frequently because of moving of nodes. Multi-hop routing algorithms for wireless networks have been proposed [6], [7], [8], [9], [10], [11].

A virtual grid network is a well-known topology model for designing algorithms of dynamic networks. A typical virtual grid network on a two-dimensional plane, which is obtained by virtually dividing a wireless network into a grid of geographical square regions are called cells of the same size. Figure 1 shows an example of the virtual grid network. Note that the size of the cells is determined by the communication range of each node.

In this paper, we consider a virtual grid network on three-dimensional (3D) space. Nodes are deployed in 3D space, each node can communicate with nodes within its communication range the same as the case of 2D plane. In a 3D virtual grid network, each cell's shape becomes a unit cube like Figure 2. Each node in 3D space has a spherical communication range instead of a circular communication range on a 2D plane and the size of each cell (unit cube) is determined by the communication range of each node. A specific single node called a router is selected at each cell (the marked node in Figure 2.) and each router communicates with routers at neighbor cells. Figure 3 illustrates an example of a 3D virtual grid network.

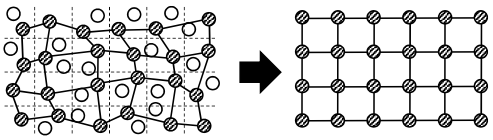


Fig. 1: A typical virtual grid network (on 2D plane)

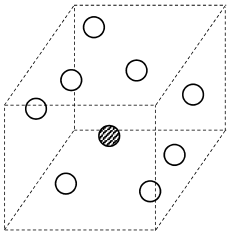


Fig. 2: A cell (unit cube) in the virtual grid network

Each node can communicate with other nodes that exist outside of the communication range using routers in a 2D/3D virtual network. We suppose two specific nodes, one named a

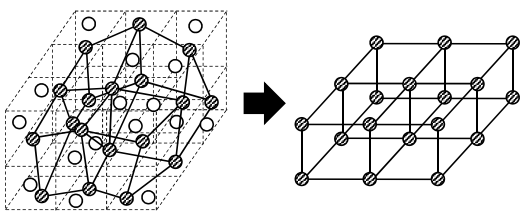


Fig. 3: A virtual grid network on 3D space

source node and the other named a *destination* node in virtual grid networks, and we allow that each node can move from its current cell to the neighbor cell freely. And we assume that there is a path, which is represented by the sequence of the routers (details will be introduced in Section 3), from a *source* node to a *destination* node. When a node moves to a neighbor cell (there are 6 neighbor cells in a 3D virtual grid network), the path is updated by extending itself to the next (neighbor) cell.

In this paper, we consider maintenance of the given path from a *source* node to a *destination* node. The power consumption is one of the important issues in wireless sensor networks, therefore, we consider that our goal is the construction of a shortest path from a *source* node to a *destination* node. Thus we propose a self-optimizing protocol which can transform an arbitrary given (initial) path from a *source* node to a *destination* node to a shortest path in a 3D virtual grid network.

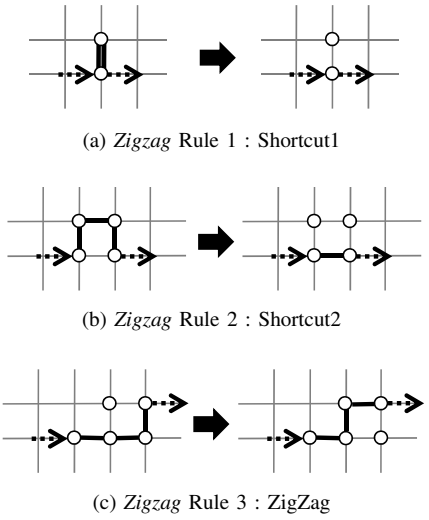
The rest of this paper is organized as follows: Section 2 presents related works and we introduce the system model we consider and explain the definition of the problem in Section 3. Our proposed self-optimizing algorithm for constructing a shortest path from a source node to a destination node in a 3D virtual grid network is represented in Section 4, and we conclude our work in Section 5.

II. Related Works

Xu et al. proposed a routing algorithm which is named GAF (Geographic Adaptive Fidelity)[5]. GAF reduces energy consumption in ad hoc wireless networks by identifying nodes that is necessary for a routing and turning off unnecessary nodes, keeping a consistent level of routing fidelity. To realize GAF protocol, each node associates itself with a virtual grid so that the entire area is divided into several square regions called cells. When each node has the communication (radio) range R, the length of each side of each grid r is set as follows: $r \leq \frac{R}{\sqrt{5}}$. Herewith all nodes in the cell can communicate with its adjacent cells. In a virtual grid network, a single node is selected as a router and it has responsibility for communication with its adjacent cells for routing and the other nodes can communicate with the router in the same cell when they need to communicate outside of their communication range (when no communication is needed, they can sleep for saving energy). A virtual grid network makes a routing problem simple because a grid architecture is simple and regular.

Takatsu et al. proposed a local-information-based self-optimizing routing algorithm in virtual grid networks which is named *Zigzag*[1]. *Zigzag* adopts a simple protocol consisting of only 3 rules. The system model of *Zigzag* is as follows:

- **Asynchronous Network**. Each node in the network can operate freely at any time.
- **No global information**. Each node does not know where a *source* node and a *destination node* are. There is no coordination system, this means that each node does not have any location information.
- **Labeled Links**. Each router maintains four communication links because it communicates with four routers in its adjacent cells. Each router labels its four communication links as UP, RIGHT, DOWN, and LEFT. All the routers have a common sense of direction. This means that if a sender sends a message through the link labeled RIGHT, a receiver receives it through the link labeled LEFT.
- **Local information within constant distance**. Each router maintains the subsequence of the routing path that contains 6 hops (directions) which consists of 3 hops back and forward along the path.



(a) *Zigzag* Rule 1 : Shortcut1

(b) *Zigzag* Rule 2 : Shortcut2

(c) *Zigzag* Rule 3 : ZigZag

Fig. 4: Three rules of *Zigzag* protocol

Zigzag uses only three rules to locally update a path. When a router finds a condition of these three rules, it locally update the subsequence of the path around it. Figure 4 simply shows the three rules of *Zigzag*.

Zigzag protocol ensures a construction of the shortest path within reasonable convergence time ($O(|P|)$, where $|P|$ is the length of initial given path) using only these three rules. *Zigzag* also guarantees the preservation of the connection between a *source* node and a *destination* node.

III. System Model and Problem Definition

In this section, we introduce our system model and define our problem.

A. System Model

We suppose all nodes in the system are deployed in 3D space. Each node has the sphere-shaped communication range with radius R, and it can communicate with the nodes within R. A 3D virtual grid network is obtained by virtually dividing an entire network into 3D grid (unit cube) cells. One node is selected as a router in each cell, each router can communicate with 6 routers in its adjacent cells. This implies that each vertex has 6 edges in a 3D virtual grid network.

There are two nodes, a *source* node v_s and a *destination* node v_t, in the system and we assume that some messages are periodically transferred from v_s to v_t. A message is sent from v_s to the router p_s which is the router in the same cell at the

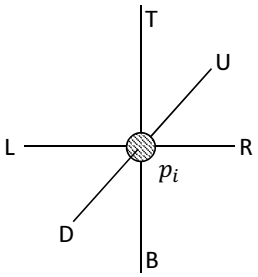


Fig. 5: A direction label of each link of the router p_i

beginning. The message is relayed by some routers to p_t which is the router in the same cell with v_t. Finally, the message is delivered to v_t. Note that v_s and p_s may be the same node, and v_t and p_t also may be the same. For ease of designing algorithms, we consider routers only in the 3D virtual grid network. The entire 3D virtual grid network is represented as $N_G = (P_G, E_G)$ (where, P_G is a set of routers and E_G is set of links of each p_i) which is undirected graph. A Router p_j is neighbor to p_i, this implies that p_i and p_j share a face of the cell, if and only if a link $(p_i, p_j) \in E_G$.

Each router p_i has 6 links because all cells are cube-shaped in a 3D virtual grid network. Each router has no knowledge of its global location, e.g. (x, y, z)-coordinates, however all the routers have a common sense of 6 directions, and they are labeled on these 6 links of each router. Figure 5 shows labels of all 6 links of p_i. Each router has 6 links which are labeled with T(Top), B(Bottom), U(Up), D(Down), L(Left) or R(Right) consistently. For example, if a router sends a message through the link labeled R, a receiving router receives the message through the link labeled L, because of common sense of direction.

A routing path P to a *destination* node p_t from a *source* node p_s can be represented by the sequence of routers as follows: $P = (p_s = p_0, p_1, p_2, \cdots, p_n = p_t)$. In this case, P consists of $(n + 1)$ routers and the length of P becomes n. We represent the length of the routing path with $|P|$, therefore $|P| = n$ implies that n-hops are required to deliver a message from p_s to p_t and $(n - 1)$ routers exist between p_s and p_t. Due to common sense of direction, each edge in P can be represented by a direction label. For example, if p_i sends a message through its link labeled R, $p_{(i+1)}$ receives it through its link labeled L inevitably. Therefore we can represent the edge of P between p_i and $p_{(i+1)}$ as a single direction label R which is p_i's outgoing link's direction. Thereafter P also can be represented by the sequence of edges or direction labels as follows: $P = (\overrightarrow{p_0, p_1}, \overrightarrow{p_1, p_2}, \overrightarrow{p_2, p_3}, \cdots, \overrightarrow{p_{(n-1)}, p_n}) = (Out(p_0), Out(p_1), Out(p_2), \cdots, Out(p_{(n-1)})$ (where $Out(p_i)$ is the direction label of p_i's output edge).

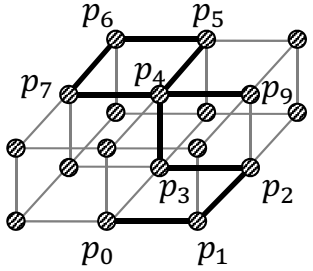


Fig. 6: An example of a routing path P

Figure 6 shows an example of the representation of P. In the case of Figure 6, $|P|$ consists of 10 routers, therefore $|P| = 9$. Note that the router p_4 and p_8 are considered as the different routers even though p_4 and p_8 are the same router. P can be represented by the sequence of the direction labels as follows: $P = (R, U, L, T, U, L, D, R, R)$. In this paper, we mainly use this notation (using direction labels) unless specifically mentioned. In addition, we use the notation D_i which means the direction label of p_i's output edge $Out(p_i)$.

Each router which belongs to the routing path from p_s to p_t maintains its routing table. A routing table consists of some records which is notated by the pair of input direction and output direction like (In, Out). For instance, p_2 in Figure 6 has the record (D, L) in its routing table because p_2 should forward a message received through the link labeled D to the link labeled L in order to transfer a message from a *source* node to a *destination* node. $p_4 (= p_8)$ stores two records in its routing table, (B, U) for p_4 and (L, R) for p_8.

We assume that each router knows routing records of preceding and following 2 routers (total 6 hops). For example, a router p_i maintains additional routing records of $p_{(i-2)}$, $p_{(i-1)}$, $p_{(i+1)}$, and $p_{(i+2)}$. A router p_i maintains P^{p_i} which is the subsequence of the current routing path P, P^{p_i} = ($Out(p_{(i-3)})$, $Out(p_{(i-2)})$, $Out(p_{(i-1)})$, $Out(p_i)$, $Out(p_{(i+1)})$, $Out(p_{(i+2)})$) and $|P^{p_i}|$ becomes 6 obviously (except some routers like p_s or p_t). Note that $Out(p_{(i-3)})$ can be derived from $p_{(i-2)}$'s routing record.

B. Problem Definition

A valid path P from p_s to p_t is initially given in a 3D virtual grid network. We say P is valid if and only if $P = (p_0, p_1, p_2, \cdots, p_n)$ fulfills the following conditions: (1) p_0 is the router in the same cell with v_s (i.e. $p_0 = p_s$), (2) p_n is the router in the same cell with v_t (i.e. $p_n = p_t$), and (3) Each p_i (except p_0 and p_n) is the router in the neighboring cell of $p_{(i-1)}$ and $p_{(i+1)}$'s. These conditions guarantee the connectivity of the path P.

The path optimization problem is, from any given initial valid path P in the 3D virtual grid network N_G, to construct the shortest path from p_s to p_t in N_G.

IV. Our Proposed Algorithm

In this section, we introduce our algorithm which solves the path optimization problem, using each router's local information only.

In the previous section, we introduced the notation of P using each router's direction label: $P = (Out(p_0), Out(p_1), Out(p_2), \cdots, Out(p_{(n-1)}))$.

Now we define an operation ($\star$) of 6 directions $D_i = \{U, D, R, L, T, B\}$ (see Figure 5) as follows. For ease of explanation, we call directions $\{U, D, R, L\}$ the *plane direction*, and directions $\{T, B\}$ the *vertical direction*. Note that this operation is not commutative.

1) **Straight Line** : When D_j is the same direction as D_i, $D_i \star D_j = 1$ (e.g. $D_i = D_j = R$).
2) **Retrace** : When D_j is the opposite direction of D_i, $D_i \star D_j = -1$ (e.g. $D_i = U$ and $D_j = D$).
3) **Bend on a Plane** : When both D_i and D_j are the *plane directions* and they are orthogonal, $D_i \star D_j = 0$ (e.g. $D_i = L$ and $D_j = U$).
4) **Horizontal Bend** : When D_i is the *vertical direction* and D_j is the *plane direction*, $D_i \star D_j = 2$ (e.g. $D_i = T$ and $D_j = U$).
5) **Null OP** : In the other case than listed above, $D_i \star D_j = \emptyset$. (e.g. $D_i = R$ and $D_j = B$).

This operation ($\star$) is basically the same as inner product of two vectors when both vectors are on a plane surface. However in the case of the bending from vertical direction to plane direction, we assign a new value 2 in our algorithm. Now we introduce our local update rules in the following subsection.

A. Local Update Rules

Let $P = (D_0, D_1, D_2, \cdots, D_{(n-1)})$ be the path to p_t from p_s (D_i means $Out(p_i)$). We define four rules of the local update on each p_i from some i as follows.

1) **Shortcut1** : If $D_{(i-1)} \star D_i = -1$, $D_{(i-1)}$ and D_i are removed from P.
2) **Shortcut2** : If $D_{(i-1)} \star D_i = 0$ and $D_{(i-1)} \star D_{(i+1)} = -1$, $D_{(i-1)}$ and $D_{(i+1)}$ are removed from P.
3) **ZigZag** : If $D_{(i-1)} \star D_i = 1$ and $D_i \star D_{(i+1)} = 0$, D_i and $D_{(i+1)}$ are exchanged their sequence in P.
4) **V-Shift** : If $D_{(i-1)} \star D_i = 2$, $D_{(i-1)}$ and D_i are exchanged their sequence in P.

Figure 7 shows the examples of four local update rules of our proposed algorithm. Note that *Shortcut2* and *ZigZag* are updated on a plane surface only. Each router p_i always checks the relation between $p_{(i-1)}$'s output direction ,which is the conter direction of p_i's input direction, and its output direction (i.e. $D_{(i-1)} \star D_i$). Figure 8 represents the flow chart of checking local update rules of p_i.

In this paper, we abbreviate how each local update is implemented. We consider each local update can be executed pseudo-atomically. To help to design atomicity, our algorithm ensures that an edge is never targeted by two or more local updates. This implies that each local update is spared the intrusion of the other local updates. Details will be mentioned in the next subsection.

B. Collision Handling of Local Updates

Some edges may be targeted for two local updates at the same time. Figure 9 shows an example of some local update rules' collision.

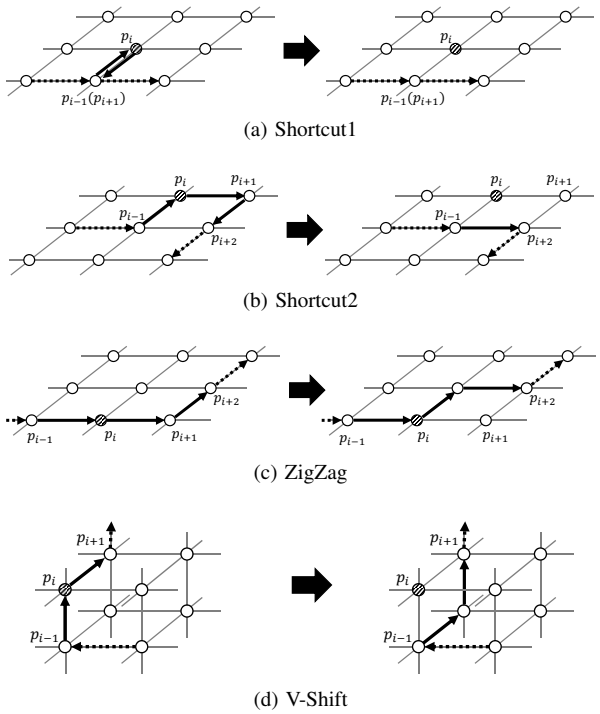


Fig. 7: Four rules of our proposed algorithm

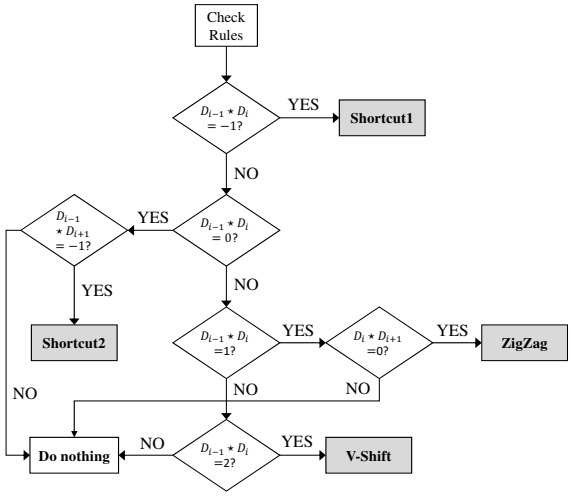


Fig. 8: Flowchart of checking local update rules

In Figure 9, p_i detects the local update *V-Shift*, $p_{(i+1)}$ detects the local update *Shortcut1*, and $p_{(i+2)}$ detects the local update *Shortcut2* at the same time. In this case, some edges may be removed or moved (exchanging its sequence) by two local updates at the same time. For example, D_i is moved by p_i's local update (*V-Shift*) and removed by $p_{(i+1)}$'s local update (*Shortcut1*) simultaneously. On the other hand, $D_{(i+1)}$ is removed by $p_{(i+1)}$'s local update (*Shortcut1*) and removed by $p_{(i+2)}$'s local update (*Shortcut2*) simultaneously.

These collision (confliction), which means the situation of which some edges are targeted by two or more local updates,

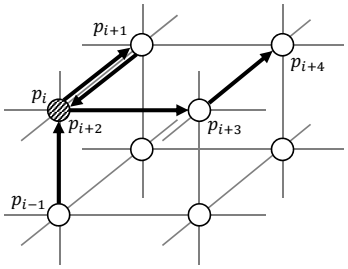
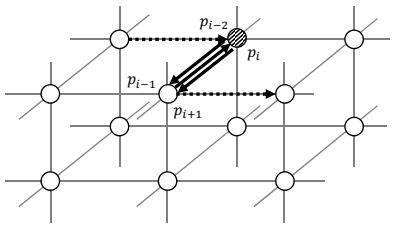
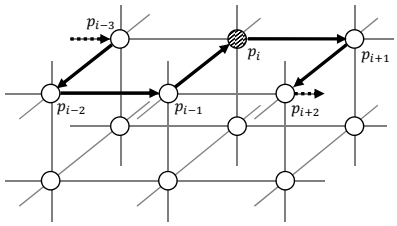


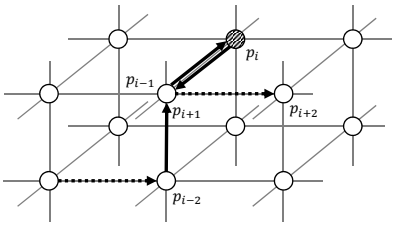
Fig. 9: Some local update rules' collision



(a) Shortcut1 precedes Shortcut1



(b) Shortcut2 precedes Shortcut2



(c) V-Shift precedes Shortcut1

Fig. 10: Examples of confliction between two local updates

may cause the disconnection of the routing path to p_t from p_s. Therefore, some exclusion rules are required to avoid disconnection of the path. A router p_i never detects two or more local update rules at the same time because the definition of local update rules. As presented in Figure 8, a router p_i detects one local update rule or not (do nothing) at some instant. Hence we only consider two different routers p_i and p_j trying to remove (or move) the same edge. To realize the mutual exclusion of local update rules, we set one simple rule: When a router p_i detects a local update rules, p_i checks some other update rules including $D_{(i-1)}$. If p_i detects some other local update rules including D_i, p_i ignores its local update rule detected. Figure 10 shows some examples of the collision (confliction) between two local update rules. In Figure 10(a),

Algorithm 1 Pseudocodes for p_i: Local-information-based routing algorithm in a 3D virtual grid network

Require: Subset of $P^{p_i} = (D_{(i-3)}, D_{(i-2)}, \cdots, D_{(i+2)})$
Ensure: Local update type of p_i

```
procedure CheckLocal(p_i)
    var update ← ⊥
    if D_(i-1) ⋆ D_i = −1 then
        update ← Shortcut1
    else if D_(i-1) ⋆ D_i = 0 & D_(i-1) ⋆ D_(i+1) = −1 then
        update ← Shortcut2
    else if D_(i-1) ⋆ D_i = 1 & D_i ⋆ D_(i+1) = 0 then
        update ← ZigZag
    else D_(i-1) ⋆ D_i = 2
        update ← V − Shift
    end if
    if CheckLocal(p_(i-1)) ≠ ⊥ then
        update ← ⊥
    else if CheckLocal(p_(i-2)) = Shortcut2
          | CheckLocal(p_(i-2)) = ZigZag then
        update ← ⊥
    end if
    if update ≠ ⊥ then
        Execute Local Update
    end if
end procedure
```

p_i detects a local update *Shortcut1* because of $D_{(i-1)} = U$ and $D_i = D$. However, p_i knows $D_{(i-2)} = D$ and it finds that $p_{(i-1)}$ detects a local update *Shortcut1*. Thus, p_i does not execute its local update. In a similar fashion, p_i in Figure 10(b) and p_i in Figure 10(c) do not execute their local update neither.

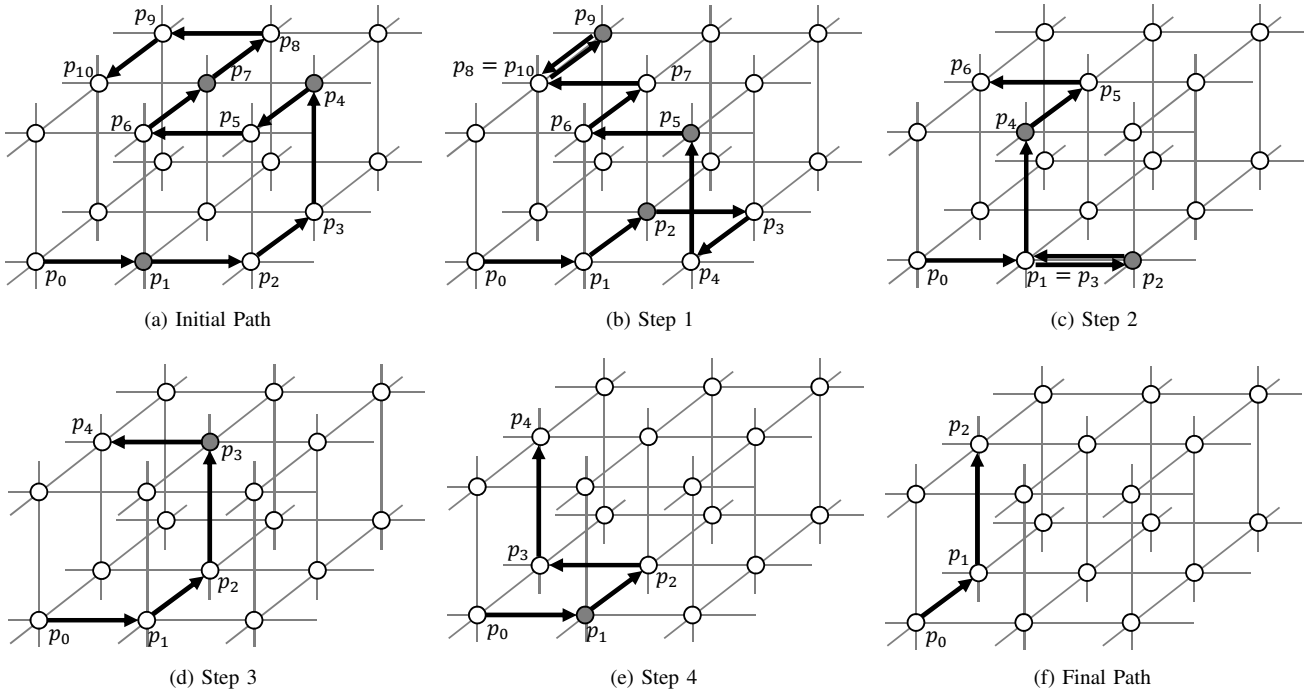
Algorithm 1 represents the pseudocodes of our proposed algorithm with a confliction handling.

Figure 11 show an example of the execution of our proposed algorithm. Our algorithm can be operated asynchronously, but this example shows the synchronous execution (each router which detects a local update rule executes it at the same time) in order to help to understand.

The initial given path $P = (R, R, U, T, D, L, U, U, L, D)$ is illustrated in Figure 11(a). And three gray routers in Figure 11(a) detect local update rules by Algorithm 1. p_1, p_4 and p_7 detect *ZigZag*, *V-Shift* and *ZigZag*, respectively. Note that p_5 ignores its local update *Shortcut2* because p_5 knows p_4 will execute the local update *V-Shift* from its local-information. Likewise, p_8 also ignores its local update.

Figure 11(b) shows the updated path after the executions of three gray routers. And three new gray routers in Figure 11(b), p_2, p_5 and p_9, execute their local updates.

Finally, our algorithm constructs the shortest path like Figure 11(f) and is terminated. This implies that there is no more local update in this path.



(a) Initial Path (b) Step 1 (c) Step 2

(d) Step 3 (e) Step 4 (f) Final Path

Fig. 11: Examples of executions of our algorithm

C. Correctness of Proposed Algorithm

In this section, we discuss the correctness of our proposed algorithm briefly. We introduce some definitions and we simply give an explanation that our proposed algorithm can construct a shortest path from p_s to p_t.

As we mentioned in the previous section, a routing path P can be represented by the sequence of each routers output link's direction like $P = (D_0, D_1, \cdots, D_{(n-1)})$ where $D_i = Out(p_i)$. Let P be a path from p_s to p_t and assume two (relaying) routers p_a and p_b in P ($0 \leq a \leq b \leq n$). $\overline{P}$ can be divided into 3 parts, the path from $p_0 (= p_s)$ to p_a (meaning $(D_0, D_1, \cdots, D_{(a-1)})$), from p_a to p_b, and from p_b to $p_n (= p_t)$. We notate these parts P^{sa}, P^{ab}, and P^{bt} respectively. Certainly, P^{sa}, P^{ab}, or P^{bt} can be an empty sequence.

Now we define the converged path $\overline{P}$ which is constructed by our proposed algorithm.

Definition 1. Converged Path $\overline{P}$ of our algorithm. *$\overline{P}$ is the converged path if and only if:*

1) *P^{sa} is alternating with two plane directions D_i and D_j, where $D_i \star D_j = 0$.*
2) *P^{ab} consists of all the same plane direction (a horizontal line) which appears in P^{sa}.*
3) *P^{bt} consists of all the same vertical direction (a vertical line).*

Figure 12 shows an example of $\overline{P}$. p_b becomes the orthogonal projection onto the same plane with p_s due to its definition.

Lemma 1. *No local update will be executed in converged path $\overline{P}$.*

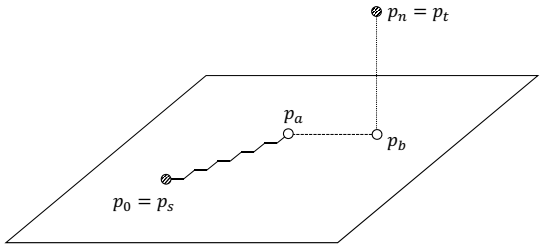


Fig. 12: An example of converged path

Proof of Lemma 1. At first, at any i ($1 \leq i < a$), all $D_{(i-1)} \star D_i$ becomes 0 in P^{sa}. However, $D_{(i-1)} \star D_{(i+1)}$ will never be -1 in P^{sa}. Hence there is no local update in P^{sa} of $\overline{P}$.

Secondly, at any i ($(a+1) \leq i < b$), all $D_{(i-1)} \star D_i$ becomes 1 in P^{sa} because P^{ab} consists of only one direction. Thus there is no local update in P^{ab} of $\overline{P}$.

Finally, there is no local update in P^{bt} of $\overline{P}$ due to the same reason of P^{ab}. □

To prove that our algorithm constructs a converged path, firstly we present the following Lemmas 2 and 3.

Lemma 2. *If there is one or more vertical directions which precede any plane directions, one or more local updates will be executed.*

Proof of Lemma 2. Assume a *vertical* direction D_i is included in P, which is the last *vertical* direction preceding some *plane* directions. This implies $D_{(i+1)}$ is a *plane* direction. Because of Algorithm 1 (line 9–10), a local update *V-Shift* is detected

by $p_{(i+1)}$. Even though there is some local updates preceding $p_{(i+1)}$, *V-Shift* is eventually executed after the other local updates are terminated. □

Lemma 3. *If both T and B are included in P, one or more local updates will be executed.*

Proof of Lemma 3. From Lemma 2, no *vertical* directions preceding any *plane* directions. We consider that D_i is the last *plane* direction in P, thus, $D_{(i+1)}$ becomes the first *vertical* direction in P. From the presumption of Lemma 3, there is D_j in P, where $(i+1) < j \leq n$ and $D_{(i+1)} \neq D_j$. In this case, p_j detects a local update *Shortcut1* because of $D_{(j-1)} \star D_j = -1$.

Therefore, if there is no more local update's execution, either T or B, but not both, is included in P. (or neither of them). □

From Lemmas 2 and 3, we have the following corollary.

Corollary 1. *If no more local update is executed in P, the subsequence $P^{bt} = (p_b, \cdots, p_n) = (D_b, \cdots, D_{(n-1)})$ (where $b \leq n$), which consists of either T or B, can be determined in P (D_b is the first vertical direction in P).*

From Corollary 1, p_b can be obtained by the orthogonal projection of p_n onto the same plane with p_s (refer to Figure 11).

To make it easier to discuss the convergence of our algorithm, now we only consider the subsequence of P, $P^{sb} = (p_s(= p_0), \cdots, p_b) = (D_0, \cdots, D_{(b-1)})$ which consists of only *plane* directions.

Lemma 4. *If 3 or more kinds of directions are included in P^{sb}, one or more local updates will be executed.*

Proof of Lemma 4. Assume D_i is the first (*plane*) direction with two kinds of different preceding directions in P^{sb}. We show that p_i detects a local updates and eventually executes it.

As the assumption, $D_{(i-1)}$ has the different directions with D_i. If $D_{(i-1)} \star D_i = -1$, p_i detects a local update $Shortcut1$. Thus, we consider $D_{(i-1)} \star D_i = 0$.

In this case, $D_{(i-2)}$ has also different direction with D_i. If $D_{(i-2)}$ is the same direction as $D_{(i-1)}$, a local update *ZigZag* is detected by p_i. If $D_{(i-2)}$ is the opposite direction as D_i, p_i detects a local update *Shortcut2*. If $D_{(i-2)}$ is the opposite direction as $D_{(i-1)}$, a local update *Shortcut1* is detected by $p_{(i-1)}$.

Therefore, if two kinds of different *plane* directions precede D_i, $p_{(i-1)}$ or p_i detects a local updates thus Lemma 4 hold. □

From Lemma 4, if two kinds of directions are in P^{sb} and no local update is detected, the directions are orthogonal. Because if not, a local update $Shortcut1$ is detected.

Before the introduction of the next lemma, we define a segment representation of P as follows.

Definition 2. Segment representation of P. *A segment representation of P is expressed by the subsequence of P. Each subsequence S_i of P consists of the consecutive same directions. $SR(P)$ is a segment representation of P, and becomes $SR(P) = (S_1, S_2, \cdots, S_m)$ ($1 \leq m \leq (n-1)$).*

A segment representation combines the consecutive same directions in P into a subsequence of P named S_i. For example, if $P = (U, U, D, L, T, T, T, R)$, the segment representation of P becomes $SR(P) = ((U, U), (D), (L), (T, T, T), (R))$. In this example, there are five segments in P ($m = 5$), and lengths of segments are $|S_1| = 2$, $|S_2| = |S_3| = |S_5| = 1$, and $|S_4| = 3$.

Lemma 5. *When a segment representation of $P^{sb} = (S_0, S_1, \cdots, S_m)$, unless $|S_i| = 1$ at any i ($0 \leq i < m$), one or more local updates will be executed.*

Proof of Lemma 5. Lemma 5 implies that only S_m (the last segment of P^{sb}) has a length of two or more ($|S_m| \geq 2$).

Assume $|S_i| > 1$ ($0 \leq i < m$) in P^{sb} and no more update is detected in P^{sb}. There is $S_{(i+1)}$, and the last direction of S_i and the first direction $S_{(i+1)}$ become orthogonal (from Lemma 4). In this case, because $|S_i| > 1$, the last two directions of S_i have the same directions. This causes the first router to detect a local update $ZigZag$ and this is a contradiction. □

From Lemmas 4 and 5, we obtain the following corollary.

Corollary 2. *If no more local update is executed in P, which consists of only plane directions, P includes only two directions which are orthogonal. Moreover, $|S_i| = 1$ at any i ($0 \leq i < m$) holds in the segment representations of $P = (S_0, S_1, \cdots, S_m)$.*

From Corollaries 1 and 2, unless P is a converged path, our algorithm retains to detect local updates and transforms P. And if P is a converged path, our algorithm stops (is terminated). Now we show our algorithm eventually makes P a converged path, this implies that the infinite executions of our algorithm never occurs. To prove this, we introduce the following definitions using a segment representation.

Definition 3. Potential function f. *Let P be a path and its segment representation $SR(P)$ is $(S_1, S_2, \cdots, S_m)$. The potential function f assigns a sequence $f(P) = (|S_{total}|, |S_1|, |S_2|, \cdots, |S_m|)$, where $|S_{total}| = \sum_{i=1}^{m} |S_i|$.*

Definition 4. Weighted Potential function f^w. *Let $SR(P) = (S_1, S_2, \cdots, S_m)$ be a segment representation of P. The weighted potential function f^w assigns a sequence $f^w(P)$ which is obtained by changing $f(P)$ with the follow rules: If a segment S_i consists of vertical directions (T or B) in $SR(P)$, a value α (defined later) is added to the length of S_i ($|S_i|$) in $f^w(P)$.*

To help to understand $f(P)$ and $f^w(P)$, we present the examples of these functions. In the above case $P = (U, U, D, L, T, T, T, R)$, $f(P)$ becomes $(|S_{total}|, |(U, U)|, |(D)|, |(L)|, |(T, T, T)|, |(R)|) = (8, 2, 1, 1, 3, 1)$. Note that the first element of $f(P)$ is equal to

the length of the path P. However, the weighted potential function $f^w(P)$ becomes $((8+\alpha), 2, 1, 1, (3+\alpha), 1)$, because S_4 of $SR(P)$ consists of *vertical* directions T.

We consider that the sequences constructed by potential function are totally ordered by the lexicographic order. In the case of the weighted potential function, we assume that α is large enough (e.g. $\alpha > |P|$). And we call the sequence, constructed by (weighted) potential function, (weighted) potential value.

Now we show that the weighted potential value becomes the minimum value when P is the converged path. And our algorithm eventually decreases the weighted potential value of the given P.

Lemma 6. *If P is a converged path, $f^w(P)$ has the minimum value.*

Proof of Lemma 6. Obviously, the minimum weighted potential value is $(|P^{min}|, 1, 1, \cdots, m)$, where $|P^{min}|$ is the length of the shortest path from p_s to p_t and $m \geq 1$ when there is no *vertical* directions in P.

In the weighted potential function, each *vertical* segment is weighted by α, which is a large value, thus it is important to reduce the number of the *vertical* segments for constructing the minimum weighted potential value, that is, there is only one *vertical* segment in $SR(P)$ when the minimum $f^w(P)$. More specifically, the *vertical* segment is positioned as the last segment of $SR(P)$.

As a result, the minimum weighted potential value becomes $(|P^{min}| + \alpha, 1, 1, \cdots, 1, m, v + \alpha)$, *vertical* directions are included the last segment only. And this becomes a converged path. □

Lemma 7. *The weighted potential value is eventually decreased by our algorithm.*

Proof of Lemma 7. Local updates *Shortcut1* and *Shortcut2* decrease the length of the given P, thus, the weighted potential value is decreased because the first element of $f^w(P)$ decreases.

Secondly we consider the local update *ZigZag*. If *ZigZag* is executed by the router p_i, D_i (output direction of p_i) is included in S_i of $SR(P)$. In this case, $|S_i| \geq 2$ due to $D_i \star D_{(i-1)} = 1$, and $|S_{(i+1)}| \geq 1$. After executing *ZigZag* on p_i, S_i is decreased by 1. Hence, $f^w(P)$ decreases.

However, local update *V-Shift* sometimes increases $f^w(P)$. For example, if $|S_i| = l$, $|S_{(i+1)}| = 1 + \alpha$ (*vertical* direction), and $|S_{(i+2)}| = 1$ of $SR(P)$ (refer to Figure 13(a)), local update *V-Shift* transforms it to $|S_i| = l+1$ and $|S_{(i+1)}| = 1+\alpha$ (refer to Figure 13(b)). This causes the increasing of $f^w(P)$. Though local update *V-Shift* moves each *vertical* direction to backward of the sequence $SR(P)$, and this shift is required for constructing the converged path to minimize the weighted potential value (Lemma 6). There is no local updates moving *vertical* directions to forward of the sequence $SR(P)$, thus, local update *V-Shift* is executed finite number of times (no infinite execution). □

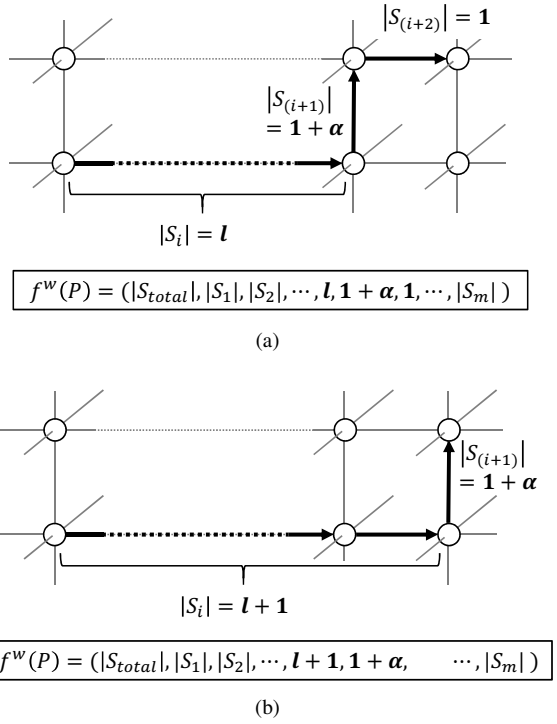


Fig. 13: Example of increasing the weighted potential value

Our algorithm continues to transform P, which is not converged path, using local update rules, and it convergs after the construction of the converged path $\overline{P}$ within finite executions of local updates (no livelock is guaranteed by Lemma 7).

We obtain the following theorem using above Lemmas and Corollaries.

Theorem 1. *If P is not a converged path, one or more local updates will be executed in P. And our algorithm constructs the converged path from P within finite number of executions.*

Finally, we show the converged path $\overline{P}$ is the shortest path from p_s to p_t using the following Lemma.

Lemma 8. *A converged path $\overline{P}$ ensures the shortest path from p_s to p_t.*

Proof of Lemma 8. We can determine p_b which has the first *vertical* directions as D_b in $\overline{P}$. p_b becomes the orthogonal projection onto the same plane with p_0, $P^{bt} = (p_b, p_{(b+1)}, \cdots, p_t)$ becomes the shortest path from p_t to the plane.

From Lemma 4, $P^{sb} = (p_s(= p_0), p_1, \cdots, p_b)$ consists of two directions which are orthogonal, hence there is no redundancy path in P^{sb}. This also implies the shortest path from p_s to p_b.

Therefore a converged path $\overline{P}$ guarantees the shortest path. □

V. Conclusion

In this paper, we proposed a self-optimizing routing algorithm which construct the shortest path in a 3D virtual grid network. Our algorithm uses each router's local information only, and has only four local update rules. We prove that our algorithm eventually constructs the shortest path from the *source* node to the *destination* node. We are considering analyzing the time complexity to converge as future work.

To guarantee the correct local updates, we assume that each router maintains 6 hops of routing information, because some confliction can occur among local updates. We expect that this routing problem in a 3D virtual grid network can be resolved with less routing information (e.g. 4 hops for each router).

Moreover, to make our algorithm a practical protocol, the design of each local update as the distributed manner is required.

Acknowledgement

This work was partly supported by JSPS KAKENHI Grant Number 15K00011.

References

[1] S. Takatsu, F. Ooshita, H. Kakugawa, and T. Masuzawa, "Zigzag: Local-information-based self-optimizing routing in virtual grid networks," Proceedings of the 33rd International Conference on Distributed Computing Systems (ICDCS) , pp. 358-368, July, 2013.

[2] Chai-Keong Toh, "Ad Hoc Mobile Wireless Networks: Protocols and Systems 1st Edition," Prentice Hall PTR, 2002.

[3] C. de Morais Cordeiro and D. P. Agrawal, "Ad Hoc and Sensor Networks: Theory and Applications (2nd Edition)," World Scientific, 2011.

[4] J. Zheng and A. Jamalipour, "Wireless Sensor Networks : A Networking Perspective," Wiley-IEEE Press, 2009.

[5] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01), pp.70–84, 2001.

[6] W. Dargie and C. Poellabauer, "Fundamentals of Wireless Sensor Networks: Theory and Practice," John Wiley & Sons, Ltd, 2010.

[7] C. Perkins and E. Royer, "Ad hoc on demand distance vector routing," in Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99), 1999, pp. 90100.

[8] J. N. Al-Karaki, A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," IEEE Wireless Communications, Vol. 11, Issue 6, pp.6–28, 2004.

[9] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00), 2000.

[10] D. Braginsky and D. Estrin, "Rumor Routing Algorithm for Sensor Networks,"in the Proceedings of the First Workshop on Sensor Networks and Applications (WSNA), 2002.

[11] F. Ye, H. Luo, J. Cheng, S. Lu, L. Zhang, "A Two-tier data dissemination model for large-scale wireless sensor networks", Proceedings of ACM/IEEE MOBICOM, 2002.

極大距離 k–独立集合問題に対する緩安定個体群プロトコル

清洲星顕　首藤裕一　角川裕次　増澤利光

大阪大学 大学院情報科学研究科 コンピュータサイエンス専攻

概要

個体群プロトコルモデルは，資源の非常に制約された小型デバイスで構成されるモバイルネットワークの抽象モデルである．デバイスが「個体」に通信が「交流」に対応し，個体間の距離が十分近づいたときに交流が行われる．自己安定とは，システムが任意の初期状況から実行を開始してもやがて所望の性質を満たす (収束性) ようになり，かつそれ以降，その性質を永遠に維持すること (閉包性) を保証する概念である．自己安定の閉包性を緩めた緩安定は，短時間（具体的には多項式の交流回数）で所望の状況に収束し，十分に長い間（具体的には指数の交流回数の間），その状況を維持することを保証する．同モデルにおいて，任意のグラフに対しリーダ選挙問題と極大独立点集合問題を解く緩安定プロトコルが提案されている．本稿では，極大な距離 k–独立集合を求める緩安定プロトコルを提案する．距離 k-独立集合とは，異なるノード間の距離が k 以下とならないようなノードの集合である．極大距離 k–独立集合問題はリーダ選挙問題と極大独立点集合問題の一般化である．

1　はじめに

個体群プロトコルモデル (population protocol)[1] とは，資源の非常に制約された小型デバイスで構成されるモバイルネットワークの抽象モデルである．デバイスが「**個体**」に、それらのネットワークが「**個体群**」に，通信は「**交流**」に対応する．個体間の交流は，互いの物理的距離が十分に近づいたときのみに発生する．例えば，通信範囲の極めて狭いデバイスを鳥の群れの一羽一羽に付けることで構成されるモバイルネットワークは，個体群プロトコルモデルで表現される．個体群プロトコルモデルは，ノードが個体，辺が交流可能な個体間を表すグラフで表現できる．確率的個体群プロトコルモデルは，個体群プロトコルモデルに確率的要素を導入したモデルである．このモデルにおいては，交流し得る全ての個体ペアから，交流するペアが等確率で選ばれる．

緩安定 [4] とは，自己安定 [2] と同様に，分散システムに耐故障性を持たせる技法のひとつである．自己安定システムは，(i) 任意の初期状況から実行を開始しても，やがて正当な状況と呼ばれる状況に達し（収束性），(ii) 一度正当な状況に達すると，それ以降，システムは永遠に所望の性質を満たし続ける (閉包性)．いくつかの問題に対しては，自己安定性の実現が不可能であることが知られており，緩安定はそれを克服するために提案された概念のひとつである．緩安定システムは，(i) 任意の初期状況から実行を開始しても，システムは比較的短時間（例えば，個体数に対して多項式時間）のうちに安全状況に達し（収束性），(ii) それ以降，システムは非常に長い間所望の性質を満たし続ける（緩閉包性）．緩安定プロトコルは，要件 (ii) における，システムが所望の性質を維持する時間が十分に長いのであれば（例えば，個体数に対して指数時間であるなど），応用上，自己安定プロトコルと同様の有用性を持つとみなすことができる．

極大距離 k–独立集合 (k-MIS) 問題とは，集合に属する異なる個体間の距離が k 以下とならないような，個体の極大集合を求める問題である．極大距離 k–独立集合問題は，リーダ選挙問題 ($(n-1)$–MIS)[4, 5, 6, 3] と極大独立点集合 (1–MIS) の一般化である．個体群プロトコルモデルにおいて，個体数 n が既知でない限り，リーダ選挙問題を解くことができないことが知られており [2]，同様の理由により，個体群プロトコル上で k-MIS 問題を自己安定的に解くことは不可能である．

そこで本稿では，個体数 n の上限値 N が与えられた上で，任意のグラフに対して k-MIS 問題を解く確率的緩安定個体群プロトコルを提案する．

2　諸定義

個体群は，単純かつ弱連結有向グラフ $G(V, E)$ で表現する．$V(|V| \geq 2)$ は個体の集合，$E(\subseteq V \times V)$ は有向辺の集合を表す．各辺は，発生し得る個体間の関係を表す．$(u, v) \in E$ のとき，個体 u，v はそれぞれ呼びかけ側，応答側として交流し得る．任意の二つの個体 $u, v \in V$ において，$(u, v) \in E \Leftrightarrow (v, u) \in E$ を満たす場合，$G(V, E)$ を無向グラフと呼ぶ．本稿では，単純無向グラフのみを考え，$n = |V|$，$m = |E|$ と定義する．

個体群プロトコルは，$P(Q, Y, I, T, O)$ と表す．Q は状態の有限集合，Y は出力シンボルの有限集合を表し，I は $I : V \to \mathcal{I}$ ($\mathcal{I}$ は個体が取りうる識別子の集合) となる関数を表す．各個体 v は固有の識別子 $I(v) \in \mathcal{I}$ を持つ．I の要素のうち，ある個体の識別子に一致するものを真の識別子，そうでないものを偽の識別子と呼ぶ．二つの個体間のあいだで交流が発生するとき，遷移関数 $T : (Q \times \mathcal{I}) \times (Q \times \mathcal{I}) \to Q \times Q$ は二つの個体の状態と識別子に基づいて次の状態を決定する．出力関数 $O : Q \times \mathcal{I} \to Y$ は各個体の出力をその状態と識別子をもとに定める．すなわち，状態 s の個体 v の出力は $O(s, v.\mathrm{id})$ となる．

状況は，各個体の状態を特定する写像 $C : V \to Q$ である．今後，プロトコル P における状況の集合を $\mathcal{C}_{\mathrm{all}}(P)$ とする．$(C'(u), C'(v)) = T(C(u), u.\mathrm{id}, C(v), v.\mathrm{id})$，かつ任意の個体 $w \in V \setminus \{u, v\}$ について $C'(w) = C(w)$ が成り立つとき，状況 C は交流 $e = (u, v)$ によって状況 C' に遷移するといい，

$C \xrightarrow{e} C'$ と表す．**スケジューラ**は，各時刻に発生する交流を決定する．本稿では，一様ランダムなスケジューラ $\Gamma = \Gamma_0, \Gamma_1, \ldots$ を仮定する．各 $\Gamma_t \in E$ は，時刻 t で発生する交流を表現する確率変数であり，各 $(u,v) \in E$ に対し $\Pr(\Gamma_t = (u,v)) = 1/m$ である．

初期状況 C_0，スケジューラ Γ が与えられたとき，プロトコル P の実行 $\Xi_P(C_0,\Gamma) = C_0, C_1, \ldots (s.t. \forall t \geq 0, C_t \xrightarrow{\Gamma_t} C_{t+1})$ は一意に決まる．今後，誤解がない場合は $\Xi_P(C_0,\Gamma)$ を単に $\Xi_P(C_0)$ と表す．

以下の二つの条件を満たす個体の集合 $S \subseteq V$ を**極大距離 k–独立集合（k-MIS）**と呼び，S の要素を**独立個体**，それ以外を**従属個体**と呼ぶ．

(i)S に属する全ての個体において，距離 k 以内に他の個体が存在しない．

(ii)S は (i) を満たす極大集合である．つまり，S に属さない任意の個体に対し，S に属する個体 g 距離 k 以内に存在する．

k-MIS 問題では，各個体は IA または DA を出力する．出力が IA，DA となる個体それぞれが独立個体，従属個体を表す．状況 C から一意に定まる個体の集合 $\{v \in V \mid O(C)(V) = \mathrm{IA}\}$ が k-MIS であるとき，状況 C が k-MIS を満たすという．以降，k-MIS を満たす状況の集合を $\mathcal{C}_{\mathrm{kMIS}}$ と表す．

$\mathrm{EHT}_P(C, \mathcal{C}_{\mathrm{kMIS}})$ をプロトコル P が状況 C からスケジューラ $\Gamma = \Gamma_0, \Gamma_1, ...$ のもとで実行を開始した後に，個体群が k-MIS を満たす状況を維持する期待交流回数（期待維持時間）とする．また，ある状況の集合 $\mathcal{C}$ に対し，$\mathrm{ECT}_P(C, \mathcal{C})$ をプロトコル P が初期状況 C からスケジューラ $\Gamma = \Gamma_0, \Gamma_1, ...$ のもとで実行を開始した後に，個体群が状況 $\mathcal{C}$ に到達するまでにかかる期待交流回数（期待収束時間）と定義する．

ここで，緩安定 k-MIS プロトコルについて定義する．直感的に，緩安定プロトコルは少ない交流回数で所望の（問題の仕様を満たす）状況に収束し，その後は非常に長い間，所望の状況を維持する．緩安定 k-MIS プロトコルを以下のように定義する．

定義 2.1.（緩安定 k-MIS プロトコル）
$\max_{C \in \mathcal{C}_{\mathrm{all}}(P)} \mathrm{ECT}_{P_{kMIS}}(C, \mathcal{S}) \leq \alpha$ かつ $\min_{C \in \mathcal{S}} \mathrm{EHT}_{P_{kMIS}}(C, \mathcal{C}_{\mathrm{kMIS}}) \geq \beta$ を満たすような状況の集合 $\mathcal{S}$ が存在するとき，個体群プロトコル $P(Q, Y, I, T, O)$ は (α, β)–緩安定 k-MIS プロトコルである．

3 プロトコル P_{kMIS}

本節では，個体数 n の上限値 N が与えられたときに，任意の無向グラフで k-MIS 問題を解く緩安定個体群プロトコル $P_{kMIS}(Q, \{\mathrm{IA}, \mathrm{DA}\}, I, T, O)$ を提案する．本プロトコルでは，各個体は変数 ind，hop，timer を持つ．個体 v の変数 var は v.var と表す．変数 ind は，距離 k 以内の独立個体の id を格納するための整数型の変数であり，変数 hop はその距離を格納するための整数型の変数である（独立個体では ind = id，hop = 0）．変数 timer は，変数 ind に記憶された識別子を持つ個体が距離 k 以内に存在しないことを検知するためのカウントダウンタイマであり，$[0, t_{\max}]$ の範囲で値をとる（$t_{\max}$ については後述）．各個体 v は，v.ind = v.id のとき IA，そうでないとき DA と出力する．

以下，P_{kMIS} (**Protocol 1**) の動きについて，説明する．本プロトコルの基本方針は，以下の二つである．

(i) 各個体は距離 k 以内の独立個体を記憶する

(ii) 距離 k 以内に独立個体が存在しない，もしくは
偽の識別子を ind に持つような従属個体は独立個体になる

(i) について，各個体は最も近い独立個体の識別子とその距離をそれぞれ変数 ind，hop に記憶する．ただし，最も近い独立個体が複数存在する場合は，その中で識別子が最小の独立個体の識別子を記憶する．各個体は交流ごとに互いの ind，hop を比較し，上記のルールによって記憶する値を決定する（9–10 行目）．これらにより，交流を繰り返すことで，各個体は最も近い独立個体の識別子とその距離を記憶することが可能となり，距離 k 以内に独立個体が存在する個体は自身から距離 k 以内の独立個体を記憶することが可能となる．また，ind が異なり双方の hop の和が k 未満となる個体間で交流が発生した場合，各個体は識別子が小さい方を優先して ind，hop を更新する（7–8 行目）．これにより，やがて異なる独立個体が距離 k 以内に隣接することがない状況に収束する．(ii) については，変数 timer を用いて実現する．各個体 v のタイマは，後述のタイマリセットと高値伝播，カウントダウンの仕組みにより，距離 k 以内に存在する独立個体の識別子が v.ind に記憶されていない限り減少し続け，やがて 0 になる．タイマの値が 0 になると，個体 v は，v.ind に対応する個体が距離 k 以内に存在しない，あるいは独立個体ではないと判断して自信が独立個体となる（タイムアウト，20–22 行目）．独立個体が交流を行う場合には，自身と交流相手のタイマの値を最大値 $t_{\max}$ に設定し（タイマリセット，11-15 行目），また，交流相手と自身の ind が等しく，かつ，交流相手の hop が自より 1 小さい場合に限り，交流相手のタイマが自信のそれより大きいのであれば，その値を自身のタイマにコピーする（高値伝搬，16–17 行目）．その後，両個体の状態に関わらず，両個体のタイマは 1 減じられる（カウントダウン，18–19 行目）．カウントダウンにより，各個体 v は，距離 k 以内に v.ind が存在しない状況が長期間続くと，タイムアウトが発生して独立個体となる．高値伝搬及びタイマリセットにより，v.ind が独立個体として距離 k 以内に存在する個体 v においては，滅多にタイムアウトが発生しない．

(i)，(ii) の実現により，$t_{\max}$ を適切に設定すれば，個体群は n に関する多項式回数の交流で k-MIS を構成し，n に関する指数回の交流の間，その k-MIS を維持する．

Protocol 1 P_{kMIS}

Variables of each agent:

$\mathrm{id} \in \mathcal{I},\ \mathrm{ind} \in \mathcal{I},\ \mathrm{hop} \in [0,k], \mathrm{timer} \in [0, t_{\max}]$

Output function O:

if $u.\mathrm{id} = u.\mathrm{ind}$ then $O(u) = \mathrm{IA}$; Otherwise, $O(u) = \mathrm{DA}$;

Interaction between initiator u_0 and responder u_1:

/* 異常検知・修正処理 */

1: **for** $i \in \{0,1\}$ **do**
2: **if** $u_i.\mathrm{id} = u_i.\mathrm{ind}$ **then**
3: $u_i.\mathrm{hop} \leftarrow 0$;
4: **if** $u_i.\mathrm{hop} = 0$ **then**
5: $u_i.\mathrm{ind} \leftarrow u_i.\mathrm{id}$;

/* ind，hop 更新処理 */

6: **for** $i \in \{0,1\}$ **do**
7: **if** $(u_0.\mathrm{hop} + u_1.\mathrm{hop} < k) \wedge (u_i.\mathrm{ind} > u_{1-i}.\mathrm{ind})$ **then**
8: $(u_i.\mathrm{ind}, u_i.\mathrm{hop}) \leftarrow (u_{1-i}.\mathrm{ind}, u_{1-i}.\mathrm{hop}+1)$;
9: **else if** $(u_i.\mathrm{hop} > u_{1-i}.\mathrm{hop}+1) \vee (u_i.\mathrm{hop} = u_{1-i}.\mathrm{hop}+1 \wedge u_i.\mathrm{ind} > u_{1-i}.\mathrm{ind})$ **then**
10: $(u_i.\mathrm{ind}, u_i.\mathrm{hop}) \leftarrow (u_{1-i}.\mathrm{ind}, u_{1-i}.\mathrm{hop}+1)$;

/* タイマリセット処理 */

11: **for** $i \in \{0,1\}$ **do**
12: **if** $u_i.\mathrm{ind} = u_i.\mathrm{id}$ **then**
13: $u_i.\mathrm{timer} \leftarrow t_{\max}$;
14: **if** $u_i.\mathrm{ind} = u_{1-i}.\mathrm{ind} = u_{1-i}.\mathrm{id}$ **then**
15: $u_i.\mathrm{timer} \leftarrow t_{\max}$;

/* 高値伝搬処理 */

16: **if** $\exists i, (u_i.\mathrm{ind} = u_{1-i}.\mathrm{ind}) \wedge (u_i.\mathrm{hop} = u_{1-i}.\mathrm{hop}+1) \wedge (u_i.\mathrm{timer} < u_{1-i}.\mathrm{timer})$ **then**
17: $u_i.\mathrm{timer} \leftarrow u_{1-i}.\mathrm{timer}$

/* カウントダウン処理 */

18: **for** $i \in \{0,1\}$ **do**
19: $u_i.\mathrm{timer} \leftarrow u_i.\mathrm{timer} - 1$;

/* タイムアウト処理 */

20: **for** $i \in \{0,1\}$ **do**
21: **if** $u_i.\mathrm{timer} = 0$ **then**
22: $(u_i.\mathrm{ind},\ u_i.\mathrm{hop}) \leftarrow (u_i.\mathrm{id},\ 0)$;
23: $u_i.\mathrm{timer} \leftarrow t_{\max}$

4 評価

本プロトコルは，首藤らによって提案された収束時間を改善する手法，the same speed timer[3] を適用することができる．

タイマの最大値 $t_{\max}$ を個体数の上階 N にのみに依存して適切に設定すると，the same speed timer を導入した場合，本プロトコルは期待収束時間が $O(kmN \log N)$ であることを予想している．一方，期待維持時間が $\Omega(Ne^N)$ であることはすでに証明できている．

5 終わりに

本稿では，極大距離 k–独立集合問題に対する緩安定個体群プロトコルを提案した．現在，期待収束時間の解析に取組んでいるところである．

参考文献

[1] D. Angluin, J Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[2] D. Angluin, J. Aspnes, M. J Fischer, and H. Jiang. Self-stabilizing population protocols. *ACM Transactions on Autonomous and Adaptive Systems*, 3(4):13, 2008.

[3] Y. Sudo, T. Masuzawa, T.A.K Datta, and L.L. Larmore. The same speed timer in population protocols. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 252–261. IEEE, 2016.

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

[5] Y. Sudo, F. Ooshita, H. Kakugawa, and T. Masuzawa. Loosely-stabilizing leader election on arbitrary graphs in population protocols. In *OPODIS*, pages 339–354, 2014.

[6] Y. Sudo, F. Ooshita, H. Kakugawa, and T. Masuzawa. Loosely-stabilizing leader election on arbitrary graphs in population protocols without identifiers nor random numbers. In *OPODIS*, 2015.

A Fast and Space-Efficient Approximate Counting in Population Protocol Model

Ryota Eguchi Taisuke Izumi

The population protocol model is known as a model of passively mobile networks. It consists of n agents and performs some distributed computation based on pairwise interactions between two agents. In this paper, we consider the approximate counting problem on the population protocol model, which requires the system to output an approximate value of n. The main contribution of this paper is to propose a new algorithm for approximate counting. It assumes a loose upper bound on n (denoted by N), and for any given $\epsilon < 1$ outputs $(1+\epsilon)$-approximate value of n within $O(n \log\log N \log^3 N)$steps. Since the proposed algorithm utilizes only $O(1/\epsilon^2 \log 1/\epsilon + \log\log N)$-bit memory space per each agent, it achieves the optimal space complexity for constant ϵ.

1 Introduction

1.1 Background and Motivation

A *passively-mobile* system is a collection of agents that move in a certain region but have no control over how they move. Since the communication range of each agent is limited, two agents can communicate only when they are sufficiently close to each other. A typical example of passively-mobile systems is the network of smart devices attached cars or animals. The *population protocol* is one of the promising models for such a system, which was initiated by Angluin et al. [4]. A population protocol consists of n agents, to which some program (algorithm) is deployed. Following the deployed algorithm, each agent changes its state by *pairwise interactions* to other agents (that is, two agents get closer to each other and update their states by exchanging information). The run of the protocol is a sequence of pairwise interactions, and all the agent must agree on some consistent outputs specified by the problem definition (e.g., electing a leader, deciding the majority of opinions, counting the total number of agents, and so on). In the last few years, the population protocol has received much attention among the distributed-computing community.

In this paper, we consider the $(1+\epsilon)$-approximate counting problem in population protocol models. Our main focus is on the time and space complexity of the (approx-

imate) counting problem. While the original model by Angluin et al. [4] incurs the restriction that each agent only equips the memory of $O(1)$ bits, superconstant-size local memory is inherently essential to make the counting problem solvable. Assuming $\Omega(\log n)$-bit space, we can construct a simple exact counting problem based on the value aggregation: Each node initially has value one, and when two agents with values x and y interacts, one of them takes value $x+y$ and the others zero. Eventually the maximum value of all agents converges to n. Unfortunately, under the uniformly-random scheduler, this algorithm takes $\Omega(n^2)$ steps to output the correct answer, where the average number of interactions required by one agent (so-called "parallel time") becomes linear. In the environment with a massive number of agents, it is quite inefficient. The primary challenge is to consider how to accelerate the running time of counting algorithms into $o(n^2)$ steps, hopefully $O(n\mathrm{polylog}(n))$ steps. In addition, yet another challenge also arises in the case of approximate counting. When considering $(1+\epsilon)$-approximate case, it suffices to identify the value i such that $(1+\epsilon)^i \leq n \leq (1+\epsilon)^{i+1}$, and thus the information theoretic bound on the memory space for the counting problem is further reduced to $\log\log_{(1+\epsilon)} n$ bits. That is, for constant ϵ, we might have an $(1+\epsilon)$-approximate counting algorithm using $O(\log\log n)$ bits (actually, there exist sequential algorithms counting value n approximately using only $O(\log\log n)$ bits [14, 17]). Overall, the main interest is summarized as follows: Can we construct a $(1+\epsilon)$-approximate counting algorithm achieving both $O(n\mathrm{polylog}(n))$ steps and $O(\log\log n)$ bits for any constant ϵ? Answering this question is obviously a non-trivial challenge.

1.2 Our Contribution

Our main contribution is to answer the question above positively. We propose a fast and small-space algorithm for $(1+\epsilon)$-approximate counting algorithm under the uniformly-random scheduler. For any constant $\epsilon < 1$, the space and time complexity of the proposed algorithm are $O(\log\log N)$ bits and $O(n\log\log N\log^3 N)$ steps respectively, where N is the known (possibly loose) upper bound on n. If $N = O(n^c)$ holds for some constant c, our algorithm attains both the optimal space complexity and polylogarithmic running time.

1.3 Organization

The paper is organized as follows: In Section 2 we state the related work. Section 3 provides the notations and definitions used in the paper. Section 4 provides the main result of $(1+\epsilon)$-approximate counting algorithm. Finally the paper is concluded in Section 5.

2 Related Work

The population protocol model is initiated by the two seminal papers by Angluin et al. [4] and Angluin et al. [6], where the main interest was to clarify the class of computable

predicates with the inputs distributed over all agents. Following the first results on computability, several directions have been considered: Self-stablization [7, 13, 9, 10, 15, 18], fault-tolerance [11], and space and time complexity [1, 2, 3, 5, 12]. In particular, revealing the feasibility of fast (i.e., $O(n\text{polylog}(n))$ steps) algorithms under the probabilistic scheduler is one of the recent trends in the study of population protocol models. The first result on this direction is a leader-based predicate computation by [5], where the constant-space population protocols with initial leaders allow faster predicate computation. The possibility of fast leader election is recentely considered [1, 2, 12]. First, a faster leader election algorithm using sublogarithmic space is presented by Alistarh and Gelashvili [2]. An almost quadratic step complexity is also proved by [12] for constant-space population protocols. Very recently, those two results are extended in the context of time-space tradeoff [1]. Except for the leader election problem, there are few studies considering fast algorithms [3]. To the best of our knowledge, this is the first paper tackling the (approximate) counting problem.

A yet another setting on the counting problem is the one where a spacial agent (called *base station* is required to compute the total number of agents. The computability and space complexity for this setting under the worst-case scheduler is also considered so far [9, 15, 8].

3 Preliminaries

Population Protocol Model A population protocol consists of $n \leq N$ agents, each of which is a state machine having a state in a set Q (whose cardinality can depends on n and/or N). Agents update their states following a transition function $\delta : Q \times Q \to Q \times Q$ when an interaction occurs. The possibility of interactions between agents is defined by an *interaction graph* G with no self-loops, whose nodes indicate the agents and whose edges indicate that two agents corresponding to its endpoints can interact. Throughout this paper, we assume G is complete, that is, any two agents can interact. An execution of a protocol is a sequence of interactions (say *steps*. At each step, the scheduler chooses an edge in G. We also assume the uniformly-random scheduler. That is, eacn edge in G (i.e., each pair of agents) is selected by the scheduler uniformly at random.

$(1+\epsilon)$-Approximation Counting The goal of the counting problem in population protocol models is that each agent outputs (i.e., stores in a special local memory) the exact value of n eventually. In its approximate version, the problem is parameterized by ϵ, and allows each agent to output a $(1+\epsilon)$-approximate value of n, where we define the $(1+\epsilon)$-approximate value of n by the value d satisfying $(1+\epsilon)^{d-1} \leq n \leq (1+\epsilon)^{d+1}$. As we stated in the introduction, the information-theoretic bound (i.e., the space requirement to store the output) on this problem is $O(\log \log_{(1+\epsilon)} n)$. Since this paper considers randomized Monte-Carlo algorithms, we also allow a sufficiently-small failure probability that the algorithm outputs incorrect values. We say that a population protocol solves $(1+\epsilon)$-approximate counting within k steps with probability p, if all the agents output

the correct approximate value with probability p at the end of step k.

4 $(1+\epsilon)$-Approximate Counting Algorithm

In this section, we present our $(1+\epsilon)$-approximate counting protocol. The key idea of our algorithm is to construct a subroutine, which tests if the value n is larger than a given value $\hat{n}$ or not. The approximate value to be outputted is decided by testing $O(\log N)$ candidates (precisely $\hat{n}_i = (1+\epsilon)^i$ for all $i \in [0, \log N]$) in the binary-search manner. We refer this testing algorithm as TestCount($\hat{n}$).

4.1 Algorithm TestCount($\hat{n}$)

Before looking at the details of that algorithm, we first explain the algorithmic idea behind it.

We first consider the following simple counting algorithm: (1) Initially, each agent randomly chooses an $O(\log N)$-bit ID (the hidden coefficient is sufficiently large so that two processes can take the same ID only with a very small probability). (2) Through a number of interactions, all-to-all ID exchange is performed. Finally the number of collected IDs is outputted as the number of agents. Since it is possible to show that $O(n \log^2 n)$-step interactions suffices for successfully achieving all-to-all exchange, this algorithm is a fast counting protocol. However, the space complexity is obviously high (i.e. $\Omega(n \log N)$ bits), which is far from our goal. The key idea of TestCount($\hat{n}$) is reduce the space by local sampling. That is, each agent first decides whether it becomes active or inactive with some probability $p(\epsilon, \hat{n})$ determined by ϵ and $\hat{n}$. Then, the only active agents propagate their IDs. The number of collected IDs is a sort of the estimation of $np(\epsilon, \hat{n})$ if no collision of IDs occur. The memory space required this mechanism clearly depends on $p(\epsilon, \hat{n})$ and the size of the ID domain. Taking a small probability saves much space, but the precision of the estimation decreases. Fortunately, for any $\epsilon < 1$, a constant-size memory attains the sufficiently accurate decision on the testing of $\hat{n} \geq n$ or not.

Algorithm 1 shows the pseudo-code of the algorithm TestCount($\hat{n}$). It works as follows:

1. Let $\epsilon' = \epsilon/4$ for short. Each agent initially decides whether it will be active or inactive with probability $p(\epsilon')/\hat{n}$, where $p(\epsilon') = \frac{192(1+\epsilon')^4}{\epsilon'^2}$.
2. Each active agent (referred as x) takes an integer value uniformly at random from $[1, r(\epsilon')]$ as its ID, and stores it into the set S_x, where $r(\epsilon') = \lceil \mathrm{e}^4(\sigma(\epsilon') - 1)^2 \rceil$ and the function $\sigma(\epsilon')$ is defined later.
3. When an interaction between agents x and y occurs, they updates their own sets S_x and S_y by those union (i.e. $S_x, S_y \leftarrow S_x \cup S_y$).
4. The agent x tests the size of set S_x when its interaction occur. If $|S_x|$ is larger than the threshold $\sigma(\epsilon') = \lceil (\frac{1+2\epsilon}{1+3\epsilon'})^2 p(\epsilon') \rceil$ then agent x concludes $\hat{n}_i \leq n$. Otherwise,

if $|S_x|$ remains under the threshold when the agent interacts $T_{out} = 18\sigma(\epsilon')\log^2 N$ times, then it concludes $\hat{n}_i > n$.

Intuitively, the reason why this strategy works well is understood as follows: In the step 1, since the expected number of active agents is $\mathrm{E}[X] = p(\epsilon') \cdot n/\hat{n}_i$, if $\hat{n}_i \leq n$ then we could expect to $X \geq p(\epsilon')$, or if $\hat{n}_i > n$ then $X < p(\epsilon')$. In the step 2, if the IDs which active agents get are all distinct, then the number of IDs is equal to the number of active agents (and such situation occurs with high constant probability, if the range of random value is sufficient large). In the execution phase each agent consequently could determine $\hat{n}_i \leq n$ according to the number of elements in its set. The formal analysis is presented in the following proof.

Algorithm 1 TestCount($\hat{n}$)

Variables of Agent x:

t_x: counter of its interactions, initializing to 0
S_x: a set stores IDs
$result_x$: variable which have the result

Constants:

T_{out}: timeout value
$\sigma(\epsilon')$: the threshold($= \lceil \frac{192(1+\epsilon')^4(1+2\epsilon')^2}{(1+3\epsilon')^2\epsilon'^2} \rceil$)
i: candidate of approximation$\hat{n}_i (= (1+\epsilon)^i)$

Auxiliary Procedure

$$coin\_tosses(i) = \begin{cases} \text{SUCCESS} & (\text{with probability } p(\epsilon)/\hat{n}_i) \\ \text{FALSE} & (\text{with probability } 1 - p(\epsilon)/\hat{n}_i) \end{cases}$$

Operation of Agent x:

initialize()

1: **if** ($coin\_tosses(i)$ ==SUCCESS) **then**
2: $S_x \leftarrow \{get\_id()\}$ //get ID and store in S_x
3: **endif**

*examine*_$\hat{n}_i(i)$

1: **when** (x interacts with some y) **do**
2: $t_x \leftarrow t_x + 1$; $S_x \leftarrow S_x \cup S_y$;
3: **if** ($count(S_x) \geq \sigma(\epsilon')$) **then**
4: return ($result_x \leftarrow$ LESS)
5: **endif**
6: **if** ($t_x \geq T_{out}$) **then**
7: return($result_x \leftarrow$ GREATER)
8: **endif**

4.2 Correctness Proof of TestCount($\hat{n}$)

In the proof, we first show that (i) active agents spread their IDs to all other agents in $O(\log^2 N)$ time with high probability, and that (ii) agents can detect with high probability the termination of spreading by counting the number of their local interactions (i.e. when some agent interacts T_{out} times, all spreading would finish with high probability).

The first Lemma 4.1 corresponds to the fact (i).

Lemma 4.1. *Suppose that there are $\xi(\leq n)$ distinct IDs in the system at step t_0. After $100\xi n \ln^2 N$-steps, all agents collect all the IDs existing in the system at t_0 with probability $1 - O(1/N^{10})$.*

Lemma 4.2 shows that when some agent interacts $O(\ln^2 N)$ times, the number of overall interactions would be $\Theta(n \ln^2 N)$.

Lemma 4.2. *For any constants $\alpha > 3$, from an arbitrary step t_0 after $40\alpha n \ln^2 N$ steps every agent interacts less than $120\alpha \ln^2 N$ times, and after $160\alpha n \ln^2 N$ steps from step t_0 every agent interacts more than $120\alpha \ln^2 N$ times with probability at least $1 - O(1/N^{10})$.*

We next show that the randomized part of this algorithm succeed with a sufficiently-high constant probability. In order to work $\mathsf{TestCount}(\hat{n})$ correctly, the number of active agents must be larger than the threshold $\sigma(\epsilon)$ if $\hat{n}_i \leq n$, and otherwise it must be less than $\sigma(\epsilon)$. In addition, if $\hat{n}_i \leq n$ the number of distinct IDs of active agents must be larger than $\sigma(\epsilon)$. Lemma 4.3 handles the event concerning the number of active agents, and Lemma 4.4 handles the event that active agents get different IDs with a sufficiently-high constant probability.

Lemma 4.3. *With probability at least $1 - \frac{1}{e^4}$, if $\hat{n}_i \leq n$ then the number of active agents will be larger than $\sigma(\epsilon)$ otherwise the number of active agents will be less than $\sigma(\epsilon)$.*

Lemma 4.4. *If the number of active agents is larger than $\sigma(\epsilon)$, then the number of distinct IDs will be at least $\sigma(\epsilon)$ with probability at least $1 - 1/e^4$.*

We summarize this analysis in Theorem 4.5.

Theorem 4.5. *For any i and $\hat{n}_i = (1+\epsilon)^i$, $\mathsf{Testcount}(\hat{n}_i)$ decides if the value $\hat{n}_i$ is larger than n or not within at most $O(n \log^2 N)$ steps, with probability at least $1 - 1/e^2$.*

4.3 Main Algorithm

The pseudocode of the main algorithm based on $\mathsf{TestCount}(\hat{n})$ is presented in **Algorithm 2**. It achieves $(1+\epsilon)$-approximation with high probability. The high-level idea of the main algorithm is very simple: It tests the values $(1+\epsilon), (1+\epsilon)^2, \ldots (1+\epsilon)^i, \ldots (1+\epsilon)^{\log_{(1+\epsilon)} N}$ in the binary-search manner. It requires $O(\log \log_{(1+\epsilon)} n)$ times of executing $\mathsf{TestCount}(\hat{n})$. To make whole of the run succeed with high probability, the success probability of each testing is amplified by $O(\log N)$-time iteration and majority voting.

The algorithm carries out multiple instances of $\mathsf{TestCount}(\hat{n})$ sequentially, which is crucial to achieve $O(\log \log N)$-bit space usage (in each instance, the local memory space is recycled). However, since population protocols cannot have any global synchronization mechanism, the sequential composition of multiple instances is generally a non-trivial task. Fortunately, in our algorithm, the following simple way resolves this matter: (1) Extend the timeout length of $\mathsf{TestCount}(\hat{n})$ into the twice of the original length, and

(2) when interacting in k-th instances, propagates the result of $(k-1)$-th instance. If an agent running in $(k-1)$-th instance received the result, it immediately adopts the received result and proceeds to the next instance.

We intuitively explain the outline of the correctness. To make the binary search work successfully, testing all candidates must return correct answers. Since each instance of TestCount($\hat{n}$) succeeds with probability 2/3, $O(\log N)$-time majority voting achieves the success probability of $1 - 1/\mathrm{poly}(N)$, which is sufficiently high to make whole of the algorithm get high success probability. Thus the remaining issue on the correctness is to see that the synchronization mechanism works correctly. Since the behavior of TestCount($\hat{n}$) is to broadcast several messages in parallel, it works correctly if there exists an interval of length T_{out} steps (i.e., the length for which information propagation correctly finishes) where all agents run the same instance. The point (2) stated above guarantees that when an agent starts k-th instance at step t_0 then every agent goes into there by $t_0 + T_{out}$, and because of the point (1) all of agents necessarily share the interval $[t_0 + T_{out}, t_0 + 2T_{out}]$ for processing k-th instance.

We look at the time and space complexity of the algorithm. Since there exists $O(\log_{(1+\epsilon)} N)$ candidates (i.e., $(1+\epsilon), (1+\epsilon)^2, \ldots (1+\epsilon)^i, \ldots (1+\epsilon)^{\log_{(1+\epsilon)} N}$) in $[1, N]$ for testing, the total number of candidates to be tested is $O(\log \log N)$ for any constant ϵ. Each testing takes $O(\log N)$-time iteration of TestCount($\hat{n}$), the running time of the whole algorithm is $O(n \log \log N \cdot \log^3 N)$ steps. For the space complexity, each instance of TestCount($\hat{n}$) takes $O(\log \log N)$-bit space, and it is recycled at each iteration. The extra requirement is the space for counting the number of iterations (and voting), and for maintaining the range of the binary search. The first part is obviously needs $O(\log \log N)$ bits. In the second part, the algorithm stores only the exponent of the the lower and upper bounds, which results in $O(\log \log N)$-bit space usage.

Algorithm 2 The approximation algorithm

variables of agent x:
r_x : counter of rounds
$less_x, greater_x$: counter of results of test_count

The modified TestCount($\hat{n}$) of Agent x

1: **when** (x interacts with some y) **do**
2: **if** $(r_x + 1 == r_y)$ **then**
3: return($result_y$) //receives a result from agent y
4: // following the operations of test_count($\hat{n}$)

The Overall Operation of Agent x

1: **while**(to finish binary searching) **do**
2: r_x++
3: **if**(the modified TestCount($\hat{n}$)==LESS) **then** $less_x$++
4: **else** $greater_x$++
5: **endif**
6: $next\_round(r_x)$ //initialize variables and if rounds are over go to the next candidate
7: **endwhile**

Lemma 4.6. *For a constant $\lambda \geq 1$, suppose that the system sequentially runs λ instances of* TestCount($\hat{n}$) *with timeout value T'_{out}. Then there exists the value T'_{out} that ensures all λ implements of modified* TestCount($\hat{n}$) *sufficient large time per one implement to operate correctly with probability at least $1 - O(1/N^9)$*

Theorem 4.7. *The approximation algorithm returns the $(1+\epsilon)$-approximation value for the number n of agents in the system with probability at least $1 - 0(1/N)$.*

5 Conclusion

In this paper, we presented a randomized approximation algorithm for the counting problem, which achieves $(1+\epsilon)$-approximation for any $\epsilon < 1$, and $O(\log \log n)$-bit space and $O(npolylog - n)$ step complexity. This algorithm uses the assumption that the system knows the loose upper bound N on the number of the agents n. Since this assumption is utilized just for each agent to detect the end of information spreading, we conjecture it is possible to remove it. We also conjecture that it is impossible to have an exact and space optimal counting algorithm with sub-linear parallel-time (i.e. $o(n^2)$ steps). It would be intriguing to prove the lower bounds of both time and space complexity for the exact and/or approximate counting problem.

References

[1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in molecular computation. In *4th workshop on Biological Distributed Algorithms(BDA)*, 2016.

[2] Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *Proc. of The 42nd International Colloquium on Automata, Languages, and Programming(ICALP)*, 2015.

[3] Dan Alistarh, Rati Gelashvili, and Milan Vojnović. Fast and exact majority in population protocols. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 47–56. ACM, 2015.

[4] 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.

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

[6] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[7] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. *ACM Transactions on Autonomous Adaptive Systtems*, 3(4):1–28, 2008.

[8] Joffroy Beauquier, Janna Burman, Simon Clavière, and Devan Sohier. Space-optimal counting in population protocols. In *Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, pages 631–646, 2015.

[9] Joffroy Beauquier, Julien Clement, Stephane Messika, Laurent Rosaz, and Brigitte Rozoy. Self-stabilizing counting in mobile sensor networks with a base station. In *Proc. of 21st International Symposium on Distributed Computing(DISC)*, pages 63–76. Springer Berlin Heidelberg, 2007.

[10] 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*, 50(3):433–445, 2012.

[11] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. *When Birds Die: Making Population Protocols Fault-Tolerant*, pages 51–66. 2006.

[12] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In *International Symposium on Distributed Computing*, pages 602–616. Springer, 2015.

[13] Michael Fischer and Hong Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Proc. of 10th International Conference on Principle of Distributed Systems(OPODIS)*, volume 4305, pages 395–409, 2006.

[14] Philippe Flajolet. Approximate counting: A detailed analysis. *BIT*, 25(1):113–134, 1985.

[15] Tomoko Izumi, Keigo Kinpara, Taisuke Izumi, and Koichi Wada. Space-efficient self-stabilizing counting population protocols on mobile sensor networks. *Theoretical Computer Science*, 552:99–108, 2014.

[16] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[17] Masatora Ogata, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. A randomized algorithm for finding frequent elements in streams using $o(loglogn)$ space. In *Proc. of 22nd International Symposium on Algorithms and Computation(ISAAC)*, pages 514–523, 2011.

[18] Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely-stabilizing leader election in a population protocol model. *Theoretical Computer Science*, 444:100–112, 2012.

個体群プロトコルモデル上での アルゴリズムの緩安定化

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．

自己安定とは，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．また緩安定とは，自己安定の制約を弱めたものであり，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が，十分に高い確率で維持することが保証されている（逆に述べると，十分に小さい確率で正当な状況を外れることを許している）．

これらのアルゴリズムが個体群プロトコル上でいずれも近年精力的に研究されており，多くの結果がそれぞれ得られている．一方で個体群プロトコル上で一般的なアルゴリズムを自己安定や緩安定アルゴリズムへの変換はあまり考えられていない．既存の研究では，個体群プロトコル上で出力が一意に決まるアルゴリズムに対して，それを自動的に自己安定化する変換アルゴリズムが提案されている [1]．本研究では，検討の一つとして，個体群プロトコル上で一般のアルゴリズムを緩安定アルゴリズムに変換することを考える．

自己安定変換器では，エージェント間のインタラクション速度を導入した半同期的スケジューラを仮定し，そのもとでの変換を検討している．このアルゴリズムでは無限のメモリ量を持つ基地局を仮定しており，またその過程が必要であることも併せて示されている．また変換対象のアルゴリズムの出力が一意という制約がある．この制約のもとで基地局からアルゴリズムを繰り返し実行させることで自己安定化を実現する．

本研究では，自己安定ではなく緩安定へと制約を緩くすることで，基地局の存在なしで変換でき，エージェントのメモリ量を有限に抑えることが可能ということを示した．これを出力が一意に決まるアルゴリズムに対してだけではなく，正当な状態がいくつもあるアルゴリズムに対してできるか，また本研究では，リーダ選挙を用いてリーダを決めてから行ったが，それを用いずにそれより高速にできるかが今後の課題である．

参考文献

[1] Joffroy Beauquier, Janna Burman, and Shay Kutten. A self-stabilizing transformer for population protocols with covering. 2010.

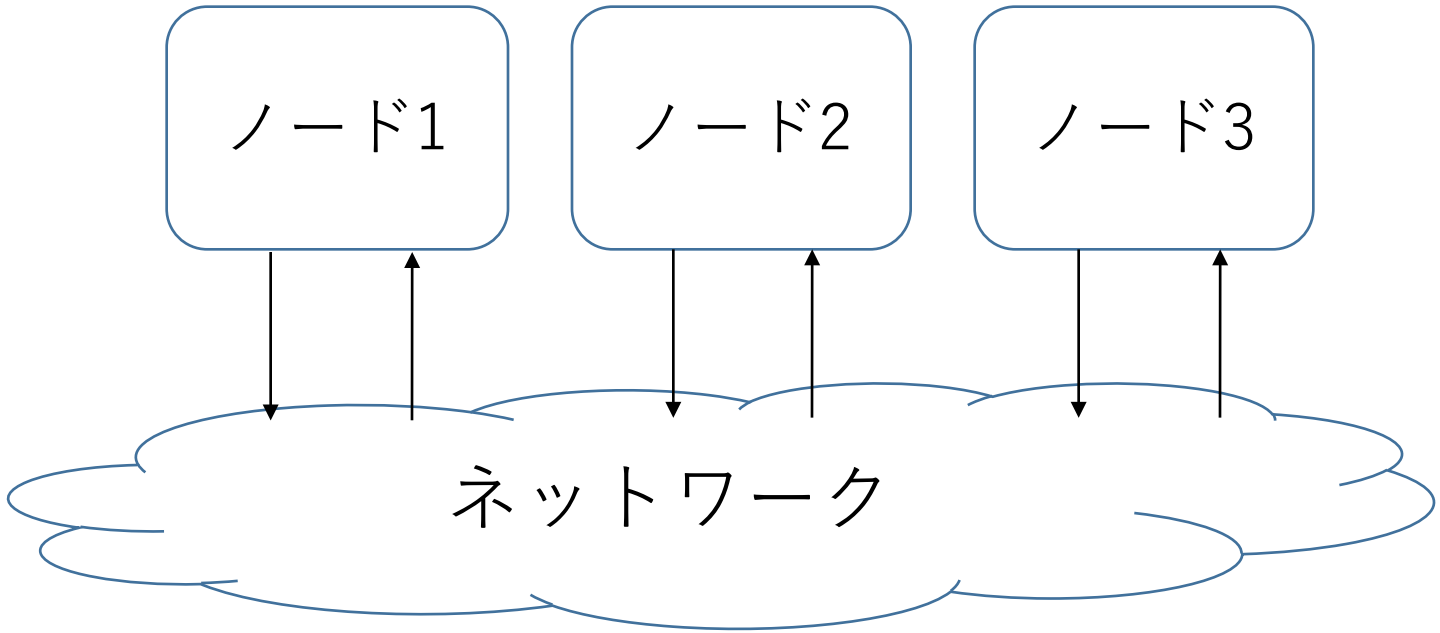
分散システム開発の支援

小貫　直之
東京工業大学

背景

➢分散システム

- 複数のノード（プロセス）が同時に実行
- ネットワークを介してノード同士がメッセージ通信



背景

➢分散システム

- 分散することで、耐故障性が上がるなどのメリット
- 一方、一貫性の問題などが発生し複雑さが増す

- 複雑さを解消するためにコンポーネント化をする

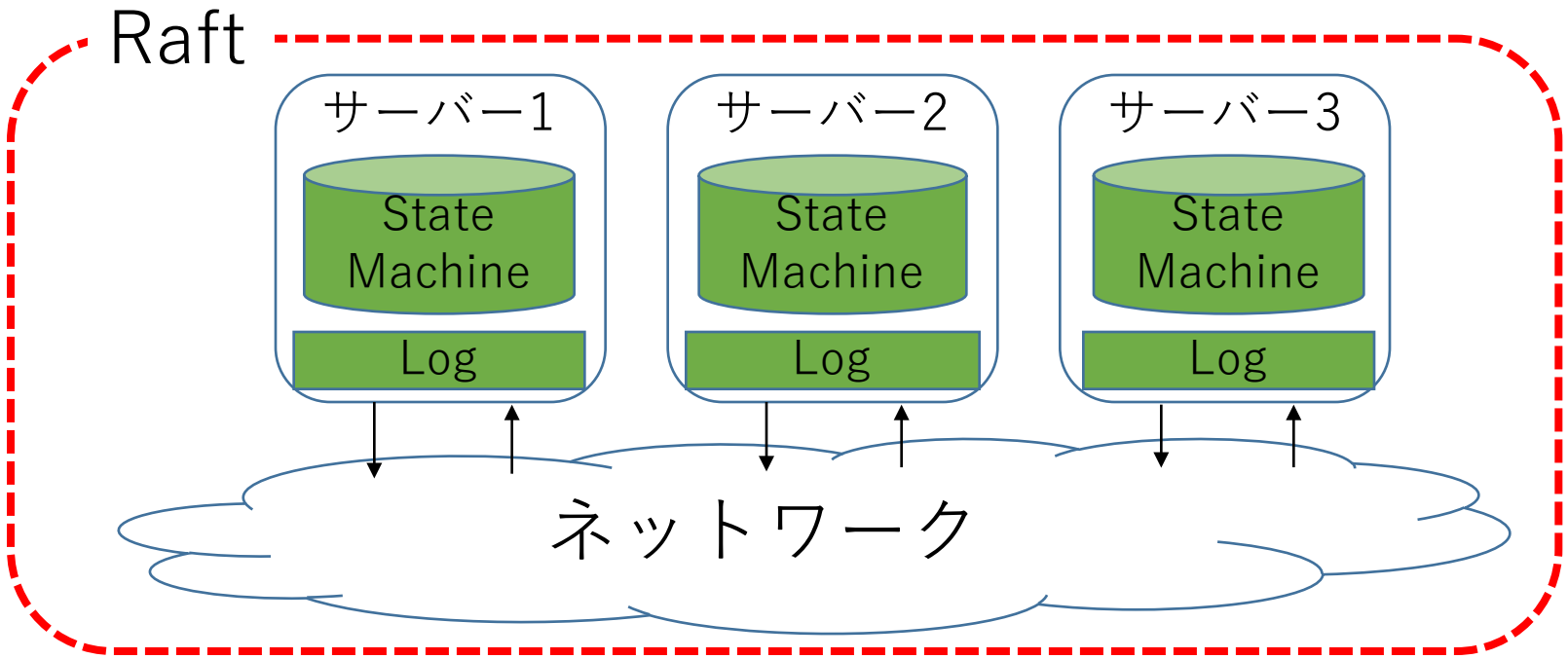
  ➡プロトコル

背景

- プロトコル
  - 機能を持ったまとまり
  - コンポーネントのようなもの
  - アルゴリズムに従ったノード内のメッセージ通信・他ノードとのメッセージの送受信を行う

背景

- 具体例：Raft
  - 分散合意アルゴリズム
  - 理解のしやすさを重視して作られた
  - ログとState Machineを持つ複数のサーバーで構成



背景

- Raftの扱う問題
  - 並列して送られてくるリクエストについて合意をとる
    - リクエスト：State Machineへの入力
  - Input : クライアントからの入力
  - Output : 各サーバーのState Machineがいつも同じ状態

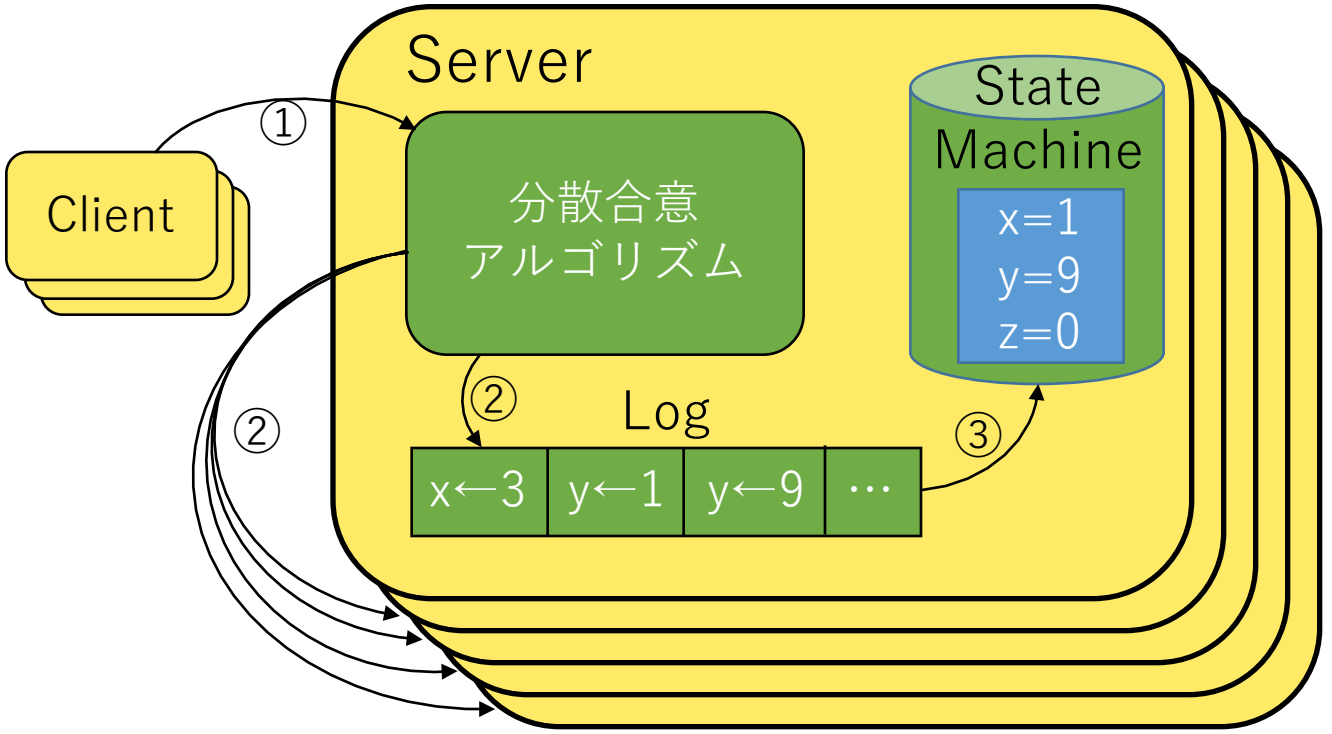
  - 各サーバーが届いたリクエストをそのまま適用するだけだと一貫性が保てない

背景

- Raftのアルゴリズム
  - 1台のリーダーと複数台のフォロワーが存在
  - 多数のクライアントからリクエストが送信される

① クライアントがサーバーにリクエストを送信
② リーダーが一括してリクエストを受け取り、ログとして各フォロワーに送信
③ ログ内のリクエストを順にState Machineに適用

➡各サーバーが同じログとState Machineを保有



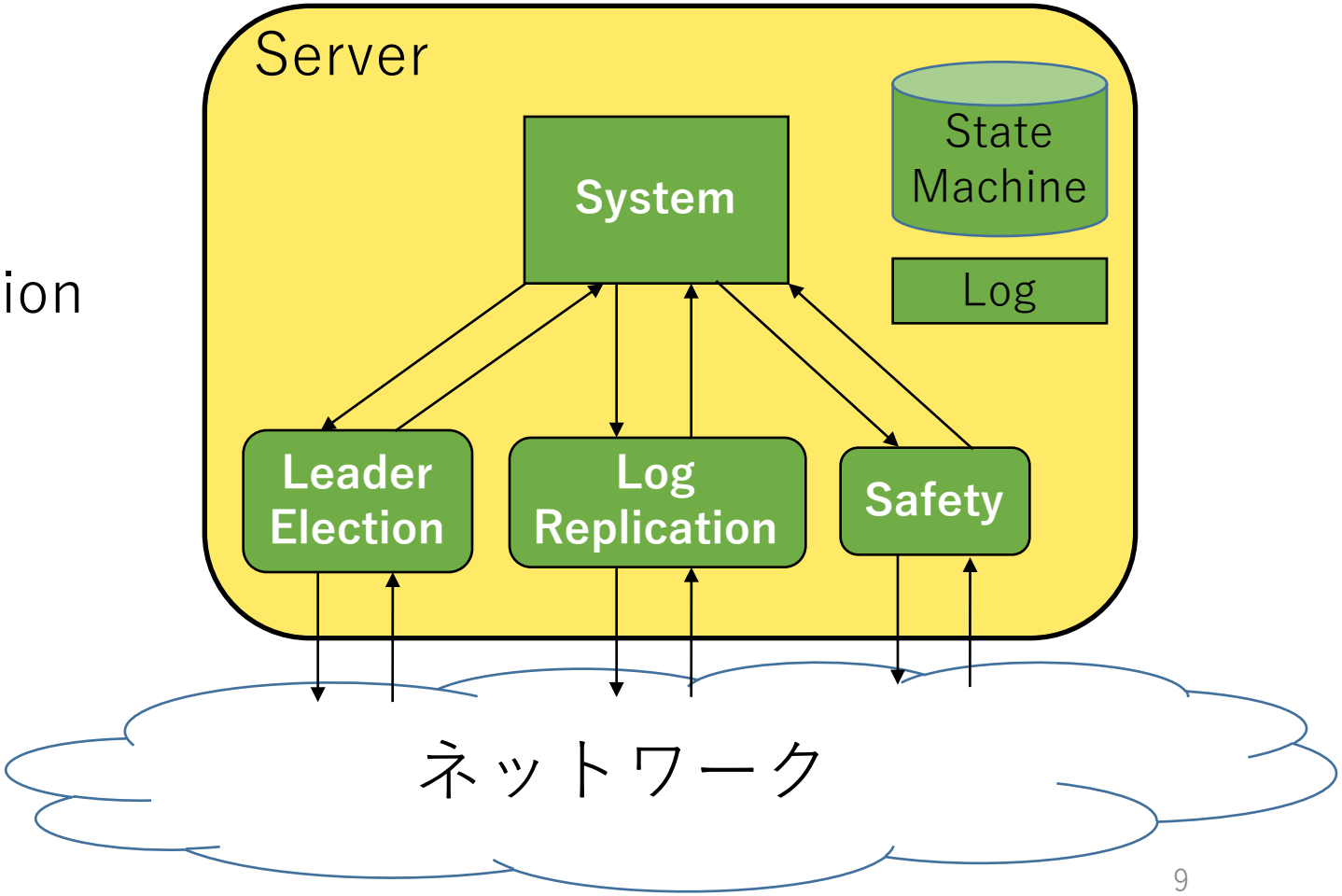
背景

- プロトコルの例(Raft)
  - Raftは分散合意アルゴリズムを３つの独立した問題に分けている
    1. リーダーの選出
    2. ログの複製
    3. 安全性の保証

➡それぞれをプロトコルとして実装できそう

背景

- プロトコルの例(Raft)
  - リーダーの選出はLeader Electionプロトコルが担当
  - ログの複製はLog Replicationプロトコルが担当



背景

コンポーネント

ノード内でのやり取りのみ

⇩

一次元

プロトコル

ノード内とノード同士の
２つのやり取り

⇩

二次元

目的

- プロトコルの作成の支援

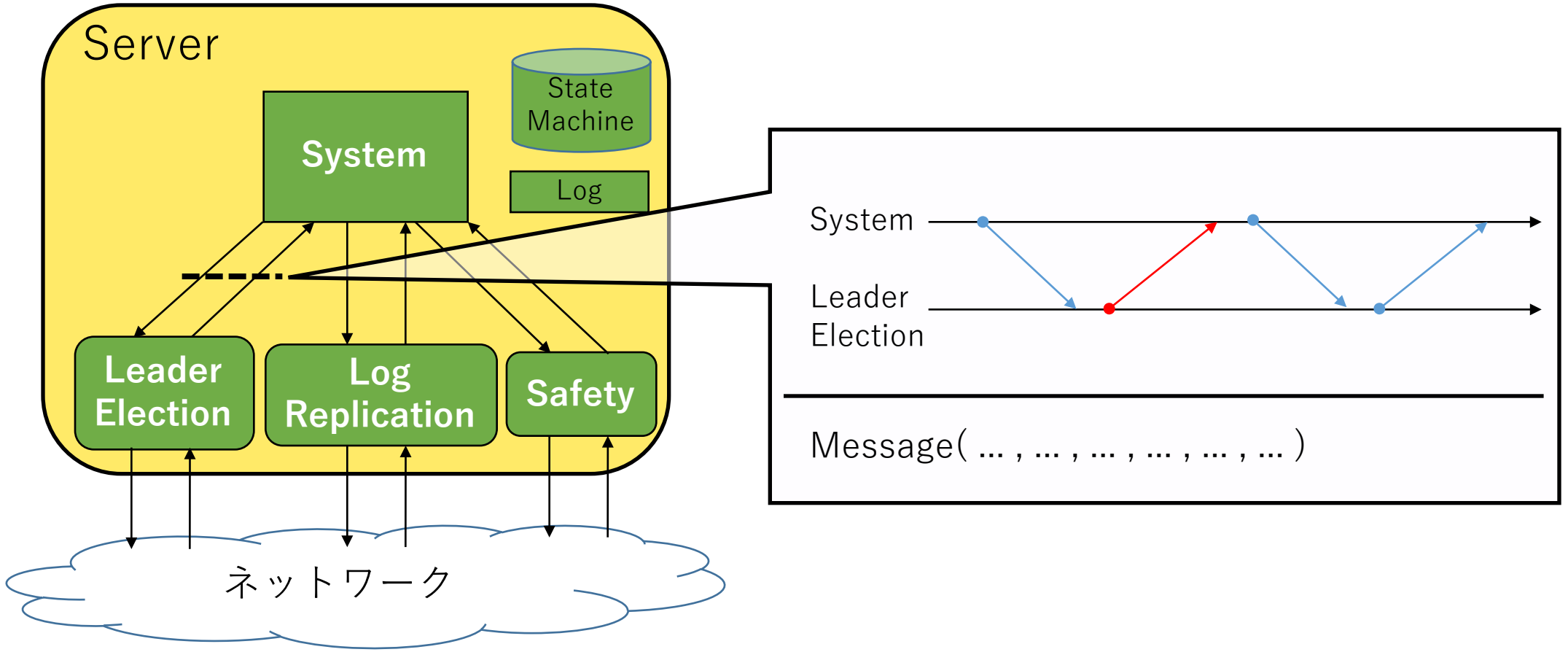
方法

以下の機能を持つ開発プラットフォームを作る

- 実行の様子の可視化
  - メッセージ送受信の確認
- プロトコルの合成
  - 簡単なプロトコルを組み合わせて複雑なプロトコルを作る

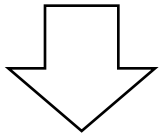
方法

- 実行の様子の可視化（イメージ）



まとめ

プロトコルの作成を支援(可視化・合成)して



分散システム開発の複雑さを軽減したい

参考

- The Raft Consensus Algorithm
  https://raft.github.io

- Diego Ongaro and John Ousterhout (2014), "In Search of an Understandable Consensus Algorithm", *USENIX ATC'14 Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference*, Pages 305-320

メッセージ通信型分散アルゴリズムの移動エージェントによる耐故障シミュレーション

五島　剛[†]　柴田　将拡[†]　大下　福仁[††]　角川　裕次[†]　増澤　利光[†]

† 大阪大学大学院情報科学研究科　〒 565-0871 大阪府吹田市山田丘 1-5
†† 奈良先端科学技術大学院大学　〒 630-0192 奈良県生駒市高山町 8916 番地の 5

あらまし　近年，分散システムの大規模化，複雑化が進んでおり，その効率の良い設計手法としてモバイルエージェントが注目を集めている．一方，多くのメッセージ通信型分散アルゴリズムが様々なタスクに対して提案されており，これらをモバイルエージェントでシミュレートできれば，モバイルエージェントでも様々なタスクを実現できる．本報告では，最大 f 体のエージェントが停止故障する環境において，総メッセージ数 M のメッセージ通信型アルゴリズムを総移動数 $O((e+M)f)$ でシミュレートするアルゴリズムを提案する (e はリンク数)．既存手法は，エージェント数 k に対し，最大 $k-1$ 体のエージェントが故障する環境において，$O((e+nM)k)$ の総移動数でシミュレーションを実現している (n はノード数)．つまり，提案手法は，以下の 2 点において既存手法を改善している．第一に，既存手法と異なり，最大故障数 f に応じて総移動数を削減することができる．第二に，$f=k-1$ の場合でも，既存手法より総移動数を削減することができる．

キーワード　分散システム，メッセージ通信モデル，モバイルエージェントモデル，耐故障シミュレーション

Move-Efficient Fault-Tolerant Simulation of Message-Passing Algorithms by Mobile Agents

Tsuyoshi GOTO[†], Masahiro SHIBATA[†], 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

1. はじめに

分散システムとは，多数の計算機 (以降，ノード) とそれらを繋ぐ通信リンク (以降，リンク) から構成されたシステムである．近年，分散システムは大規模化が進んでおり，分散システムの設計はますます困難になっている．そのため，大規模化・複雑化する分散システムを設計するための有効な手法として，モバイルエージェントを利用した設計方法が注目を集めている [3]．モバイルエージェント (以降，エージェント) とは，ネットワーク中の計算機を移動しながら，自律的に動作するソフトウェアである．リーダー選挙，エージェント配置，集合，自己安定，終了検知，探索，トポロジ検査，など，様々な問題に対して，モバイルエージェントアルゴリズムが提案されている．また，セキュリティに関する，侵入者検知 [1，2]，ネットワーク浄化 [10，12] などの問題に対するアルゴリズムも提案されている．

上記で挙げたアルゴリズムでは，そのほとんどが，ノード及びエージェントが故障しないと仮定しているが，大規模化が進む近年の分散システムでは，ノード及びエージェントの故障は不可避である．それに伴い，故障するノード [6，7] や，訪問したエージェントを破壊するノード [8，11] の存在を仮定した研究が行われるようになっている．また，ノードだけでなく，エージェント自身が故障することを仮定した研究も行われている．例えば，現在，システムは世界規模になっていて，エージェントが物理リンクを使って長距離移動することも多くある．物理リンクに故障が起これば，移動の間にエージェントが故障する．したがって，エージェントの故障を考えることが要求されている．

エージェントで様々なタスクを実行する手法として，メッセージ通信モデルの分散アルゴリズムをシミュレートする方法がある [5，13，4]．メッセージ通信モデルは，メッセージ通信を行う多数の計算機をモデル化したもので，これまでに多数の分散アルゴリズムが提案されている [9]．メッセージ通信モデルのアルゴリズムをエージェントが効率よくシミュレートすることで，既存の分散アルゴリズムをモバイルエージェントシステムで利用可能となる．現在提案されているエージェント故障を仮定したメッセージ通信アルゴリズムのシミュレーションは，Das et al. [5] によるものだけである．[5] では，エージェント数 k に対し，最大 $k-1$ 体のエージェントが停止故障すると仮定して，メッセージ通信モデルの任意のアルゴリズムをシミュレートするアルゴリズムを 2 つ提案している．一つ目のアルゴリズムは，識別子のあるネットワークにおいて，総メッセージ数 M のメッセージ通信アルゴリズムを総移動数 $O((e+nM)k)$ でシミュレートする (n はノード数，e はリンク数)．もう一つは，識別子の無いネットワークにおいて $O((e+nk)M)$ の総移動数でシミュレーションを実現する．これら 2 つのアルゴリズムでは，メッセージを 1 つ送信するために $O(nk)$ 回の移動数が必要となる．

本報告では，最大 $f \leqq k-1$ 体のエージェントが停止故障する識別子のあるネットワークにおいてメッセージ通信型アルゴリズムのシミュレーションを実現するモバイルエージェントアルゴリズムを提案する．ただし，エージェントは識別子を持ち，f は既知としている．エージェントの総移動数は $O((e+M)f)$ であり，$e \leqq M$ のとき 1 メッセージ当たりの移動数を $O(f)$ で抑えることができる．これは，メッセージ 1 つに対して必要となる移動数としては，最適なものである．提案アルゴリズムは，$f = k-1$ の場合でも，既存手法と比較して総移動数を削減できる．本アルゴリズムでエージェントが使用するメモリは，メッセージ通信アルゴリズムにおけるメッセージの最大サイズと同一であり，最適なものとなっている．

2. 諸 定 義

2.1 ネットワークモデル

ネットワークは無向グラフ $G(V, E, \lambda)$ で表される．V はネットワーク内のノードの集合，E はネットワーク内のリンクの集合とする．ネットワークのノード数を $n = |V|$ とする．E に属するリンクは V に属する異なる 2 つのノードを接続している．ノード u，ノード v 間のリンクを e_{uv}，または e_{vu} と表す．deg_u をノード u に接続するリンクの数とし，u の次数と呼ぶ．G の最大次数を Δ とする．ノード u に接続するリンクと直接つながっているノードを，ノード u の隣接ノードと呼び，ノード u の隣接ノードの集合を N_u と表す．各ノードに接続するリンクはノード内でラベル関数 $\lambda_u : \{(u, v) : v \in N_u\} \to \{1, 2, \cdots, deg_u\}$ によって局所的にラベル付けされており，辺 e_{uv}，$e_{uw} (v \neq w)$ について $\lambda_u(e_{uv}) \neq \lambda_u(e_{uw})$ にが成り立つ．$\lambda_u(e_{uv})$ を，ノード u における e_{uv} のポート番号 (または単にポート) と呼ぶ．

2.2 メッセージ通信モデル

各ノードはミーリー型有限状態機械 (S, δ) で定義される．S はノードの状態集合を表し，ノードの状態はノードが保持する変数によって定義される．状態集合 S の中に初期状態が含まれる．状態遷移関数 δ は $\delta : S \times (M \cup \perp) \times P \to S \times 2^{M \times P}$ であり，ノードの状態，受信メッセージ，受信ポートから，次のノードの状態，次に送信するメッセージとその送信ポートの対の集合を決める関数である．M は可能なメッセージ全ての集合を表す．$\perp$ はメッセージ受信なしに動作することを表している．ただし，$\delta(s, \perp)$ が定義されるのは s が初期状態のときだけで，状態遷移を一度でも起こせば，状態が初期状態になることはないものとする．P はポート番号の集合を表す．各ノードの状態遷移関数は同じものとする．ノードが識別子を持つ場合も持たない場合も想定しており，識別子を持つ場合には，識別子が状態の一部に含まれている (コーディングされている) ものとする．

各ノードは 1 ステップで，以下の動作をアトミックに行う：1) メッセージの受信により，または，初期状況から動作を開始，2) 局所計算の実行，3) メッセージを送信．ネットワーク内には，少なくとも 1 つのイニシエータと呼ばれるノードが存在し，分散アルゴリズムはイニシエータから開始される．イニシエータになるノードの集合 $INIT = \{v \in V; v$ はイニシエータ $\}$ は初期状況で決定されているものとする．すなわち，イニシエータはアルゴリズム開始と同時に一斉に動作を開始する．イニシエータが最初に動作を開始する場合を除いて，各ノードはメッセージを受信した場合のみ動作する．

メッセージ通信は信頼性があるものと仮定する．すなわち，メッセージの送受信において以下が成立する．

[A1] ノード v が隣接ノード u に送信したメッセージは，必ず一度だけ u に受信される．

[A2] ノード v が隣接ノード u からメッセージを受信した場合，u はそのメッセージを送信している．

ネットワークのリンクは FIFO とする．すなわち，ノード u からノード v に送信されたメッセージは，u が送信した順に v に受信される．本稿では，非同期システムを想定する．すなわち，メッセージの送信から受信までの時間は有限であるが上界はない．

2.3 モバイルエージェントモデル

ネットワーク中にはエージェントが k 体存在し，エージェントの集合を $A = \{a_0, a_1, \cdots, a_{k-1}\}$ とする．エージェントは自身のメモリを持ち，そのメモリをノートブックと呼ぶ．このモデルではノードはレポジトリの機能しか持たず，ノードのメモリをホワイトボードと呼ぶ．

各エージェントは互いに異なる ID を持つ．この ID は $O(\log k)$ ビットで表されるものとする．また，他のエージェントの ID，ノード数 n，エージェント数 k についての情報を持たない．各エージェントは，最初にいずれかのノード $v \in V$ に存在するとする．エージェントが最初に位置しているノードを，そのエージェントのホームベースと呼ぶ．異なるエージェントのホームベースは異なるものとする．

エージェントはミーリー型有限状態機械 (S, δ) で定義される．S はエージェントの状態集合を表し，エージェントの状態

はノートブックの内容とエージェントの ID によって定義される．状態集合 S の中に各エージェントの初期状態が含まれる．(エージェントは相異なる ID を持つので，その初期状態も一般的に相異なる．)

状態遷移関数 δ は $\delta : S \times W \times P \to S \times W \times P$ であり，エージェントの状態，訪問中のノードのホワイトボードの内容，移動してきたポート番号から，次のエージェントの状態，訪問中のノードのホワイトボードの更新後の内容，次に通るポート番号を決める関数である．W はホワイトボードの可能な状態全ての集合を表す．P はポート番号の集合を表す．移動してきたポート番号は，0 の場合，ノード上で動作を開始したことを表し，$p > 0$ の場合，ポート p からノードに到着したことを表す．次に通るポート番号が 0 の場合はエージェントはノードにとどまり，$p > 0$ の場合は，ポート p を通って次のノードに移動する．状態に識別子が含まれるため，エージェントの状態空間は互いに素にできる．つまり，エージェントごとに異なる状態遷移を考えているのと同じである．

各エージェントは 1 ステップで，以下の動作をアトミックに行う：1) ノードに到着する，または，ノード上で動作を開始する (ホームベースの場合，あるいは，ノードにとどまっていた場合)，2) 局所計算を行い，自身の状態，ホワイトボードを更新する，3) 移動する場合，ノードから出ていく．

エージェントはリンクを移動中に停止故障 (消失) する場合がある．ただし，上記の 1 ステップを実行している間は故障しないものとする．エージェントの故障数の最大値は $f \leqq k-1$ であり，各エージェントは f を知っているものとする．リンクを移動している間に故障しなかったエージェントは，必ず有限時間内に目的ノードに到達する．故障したエージェントは再び動作することはなく，永久にネットワーク内から消える．

ネットワークのリンクは FIFO とする．すなわち，同じリンクを通るエージェントは，そのリンクに入った順に目的ノードに到達する．また，エージェントは非同期的に動作する．すなわち，エージェントがリンクの移動に要する時間，および，ノードにとどまっているエージェントが次の動作を開始するまでに要する時間は有限であるが上界はない．

3. 提案アルゴリズム

本節では，メッセージ通信アルゴリズムをシミュレートするモバイルエージェントアルゴリズムを提案する．ここでシミュレートするメッセージ通信アルゴリズムは，2.2 で定義されたモデル上で正しく動作するアルゴリズムであり，以降，シミュレートするメッセージ通信アルゴリズムを Z と表す．また，モバイルエージェントアルゴリズムは，2.3 で定義されたモデル上で正しく動作するアルゴリズムである．

提案アルゴリズムでは，Z がメッセージを送信する回数を有限であると仮定する．

アルゴリズムは以下の疑似コードで表される．

提案アルゴリズムにおいて，エージェントは大きく分けて，1) イニシエータの探索，2) アルゴリズム Z のシミュレート，2 つの動作を行う．

Algorithm 1 使用する関数・変数

```
1: //ノードが持つ変数
2: array [int] v.agent_transmit, v.agent_search;//エージェントの名前を記憶
3: boolean v.init;//イニシエータかどうかを判別するための変数
4: boolean v.root;//イニシエータに最初に訪問したエージェントの ID を記憶
5: queue v.send;//生成されたメッセージを記憶
6: array [int] v.receive;//各ポートから受信した最新のメッセージを記憶
7: array [int] v.parent_transmit, v.parent_search;//訪問時のポート (戻るポート) を記憶
8: array [int] v.port;//通っていない (未探索の) ポートを記憶
9: //エージェントが持つ変数
10: a.msg;//メッセージを記憶 (m, p) m:メッセージ，p:進むポート番号
11: a.tree;//伝送モードのエージェントの仮 ID
Function: Process(m, q)//イニシエータの動作及び，メッセージ受信時の動作をシミュレート
12: if (v.init = true) then
13:    v.init ← false;
14:    v.root ← true;
15:    v.tree ← a.tree;
16:    (s, M) ← (v.state_n, ∅);
17:    v.state_n ← s; M を v.send にエンキュー ;
18: if (m ≠ ∅) ∧ (m ≠ v.receive[q]) then
19:    v.receive[q] ← m;
20:    (s, M) ← (v.state_n, m);
21:    v.state_n ← s; M を v.send にエンキュー ;
```

まず，Algorithm2 について説明する．Algorithm2 によって，エージェントは 1) イニシエータの探索，を行う．具体的には，イニシエータであるノード v は，$v.init$ が真なので，$v.init$ が真であるノードを探索する．ただし，$v.init$ は，エージェントが 1 体でもシミュレーションを開始すれば偽となるので，2 体目以降に訪問したエージェントはシミュレーションを開始できない．2 体目から $f+1$ 体目に訪問したエージェントもシミュレーションを開始できるようにするために，$v.root$ を使用する．$v.init$ が偽になるときに，$v.root$ を真にすることで，イニシエータの動作は一回しか行われず，また，2 体目以降にそのノードに訪れたエージェントもシミュレーションを開始することができる．$v.init$ または，$v.root$ が真であるノードを発見すれば，$Transmit()$ を実行してシミュレーションを開始する．$Transmit()$ の詳細については後述する。

探索は深さ優先で行われる．探索の際のエージェントの経路は，$v.parent_{search}$ に記憶される．すなわち，エージェント a が初めてノード v にポート q から訪問した際は，$v.parent_{search}[a.id]$ に q が記憶される．探索を開始したノードの $v.parent_{search}[a.id]$ には 0 が記憶される．つまり，$v.parent_{search}[a.id]$ が 0 となっているノードに戻ってきたとき，そのノードのポートが全て探索されていれば，イニシエータの探索を終了する．

Algorithm 2 main

```
1: a.tree ← a.id;
2: v.port[a.id] ← {0, ··· , deg_v − 1}
3: v.parent_search[a.id] ← 0;
4: while (1)
5:   if ((v.init = true) ∨ (v.root = true)) then //訪問ノードがイニシエータまたは木の根
6:     Transmit();
7:   if (v.port[a.id] ≠ ∅) then //未探索の辺が存在すれば探索する
8:     v.port[a.id] ← v.port[a.id]/{p};
9:     ポート p を通り移動 ;
10:    ポート q から到着したとする ;
11:    if (|v.parent_search| = f + 1) then //f+1 体訪問していたとき
12:      ポート q を通り移動 (引き返す);
13:    else
14:      if (v.parent_search[a.id] ≠ ⊥) then //閉路ができたとき
15:        v.port_search ← v.port_search/{q}
16:        ポート q を通り移動 (引き返す);
17:      else //初めてノード v に初めて訪問したとき
18:        v.parent_search[a.id] ← q;
19:        v.port[a.id] ← {0, ··· , deg_v − 1}/{q}
20:  else //すべての辺が探索済みなので戻る，またはアルゴリズム終了
21:    p ← v.parent_search[a.id];
22:    if (p = 0) then
23:      break;
24:    else
25:      ポート p を通り移動 (深さ優先の戻る動作);
26: end while
```

探索時に同じノードに訪問した際は引き返す．これによって探索時にエージェントが記憶する経路の長さを最大 $n-1$ に抑えることができる．また，既に $f+1$ 体のエージェントが訪問していた場合も，同様に引き返す．

以上の動作によって，ネットワーク内のイニシエータをすべて探索できることは後で証明する．

次に Algorithm3 について説明する．Algorithm3 の $Transmit()$ によって，エージェントは 2) アルゴリズム Z のシミュレートする．$Transmit()$ はエージェントがイニシエータを発見したときに開始される．$Transmit()$ を実行しているエージェントは，各ノードで，その動作をシミュレートし，メッセージ送信を続ける．提案アルゴリズムでは，深さ優先的にメッセージ送信を行っている．すなわち，メッセージ送信先ノードにおいて，メッセージが存在する場合，及び，ノードの動作のシミュレーションの際にメッセージが生成された場合，メッセージを続けて送信し続ける．メッセージ送信が終われば，送信元ノードに引き返す．つまり，故障せずに $Transmit()$ を終了した際，エージェントは $Transmit()$ を開始したノードにいる．送信元に向かうポートを覚えておくため，イニシエータ探索時と同様に，エージェント a がノード v にポート p からメッセージを送信した際は，$v.parent_{tramsmit}[a.id]$ に p を記憶する．$v.parent_{tramsmit}[a.id]$ が 0 であるとき，v は $Transmit()$ を開始したノードである．

Algorithm 3 伝送モードの動作

```
Function: Transmit()//メッセージを深さ優先的に送信
1: Process(∅, ∅);//イニシエータが未処理ならば処理
2: a.tree ← v.tree;//自身が属する木を決定
3: v.parent_transmit[a.id] = 0;//Transmit() の開始ノードにしるしをつける
4: while (1)
5:   if ((v.send ≠ ∅) ∧ (v.tree = a.tree)) then //木の ID が同じでメッセージが存在すれば送信
6:     a.msg ← head(v.send); //メッセージを送信したエージェントの数を管理
       //送信したエージェントの数が f 未満の場合
7:     if (|v.agent_transmit| < f) then
8:       v.agent_transmit ← v.agent_transmit ∪ {a.id};
       //送信したエージェントの数が f の場合
9:     else
10:      v.agent_transmit ← ∅;
11:      dequeue(v.send);
12:    a.msg = (m, p) とする ;
13:    ポート p を通り移動 ;
14:    ポート q からノードに到着したとする ;
15:    Process(a.msg, q);
16:    if (v.parent_transmit[a.id] ≠ ⊥) ∧ ((v.tree = a.tree) ∨ (v.tree = ⊥))) then //木の ID が同じかまだ ID が付けられていない、かつ閉路ができていなければ戻らない
17:      v.parent_transmit[a.id] ← q;
18:      v.tree ← a.tree;
19:    else //違う木の ID が既に書き込まれているか、閉路ができれば戻る
20:      ポート q を通り移動 (引き返す);
21:      if (a.id ∈ v.agent_transmit) then //送信したメッセージが削除されていなければ削除
22:        dequeue(v.send);
23:        v.agent_transmit ← ∅;
24:  else //メッセージキューが空なので戻る，またはメッセージ送信を終了
25:    v.tree ← ⊥;
26:    p ← v.parent_transmit[a.id];
27:    v.parent_transmit[a.id] ← ⊥;
28:    if (p = 0) then //Transmit() を開始したノードならばメッセージ送信を終了
29:      v.root ← ⊥;
30:      a.tree ← a.id;
31:      return;
32:    else //開始ノードでなければ親ノードに移動
33:      ポート p を通り移動 (深さ優先の戻る動作);
34:      if (a.id ∈ v.agent_transmit) then
35:        dequeue(v.send);
36:        v.agent_transmit ← ∅;
37: end while
```

イニシエータを訪問した最初のエージェントは，$Transmit()$ 開始時に，自身の ID をイニシエータの $v.root$，$v.tree$ に書き込み，メッセージの送信先にも，同じ ID を書き込んでいく．イ

ニシエータを 2 番目以降に訪問したエージェントは，既に書かれている ID を自身の仮の ID として $a.tree$ に書き込み，メッセージ送信を続ける．$Transmit()$ の終了時，$a.tree$ には，再び自身の ID が書き込まれる．メッセージ送信先に既に別の ID が書き込まれている場合，及び，すでに自身が訪問していた ($v.parent_{transmit}[a.id]$ が $\perp$ ではない) 場合，エージェントは，そのノードのメッセージを送信せず，送信元ノードに引き返す．これによって，メッセージ送信の際に記憶されるエージェントの経路の長さが最大 $n-1$ に抑えられる．

メッセージは，$f+1$ 体のエージェントが送信したとき，及び，そのメッセージを送信したエージェントが戻ってきたときにキュー $v.send$ から削除される．

戻る動作によって訪問したノードに，自身の $a.tree$ と異なる ID が書き込まれていた場合，$a.tree$ が書き込まれていた時のメッセージが全て送信され，さらに別のイニシエータからやって来たエージェントが訪問している．この場合，別のイニシエータからやって来たエージェントがメッセージを送信するので，何もせずさらに戻る．

メッセージ送信の際に，未処理のイニシエータを訪問した場合は，そのイニシエータには自身の仮 ID$a.tree$ を書き込む．すなわち，そのイニシエータの $v.root$ は真にはならず，同じイニシエータからやって来た $a.tree$ を持つエージェントがそのイニシエータのメッセージを送信する．

以上がアルゴリズムの動作の概要である．

3.1 正当性の証明

提案アルゴリズムについて，以下の定理が成り立つ．

[定理 3.1] 提案アルゴリズムは，高々 f 体のエージェントが停止故障するという仮定のもとで，アルゴリズム Z をシミュレートする．

定理 3.1 を証明するために，いくつかの補題を証明する．

以降では，エージェント a が Algorithm1 の $Process()$ または Algorithm3 の $Transmit()$ を実行している場合に，a はシミュレーションモードであるという．また，a がシミュレーションモードでない場合，a は探索モードであるという．

[補題 3.1] 提案アルゴリズムによって，各ノードは少なくとも 1 体の故障しない探索モードのエージェントによって訪問される．また，全てのイニシエータはアルゴリズム Z を開始する．

証明 探索モードのエージェントは，動作を開始していないイニシエータを訪問すると，その動作を開始させる．そのため，すべてのノードに故障しない探索モードのエージェントが少なくとも 1 体訪問することを示す．

背理法を用いて証明する．提案アルゴリズム終了時に，故障しない探索モードのエージェントが，1 体も訪問していないノードが存在すると仮定する．$f \leqq k-1$ より，少なくとも 1 体のエージェントは故障せず探索を行っているので，ネットワーク内には，故障しないエージェントが訪問したノードの集合と，訪問していないノードの集合が存在することになる．ネットワーク全体は連結なので，これらのノードの集合はどこかで隣接している．隣接しているノードのうち，故障しないエージェントが訪問しているノードを v_1，故障しないエージェントが訪問していないノードを v_2 とする．ここで，提案アルゴリズムより，故障しないエージェントは，深さ優先探索でネットワーク全体を探索しようとしていて，自身が訪問済みか，すでに $f+1$ 体のエージェントが訪問しているノードに訪問した場合にのみ何もせず引き返す．よって故障しないエージェントは v_2 を訪問する前に v_1 で引き返したことになる．しかし，エージェントが引き返したということは，v_1 には $f+1$ 体のエージェントが訪問していたということなので，$f+1$ 体のうち少なくとも 1 体のエージェントが故障せず v_2 を訪問する．これは v_2 の仮定に矛盾する．従って，故障しない探索モードのエージェントが，すべてのノードに少なくとも 1 体は訪問する． ■

[補題 3.2] 提案アルゴリズムは，信頼性のあるメッセージ通信をシミュレートする．

証明 まず，全てのメッセージが目的ノードに受信されることを証明する．補題 3.1 より，すべてのイニシエータに，故障しないエージェントが少なくとも 1 体訪問する．また，提案アルゴリズムより，メッセージ送信 (アルゴリズムのシミュレーション) を始められるのは，イニシエータに訪問したエージェントだけである．また，すべてのメッセージはイニシエータから，直接的 (アルゴリズムの実行開始時)，または間接的 (メッセージ連鎖のあるメッセージ受信時) に生成されている．よって，メッセージキューが空でないノードには，イニシエータ同様に，故障しないエージェントが訪問し，メッセージ送信を行う．故障しないエージェントによって送信されたメッセージは必ず受信される．よって，全てのメッセージが目的ノードに受信される．

また，エージェントは一度の移動で 1 つのメッセージしか運ばず，示されたポートにのみにメッセージを送信するので，メッセージは目的ノードにのみ受信される．送信したメッセージが送信先ノードで既に受信済みであれば処理は行わない．よって 2.2 節の条件 [$A1$] を満たす．また，受信されたメッセージは，エージェントによって運ばれたメッセージのみである．よって条件 [$A2$] を満たす．よって信頼性のあるメッセージ通信の条件を満たす． ■

[補題 3.3] 提案アルゴリズムによるメッセージ通信は，FIFO を満たす．

証明 提案アルゴリズムにおいて，エージェントがあるノード v のメッセージキューの先頭のメッセージ m をポート p に送信するとする．このとき，m より前に v から p に送信されたメッセージ m' が存在するとする．m が v のメッセージキューの先頭だったことから，m' は v のメッセージキューから削除されている．m' が削除されるのは，m' を送信したエージェントが戻ってきたとき，または，$f+1$ 体のエージェントが m' を送信したときのみである．前者の場合，明らかに m' が先に受信されている．また，後者の場合，m' は有限時間内に必ず目的ノードに到着し，エージェントは FIFO 順でリンクを移動するため，必ず m' は m より先に受信される．よって，メッセージは送信された順に受信されるので，提案手法によるメッセージ通信は FIFO を満たす． ■

［補題 3.4］ 提案アルゴリズムによるエージェントの動作によって，各ノードの動作は正しくでシミュレートされる．

2.2 より，シミュレートするメッセージ通信モデルにおいて，各ノードは 1 ステップで，以下の動作をアトミックに行う：1) メッセージの受信により，または，初期状況から動作を開始 (*receive*)，2) 局所計算の実行 (*execute*)，3) メッセージを送信 (*send*)．このアトミックステップが連続して実行されるとき，この動作を外部から観察すると，(1) 各アトミックステップ内の *receive*，*execute*，*send* 動作の順序，(2) アトミックステップ間の *receive* 動作の順序，(3)*execute* 動作の順序，(4)*send* 動作の順序，さえ守られていれば，どのようにステップが実行されても良い．例えば，アトミックステップ AS_1，AS_2 が存在するとする．このとき，各アトミックステップは，$AS_{1-receive}$，$AS_{1-execute}$，AS_{1-send}，及び，$AS_{2-receive}$，$AS_{2-execute}$，AS_{2-send}，のステップからなる．これらは，メッセージ通信モデルにおいて実行される場合，アトミック動作の仮定より，$AS_{1-receive}$，$AS_{1-execute}$，AS_{1-send}，$AS_{2-receive}$，$AS_{2-execute}$，AS_{2-send} の順に実行される．しかし，これらは，$AS_{1-receive}$，$AS_{2-receive}$，$AS_{1-execute}$，$AS_{2-execute}$，AS_{1-send}，AS_{2-send}，という順序で実行されていても，同じ状況に到達するので，同じ動作をしていると考えてよい．

次に，提案アルゴリズムによるエージェントの動作について考える．2.3 より，各エージェントは 1 ステップで，以下の動作をアトミックに行う：1) ノードに到着する，または，ノード上で動作を開始する (*visit*)，2) 局所計算を行い，自身の状態，ホワイトボードを更新する (*execute*)，3) 移動する場合，ノードから出ていく (*leave*)．ノードにおける *receive*，*execute* 動作は，エージェントの *visit*，*execute* 動作によってまったく同様に行われる．また，メッセージ送信の役目を果たす *leave* 動作は，必ずエージェントの *execute* 動作の後に行われ，エージェントの動作は相互排他で行われるので，(1)，(2)，(3) の順序は満たされている．そして，生成されたメッセージはまずメッセージキューに格納され，生成された順にエージェントに送信されるので，(4) の順序も満たされている．よって，提案アルゴリズムのエージェントの動作によって，各ノードの動作は正しくシミュレートされる．

補題 3.1，補題 3.2，補題 3.3 より，提案アルゴリズムによって，すべてのイニシエータで処理が開始され，ノードの動作及びリンクの性質が正しくシミュレートされている．以上により定理 3.1 が成り立つ．

3.2 評　　価

本節では，提案アルゴリズムの総移動数，及びメモリについての評価を行う．提案アルゴリズムはメッセージ通信モデルのアルゴリズム Z をシミュレートするため，総移動数，メモリ量はアルゴリズム Z に依存する．アルゴリズム Z におけるノードの状態を格納するために必要なメモリ量を S，メッセージ数を M，メッセージの最大サイズを L とする．

［定理 3.2］ 提案アルゴリズムにおけるエージェントの総移動数は $O((e+M)f)$，エージェントのメモリは $O(L+\log k)$，ノードのメモリは $O(S+ML+(L+f)\Delta+M\log(n\Delta))$ となる．

証明 まず，エージェントの総移動数について評価する．探索モードでは，各リンクの各方向について，1 回進んだエージェントは 1 回戻る．各方向で高々 $f+1$ 体のエージェントが探索するので，総移動数は $2\cdot 2\cdot e\cdot(f+1)=4e(f+1)$ となる．シミュレーションモードでは，アルゴリズム Z で生成されたメッセージ 1 つにつき，高々 $f+1$ 体のエージェントが送信処理を行う．1 回の送信処理に対して 1 回戻る処理が発生するため，総移動数は $2M(f+1)$ となる．よって総移動数は $4e(f+1)+2M(f+1)=2(2e+M)(f+1)=O((e+M)f)$ となる．

次に，エージェントのメモリについて評価する．エージェントは，アルゴリズム Z によって生成されるメッセージ及び，エージェント ID を記憶する．エージェントは同時に高々 1 つのメッセージしか持たず，また，エージェント ID は $O(\log k)$ で表されることから，エージェントのメモリは $O(L+\log k)$ となる．

最後にノードのメモリについて評価する．ノードが記憶するものは，アルゴリズム Z におけるノードの状態，及び提案アルゴリズムで使用する変数である．アルゴリズム Z におけるノードの状態を保持する $v.state$ は $O(S)$ で表される．メッセージキュー $v.send$ は v が送信するすべてのメッセージを格納するため $O(ML)$ で表される．

$v.agent_{search}$ は，たかだか f 体のエージェントが ID を書き込み，エージェントの ID の書き込みには $O(\log k)$ のメモリが必要である．よって必要なメモリは $O(f\log k)$ となる．$v.agent_{transmit}$ は，最大で k 体のエージェントが ID を書き込むので，$v.agent_{search}$ と同様に考えると，必要なメモリは $O(k\log k)$ となる．$v.receive$ は，各ポートに対して最新の受信メッセージを格納するため，必要メモリは $O(L\Delta)$ となる．変数 $v.port$ は，$v.port$ に名前を書き込んだエージェント 1 体につき，ポート数分必要となる．ポートが探索済みかどうかは 1 ビットのメモリで表現できる．よって，必要なメモリは $O(f\Delta)$ となる．変数 $v.parent_{search}$ に必要な要素数は，$f+1$ となる．これは，高々 $f+1$ 体の探索モードのエージェントの深さ優先木の親に接続するポートを保持すれば十分だからである．また配列の 1 つの要素には $O(\log(n\Delta))$ 必要となる．よって必要なメモリは $O(f\log(n\Delta))$ である．変数 $v.parent_{transmit}$ に必要な要素数は，たかだか k である．また探索モードと同様に，配列の 1 つの要素には $O(\log(n\Delta))$ 必要となる．よって必要なメモリは $O(k\log(n\Delta))$ である．

すべてのメモリを加えると，

$$\begin{aligned}&S+ML+f\log n+k\log n+L\Delta+f\Delta+f\log(n\Delta)+k\log(n\Delta)\\&\quad=O(S+ML+(L+f)\Delta+(f+k)\log(n\Delta))\end{aligned}$$

よってノードに必要なメモリは $O(S+ML+(L+f)\Delta+k\log(n\Delta))$ となる． ■

4. ま と め

本稿では，k 体の非匿名エージェントを用いてメッセージ通信アルゴリズムをシミュレートするアルゴリズムを提案した．

提案手法は，最大故障数 $f \leq k-1$ が与えられた環境において，総移動数が $O((e+M)f)$ となっている．既存手法は，エージェント数 k に対し，最大 $k-1$ 体のエージェントが故障する環境において，$O((e+nM)k)$ の総移動数でシミュレーションを実現している．提案手法では，$f = k-1$ の場合でも，総移動数は $O((e+M)k)$ となり，既存手法より総移動数を削減することができている．

文　　献

[1] Barriére, L., Flocchini, P., Fraigniaud, P., Santoro, N.: Capture of an intruder by mobile agents. In: Proc. 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA' 02), pp. 324〜332 (2002)

[2] Blin, L., Fraigniaud, P., Nisse, N., Vial, S.: Distributed chasing of network intruders. In: Proc. 13th Colloquium on Structural Information and Communication Complexity (SIROCCO' 06), pp. 70〜84 (2006)

[3] Cao, J., Das, S. K.: Mobile Agents in Networking and Distributed Computing. John Wiley & Sons (2012)

[4] Chalopin, J., Godard, E., M ´ etivier, Y., Ossamy, R.: Mobile agents algorithms versus message passing algorithms. In: Proc. 10th Int. Conf. on Principles of Distributed Systems (OPODIS' 06), pp. 187〜201 (2006)

[5] Das, S., Flocchini, D., Santoro, N., Yamashita, M.: Fault-Tolerant Simulation of Message-Passing Algorithms by Mobile Agents. In: Proc. Colloquium on Structural Information and Communication Complexity (SIROCCO' 07), pp. 289〜303 (2007)

[6] Das, S., Mihalak, M., Sramek, R., Vicari, E., Widmayer, P.: Rendezvous of mobile agents when tokens fail anytime. In: Proc. 10th Int. Conf. on Principles of Distributed Systems (OPODIS' 08), pp. 463〜480 (2008)

[7] Dieudonneé, Y., Pelc, A.: Deterministic network exploration by a single agent with Byzantine tokens. Information Processing Letters 112, pp. 467〜470 (2012)

[8] Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Finding a black hole in an arbitrary network: optimal mobile agents protocols. Distributed Computing, to appear. Preliminary version. In: Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC' 02), pp. 153〜162 (2002)

[9] Erciyes, K.: Distributed Graph Algorithms for Computer Networks. Springer Science & Business Media (2013)

[10] Flocchini, P., Luccio, F.L., Huang, M.: Decontamination of chordal rings and tori using mobile agents. International Journal of Foundation of Computer Science 18, No. 03, pp. 547〜563 (2007)

[11] Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Hardness and approximation results for black hole search in arbitrary networks. Theoretical Computer Science 384, pp. 201〜221 (2007)

[12] Luccio, F., Pagli, L., Santoro, N.: Network decontamination in presence of local immunity. International Journal of Foundation of Computer Science 18, No. 03 pp. 457〜474 (2007)

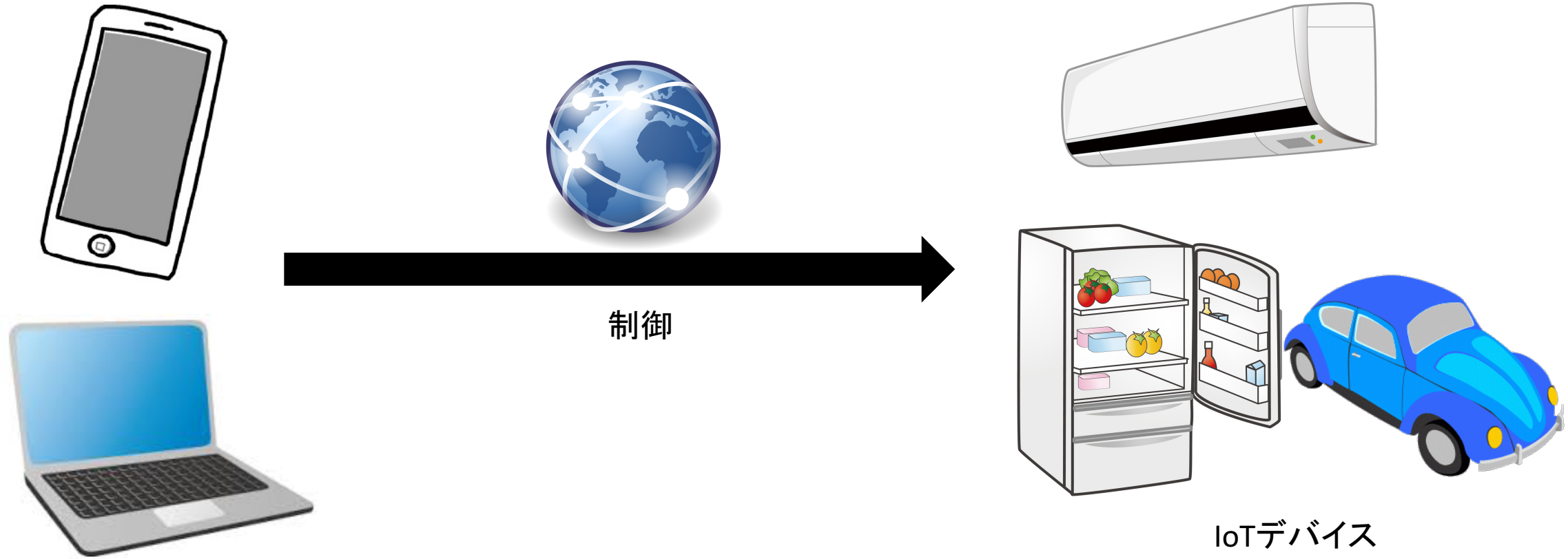
[13] Suzuki, T., Izumi, T., Ooshita, F., Kakugawa, H., Masuzawa, T.: Move-optimal gossiping among mobile agents. Theoretical Computer Science 393, pp. 90〜101 (2008)

IoTのための耐侵入ミドルウェア開発

木下　崇央
東京工業大学

IoTとは

電気機器や電化製品をインターネットを通じて制御



もし、システムに侵入されたら……

システムに侵入

インフラの破壊、乗っ取り

IoTの主なセキュリティー対策①

IoTデバイスにおけるファイヤーウォール、認証

権限のないユーザがIoTデバイスにアクセスするのをブロックする

→権限を持っているように偽装して侵入される恐れがある

IoTの主なセキュリティー対策②

侵入検知(Intrusion detection)

-シグニチャ型

-異常検出型

シグニチャ型→新しい特徴を持つ攻撃に対応できない

異常検出型→精度が高くない

目的

耐侵入性に優れたミドルウェアの作成

耐侵入性（intrution tolerance）

悪意のある第三者によってシステムに侵入された際、**システムが破壊される、勝手に操作される**のを**防ぐ**

方向性

一貫性をもとにした合意を形成する

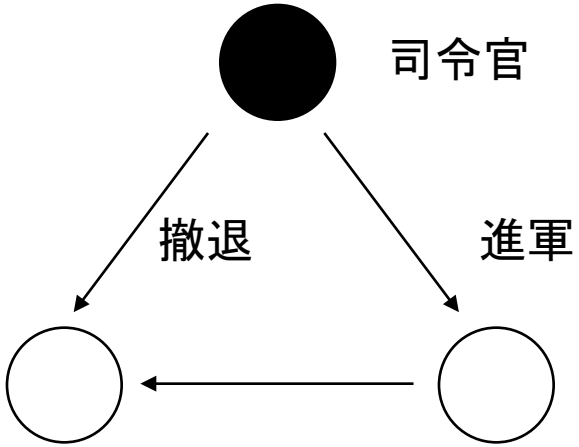
<一貫性>
-ビザンチン合意問題
-ビザンチン・フォールト・トレランス

ビザンチン合意問題

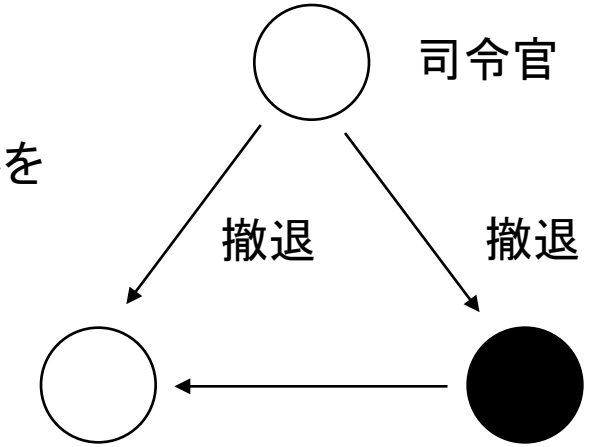
f:欠陥のあるプロセス数　　n:全体のプロセス数

⇒3f+1 <= nでなければ正しい合意に至ることができない

(例)f=1、n=3の場合

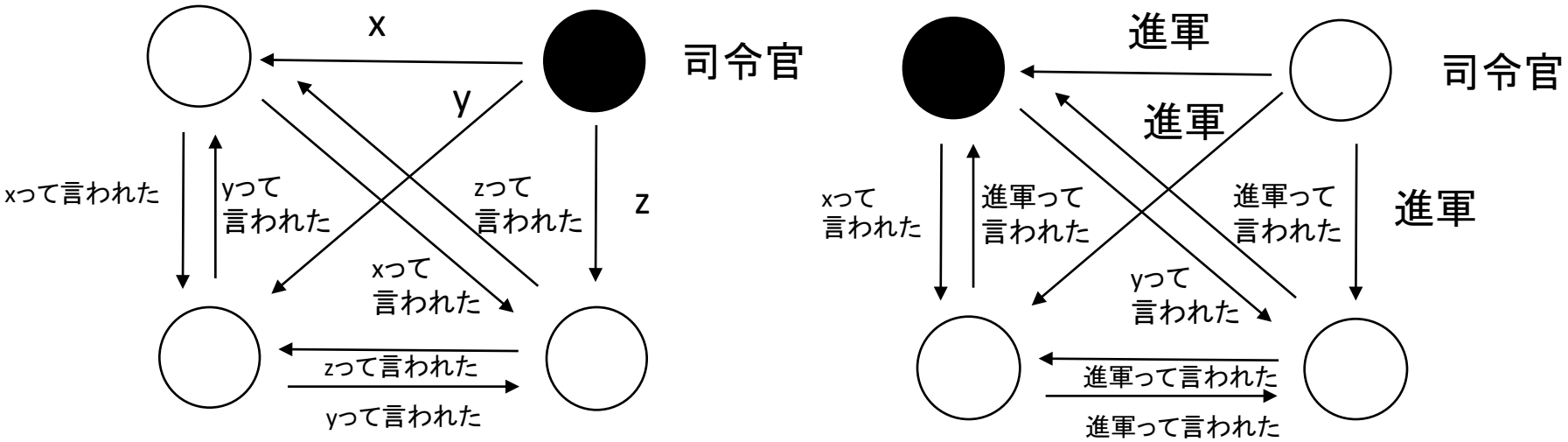


どっちが裏切り者かを
区別できない



ビザンチン合意問題

(例)f=1、n=4の場合



x,y,z ∈ {撤退,進軍}

プロセスは多数派の意見を採用するため、
x,y,zにどのような値が送られても裏切り者でない
3つのプロセスは合意に至ることができる。

ビザンチン・フォールト・トレランス

ビザンチン合意問題　→　同期通信のみ

ビザンチン・フォールト・トレランス　→　非同期通信でも有効

手順

学部

IoTのテストベッドの作成

大学院

IoTテストベッドで生じた問題を改善した耐侵入ミドルウェアの作成

まとめ

-IoTセキュリティ対策としてファイヤウォールや侵入検知だけでは不十分

-性能が低いIoTデバイスにおいて侵入に強いミドルウェアを作成、侵入検知の前にシステムが破壊されるのを防ぐ

参考

[1] CSA :
Security Guidance for Early Adopters of the Internet of things(IoT) (2015)

[2] Leslie Lamport, Robert E. Shostak, Marshall C. Pease:
The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst. 4(3): 382-401 (1982)

[3] Miguel Castro, Barbara Liskov:
Practical Byzantine Fault Tolerance. OSDI 1999: 173-186

[4] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, Edmund L. Wong:
Zyzzyva: Speculative Byzantine fault tolerance. ACM Trans. Comput. Syst. 27(4) (2009)

On the Maximum Weight Minimal Separator

Tesshu Hanaka[1], Hans L. Bodlaender[2], Tom C. van der Zanden[2], and Hirotaka Ono[1]

1 **Department of Economic Engineering, Kyushu University, 6-19-1, Hakozaki, Higashi-ku, Fukuoka, 812-8581, Japan**
3EC15004S@s.kyushu-u.ac.jp, hirotaka@econ.kyushu-u.ac.jp
2 **Department of Computer Science, Utrecht University, Padualaan 14, De Uithof PO Box 80089 3508 TB Utrecht, The Netherlands**
{h.l.bodlaender,t.c.vanderzanden}@uu.nl

Abstract

Given an undirected and connected graph $G = (V, E)$ and two vertices $s, t \in V$, a vertex subset S that separates s and t is called s-t separator, and an s-t separator is called minimal if no proper set of S separates s and t. In this paper, we consider finding a minimal s-t separator with maximum weight on a vertex-weighted graph. We first prove this problem is NP-hard. Then, we propose an $\mathbf{tw}^{O(\mathbf{tw})}n$-time deterministic algorithm based on tree decomposition. Moreover, we also propose an $O^*(9^{\mathbf{tw}} \cdot W^2)$-time randomized algorithm to determine whether there exists a minimal s-t separator where W is its weight and $\mathbf{tw}$ is treewidth of G.

Keywords and phrases parameterized algorithm, minimal separator, treewidth

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Given a connected graph $G = (V, E)$ and two vertices $s, t \in V$, a set $S \subseteq V$ of vertices is called an *s-t separator* if s and t belong to different connected components in $G \setminus S$, where $G \setminus S = (V \setminus S, E)$. If a set S is an s-t separator for some s and t, it is simply called a *separator*. If an s-t separator S is minimal in terms of set inclusion, that is, no proper subset of S separates s and t, it is called a *minimal s-t separator*. Similarly, if a separator is minimal in terms of set inclusion, it is called a *minimal separator*.

Separators and minimal separators have been considered important in several contexts and have been intensively studied indeed. For example, they are obviously related to the connectivity of graphs, which is an important notion in many practical applications, such as network design, supply chain analysis and so on. From a theoretical point of view, minimal separators are related to treewidth or potential maximal cliques, which play key roles in designing fast algorithms [4, 6].

In this paper, we consider the problem of finding the most important minimal separator of a given weighted graph. More precisely, the problem is defined as follows: Given a connected graph $G = (V, E)$, vertices $s, t \in V$ and a weight function $w : V \to \mathbb{N}^+$, find a minimal s-t separator whose weight $\sum_{v \in S} w(v)$ is maximum. The decision version of the problem is to decide the existence of minimal s-t separator with weight W. We name the problems Maximum Weight Minimal s-t Separator.

This problem is motivated in the context of supply chain network analysis. When a weighted network represents a supply chain where a vertex represents an industry, s and t are virtual vertices respectively represents source and sink, and its weight of a vertex represents its financial importance, the maximum weight minimal s-t separator is interpreted as the set of industries that is most significant but vulnerable in the supply chain network.

Unfortunately, the problem is shown to be NP-hard, and we then design an FPT algorithm with respect to treewidth. It should be noted that since the condition of s-t connectivity can be written with Monadic Second Order Logic, it can be solved in $f(\mathbf{tw}) \cdot n$ time by Courcelle's meta-theorem, where f is a computable function and $\mathbf{tw}$ is treewidth of the graph. However, the function f forms a tower of exponentials; the existence of an FPT algorithm with better running time is not obvious.

In this paper, we propose two parameterized algorithm with respect to treewidth. One is a $\mathbf{2}^{O(\mathbf{tw} \log \mathbf{tw})}n$-time deterministic algorithm and the other is an $O^*(c^{\mathbf{tw}} \cdot W^2)$-time randomized algorithm for the decision version, where c is a constant and O^* is the order notation omitting the polynomial factor. The former algorithm is based on a standard dynamic programming approach, whereas the latter utilizes two techniques recently developed. The first technique is called *Cut & Count*, and by using this, the running time is bounded by a single exponential of treewidth. Furthermore, by applying the second technique called fast convolution, we improve the running time by reducing the base of the exponent from $c = 21$ to $c = 9$; the total running time of the resulting algorithm is $O^*(9^{\mathbf{tw}} \cdot W^2)$, which could be much faster if the graph is unweighted.

1.1 Related work

1.1.1 The number of minimal separators

Minimal separators have been investigated long time from many aspects. As mentioned above, they are related to treewidth or potential maximal cliques, for example [4, 6]. In general, a graph has exponentially many minimal separators, and in fact there exists a graph with $\Omega(3^{n/3})$- minimal separators [9]. Recently, it was improved on $\omega(1.4521^n)$ [11]. On the other hand, some graph classes have only polynomially (even linearly) many minimal separators. For example, Bouchitté showed weakly triangulated(chordal) graph that is superclass of them also has polynomial number of separators [5]. As other graph class with polynomial minimal separators, there are circular and circular arc graphs [20, 14], polygon circle graph which is superclass of circle graph [19]. For such a class of graphs, Maximum Weight Minimal s-t Separator can be solved in polynomial time, because there exists an $O(n^3 \cdot R_{sep})$-time algorithm enumerating all the minimal separators where R_{sep} is the number of them and we just evaluate weights of them [2].

1.1.2 The relationship between minimal separators and treewidth

Minimal separators and treewidth have very important relationship between each other. As for the number of minimal separators, if a graph has only polynomial minimal separators, we can compute treewidth in polynomial time [5, 6]. Such graph classes include circular-arc ($O(n^2)$ [13, 14]), polygon circle ($O(n^2)$ [19]), weakly triangulated ($O(n^2)$ [5]) and so on. On the other hand, computing treewidth is fixed parameter tractable with respect to maximum size of minimal separators [18]. It corresponds to a solution size of Maximum Weight Minimal s-t Separator on unweighted graphs. In the sense, we show that Maximum Weight Minimal s-t Separator on unweighted graphs is fixed parameter tractable with respect to treewidth in this paper, conversely.

The remainder of the paper is organized as follows. In Section 2, we first give basic terminology and basic notions of designing algorithms. In section 3, after mentioning that our problem is NP-hard, and we then design a standard dynamic programming algorithm. In Section 4, we propose randomized algorithms based on *Cut & Count* technique.

2 Preliminaries

In this section, we give notations, definitions, and some basic concepts. Let $G = (V, E)$ be an undirected and vertex-weighted graph. For $V' \subseteq V$, let $G[V']$ denote a subgraph of G induced by V'. Furthermore, we denote the set of neighbors of v by $N(v)$ for a vertex v. We define the function $[p]$ such that if p is true, it returns 1, otherwise 0.

2.1 Tree Decomposition

Our algorithms proposed in Sections 4 and 5 are based on dynamic programming on tree decomposition. In this subsection, we give the definition of tree decomposition.

▶ **Definition 1.** A *tree decomposition* of a 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 $\forall \{u, v\} \in E$, there exists 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 term "nodes" (not "vertices") for T to avoid a 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 sometimes use the notation $\mathbf{tw}$ instead of $\mathbf{tw}(G)$ for simplicity.

In general, computing $\mathbf{tw}(G)$ of a given G is NP-hard [1], but fixed-parameter tractable with respect to itself and there exists a linear time algorithm if treewidth is fixed [3]. In the following, we assume that a decomposition tree with the minimum treewidth is given.

Kloks introduced a very useful tree decomposition for some algorithms, called *nice tree decomposition*[13]. In the sense, it is a special binary tree decomposition which has four types of nodes, named *leaf* , *introduce vertex* , *forget* and *join*. Moreover, Cygan et al. added a new type of node named *introduce edge* [7].

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

1. T is rooted at a designated node $X_r \in \mathcal{X}$, called root node.
2. Every node of the tree T has at most two children nodes.
3. The nodes of T hold one of the following five node types:
   - A *leaf* node i which has no children and whose bag X_i satisfies $|X_i| = 0$.
   - An *introduce vertex* node i which has one child j with $X_i = X_j \cup \{v\}$ for a vertex $v \in V$.
   - An *introduce edge* node i which 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 which has one child j and satisfies $X_i = X_j \setminus \{v\}$ for a vertex $v \in V$.
   - A *join* node i which 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 in polynomial time [8]. Here, given a tree decomposition $\langle \mathcal{X}, T \rangle$, we define a subgraph $G_t = (V_t, E_t)$ for each node t where V_t is the set of vertices added in each introduce vertex node and E_t is the set of edges added in each introduce edge node until node t on a decomposition tree.

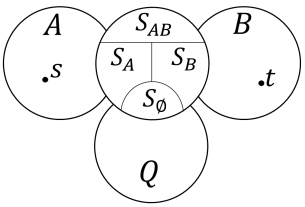


Figure 1 Connection between vertex sets

3 A standard dynamic programming algorithm based on tree decomposition

In this section, we propose an FPT algorithm that solves Maximum Weight Minimal s-t Separator. Before explaining the algorithm, we first mention that Maximum Weight Minimal s-t Separator is NP-hard even for unweighted graphs. The omitted proof is shown in Appendix.

▶ **Theorem 3.** Maximum Weight Minimal s-t Separator *is NP-hard even if all the vertex weights are identical.*

From here, we show how we design an FPT algorithm with respect to treewidth. It is a standard dynamic programming algorithm based on tree decomposition, and the running time is $\mathbf{tw}^{O(\mathbf{tw})}$-time. The running time $\mathbf{tw}^{O(\mathbf{tw})}$ appears in some connectivity problems, for example Steiner Tree, Feedback Vertex Set and Connected Vertex Cover [7, 8].

We first show that Maximum Weight Minimal s-t Separator is considered a connectivity problem. To show this, we define partitioned solution for Maximum Weight Minimal s-t Separator.

▶ **Definition 4.** A *partitioned solution* of weight W is a partition (S, A, B, Q) of V such that: (1) $s \in A, t \in B$, (2) $G[A]$ is connected, (3) $G[B]$ is connected, (4) $\sum_{v \in S} w(v) = W$, (5) for $\forall v \in S$, there exist vertices $a \in A, b \in B$ such that $(a, v) \in E, (v, b) \in E$ and (6) for sets A, B, Q, there does not exist an edge (u, v) such that u and v are in different sets.

If a partitioned solution is equivalent to a solution of Maximum Weight Minimal s-t Separator, then we can consider it as a kind of connectivity problem. In fact, we have the following theorem.

▶ **Theorem 5.** *There exists a minimal s-t separator of weight W if and only if there exists a partitioned solution (S, A, B, Q) of weight W.*

Using partitioned solutions, we design an $\mathbf{tw}^{O(\mathbf{tw})}$-time algorithm for Maximum Weight Minimal s-t Separator. First, we partition S into $S_\emptyset, S_A, S_B$ and S_{AB}(See Figure 1). They are needed for updating process on the dynamic programming. Set $S_\emptyset$ is the one of vertices in S that have no neighbor of A and B, but may have neighbors in S_A, S_B, S_{AB}, Q. Set S_A (resp., S_B) is the one of vertices in S that has at least one neighbor of A (resp., B), but no neighbor of B (resp., A). They may have neighbors of S_A, S_B, S_{AB}, Q. Set S_{AB} is the one of vertices in S that have neighbors of both A and B and may have neighbors of S_A, S_B, S_{AB}, Q. From these sets, we define *partial solution* as follows.

▶ **Definition 6.** Given a node t of the tree decomposition of G, a *partial solution* for node t is a partition $(S_\emptyset, S_A, S_B, S_{AB}, A, B, Q)$, such that:

- $V_t = S_\emptyset \cup S_A \cup S_A \cup S_{AB} \cup A \cup B \cup Q$,
- $\forall v \in S_\emptyset$, $N(v) \cap (A \cup B) = \emptyset$,
- $\forall v \in S_A$, $N(v) \cap B = \emptyset$ and $N(v) \cap A \neq \emptyset$,
- $\forall v \in S_B$, $N(v) \cap B \neq \emptyset$ and $N(v) \cap A = \emptyset$ and
- $\forall v \in S_{AB}$, $N(v) \cap B \neq \emptyset$ and $N(v) \cap A \neq \emptyset$
- $s \in V_t \Rightarrow s \in A$
- $t \in V_t \Rightarrow t \in B$.

We design DP tables in each node. For a representation of the state of v, we define two functions: the *coloring* function $c : V \to \{s_\emptyset, s_A, s_B, s_{AB}, a, b, q\}$ and the *group coloring* $g : V \to \{1, 2, \ldots, \mathbf{tw}\}$. Each element of $\{s_\emptyset, s_A, s_B, s_{AB}, a, b, q\}$ and $\{1, 2, \ldots, \mathbf{tw}\}$ is called *state*. For colorings c, g, we denote the state of coloring of v by $c(v)$ and the state of group coloring by $g(v)$. The states of a coloring c represent which set a vertex is in, for example, v is in $S_\emptyset$ if $c(v) = s_\emptyset$. The group coloring is needed to confirm connectivity of A and B. We consider all partitions of vertices in a bag and there are $|X_i|^{|X_i|}$ partitions. The group coloring indicates a connected component where each vertex exists on updating process. Intuitively, connected components of $G[A]$(resp., $G[B]$) are merged on updating process and all vertices in A(resp., B) form one connected component in a root node if this partition is a partitioned solution. Note that we only use $|X_i|$ states of group coloring for each vertex in X_i since we only check partitions of vertices in X_i. Therefore, the size of DP table is $(7|X_i|)^{|X_i|} \times |X_i|$ in each node.

Suppose that introduce vertex nodes, introduce edge nodes and forget nodes have one child node j respectively, and join nodes have two children nodes j_1, j_2. Let c_i be the coloring in node i and g_i be the group coloring in node i. Moreover, let $c_i(v)$ be the state of coloring of v and $g_i(v)$ be the state of group coloring. Here, we transform a nice tree decomposition by adding $\{s, t\}$ to all bags and suppose that the root bag X_r contains only two vertices s, t. The width of this tree decomposition is at most $\mathbf{tw} + 2$. We can transform any tree decomposition into such a tree decomposition in polynomial time. Then we define the value function $f_i : \{s_\emptyset, s_A, s_B, s_{AB}, a, b, q\}^{|V_i|} \times \{1, 2, \ldots, |X_i|\}^{|V_i|} \to \mathbb{N} \cup \{-\infty\}$ for a node i, a coloring c and a group coloring g as follows:

$$f_i(c, g) := \sum_{v \in (S_\emptyset \cup S_A \cup S_B \cup S_{AB}) \cap V_i} w(v).$$

This represents the weight of a partial solution.

We then define recursive formulas for each node. For a vertex v, we sometimes denote $f_i(c \times \{c(v)\}, g \times \{g(v)\})$ to emphasize the state of v. In a root node, $\max_{g_r} f_r(\{a\} \times \{b\}, g_r)$ is an optimal value because $X_r = \{s, t\}$. Let an index i be a parent node's index and an index j be a child node's index.

Leaf node:

In leaf nodes, we define $f_i(\{a\} \times \{b\}, g_i) := 0$, otherwise $f_i(c_i, g_i) := -\infty$ since there are only two vertices s, t in X_i.

Introduce vertex v node:

In introduce vertex nodes, we consider three cases. If $c(v) = s_\emptyset$, we add $w(v)$ to $f_j(c, g)$ because v is added in S. If $c(v) \in \{a, b, q\}$, the value of f_i does not change since $v \notin S$.

Finally, if $c_i(v) \in \{s_A, s_B, s_{AB}\}$, a partial solution is invalid by the definition because v has no edge and hence no neighbor in A and B. Therefore, we define f_i as follows:

$$f_i(c \times \{c_i(v)\}, g \times \{g_i(v)\}) := \begin{cases} f_j(c,g) + w(v) & if\ c_i(v) = s_\emptyset \\ f_j(c,g) & if\ c_i(v) = a, b, q \\ -\infty & if\ c_i(v) = s_A, s_B, s_{AB}. \end{cases}$$

Introduce edge (u, v) node:

In introduce edge nodes, we define f_i for some cases of colorings. Here, we denote the state of v in a node i by $(c_i(v), g_i(v))$.

- Let $g' \in g_i$ be a state of the group coloring. If $(c_i(u), g_i(u)) = (a, g')$ and $(c_i(v), g_i(v)) = (a, g')$, or $((c_i(u), g_i(u)) = (b, g')$ and $(c_i(v), g_i(v)) = (b, g'))$, u, v is in A(resp., B) and the same connected component in $G[A]$(resp., $G[B]$). Because a connected component containing u and another one containing v are connected each other by adding edge (u, v), we choose maximum $f_j(c_j, g_j)$ such that after two connected component are merged by adding (u, v), g_j correspond with g_i. Therefore, if there is no grouping color satisfying these conditions, we define $f_i(c, g_i) := -\infty$. Otherwise, let D be the set of grouping colorings satisfying these conditions. Thus, we define f_i as follows:

  $$f_i(c, g_i) := \max_{g_y^* \in D} f_j(c, g_y^*).$$

- If $(c_i(u), g_i(u)) = (s_\emptyset, *)$, $(s_A, *)$, $(s_B, *)$, $(s_{AB}, *)$, $(q, *)$ and $(c_i(v), g_i(v)) = (s_\emptyset, *)$, $(s_A, *)$, $(s_B, *)$, $(s_{AB}, *)$, $(q, *)$ where $*$ means arbitrary group state, the value $f_i(c, g)$ equals to $f_j(c, g)$ for all combinations of $(c_i(u), g_i(u))$ and $(c_i(v), g_i(v))$, because this case is indifferent from A, B and connected components in A and B are not merged. Hence, we define f_i as follows:

  $$f_i(c, g) := f_j(c, g).$$

- If $(c_i(u), g_i(u)) = (s_A, *)$ and $(c_i(v), g_i(v)) = (a, *)$, or $(c_i(u), g_i(u)) = (a, *)$ and $(c_i(v), g_i(v)) = (s_A, *)$, we consider two cases. One case is that $u \in S_A$ and $v \in A$ in a children node and another case is $u \in S_\emptyset$ and $v \in A$ in a children node. In the latter case, u is moved from $S_\emptyset$ into S_A by adding (u, v), because u has a neighbor v in A. Thus, we define f_i as follows:

  $$f_i(c \times \{s_A\} \times \{a\}, g) := \max\{f_j(c \times \{s_A\} \times \{a\}, g), f_j(c \times \{s_\emptyset\} \times \{a\}, g)\}.$$

- If $(c_i(u), g_i(u)) = (s_A, *)$ and $(c_i(v), g_i(v)) = (a, *)$, or $(c_i(u), g_i(u)) = (a, *)$ and $(c_i(v), g_i(v)) = (s_A, *)$, we consider almost the same cases as above ones, that is, $u \in S_B$, $v \in B$ and $u \in S_\emptyset$ and $v \in B$ in a children node.

  $$f_i(c \times \{s_B\} \times \{b\}, g) := \max\{f_j(c \times \{s_B\} \times \{b\}, g), f_j(c \times \{s_\emptyset\} \times \{b\}, g)\}.$$

- If $(c_i(u), g_i(u)) = (s_{AB}, *)$ and $(c_i(v), g_i(v)) = (a, *)$, or $(c_i(u), g_i(u)) = (a, *)$ and $(c_i(v), g_i(v)) = (s_{AB}, *)$, there are two cases that $u \in S_{AB}$ and $v \in A$ in a children node and that $u \in S_B$ and $v \in A$ in a children node. In the latter case, u is moved from S_B to S_{AB} by adding (u, v), because u has a neighbor v in B. Therefore, we define f_i as follows:

  $$f_i(c \times \{s_{AB}\} \times \{a\}, g) := \max\{f_j(c \times \{s_{AB}\} \times \{a\}, g), f_j(c \times \{s_B\} \times \{b\}, g)\}.$$

- If $(c_i(u), g_i(u)) = (s_{AB}, *)$ and $(c_i(v), g_i(v)) = (b, *)$, or $(c_i(u), g_i(u)) = (b, *)$ and $(c_i(v), g_i(v)) = (s_{AB}, *)$, we consider almost the same cases as above ones, that is, $u \in S_{AB}$, $v \in B$ and $u \in S_A$ and $v \in B$ in a children node.

$$f_i(c \times \{s_{AB}\} \times \{b\}, g) := \max\{f_j(c \times \{s_{AB}\} \times \{b\}, g), f_j(c \times \{s_A\} \times \{b\}, g)\}.$$

- Otherwise, we define $f_i(c, g) := -\infty$ because the rest of cases is invalid. Recall the definition of partitioned solution and the meaning of states.

Forget v node:

In a forget v node, if $c_j(v) = a$, there exists at least one vertex u in A such that $g_j(u) = g_j(v)$. Otherwise, a connected component in $G[A]$ containing u would not connect to other connected components in $G[A]$. Thus this is not a partitioned solution. The case that $c_j(v) = b$ is almost the same. We define the subset of tuples of (c_j, g_j) that satisfies these and denote it by D. We then define f_i as follows:

$$f_i(c, g) := \begin{cases} \max_{(c \times c_j(v), g \times g_j(v)) \in D} f_j(c \times c_j(v), g \times g_j(v)) & if\ D \neq \emptyset \\ -\infty & if\ D = \emptyset \end{cases}$$

Join node:

For i, j_1, j_2, we denote each coloring by c_i, c_{j_1}, c_{j_2}. We then define the subset D of tuples of $((c_{j_1}, g_{j_1}), (c_{j_2}, g_{j_2}))$ as the combinations of colorings for c_{j_1}, c_{j_2} such that (See Table 1):

- $\forall v \in c_i^{-1}(\{s_\emptyset, a_l, a_r, b_l, b_r, q\})$, $(c_{j_1}(v), c_{j_2}(v)) = (c_i(v), c_i(v))$,
- $\forall v \in c_i^{-1}(\{s_A\})$, $(c_{j_1}(v), c_{j_2}(v)) = (s_A, s_\emptyset), (s_\emptyset, s_A), (s_A, s_A)$,
- $\forall v \in c_i^{-1}(\{s_B\})$, $c_{j_1}(v), c_{j_2}(v)) = (s_B, s_\emptyset), (s_\emptyset, s_B), (s_B, s_B)$, and
- $\forall v \in c_i^{-1}(\{s_{AB}\})$, $(c_{j_1}(v), c_{j_2}(v)) = (s_{AB}, s_\emptyset), (s_{AB}, s_A), (s_{AB}, s_B), (s_{AB}, s_{AB}), (s_\emptyset, s_{AB})$, (s_A, s_{AB}), (s_B, s_{AB}), (s_A, s_B), (s_B, s_A) and
- two grouping colorings g_{j_1}, g_{j_2} correspond with g_i after they are merged.

If $c_i(v) \in \{s_\emptyset, a, b, q\}$, $c_{j_1}(v) = c_i(v)$ and $c_{j_2}(v) = c_i(v)$. If $c_i(v) = s_A$, there are three combinations such that $(c_{j_1}(v), c_{j_2}(v)) = (s_A, a)$, $(c_{j_1}(v), c_{j_2}(v)) = (a, s_A)$ and $(c_{j_1}(v), c_{j_2}(v)) = (s_A, s_A)$. For g_i, g_{j_1}, g_{j_2} correspond to g_i after they are merged in a join node. Let S^* be the vertex subset $c_i^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\})$. We then define f_i as follows:

$$f_i(c_i, g_i) := \begin{cases} \max_{((c^*_{j_1}, g^*_{j_1}), (c^*_{j_2}, g^*_{j_2})) \in D} \{f_{j_1}(c^*_{j_1}, g^*_{j_1}) + f_{j_2}(c^*_{j_2}, g^*_{j_2}) - w(S^*)\} & if\ D \neq \emptyset \\ -\infty & if\ D = \emptyset \end{cases}.$$

We recursively calculate f_i on decomposition tree. Note that all bags have $|X_i|$'s vertices and the number of colorings (c, g) in each node is $7^{|X_i|}|X_i|^{|X_i|} = \mathbf{tw}^{O(\mathbf{tw})}$. The running time to compute all f_i's in X_i is dominated by join nodes and it is roughly $(\mathbf{tw}^{O(\mathbf{tw})})^3 = \mathbf{tw}^{O(\mathbf{tw})}$ since we scan every coloring in two children nodes X_{j_1} and X_{j_1} for each coloring c_i. Therefore, total running time is $\mathbf{tw}^{O(\mathbf{tw})}n$ and we conclude with the following theorem.

▶ **Theorem 7.** *For graphs of treewidth at most* $\mathbf{tw}$*, there exists an algorithm that solves* Maximum Weight Minimal s-t Separator *in time* $\mathbf{tw}^{O(\mathbf{tw})}n$.

	$s_\emptyset$	s_A	s_B	s_{AB}	a	b	q
$s_\emptyset$	$s_\emptyset$	s_A	s_B	s_{AB}			
s_A	s_A	s_A	s_{AB}	s_{AB}			
s_B	s_B	s_{AB}	s_B	s_{AB}			
s_{AB}	s_{AB}	s_{AB}	s_{AB}	s_{AB}			
a					a		
b						b	
q							q

Table 1 This table represents combinations of states of two children nodes j_1, j_2 for each vertex in $X_i = X_{j_1} = X_{j_2}$. The row and column correspond to states of j_1, j_2 respectively and inner elements correspond to states of x. For example, if $c_i(v) = s_A$, there are three combinations such that $(c_{j_1}(v), c_{j_2}(v)) = (s_A, a)$, $(c_{j_1}(v), c_{j_2}(v)) = (a, s_A)$ and $(c_{j_1}(v), c_{j_2}(v)) = (s_A, s_A)$.

4 Algorithms using *Cut & Count*

In this section, we give an algorithm that solves decision version Maximum Weight Minimal s-t Separator to decide the existence of minimal s-t separator with weight W in time $O^*(9^{\mathbf{tw}} \cdot W^2)$ for graphs of treewidth at most **tw**. This algorithm is based on the *Cut & Count* technique.

4.1 Isolation Lemma

In this subsection, we explain the Isolation lemma introduced by Mulmuley et al.[16]. On the *Cut & Count* technique, it is used for obtaining a single solution with high probability. Therefore, the Isolation lemma allows us to count objects modulo 2.

▶ **Definition 8** ([16]). A function w' : $U \to \mathbb{Z}$ *isolates* a set family $\mathcal{F} \subseteq 2^U$ if there is a unique $S' \in \mathcal{F}$ with $w'(S') = \min_{S \in \mathcal{F}} w'(S)$ where $w'(X) = \sum_{u \in X} w'(u)$.

▶ **Lemma 9** (Isolation Lemma[16]). *Let $F \subseteq 2^U$ be a set family over a universe U with $|F| > 0$. For each $u \in U$, choose a weight $w'(u) \in \{1, 2, \dots N\}$ uniformly and independently at random. Then*

$$Pr[w' \text{ isolate } \mathcal{F}] \geq 1 - \frac{|U|}{N}.$$

4.2 Cut & Count

The *Cut & Count* technique was introduced by Cygan et al. for solving connectivity problems[7]. The concept of *Cut & Count* is counting the number of relaxed solutions such that we do not consider whether they are connected or disconnected. Then we compute the number of relaxed solutions modulo 2 and we determine whether there exists a connected solution by cancellation tricks. Now, we define a *consistent cut* to explain the detail of *Cut & Count*.

First, we explain a *consistent cut*.

▶ **Definition 10** ([7]). A cut (V_1, V_2) of $V' \subseteq V$ such that $V_1 \cup V_2 = V'$ and $V_1 \cap V_2 = \emptyset$ is *consistent* if $v_1 \in V_1$ and $v_2 \in V_2$ implies $(v_1, v_2) \notin E$.

This means that consistent cut (V_1, V_2) of V' has no edge between V_1 and V_2. We fix an arbitrary vertex v in V_1. Then, if $G[V]$ is connected, then there only exists one consistent

cut, that is, $(V_1, V_2) = (V, \emptyset)$. Therefore, the number of consistent cuts is odd. By this fact, we only compute the number of consistent cuts modulo 2 on decomposition tree and return yes if the number of consistent cuts is odd, otherwise no in a root node. The Isolation lemma is useful for us to consider only an unique solution, and hence we can use the modulo 2 trick.

Let $\mathcal{S} \subseteq 2^U$ be a set of solutions. According to [7, 8], the *Cut & Count* is divided into two parts as follows.

- **The Cut part** : Relax the connectivity requirement by considering the set $\mathcal{R} \supseteq \mathcal{S}$ of possibly connected or disconnected candidate solutions. Moreover, consider the set $\mathcal{C}$ of pairs $(X; C)$ where$X \in \mathcal{R}$ and C is a consistent cut of X.
- **The Count part** : Isolate a single solution by sampling weights of all elements in U with high probability by the isolation lemma. Then, compute $|\mathcal{C}|$ modulo 2 using a sub-procedure. Disconnected candidate solutions $X \in \mathcal{R} \setminus \mathcal{S}$ cancel since they are consistent with an even number of cuts. If the only connected candidate $x \in \mathcal{S}$ exists, we obtain the odd number of cuts .

Given a set U and a tree decomposition $\langle \mathcal{X}, T \rangle$, the general scheme of *Cut & Count* is as follows:

Step 1. Set the integer weight for every vertex uniformly and independently at random by $w' : U \to \{1, \ldots, 2|U|\}$.

Step 2. For each integer weight $0 \leq W' \leq 2|U|^2$, compute the number of relaxed solutions of weight W' with consistent cuts modulo 2 on a decomposition tree. Then return yes if it is odd, otherwise no in the root node.

To determine whether there exists a partitioned solution (S, A, B, Q) of weight W so that A and B are connected, we use a variation on the *Cut & Count* technique. Required connectedness of both A and B, we consider consistent cuts of A and B.

▶ **Definition 11.** Given a node t of the tree decomposition of G, a *partial solution* for that node is a tuple $(S_\emptyset, S_A, S_B, S_{AB}, A_l, A_r, B_l, B_r, Q, w)$, such that:

- $V_t = S_\emptyset \cup S_A \cup S_A \cup S_{AB} \cup A_l \cup A_r \cup B_l \cup B_r \cup Q$,
- (A_l, A_r) is a consistent cut: there exists no edge $(u, v) \in E$ such that $u \in A_l$ and $v \in A_r$,
- (B_l, B_r) is a consistent cut: there exists no edge $(u, v) \in E$ such that $u \in B_l$ and $v \in B_r$,
- $w = \Sigma_{v \in S} w(v)$,
- $\forall v \in S_\emptyset$, $N(v) \cap (A_l \cup A_r \cup B_l \cup B_r) = \emptyset$,
- $\forall v \in S_A$, $N(v) \cap (B_l \cup B_r) = \emptyset$ and $N(v) \cap (A_l \cup A_r) \neq \emptyset$,
- $\forall v \in S_B$, $N(v) \cap (B_l \cup B_r) \neq \emptyset$ and $N(v) \cap (A_l \cup A_r) = \emptyset$ and
- $\forall v \in S_{AB}$, $N(v) \cap (B_l \cup B_r) \neq \emptyset$ and $N(v) \cap (A_l \cup A_r) \neq \emptyset$.
- $s \in V_t \Rightarrow s \in A_l$
- $t \in V_t \Rightarrow t \in B_l$

Here, we set another weight $w'(v)$ for each vertex by choosing from $\{1, \ldots, 2|V|\}$ uniformly and independently at random for the Isolation lemma. We also set the *coloring* $c : V \to \{s_\emptyset, s_A, s_B, s_{AB}, a_l, a_r, b_l, b_r, q\}$. Then we give a dynamic programming algorithm that computes the number of partial solutions. To count the number of relaxed solutions with consistent cuts, for each c, w and w' we define the counting function $h_i : \{s_\emptyset, s_A, s_B, s_{AB}, a_l, a_r, b_l, b_r, q\}^{|X_i|} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ in each node i on a nice tree decomposition as follows.

Leaf node:

In a leaf node, we define $h_i(\emptyset, 0, 0) = 1$, if $S_\emptyset = S_A = S_B = S_{AB} = A_l = A_r = B_l = B_r = \emptyset$ and $w, w' = 0$. Otherwise, $h_i(c, w, w') = 0$.

Introduce vertex v node:

The function h_i has five cases in introduce vertex nodes. Note that we only add one vertex v without edges. Thus, if $c(v) \in \{s_A, s_B, s_{AB}\}$, a partial solution is invalid by the definition because v has no neighbor. If $c(v) = s_\emptyset$, vertex v is chosen as a vertex of S, and we hence add each weight $w(v)$, $w'(v)$ to w, w', respectively. Moreover, v must not be s, t because s(resp., t) should be in A_l(resp., B_l). If not so, it is not a partitioned solution. Similarly, if $c(v) = a_l$(resp., b_l), we check whether v is not t(resp., s). As for $c(v) \in \{a_r, b_r, q\}$, we also check whether v is neither s nor t. Therefore, we define h_i in introduce vertex nodes as follows:

$$h_i(c \times \{c(v)\}, w, w') := \begin{cases} [v \neq s, t]h_j(c, w - w(v), w' - w'(v)) & if\ c(v) = s_\emptyset \\ [v \neq t]h_j(c, w, w') & if\ c(v) = a_l \\ [v \neq s]h_j(c, w, w') & if\ c(v) = b_l \\ [v \neq s, t]h_j(c, w, w') & if\ c(v) \in \{a_r, b_r, q\} \\ 0 & otherwise. \end{cases}$$

Introduce edge (u, v) node:

In introduce edge nodes, we check each state of endpoints of edge (u, v) and define f_i for some cases.

- If $c(u) = s_\emptyset$, vertex u has no vertices in A, B. Hence, we define the function h_i in this case as follows:

$$h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') \quad := \quad [c(v) \notin \{a_l, a_r, b_l, b_r\}]h_j(c \times \{s_\emptyset\} \times \{c(v)\}, w, w').$$

- If $c(u) = s_A$, vertex u has neighbors of A but no neighbor of B. In this case, we have two cases. One case is that $u \in S_\emptyset$ and $v \in A$ in a children node, because adding edge (u, v) in the introduce edge (u, v) node, vertex u is moved from $S_\emptyset$ to S_A. One case is that $u \in S_A$ and $v \notin B$ in a children node. If $v \in B$, vertex u is in S_{AB} in the parent node. We define h_i as follows. Note that only if $c(v) \in \{a_l, a_r\}$, we sum up two cases. If $c(v) \in \{b_l, b_r\}$, $h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := 0$, otherwise $h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := h_j(c \times \{s_A\} \times \{c(v)\}, w, w')$.

$$\begin{aligned} h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') \quad &:= \quad [c(v) \in \{a_l, a_r\}]h_j(c \times \{s_\emptyset\} \times \{c(v)\}, w, w') \\ &+ \quad [c(v) \notin \{b_l, b_r\}]h_j(c \times \{s_A\} \times \{c(v)\}, w, w'). \end{aligned}$$

- If $c(u) = s_B$ is almost the same as above case, that is, we replace A(resp., B) to B(resp., A).

$$\begin{aligned} h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') \quad &:= \quad [c(v) \in \{b_l, b_r\}]h_j(c \times \{s_\emptyset\} \times \{c(v)\}, w, w') \\ &+ \quad [c(v) \notin \{a_l, a_r\}]h_j(c \times \{s_B\} \times \{c(v)\}, w, w'). \end{aligned}$$

- If $c(u) = s_{AB}$, we consider three cases: $u \in S_A$ and $v \in B$, $u \in S_B$ and $v \in A$, and $u \in S_{AB}$ and v is in arbitrary set in the children node. For first and second cases, vertex u is moved from $S_A(S_B)$ into S_{AB} by adding edge (u, v). If $u \in S_{AB}$, v is allowed to be in any set because a vertex in S_{AB} could connect to all sets. Therefore, we define f_i as follows:

$$\begin{aligned} h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') &:= [c(v) \in \{b_l, b_r\}] h_j(c \times \{s_A\} \times \{c(v)\}, w, w') \\ &+ [c(v) \in \{a_l, a_r\}] h_j(c \times \{s_B\} \times \{c(v)\}, w, w') \\ &+ h_j(c \times \{s_{AB}\} \times \{c(v)\}, w, w'). \end{aligned}$$

- If $c(u) \in \{a_l, a_r\}$, $c(v) \notin \{b_l, b_r, q\}$ since there is no edge between A, B and Q by the definition of partitioned solution. There is also no edge between A_l and A_r because (A_l, A_r) is a consistent cut. Therefore, if u is in $a_l(a_r)$, then v are in the same set of u or separator sets S_A, S_{AB}. Note that because u is in A, v is not in $S_\emptyset, S_B$.

$$\begin{aligned} h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') &:= [c(v) = c(u)] h_j(c \times \{c(u)\} \times \{c(v)\}, w, w') \\ &+ [c(v) \in \{s_A, s_{AB}\}] h_j(c \times \{c(u)\} \times \{c(v)\}, w, w'). \end{aligned}$$

- The case that $c(u) \in \{b_l, b_r\}$ is almost the same as above case, that is, we replace A to B.

$$\begin{aligned} h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') &:= [c(v) = c(u)] h_j(c \times \{c(u)\} \times \{c(v)\}, w, w') \\ &+ [c(v) \in \{s_B, s_{AB}\}] h_j(c \times \{c(u)\} \times \{c(v)\}, w, w'). \end{aligned}$$

- If $c(u) = q$, vertex u is in Q. Hence, v must be in $S_\emptyset, S_A, S_B, S_{AB}$, or Q because a vertex in Q has no neighbor of A and B by the definition of partitioned solution.
  $h_i(c \times \{c(u)\} \times \{c(v)\}, w, w') := [c(v) \in \{s_\emptyset, a_A, s_B, s_{AB}, q\}] h_j(c \times \{c(u)\} \times \{c(v)\}, w, w')$.

Forget v node:

For forget nodes, the state of v would never change forward. Thus, if $c(v) \in \{s_\emptyset, s_A, s_B\}$, a partial solution does not satisfy the condition of partitioned solution because any $v \in S$ must have neighbors of both A and B. For this reason, we only sum up for each state $c(v) \in \{s_{AB}, a_l, a_r, b_l, b_r, q\}$. The function h_i in forget nodes is defined as follows:

$$h_i(c, w, w') := \sum_{c(v) \in \{s_{AB}, a_l, a_r, b_l, b_r, q\}} h_j(c \times \{c(v)\}, w, w').$$

Join node:

We denote each coloring by c_i, c_{j_1}, c_{j_2} and weights of partial solutions in i, j_1, j_2 by w_i, w_{j_1}, w_{j_2}, w'_i, w'_{j_1}, w'_{j_2}. Moreover, for a state subset $L \subseteq \{s_\emptyset, s_A, s_B, s_{AB}, a_l, a_r, b_l, b_r, q\}$, we define $c^{-1}(L)$ as the vertex set such that all vertices satisfy $c(v) \in L$. For a coloring c_i, we also define the subset D of tuples of (c_{j_1}, c_{j_2}) as the combinations of colorings of c_{j_1}, c_{j_2} like Section 4 such that (See Table 2):

- $\forall v \in c_i^{-1}(\{s_\emptyset, a_l, a_r, b_l, b_r, q\})$, $(c_{j_1}(v), c_{j_2}(v)) = (c_i(v), c_i(v))$,
- $\forall v \in c_i^{-1}(\{s_A\})$, $(c_{j_1}(v), c_{j_2}(v)) = (s_A, s_\emptyset), (s_\emptyset, s_A), (s_A, s_A)$,
- $\forall v \in c_i^{-1}(\{s_B\})$, $c_{j_1}(v), c_{j_2}(v)) = (s_B, s_\emptyset), (s_\emptyset, s_B), (s_B, s_B)$, and
- $\forall v \in c_i^{-1}(\{s_{AB}\})$, $(c_{j_1}(v), c_{j_2}(v)) = (s_{AB}, s_\emptyset), (s_{AB}, s_A), (s_{AB}, s_B), (s_{AB}, s_{AB}), (s_\emptyset, s_{AB})$, (s_A, s_{AB}), (s_B, s_{AB}), (s_A, s_B), (s_B, s_A).

Let S^* be the vertex subset $c_i^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\})$. To sum up all combinations of vertex states and weights for counting, we define the function h_i. If $D = \emptyset$, we define $h_i(c_i, w_i, w_i') := 0$. Otherwise,

$$h_i(c_i, w_i, w_i') := \sum_{\substack{w_{j_1}+w_{j_2} \\ =w_i+w(S^*)}} \sum_{\substack{w'_{j_1}+w'_{j_2} \\ =w'_i+w'(S^*)}} \sum_{(c^*_{j_1}, c^*_{j_2})\in D} h_{j_1}(c^*_{j_1}, w_{j_1}, w'_{j_1}) h_{j_2}(c^*_{j_2}, w_{j_2}, w'_{j_2}).$$

From now, we analyze the running time of this algorithm. In leaf, introduce vertex, introduce edge, and forget nodes, we can compute f_i for each coloring c and weight w, w' in $O(1)$-time because we only use $O(1)$-operations. Therefore, Total running time in them is $O^*(9^{\mathbf{tw}} \cdot W \cdot W')$. However, in join nodes, we sum up all weight combinations and coloring combinations satisfying some conditions. There are 21 coloring's combinations for each vertex according to Table 2, and W, W' weight's combinations. Therefore, we compute all f_i's in a join node in time $O^*(21^{\mathbf{tw}} \cdot W^2)$. Note that by the definition, $O(W'^2)$ is polynomial factor.

▶ **Theorem 12.** *For graphs of treewidth at most* **tw***, there exists a Monte-Carlo algorithm that solves decision version* Maximum Weight Minimal s-t Separator *in time* $O^*(21^{\mathbf{tw}} \cdot W^2)$*. It cannot give false positives and may give false negatives with probability at most* $1/2$*.*

Moreover, we show that there exists a Monte-Carlo algorithm that solves Maximum Weight Minimal s-t Separator in time $O^*(9^{\mathbf{tw}} \cdot W^2)$ using the convolution technique[17]. For the detailed explanation, see Appendix.

▶ **Theorem 13.** *For graphs of treewidth at most* **tw***, there exists a Monte-Carlo algorithm that solves decision version* Maximum Weight Minimal s-t Separator *in time* $O^*(9^{\mathbf{tw}} \cdot W^2)$*. It cannot give false positives and may give false negatives with probability at most* $1/2$*. If the input graph is unweighted, the running time is* $9^{\mathbf{tw}} \cdot |V|^{O(1)}$*.*

References

1 S. Arnborg, D. G. Corneil, A. Proskurowski : Complexity of finding embeddings in a k-tree. In: *SIAM Journal on Algebraic Discrete Methods*, 8(2), 277–284 (1987)

2 A. Berry, J. P. Bodat, O. Cogis: Generating all the minimal separators of a graph. In: *International Journal of Foundations of Computer Science*, 11(3), 397–404 (2000)

3 H.L. Bodlaender: A linear-time algorithm for finding tree-decompositions of small treewidth. In: *SIAM Journal on Computing* 25(6), 1305–1317 (1996)

4 H. L. Bodlaender, T. Kloks, D. Kratsch: Treewidth and pathwidth of permutation graphs. In: *SIAM Journal on Discrete Mathematics*, 8(4), 606–616 (1995)

5 V. Bouchitté, I. Todinca: Treewidth and minimum fill-in: grouping the minimal separators. In: *SIAM Journal on Computing*, 31(1), 212–232 (2001)

6 V. Bouchitté, , I. Todinca: Listing all potential maximal cliques of a graph. In: *Theoretical Computer Science*, 276(1-2), 17–32(2002)

7 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: *Proceeding FOCS '11 Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, 150–159 (2011)

8 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk and S. Saurabh : Parameterized Algorithms, Springer International Publishing (2015)

9 F. V. Fomin, D. Kratsch, I. Todinca, Y. Villanger: Exact algorithms for treewidth and minimum fill-in. In: *SIAM Journal on Computing*, 38(3), 1058–1079 (2008).

10 M. R. Garey, D. S. Johnson, and L. Stockmeyer: Some simplified NP-complete graph problems. In: *Theoretical computer science*, 1(3), 237–267 (1976).

11 S. Gaspers, S. Mackenzie: On the Number of Minimal Separators in Graphs. *arXiv:1503.01203v2* (2015)

12 M. C. Golumbic: Algorithmic graph theory and perfect graphs. Academic Press, New York, United States (1980)

13 T. Kloks: Treewidth, Computations and Approximations. *Lecture Notes in Computer Science*, 842, Springer-Verlag Berlin Heidelberg (1994).

14 T. Kloks: Treewidth of circle graphs. In: *International Journal of Foundations of Computer Science*, 7(2), 111 (1996)

15 T. Kloks, D. Kratsch: Treewidth of chordal bipartite graphs. In: *Journal of Algorithms*, 19(2), 266–281 (1995)

16 K. Mulmuley, U. V. Vazirani, V. V. Vazirani: Matching is as easy as matrix inversion. In: *Combinatorica*, 7(1), 105–113 (1987)

17 J. M. M. van Rooij, H. L. Bodlaender, P. Rossmanith: Dynamic programming on tree decompositions using generalised fast subset convolution. In: *17th Annual European Symposium on Algorithms, ESA 2009, Lecture Notes in Computer Science* 5757, Springer Berlin Heidelberg, 566–577 (2009)

18 K. Skodinis: Efficient Analy sis of Graphs with Small Minimal Separators. In: *Graph-Theoretic Concepts in Computer Science- 25th International Workshop, WG'99, Lecture Notes in Computer Science* 1665, Springer Berlin Heidelberg, 155–166 (1999)

19 K. Suchan: Minimal Separators in Intersection Graphs, *Masters thesis, Akademia Gorniczo-Hutnicza im. Stanislawa Staszica w Krakowie, Cracow* (2003)

20 R. Sundaram, K. S. Singh, C. P. Rangan: Treewidth of circular-arc graphs. In: *SIAM Journal on Discrete Mathematics*, 7(4), 647–655 (1994)

Appendix

A Proof of Thorem 3

Proof. We show the reduction from a well-known NP-hard problem, MAX CUT ([10]) of unweighted graph $G = (V, E)$, which asks the existence of cut $(C, V \setminus C)$ whose value $|\{(u, v) \in E \mid u \in C, v \in V \setminus C\}|$ is at least k.

In the following, we construct an instance of Maximum Weight Minimal s-t Separator from $G = (V, E)$ and positive integer k. Although the instance is weighted, the proof can be easily modified to an unweighted case. Let $p = (3n+1)k$ and $G' = (V \cup E \cup V' \cup V'' \cup \{s, t\}, E_1 \cup E_2)$, where $n = |V|$, $V' = \{v' \mid v \in V\}$, $V'' = \{v'' \mid v \in V\}$, $E_1 = \bigcup_{e=(u,v)\in E}\{(u, e), (v, e)\}$ and $E_2 = \bigcup_{u\in V}\{(s, u'), (u', u)\} \cup \bigcup_{u\in V}\{(t, u''), (u'', u)\}$. The vertex weights of G' are defined by $w_v = 3n + 1$ for $v \in E$ and 1 otherwise.

We now show that if G has a cut C of weight at least k, G' has a minimal s-t separator S whose weight is at least $p = (3n + 1)k$. We define $S = \{u'' \in V'' \mid u \in V \cap C\} \cup \{v' \in V' \mid v \in V \setminus C\} \cup \{e = (u, v) \in E \mid u \in C, v \in V \setminus C\}$. The weight of S is at least $(3n + 1)k$ by $|\{e = (u, v) \in E \mid u \in C, v \in V \setminus C\}| \geq k$. We can see that S is a s-t separator, since any $u \in C$ is reachable from s even after removing S from G' but not reachable from t. The minimality also holds because by adding any vertex in S we obtain a path from s to t.

We next show that if G' has a minimal s-t separator S whose weight is at least $p = (3n+1)k$, G has a cut C of weight at least k. By the weighting, S contains at least k vertices in E. Note that for any $v \in V(G)$, at least one of v, v', v'' is included in S, otherwise S does not separate s and t. If S does not contain any $v \in V$, let C be vertices in V that are reachable from s after removing S; C is actually a cut, and its weight is k. Otherwise, S contains a vertex $v \in V$. In this case, S does not contain any e forming $e = (v, x)$ because otherwise it contradicts the minimality. Then, we construct $S' := S \setminus \{v\} \cup \{v'\} \cup \{e = (v, x) \in E\}$. By repeating this procedure, we obtain S that does not contain any $v \in V$. This completes the correctness of the reduction.

As mentioned above, this reduction can be modified to an unweighted case. In the modification, we construct graph $G' = (V \cup E' \cup V' \cup V'' \cup \{s, t\}, E'_1 \cup E_2)$ instead of $G' = (V \cup E \cup V' \cup V'' \cup \{s, t\}, E_1 \cup E_2)$, where $E' = \{e^{(1)}, \ldots, e^{(3n+1)} \mid e \in E\}$ and $E'_1 = \bigcup_{e=(u,v)\in E}\bigcup_{i=1}^{3n+1}\{(u, e^{(i)}), (v, e^{(i)})\}$. The weight of every vertex is one. In the new reduction, to block the path between u and v, we need to remove $3n + 1$ copies of $e(= (u, v))$, which plays the same role of weight $3n + 1$. ◀

B Proof of Theorem 5

To prove Theorem 5, we use the following lemma. This lemma appears in many papers and books, for example, an exercise in [12].

▶ **Lemma 14.** *Let S be a minimal s-$t$separator and A, B be the connected components of $G[V \setminus S]$ containing s and t, respectively. Then every vertex of S has a neighbor in A and a neighbor in B.*

By using this lemma, we can prove Theorem 5.

Proof. ($\Rightarrow$) Let S be a minimal s-t separator of weight W. Let A be the set of vertices of $V \setminus S$ such that $G[A]$ is the connected component containing s and B be the set of vertices such that $G[B]$ is the connected component containing t. Moreover, let $Q = V \setminus A \setminus B \setminus S$. Note that $S \cap A \cap B \cap Q = \emptyset$ and there is no edge between A, B and Q. By lemma 14,

all vertices in S have both neighbors of vertices in A and B. Therefore, (S, A, B, Q) is a partitioned solution of weight W.
($\Leftarrow$) Suppose that (S, A, B, Q) is a partitioned solution of weight W such that A and B are connected. We claim that S is a minimal s-t separator. To show this by contradiction, we suppose that there exists a vertex $v \in S$ such that $S \setminus \{v\}$ separates s and t. Then there exist vertices $a \in A, b \in B$ so that $(a, v) \in E, (v, b) \in E$. Since $G[A]$ is connected, there exist a path (in $G[A]$) from s to a. Similarly, there exist a path (in $G[B]$) from b to t. By joining these paths with the edges (a, v) and (v, b), we obtain a path from s to t, which contradicts that $S \setminus \{v\}$ is a separator. Hence S is a minimal s-t separator, and by definition it is of weight W. ◀

C Fast Convolution

First, we set the new coloring $\hat{c} : V \to \{s_{\bar{A}\bar{B}}, s_{\bar{A}}, s_{\bar{B}}, s_{all}, a_l, a_r, b_l, b_r, q\}$. The state $s_{\bar{A}\bar{B}}$ represents that a vertex v is in S and has no neighbor of A and B. The state $s_{\bar{A}}$(resp., $s_{\bar{B}}$) represents a vertex v is in S and has no neighbor of A(resp., B), respectively. Finally, the state s_{all} represents a vertex v is in S without constraints.

Then, we show the following lemma to transform between c and $\hat{c}$.

▶ **Lemma 15.** *Let x be a node of a tree decomposition and $h_i(c, w, w')$ be a counting function to count the number of partial solutions of* Maximum Weighted Minimal s-t Separator *of each weight w, w', corresponding to each coloring c, $\hat{c}$ of a node x. Then we can transform from one coloring to another coloring for each function without loss of information. Moreover, it is computed in $O(W \cdot W' \cdot 9^{\mathbf{tw}} \cdot |X_i|)$.*

Proof. This proof scheme follows [17]. We consider an immediate i-th step in the transformation. For $h_i(c \times \{c(v)\} \times \hat{c}, w, w')$, v is a vertex which turns into the state of another coloring in i-th step and c is a partial coloring of size $i - 1$ and $\hat{c}$ is also a partial coloring of size $|X_i| - i$. Here, for simplicity, we denote $h_i(c \times \{c(v)\} \times \hat{c}, w, w')$ and $h_i(c \times \{\hat{c}(v)\} \times \hat{c}, w, w')$ by $h_i(c(v))$ and $h_i(\hat{c}(v))$.

Since h_i is the number of partial solutions with consistent cuts, the transformation from c to $\hat{c}$ of h_i in i-th step is as follows:

- $h_i(s_{\bar{A}\bar{B}}) = h_i(s_\emptyset)$
- $h_i(s_{\bar{A}}) = h_i(s_\emptyset) + h_i(s_B)$
- $h_i(s_{\bar{B}}) = h_i(s_\emptyset) + h_i(s_A)$
- $h_i(s_{all}) = h_i(s_\emptyset) + h_i(s_A) + h_i(s_B) + h_i(s_{AB})$.

Conversely, we can transform from $\hat{c}$ to c as follows:

- $h_i(s_\emptyset) = h_i(s_{\bar{A}\bar{B}})$
- $h_i(s_A) = h_i(s_{\bar{B}}) - h_i(s_{\bar{A}\bar{B}})$
- $h_i(s_B) = h_i(s_{\bar{A}}) - h_i(s_{\bar{A}\bar{B}})$
- $h_i(s_{AB}) = h_i(s_{all}) - h_i(s_{\bar{A}}) - h_i(s_{\bar{B}}) + h_i(s_{\bar{A}\bar{B}})$.

These equations follow from equations of transformation from c to $\hat{c}$. For each colorings $c, \hat{c}$, each transformation can be performed in $|X_i|$-step. Thus, total running time of each direction is $O(W \cdot W' \cdot 9^{\mathbf{tw}} \cdot |X_i|)$-time. ◀

Therefore, we firstly transform from the original coloring to the new coloring in $O(W \cdot W' \cdot 9^{\mathbf{tw}} \cdot |X_i|)$-time. Then we compute the following function h_i for the new coloring $\hat{c}$ in $O(9^{\mathbf{tw}} \cdot W^2)$-time:

$$h_i(\hat{c}, w, w'_i) := \sum_{w_{j_1} + w_{j_2} = w_i + w(\hat{S}^*)} \sum_{w'_{j_1} + w'_{j_2} = w'_i + w'(\hat{S}^*)} h_{j_1}(\hat{c}, w_{j_1}, w'_{j_1}) h_{j_2}(\hat{c}, w_{j_2}, w'_{j_2})$$

	$s_\emptyset$	s_A	s_B	s_{AB}	a_l	a_r	b_l	b_r	q
$s_\emptyset$	$s_\emptyset$	s_A	s_B	s_{AB}					
s_A	s_A	s_A	s_{AB}	s_{AB}					
s_B	s_B	s_{AB}	s_B	s_{AB}					
s_{AB}	s_{AB}	s_{AB}	s_{AB}	s_{AB}					
a_r					a_r				
a_l						a_l			
b_r							b_r		
b_l								b_l	
q									q

Table 2 Combinations of the original coloring in join node

	s_{all}	$s_{\bar{A}}$	$s_{\bar{B}}$	$s_{\bar{A}\bar{B}}$	a_l	a_r	b_l	b_r	q
s_{all}	s_{all}								
$s_{\bar{A}}$		$s_{\bar{A}}$							
$s_{\bar{B}}$			$s_{\bar{B}}$						
$s_{\bar{A}\bar{B}}$				$s_{\bar{A}\bar{B}}$					
a_r					a_r				
a_l						a_l			
b_r							b_r		
b_l								b_l	
q									q

Table 3 Combinations of the new coloring in join node

where $\hat{S}^* = \hat{c}^{-1}(\{s_\emptyset, s_A, s_B, s_{AB}\}) \subseteq V$. Note that $\hat{c}_i, \hat{c}_{j_1}, \hat{c}_{j_2}$ are the same coloring. Finally, we transform the coloring conversely. Thus, total running time of this algorithm is $O^*(9^{\mathbf{tw}} \cdot W^2)$.

分割不可能な財のオンライン配分方式

An Online Allocation Algorithm of Indivisible Goods

清水 航平　　真鍋 義文
Kohei Shimizu　　Yoshifumi Manabe

工学院大学大学院 工学研究科 情報学専攻
Department of Computer Science, Graduate School of Informatics, Kogakuin University

概 要

本論文では、分割不可能な財のオンライン配分方式を提案する。ここで言うオンラインアルゴリズムとは、参加者がどの時刻からでも参加することが可能であり、財が割り当てられた時に退出するものである。参加者の各財に感じる価値の合計は同じであると仮定する。分割可能な財に対する配分問題であるオンラインケーキ分割問題において、その場にいる他の参加者に対して嫉妬が発生しない状態を意味する Immediately Envy-free という好ましい特性が存在するが、分割不可能な財に対するオンライン配分アルゴリズムにおいてこの特性を満たすことは困難である。そこでその場にいる自分より後から来た参加者に対して嫉妬が発生しない状態を意味する Weakly Immediately Envy-free という特性を提案する。提案アルゴリズムは、その特性を満たしつつ、割り当てられた財の価値の合計が全参加者の中で最も少ない参加者の獲得した財の合計価値を最大化することを目標とする。この問題は NP 完全である分割問題から帰着可能な内容を含んでいる為、NP 困難であることを証明している。従って、本稿ではこの問題に対する近似解を求めるアルゴリズムを提案し、同時に近似度に関する証明を行う。

Keywords - allocation; envy-free; algorithm; indivisible goods

1　はじめに

複数の参加者間で複数の財を分け合う公平分割問題、その中でもパーティでのプレゼント分配や、同様の行動に対する物品での報酬支給の様な場合に適用可能な分割不可能な財のオンライン分割問題を考察する。本稿では、その場にいる自分より後から来た参加者に対して嫉妬が発生しない状態を意味する Weakly Immediately Envy-free という特性を提案し、その特性を満たしつつ、割り当てられた財の価値の合計が全参加者の中で最も少ない参加者の獲得した財の合計価値を最大化することを目標とする。分割不可能な財に対する配分方式としては、maximin share guarantee[2][3][4] の様にいくつかの近似アルゴリズムがすでに提案されているものの、各参加者が獲得した財の価値の合計の総計を最大化することに焦点を置いており、またオフライン分割問題のアルゴリズムである。オンラインケーキ分割問題に関するアルゴリズムは提案されている [1] が、我々の知る限りでは分割不可能な財に対するオンライン配分方式は存在していない。この問題は NP 完全である分割問題から帰着可能な内容を含んでおり NP 困難である。従って、本稿ではこの問題に対する近似解を求めるアルゴリズムを提案し、同時に近似度に関する証明を行う。またコンピュータシミュレーションによって近似度の検証も行う。

2　問題定義

分割不可能な財のオンライン配分問題は以下のように定義する。

参加者の集合:$X = \{x_1, x_2, \ldots, x_n\}$.
財の集合:$V = \{v_1, v_2, \ldots, v_m\}$.
財に対する各参加者の評価関数 $P_i : V \to \mathbb{N}$, ただし
$\forall i, j (1 \leq i \leq n, 1 \leq j \leq m) P_i(v_j) \geq 1$,
$\sum_{j=1}^{m} P_i(v_j) = P (i = 1, \ldots, n)$.
各参加者にとって、すべての財の価値の合計は同一。各参加者がそれぞれの財に感じる価値は同じとは限らない。本稿では財に対する評価関数の最小単位を P（ポイント）と呼ぶ。どの参加者、どの財においても無価値である財は存在しないとする。また参加者はリスク回避型プレイヤーとする。各参加者は、参加者は全体で何人であるか、割り当てが終了していない参加者は現在何人であるか知ることが出来るものとする。
割り当て $A \to 2^V (V$ の部分集合を与える).
割り当て A において x_i に割り当てられた財の合計 Point:$u_i(A(x_i)) = \sum_{v_i \in A(x_i)} P_i(v_i)$.
割り当て A の u_i の最小値:$u(A) = \min_{x_i \in X} u_i(A(x_i))$.
$u(A)$ を最大にする割り当て:$\tilde{A}$

本稿では、最も好ましい割り当ては以下の状態であるとする。(1) 割り当ては、その場にいる自分より後から来た参加者に対して嫉妬が発生しない状態である Weakly Immediately Envy-free を必ず満たすものである。(2)(1) を満たす割り当てが複数存在するならば、その中でも $u(A)$ が最大となるもの。

多くのオンライン配分問題において、次の参加者が到着する前に割り当てを退出する先の参加者はアルゴリズムに悪い影響を及ぼす傾向 [1] がある。その為本稿ではペナルティーとして、そのような参加者に対しては Weakly Immediately Envy-free を考慮しないこととする。

3　NP 完全性

参加者がどのように到着するかに関わらず未だ財を割り当てられていない参加者が 2 人となる瞬間が存在し、

以下で示す通りこの配分問題はNP完全である計算をする必要があるケースが含まれている。

NP 完全性の証明の為、以下の決定問題を考える[5][6][7]。

初期状態: X, V, P_i、整数 k
質問: $u(A) \geq k$ となるような A は存在するか？

この決定問題がNPに属することは明白である。分割問題に関する決定問題の還元により、NP困難性を証明することが出来る。

初期状態: $\sum_{s_i \in \chi} s_i = 2L$ を満たす整数の集合 $\chi = \{s_1, s_2, \ldots, s_p\}$
質問: $\chi = \chi_1 \cup \chi_2, \chi_1 \cap \chi_2 = \emptyset$ かつ $\sum_{s_i \in \chi_1} s_i = \sum_{s_i \in \chi_2} s_i$ を満たすような配分 (χ_1, χ_2) は存在するか？

$n = 2$, $m = p$, $P_1 = P_2$ かつ $k = L$ のとき、配分問題が解を持つときのみ分割問題は解を持つ。例として、分割問題 ($\chi = \{3,8,4,12,5,9,1,2,2,6\}$, $2L = 52$) は ($n = 2$, $m = 10$, $P = 52$, $P_1 = \{3,8,4,12,5,9,1,2,2,6\}$, $P_2 = \{3,8,4,12,5,9,1,2,2,6\}$, $k = 2L/2 = 26$) という配分問題に置き換えることが出来る。この場合、次のような配分 A が存在する。$A(x_1) = \{v_1, v_3, v_4, v_7, v_{10}\}$, $A(x_2) = \{v_2, v_5, v_6, v_8, v_9\}$, $u(A) = 26$,$\chi_1 = \{3,4,12,1,6\}$, $\chi_2 = \{8,5,9,2,2\}$
対応する配分問題が解を持つときのみ分割問題が解を持つことは明白である。分割問題がNP完全であることから、配分問題はNP完全な計算が必要な場合を含んでいる。NP完全性より、本稿では最適解に出来るだけ近づくような近似解を求めるアルゴリズムを考察する。

4 提案アルゴリズム

この問題はNP困難であるため、近似アルゴリズムを提案する。各参加者は問題定義を満たす P_i という評価関数を所持している。

1. 参加者が二人以上揃った場合に割り当てを開始し、参加者が揃う前に退出する場合は財の中から一つのみを選んで取得し退出する。

2. 各参加者は現在残っている財を確認し、満足できる分を取得するためには最低何個財が必要かを提示する。(全ての参加者は値 T を超える個数を提示してはならない。T は残りの財の個数を残りの参加者の人数で割った値を、小数点以下繰り上げしたものである。)

3. 提示された数を確認し、最も少ない個数を提示した参加者の中でも先に到着していた参加者が、その個数だけ好きな財を取得し退出する。

4. 財を割り当てられていない参加者が一人になるまでStep1,2,3を繰り返す。

5. 最後の参加者が残ったすべての財を取得し退出する。

提案アルゴリズムは、未だ到着していない参加者の為に出来るだけ多くの財を残そうとするアルゴリズムである。未だ到着していない参加者の P_i を知ることは不可能であり、また価値が0の財は存在しない為、多くの財を残そうとする働きはより良い割り当てを導くことのできる傾向がある。

他者の評価関数を知ることは出来ないため、虚偽の申告をした場合参加者は本来取得できるはずだった財よりも低い価値の財を取得することになる可能性がある。少なく申請すればそのまま満足できる量よりも少ない財を取得し退出することになる可能性があり、多く申請すれば本来自分が満足出来た量以上の財を他の参加者に取得され残りの財の価値を低くする可能性がある。よってリスク回避型プレイヤーである参加者は、虚偽の申告をすることはない。

もし提案アルゴリズムがWeakly Immediately Envy-freeでないとした場合、自分より後から来た参加者の割り当てを知っている参加者が、その参加者に対して嫉妬が発生している状態である。Step2とStep3の働きにより、提案アルゴリズムでは割り当て開始時にその場にいる参加者の中で最も先に到着している参加者は必ず満足できる量以上の財を取得することが出来る。もしその参加者が最小の個数を提示した場合は満足できる量以上の財を取得し、他の参加者がその個数よりも少ない値を提示したならば先に到着した参加者にとって満足できる量より少ない財を取得し退出する。よってこのアルゴリズムは常にWeakly Immediately Envy-freeを満たす割り当てを導く事が可能である。

先に到着していた参加者は、提案アルゴリズムのStep3の条件より、次の割り当てにおいて財1つであれば必ずその財を取得することが出来る。そのため途中退室での財の中から一つのみを選んで退出可能とする内容は、他者の割り当てに不利益を及ぼすことはない。

5 提案アルゴリズム実行例

5.1 実行例 1

初期状態: $n = 3$, $m = 4$, $P = 100$, $P_1 = \{30,30,20,20\}$, $P_2 = \{80,10,5,5\}$, $P_3 = \{30,10,20,40\}$.
全探索を用いて求めたこの例における最適解は以下のとおりである。$\tilde{A}(x_1) = \{v_2, v_3\}$, $\tilde{A}(x_2) = \{v_1\}$, $\tilde{A}(x_3) = \{v_4\}$, $u_1(\tilde{A}) = 50$, $u_2(\tilde{A}) = 80$, $u_3(\tilde{A}) = 40$, $u(\tilde{A}) = 40$.

最初の参加者 x_1 と二番目の参加者 x_2 が到着する。割り当てされていない参加者の人数は現時点では残り3人である。x_1 にとっての残りの財に対する価値の合計は100であり、x_1 は $\frac{100}{3}$ ポイント以上の財を獲得すれば満足できる。よって x_1 は財2つで満足できると提示する。

例としては、$A(x_1) = \{v_1, v_2\}$ となる。x_2 にとっての残りの財に対する価値の合計は 100 であり、x_2 は $\frac{100}{3}$ ポイント以上の財を獲得すれば満足できる。よって x_2 は財 1 つで満足できると提示する。例としては、$A(x_2) = \{v_1\}$ となる。 x_2 が x_1 よりも少ない個数を提示したため、x_2 が 1 つ好きな財を取得し退出する。x_2 は v_1 を取得し退出する。x_2 の退出後、三番目の参加者 x_3 が到着する。割り当てされていない参加者の人数は現時点では残り 2 人である。x_1 にとっての残りの財に対する価値の合計は 70 であり、x_1 は $\frac{70}{2}$ ポイント以上の財を獲得すれば満足できる。よって x_1 は財 2 つで満足できると提示する。例としては、$A(x_1) = \{v_2, v_3\}$ となる。x_3 にとっての残りの財に対する価値の合計は 70 であり、x_3 は $\frac{70}{2}$ ポイント以上の財を獲得すれば満足できる。x_3 は財 1 つで満足できると提示する。例としては、$A(x_3) = \{v_4\}$ となる。x_3 が x_1 よりも少ない個数を提示したため、x_3 が 1 つ好きな財を取得し退出する。x_3 は v_4 を取得し退出する。割り当てされていない参加者の人数は現時点では残り 1 人である。x_1 は残りの財 v_2 と v_3 を取得し退出する。

$A(x_1) = \{v_2, v_3\}$, $A(x_2) = \{v_1\}$, $A(x_3) = \{v_4\}$, $u_1(A) = 50$, $u_2(A) = 80$, $u_3(A) = 40$, $u(A) = 40$ となる。この例では、提案アルゴリズムは最適解を導いている。

5.2 実行例 2

初期状態: $n = 4$, $m = 8$, $P = 100$, $P_1 = \{50, 20, 20, 2, 2, 2, 2, 2\}$, $P_2 = \{10, 20, 20, 10, 10, 10, 10, 10\}$, $P_3 = \{20, 20, 10, 10, 10, 10, 10, 10\}$, $P_4 = \{5, 5, 5, 5, 10, 20, 30, 20\}$.
全探索を用いて求めたこの例における最適解は以下のとおりである。$\tilde{A}(x_1) = \{v_1\}$, $\tilde{A}(x_2) = \{v_2, v_3\}$, $\tilde{A}(x_3) = \{v_4, v_5, v_8\}$, $\tilde{A}(x_4) = \{v_6, v_7\}$, $u_1(\tilde{A}) = 50$, $u_2(\tilde{A}) = 40$, $u_3(\tilde{A}) = 30$, $u_4(\tilde{A}) = 50$, $u(\tilde{A}) = 30$.

最初の参加者 x_1 と二番目の参加者 x_2 が到着する。割り当てされていない参加者の人数は現時点では残り 4 人である。x_1 にとっての残りの財に対する価値の合計は 100 であり、x_1 は $\frac{100}{4}$ ポイント以上の財を獲得すれば満足できる。よって x_1 は財 1 つで満足できると提示する。例としては、$A(x_1) = \{v_1\}$ となる。x_2 にとっての残りの財に対する価値の合計は 100 であり、x_2 は $\frac{100}{4}$ ポイント以上の財を獲得すれば満足できる。よって x_2 は財 2 つで満足できると提示する。例としては、$A(x_2) = \{v_2, v_3\}$ となる。 x_1 が x_2 よりも少ない個数を提示したため、x_1 が 1 つ好きな財を取得し退出する。x_1 は v_1 を取得し退出する。x_1 の退出後、三番目の参加者 x_3 が到着する。割り当てされていない参加者の人数は現時点では残り 3 人である。x_2 にとっての残りの財に対する価値の合計は 90 であり、x_2 は $\frac{90}{3}$ ポイント以上の財を獲得すれば満足できる。よって x_2 は財 2 つで満足できると提示する。例としては、$A(x_2) = \{v_2, v_3\}$ となる。x_3 にとっての残りの財に対する価値の合計は 80 であり、x_3 は $\frac{80}{3}$ ポイント以上の財を獲得すれば満足できる。よって x_3 は財 2 つで満足できると提示する。例としては、$A(x_3) = \{v_2, v_3\}$ となる。x_2 と x_3 が共に同数を提示したため、先に到着した参加者である x_2 が 2 つ好きな財を取得し退出する。x_1 は v_1 を取得し退出する。x_2 は v_2 と v_3 を取得し退出する。x_2 の退出後、四番目の参加者 x_4 が到着する。割り当てされていない参加者の人数は現時点では残り 2 人である。x_3 にとっての残りの財に対する価値の合計は 50 であり、x_2 は $\frac{50}{2}$ ポイント以上の財を獲得すれば満足できる。よって x_3 は財 3 つで満足できると提示する。例としては、$A(x_3) = \{v_4, v_5, v_6\}$ となる。x_4 にとっての残りの財に対する価値の合計は 85 であり、x_4 は $\frac{85}{2}$ ポイント以上の財を獲得すれば満足できる。よって x_4 は財 2 つで満足できると提示する。例としては、$A(x_4) = \{v_6, v_7\}$ となる。x_4 が x_3 よりも少ない個数を提示したため、x_4 が 2 つ好きな財を取得し退出する。x_4 は v_6 と v_7 を取得し退出する。割り当てされていない参加者の人数は現時点では残り 1 人である。x_3 は残りの財 v_4 と v_5 と v_8 を取得し退出する。

$A(x_1) = \{v_1\}$, $A(x_2) = \{v_2, v_3\}$, $A(x_3) = \{v_4, v_5, v_8\}$, $A(x_4) = \{v_6, v_7\}$, $u_1(A) = 50$, $u_2(A) = 40$, $u_3(A) = 30$, $u_4(A) = 50$, $u(A) = 30$ となる。この例では、x_1 は $A(x_1)$ しか分からない。よって x_1 はどの参加者に対しても嫉妬を感じることはない。x_2 は自分の割り当てのほかに $A(x_1)$ を知ることが出来る。 しかし $u_2(A) = 40$、$u_2(A(x_1)) = 10$ であるため、x_2 は x_1 に対しても嫉妬を感じることはない。x_3 は自分の割り当てのほかに $A(x_4)$ を知ることが出来る。しかし $u_3(A) = 30$、$u_3(A(x_4)) = 30$ であるため、x_3 は x_4 に対しても嫉妬を感じることはない。x_4 は退出時の状況から $A(x_3)$ を推測することが可能である。しかし $u_4(A) = 50$、$u_4(A(x_3)) = 35$ であるため、x_4 は x_3 に対しても嫉妬を感じることはない。この例において、x_3 は自分より後から来た参加者への割り当て $A(x_4)$ を知ることが出来る。もし x_3 が x_4 に対して嫉妬を感じるような場合であれば、それは Weakly Immediately Envy-free を満たしていないことになる。しかしながら x_3 は x_4 に対して嫉妬を感じていない。よって提案アルゴリズムは Weakly Immediately Envy-free を満たす最適解を導いている。

6 近似度

本稿では近似度を、提案アルゴリズムによって求めた値 $u(A)$ をオフライン全探索によって求めた値 $u(\tilde{A})$ で割った値であると定義する。その近似度について、$n = m$、$2n \leq m$、$n + 1 \leq m \leq 2n - 1$ の 3 つの場合に分けて証明を行う。

6.1 $n = m$

$n = m$ であるならば、全ての参加者は財 1 つで満足できると提示する。すべての参加者がそれぞれの最も望む財を取得した場合において、$u(\tilde{A})$ は $\frac{P}{2}$ 以上となり得る。もし各参加者の最も望む財が異なるものであるならば、提案アルゴリズムの Step2 と Step3 の働きにより、全参加者は最も望む財を取得することが可能である。もし最も望む財が被った参加者が存在した場合、少なくとも一人の参加者が最も望む財を取得することが出来ない。よってその場合 $u(\tilde{A}) \leq \frac{P}{2}$ となる。各参加者は 1 つの財を取得可能であり無価値の財は存在しないため $u(A) \geq 1$ である。以上より、$n = m$ において近似度は $\frac{1}{\frac{P}{2}} = \frac{2}{P}$ 以上であると証明される。

6.2 $2n \leq m$

提案アルゴリズムには提示個数の上限である T が存在する。この上限の働きにより、$2n \leq m$ において各参加者は最も望まない財 2 つの価値以上の財を必ず取得することが出来る。よって各参加者は少なくとも 2 ポイント以上の価値の財を取得することが出来る。$u(\tilde{A})$ は P 以下であることは自明である。以上より、$2n \leq m$ において近似度は $\frac{1}{\frac{P}{2}} = \frac{2}{P}$ 以上であると証明される。

6.3 $n + 1 \leq m \leq 2n - 1$

提案アルゴリズムには、その時点で最も先に到着した参加者は必ず満足できる量以上の価値の財を取得できる働きがある。もし $u(A) = 1$ であるならば、財を一つしか取得できなかった参加者が存在する。そのような場合、複数の参加者が希望個数 2 つと同数を提示し、誰かが財を 2 つ取得したラウンドが必ず存在する。参加者が希望個数 2 個と提示するような場合は、n' はその時点でまだ財を割り当てられていない参加者の人数としたとき、残っているすべての財に対して感じる価値がそれぞれ $\frac{P}{n'}$ よりも少ない場合である。$n' = 2$ となったとき、3 つのケースが存在する。1 つ目のケースは残っている財の数が 2 つの場合である。このケースでは、複数の参加者が希望個数 2 つと同数を提示し、誰かが財を 2 つ取得したラウンドが必ず存在する。その際は $3 \leq n'$ であるため、$u(\tilde{A})$ は最大で $\frac{P}{n'} * 2 = \frac{2P}{3}$ となる。2 つ目のケースは残っている財の数が 3 つの場合である。$n + 1 = m$ であるならば、複数の参加者が希望個数 2 つと同数を提示し、誰かが財を 2 つ取得したラウンドは存在しない。この場合提案アルゴリズムの $u(A)$ が 1 となるならば、最後の参加者ともう一人の参加者が同じ財 1 つを希望する場合である。この場合においては $n = m$ に関する証明の内容より、$u(\tilde{A}) \leq \frac{P}{2}$ となる。$n + 2 \leq m$ であるならば、複数の参加者が希望個数 2 つと同数を提示し、誰かが財を 2 つ取得したラウンドが必ず存在する。よって $u(\tilde{A})$ は最大で $\frac{2P}{3}$ となる。3 つ目のケースは残っている財の数が 4 つ以上の場合である。この場合は $2n \leq m$ で証明されているとおり、少なくとも 2 ポイント以上の価値の財を取得することが出来る。全ての場合において $u(\tilde{A}) \leq P$ であるため、$u(A)$ が 2 以上であるならば近似度は $\frac{2}{P}$ 以上となる。以上より、$n + 1 = m$ において近似度は $\frac{1}{\frac{P}{2}} = \frac{2}{P}$ 以上、$n + 2 \leq m \leq 2n - 1$ において近似度は $\frac{1}{\frac{2P}{3}} = \frac{3}{2P}$ 以上と証明される。

7 検証結果

提案アルゴリズムにおいて、各 1000 パターンのランダムな評価関数を用いて、$n = 3, 4, 5$ と $m = 3, 4, 5, 6, 7, 8, 9, 10$ の範囲で $P = 100$ としてコンピュータシミュレーションにより検証を行った結果が以下の表のとおりである。表の値は近似度の平均値である。

表 1 検証結果

	$n = 3$	$n = 4$	$n = 5$
$m = 3$	0.7498	none	none
$m = 4$	0.7440	0.5716	none
$m = 5$	0.7494	0.5611	0.4146
$m = 6$	0.7177	0.5623	0.4187
$m = 7$	0.7340	0.5534	0.4279
$m = 8$	0.7203	0.5520	0.4470
$m = 9$	0.7188	0.5830	0.4972
$m = 10$	0.7375	0.6013	0.5145

表を読み取ると、n の値が増えるほど近似度の値が減少する傾向がある。これはオンライン配分問題であるがゆえに、参加人数が増えるほど後の参加者の評価関数が未知のまま割り当てを実行しなければならない回数が増える為である。提案アルゴリズムと比較を行っている値は Weakly Immediately Envy-free を無視したオフライン全探索によって導いた値であることを考えると、提案アルゴリズムは良い結果を出すことが出来ていると考えられる。

謝辞

この研究は JSPS 科研費 JP26330019 の助成を受けたものです。

参考文献

[1] Walsh,T. : “Online Cake Cutting”, Algorithmic Decision Theory, COMSOC 2010, pp 292-305.

[2] Jonathan, G., Ariel, D. P. : “Spliddit: Unleashing Fair Division Algorithms”, ACM SIGecom Exchange, Vol.13, No.2, 2014, pp. 41-46.

[3] Procaccia, A. D., Wang, J. : “Fair enough: Guaranteeing approximate maximin shares”, 14th ACM Conference on Economics and Computation, 2014, pp. 675-692.

[4] Haris, A., Serge, G., Simon, M., Toby, W. : “Fair Assignment of Indivisible Objects under Ordinal Preference”, AAMAS ’14, 2014, pp. 1305-1312.

[5] Sylvain, B., Michel, L. : “Characterizing Conflicts in Fait Division of Indivisible Goods Using a Scale of Criteria”, AAMAS ’14, 2014, pp. 1321-1328.

[6] Richard, L., Evangelos, M., Elchanan, M., Amin, S. : “On approximately fair allocations of indivisible goods”, 5th ACM conference on Electronic commerce, 2004, pp. 125-131.

[7] Shimizu,K., Manabe,Y. : “An Allocation Algorithm of Indivisible Goods”, 10th APSITT, August 2015, pp 112-114.

一様な分布を持つパチンコ台の釘配置

北村直樹 , 川端 祐也, 泉 泰介

2016 年 7 月 1 日

パチンコは，日本で親しまれているギャンブルゲームである．本研究では，パチンコを数学的なモデルとして取り扱う．Akiyama，Ruiz ら [1] によって数学的なパチンコのモデルが提唱され，最新の研究では，Akitaya ら [2] によって発展的内容が取り扱われている．
問題設定として，正三角形が敷き詰められたグリッドの格子点上に釘を配置し，盤面の中央上部からボールを投入すると仮定する．ボールが釘に当たると，左右に等確率でボールが三角形の辺に沿って移動する．例を図 1 に示す．この問題の目的は，盤面上に釘を配置することによって盤面の最下層におけるボールの落下位置の確率分布を一様にすることである．

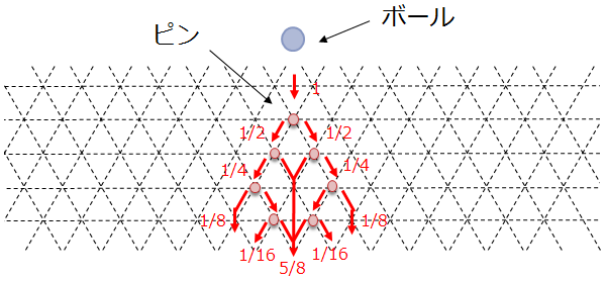


図 1: パチンコの数学的なモデル

Akitaya らの研究 [2] では，小さな誤差を含む任意の確率の一様分布を構成するアルゴリズムが存在することが明らかにされている．また同研究において，$n = 1, 2, 3, 4$ の場合における $\frac{1}{2^n}$ の一様確率分布を構成できることが明らかにされている．

本研究では，任意の整数 $n \geq 1$ において，$\frac{1}{2^n}$ の一様確率分布及びそれを構成するアルゴリズムが存在することを発見した．我々が提案するアルゴリズムでは，整数 $n \geq 5$ の場合における $\frac{1}{2^n}$ の一様確率分布について，$\frac{1}{2^{n-1}}$ の一様確率分布から構成する．先行研究 [2] において $n \leq 4$ の場合については構成できることが明らかになっているため，アルゴリズムを再帰的に実行することで目的の結果を得られる．

参考文献

[1] Jin Akiyama and Mari-Jo P. Ruiz. Pachinko math. In A Day ’s Adventure in Math Wonderland, 2008.

[2] Hugo A. Akitaya, Erik D. Demaine, Martin L. Demaine, Adam Hesterberg, Ferran Hurtado, Jason S. Ku, and Jayson Lynch. Pachinko. In CoRR, abs/1601.05706, 2016.

繁殖戦略解析のための考察

Consideration for the breeding strategy analysis

川邊 茂和　　真鍋 義文
Shigekazu Kawabe　　Yoshinumi Manabe

工学院大学大学院　工学研究科 情報学専攻
Department of Computer Science, Faculty of Informatics, Kogakuin University

1　あらまし

反応個体を用意し, その中で起きる化学反応の行き着いた状態からある特定の反応個体が存在するかしないかということを化学反応の行き着いた状態から判別する CRD(chemical reaction decider)[1] というプロトコルがある. このプロトコルでは化学反応のみを扱っていたので他の現象についても扱うことはできないのかと考えた. 本稿では, 進化ゲームに対する CRD と同様な定式化と分析の可能性について考察する. 本稿では一例として, 繁殖戦略を持つ鳥を題材としてモデル化を試みた結果を示す.

2　CRD

CRD は CRN(chemical reaction networks) に対する判定問題として定義されている.CRN は $N = (\Lambda, R)$ で定義される.Λ は $\Lambda = \{A, B, C\}$ のように表されどのような種類の化学式があるかを示している.R は $A+2B \to A+3C$ の場合 $\langle(1,2,0)(1,0,3)\rangle$ のように表すような化学物質の集合である.CRD は $D = (\Lambda, R, \Sigma, \Upsilon, \phi, s)$ で定義される.$k = |\Sigma|$ である.s は N^k であり, 初期状態で Σ の各化学物質が何単位存在するのかを表す.Σ は Λ の部分集合で, 入力とする化学物質の集合である.Υ は Λ の部分集合で, 反応終了時点での存在, 非存在を判定する化学物質の集合である.ϕ は反応終了時点での化学物質の存在, 非存在を YES,NO で返す関数である.

3　繁殖戦略

繁殖戦略の例としてジュズカケバトのつがいを作るが浮気を行うという繁殖の動き [2] を参考にしておりモデル化するにあたって問題を定式化するために本章の仮定をおいた. 繁殖戦略では, ジュズカケバトが年 2 回繁殖を行うため繁殖フェーズを 2 回行う, 次に死滅フェーズを行い最後に判定フェーズを行うという流れを 1 ループとし, 条件が満たされるまでループし続ける. つがい以外のメスに繁殖を持ちかける一定の確率を表す $O(\%), T(\%)$ がある. 戦略 A は確率 O, 戦略 B は確率 T に従って繁殖をする. 戦略 A, B の個体はそれぞれ $a_{n,m}, b_{n,m}$ のように表され,n は性別を表し 1 の場合オス,2 の場合メス,m はつがいのラベルというように表す. 上記のような一定の戦略を持つ A, B で行う繁殖フェーズでは以下のような条件に従うことになる.

1. 全オスはつがいまたはつがい以外に繁殖を持ちかける.
2. 繁殖を持ちかけた相手がつがいの場合は繁殖が必ず成功する.
3. 繁殖を持ちかけた相手がつがい以外の場合には繁殖できるかどうか, つがい以外に繁殖を持ちかけ失敗する確率 $R(\%)$ によって判定する.
4. 繁殖の持ちかけが成功した場合, 子をなしたかどうか確率 $P(\%)$ によって子をなす.
5. 子はオス, メス 1 羽ずつ生まれ, 繁殖をしたオスの方の戦略を引き継ぐ.

繁殖フェーズでなした子は, なされた段階の 3 ループ後から繁殖フェーズに加わる.

死滅フェーズでは, 繁殖戦略を開始した時の全体の個体数と同数になるように全体から通常個体 4, 子の個体 6 の割合でランダムに個体を死滅させていく.

判定フェーズでは A もしくは B どちらかの戦略の個体がすべて死滅しているかどうかを判定し死滅している場合はループを終え, 死滅していない場合には再度繁殖フェーズから始める.

O,T,R,P の値を変更してシミュレーションを実行した結果, 戦略 A の個体が 1000 ループ以内に全て死滅する条件の一つとして以下のものが求められた.

- $O \geq T$
- $O - T \geq 44$
- $R \geq 70$
- $P \geq 70$

上記の条件に従った繁殖戦略であればシミュレーションを使わずに戦略 O を持つ個体が淘汰されるということが式から判定できる. しかし $O = 60, T = 10, R \geq 70, P \geq 70$ という上記の条件に沿った状況で、A,B 各オスメスの個体数を 10 体の状態で計算した結果 0.34 の確率で 1 回の繁殖フェーズを終えた際に A のなした子の方が多いという状況になるという問題がある.CRD ではこのような確率は扱わないので式にどのように当てはめていくか, また条件に沿った他の値で計算した場合確率は変わるのかということについて考えていく.

4 おわりに

本稿では, 繁殖戦略とはどのような戦略モデルなのか, 繁殖戦略で片方の戦略が死滅するのはどのような条件なのかということについて考察した. 今後は繁殖戦略を CRD のような判定問題として解く場合に, 繁殖戦略は確率的な挙動を持つため, どのような確率で発生する絶滅をもって「絶滅が起きる」と判定するべきであるかを考える必要がある. また, 繁殖戦略を CRD のような判定問題として解く場合に,A の戦略が絶滅する条件で A のなした子の方が多いという例外がどれほど影響を及ぼすのかということを考えていくことが課題となる.

参考文献

[1] Ho-Lin Chen,Rachel Cummings,and David Doty,:pp.16-30“Speed faults in computation by chemical reaction networks,” DISC. 2014,LNCS 8784

[2] 高橋修,“多摩の自然と災害　－自然との共存をはかる－”⟨http://www.fsifee.u-gakugei.ac.jp/gp/pdf/H18/report/p166-169.pdf⟩ (2016/9/10 アクセス)

単貧民における最適戦略と必勝戦略に関する考察

九州大学経済学部経済工学科4年
木谷　裕紀
九州大学大学院経済学研究院
小野　廣隆

今日の流れ

1.大貧民と単貧民
2.単貧民における必勝手順
3.本研究
4.今後の展望

今日の流れ

1.大貧民と単貧民
2.単貧民における必勝手順
3.本研究
4.今後の展望

大貧民について

・大貧民はトランプで遊ぶカードゲームのひとつ.
「大富豪」,「階級闘争」などとも呼ばれる.
海外にも類似した遊びがある.
・カードを参加者にすべて配り,
手持ちのカードを順番に場に出して
早く手札をなくすことを競うゲーム.
・不完全情報多人数ゲーム
・電通大で毎年コンピューター大貧民大会が開催.

単貧民とは（1）

単貧民とは大貧民と類似した完全情報ゲーム.
西野（2007）が定義.
大貧民との主な違いは以下の三つ.

特殊ルールが一切なし.
１枚出しのみ.
手札は公開で行われる.

単貧民とは（2）

ゲームの開始	各プレイヤーに手札を与える.
各手番の行動	各手番のときに各プレイヤーは一枚のみ場に札を出す. 又はパスを選択することができる.
場に出せる札	場に出せるカードは 既に場に札がおかれている場合, その札より強くなければならない.
勝利条件	手札が先になくなった方が勝利.

本研究

本研究では

必勝判定問題…単貧民がどのプレイヤーが勝利することができるのか判定する問題
必勝戦略…勝つことのできるプレイヤーが必ず勝つための戦略

と定義.

7

本研究と単貧民

単貧民の必勝戦略,必勝判定に関しては
手札が10枚以下の場合（西野 2007）
互いの手札が同一で且つ同じ強さの札が 2 枚以下（木谷　小野　2016）のみ.

より一般の手札における最適アルゴリズムを
本研究では考察.

8

今日の流れ

1.大貧民と単貧民
2.単貧民における必勝手順
3.本研究
4.今後の展望

9

今日の流れ

1.大貧民と単貧民
2.単貧民における必勝手順
3.本研究
4.今後の展望

10

単貧民における必勝手順例（1）

プレイヤーAの手札 3,5,10
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
A→B→C→A…の順の時
プレイヤーAはどの手札からだすことによって
必ず勝つことができるか？

11

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと

12

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 ~~6~~,7,8,9
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがいずれかの札を出せば

13

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,~~7~~,8,9
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがいずれかの札を出せば

14

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,7,~~8~~,9
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがいずれかの札を出せば

15

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,7,8,~~9~~
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがいずれかの札を出せば

16

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,7,8,~~9~~
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがいずれかの札を出せば
プレイヤーCは場に札を出すことができないので

17

3から出した場合

プレイヤーAの手札 ~~3~~,5,~~10~~
プレイヤーBの手札 6,7,8,~~9~~
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがいずれかの札を出せば
プレイヤーCは場に札を出すことができないので
プレイヤーAは場にいかなる札があっても
10を出すことができる.

18

3から出した場合

プレイヤーAの手札 ~~3,5,10~~
プレイヤーBの手札 6,7,8,~~9~~
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがいずれかの札を出せば
プレイヤーCは場に札を出すことができないので
プレイヤーAは場にいかなる札があっても
10を出すことができる.
その後5を出して勝利

19

3から出した場合

プレイヤーAの手札 ~~3,5,10~~
プレイヤーBの手札 6,7,8,~~9~~
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがいずれかの札を出せば
プレイヤーCは場に札を出すことができないので
プレイヤーAは場にいかなる札があっても
10を出すことができる.
その後5を出して勝利…ですが

20

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがパスを選択した場合

21

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがパスを選択した場合
プレイヤーCは4を出して
手札をすべてなくすことができる.

22

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
プレイヤーAが3を場に出したあと
プレイヤーBがパスを選択した場合
プレイヤーCは4を出して
手札をすべてなくすことができる.
したがってプレイヤーCが勝つ場合がある.
→必勝ではない

23

3から出した場合

プレイヤーAの手札 ~~3~~,5,10
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
プレイヤーAが3を場に出した場合

→必勝ではない

24

5から出した場合

プレイヤーAの手札 3,~~5~~,10
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
5から出した場合

25

5から出した場合

プレイヤーAの手札 3,~~5~~,10
プレイヤーBの手札 6,7,8,~~9~~
プレイヤーCの手札 4
5から出した場合
プレイヤーBが如何なる手を選択しても
プレイヤーCは場に札を出すことができないが

26

5から出した場合

プレイヤーAの手札 3,~~5~~,~~10~~
プレイヤーBの手札 6,7,8,~~9~~
プレイヤーCの手札 4
5から出した場合
プレイヤーBが如何なる手を選択しても
プレイヤーCは場に札を出すことができないが
その手に対しプレイヤーAは10を出し

27

5から出した場合

プレイヤーAの手札 ~~3,5,10~~
プレイヤーBの手札 6,7,8,~~9~~
プレイヤーCの手札 4
5から出した場合
プレイヤーBが如何なる手を選択しても
プレイヤーCは場に札を出すことができないが
その手に対しプレイヤーAは10を出し
その後3を出すことによって
プレイヤーAは勝つことができる→必勝

28

10から出した場合

プレイヤーAの手札 3,~~5~~,~~10~~
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
10から出した場合
プレイヤーAがそのあと3,5どちらからだしても
プレイヤーBが6～9を出すことによって
プレイヤーBが先に手札をなくすことができる.

29

10から出した場合

プレイヤーAの手札 3,~~5~~,~~10~~
プレイヤーBの手札 ~~6,7,8,9~~
プレイヤーCの手札 4
10から出した場合
プレイヤーAがそのあと3,5どちらからだしても
プレイヤーBが6～9を出すことによって
プレイヤーBが先に手札をなくすことができる.

30

単貧民における必勝手順例（1）

プレイヤーAの手札 3,5,10
プレイヤーBの手札 6,7,8,9
プレイヤーCの手札 4
A→B→C→A…の順の時
プレイヤーAはどの手札からだすことによって
必ず勝つことができるか？

A. 5 から出すことによってのみ必勝

31

単貧民における必勝手順例（2）

プレイヤーAの手札 5,7,7,7,9
プレイヤーBの手札 4,6,8,10

A→B→A…の順の時
プレイヤーAはどの手札からだすことによって
必ず勝つことができるか？

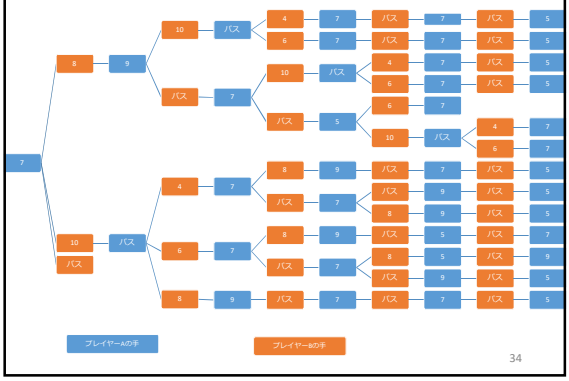
32

単貧民における必勝手順例（2）

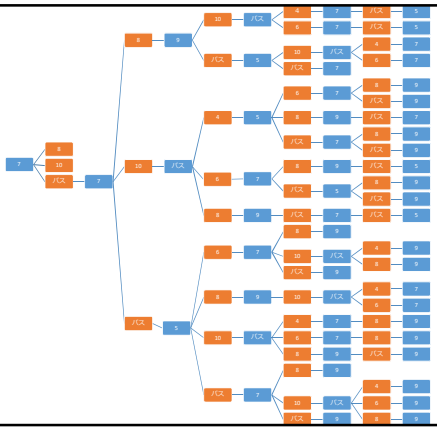
プレイヤーAの手札 5,~~7~~,7,7,9
プレイヤーBの手札 4,6,8,10

7から出すのが唯一の必勝手順

33



34



35

単貧民における必勝手順例（3）

プレイヤーAの手札 5,7,7,7,9
プレイヤーBの手札 4,6,8,10

どの札から出せばいいのか判断することは難しい

36

単貧民における最適手順

・このゲームにおいて必勝戦略を
求めるだけならゲーム木が有効
・しかし,枚数が多くなるとゲーム木のサイズ
は指数的に増大

37

単貧民における最適手順

・このゲームにおいて必勝戦略を
求めるだけならゲーム木が有効
・しかし,枚数が多くなるとゲーム木のサイズ
は指数的に増大

単貧民において必勝手順あるいは必勝判定を比較的早く解くアルゴリズムはないのか？

38

今日の流れ

1.大貧民と単貧民
2.単貧民における必勝手順
3.本研究
4.今後の展望

39

今日の流れ

1.大貧民と単貧民
2.単貧民における必勝手順
3.本研究
4.今後の展望

40

本研究

本研究では二人プレイヤー単貧民のうち
二種類以下の手からなる単貧民についてどちらのプレイヤーが必勝戦略をもつか示した上で
k種類の手からなる単貧民の必勝判定アルゴリズムの予想を示し,kが3以下のとき成立することを示す.

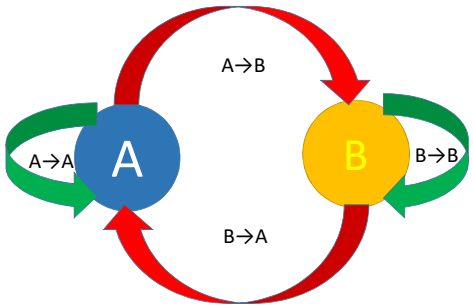
41

準備

先手プレイヤーをA,後手プレイヤーをB
各プレイヤーの手札を弱いほうから1～n
Aの強さが1番弱い札の所持札数はA1, 強さが2番目に弱い札をA2…とする.
Bの強さが1番弱い札の所持札数はB1, 強さが2番目に弱い札をB2…とする.
Aが場に出したカードに対しBが何らかの札を出すことを手番がAからBに変更すると表現する

42

準備（補題）



43

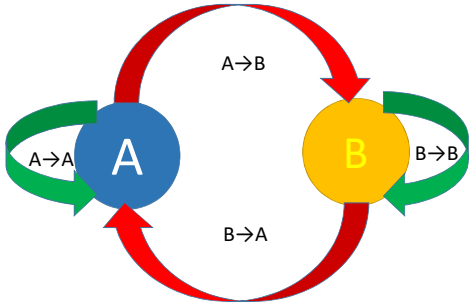
準備（補題）

ある単貧民1ゲームにおいてBからAに手番が変わる回数A→BとAからBに手番が変わる回数B→A
において以下は同値である

- A→B ≧B→Aが成立している
- その試合の勝者はAである.

44

準備（補題）



45

補題の証明

$a \geq b$ が成立しているならばその試合の勝者は A である.
その試合の勝者は A であるならば $a \geq b$ が成立している.
この両方を示せばよい
これは背理法で容易に証明可能

46

二種類以下の手からなる単貧民

一種類の手からなる単貧民
→明らかに先手必勝

二種類の手からなる単貧民からまず議論していく.

47

.二種類の単貧民の必勝戦略

強さ１の札が残り2枚以上
場に札が出すことが可能なできるだけ弱い札をだす.

上記以外
強い札から出す.

48

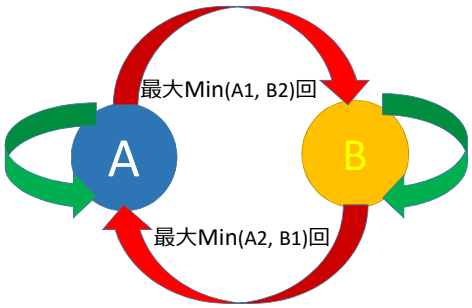
必勝判定問題

	Aの手札	Bの手札
2の札の枚数	A2	B2
1の札の枚数	A1	B1

	A1>B2	A1≦B2
B1>A2	Min(A2, B1) ≧Min(A1, B2) ↔Aが必勝	Min(A2, B1) +1≧ Min(A1, B2) ↔Aが必勝
B1≦A2	Min(A2, B1) > Min(A1, B2) ↔Aが必勝	Min(A2, B1) ≧ Min(A1, B2) ↔Aが必勝

49

証明の概要 1



50

補題の証明の概略

Min(*MaxA*→*B*)のためのAの行動は
強さ1の札をできる限り出さないことである.
また, 強さ1 の札は最後の一枚として出すことによって手番を相手に渡さない (でゲームを終了させる) ことができるので *A*1 ≤ *B*2 のとき = *Z* − 1 *Min*(*MaxA*→*B*)である.
*A*1 > *B*2 のときは (*A*1 − 1) 枚の札を出す前に B が 2 を出し終わることが可能となってしまうので *Min*(*MaxA*→*B*)= *Z* となる.

51

必勝判定問題

	Aの手札	Bの手札
2の札の枚数	A2	B2
1の札の枚数	A1	B1

	A1>B2	A1≦B2
B1>A2	Min(A2, B1) ≧Min(A1, B2) ↔Aが必勝	Min(A2, B1) +1≧ Min(A1, B2) ↔Aが必勝
B1≦A2	Min(A2, B1) > Min(A1, B2) ↔Aが必勝	Min(A2, B1) ≧ Min(A1, B2) ↔Aが必勝

52

準備

Aの手札集合をU(u∈U), Bの手札集合をV(v∈V)とする.
u,vはそれぞれ強さを示すラベルを持ちそのラベルをLu,Lvとする

53

手札が n 種類の単貧民

1.*G* = (*U*,*V*) をコピーしたグラフ *G*′ = (*U*′,*V*′) を作成する.
2.G において *Lu* > *Lv* の点に対し辺 E を作る.
3.辺が引かれた G に対し最大マッチングをとる
4.G′ において *Lv*′ > *Lu*′ の点に対し辺 E′ を作る.
5.辺が引かれた G′ に対しの最大マッチングをとる.
6.G における最大マッチングの数を UM, G′ における最大マッチングの数を VM とする.

54

N(Uf) に含まれる要素数を UM’ とする.

N(V’f) に含まれる要素数を VM’ とする.

$UM' > 1$ならばa=UM, $UM' = 1$ならばa=UM-1 UM’=0 ならばa=0 とする.

$V M' > 1$ ならば b=VM, $V M' = 1$ ならば b=VM-1 $V M' = 0$ ならば b=0 とする.

$a \leq b$ なら先手に必勝戦略が存在, $a < b$ なら後手に必勝戦略が存在.

55

今日の流れ

1.大貧民と単貧民
2.単貧民における必勝手順
3.本研究
4.今後の展望

56

今日の流れ

1.大貧民と単貧民
2.単貧民における必勝手順
3.本研究
4.今後の展望

57

まとめと今後の展望

二人単貧民の一部初期手札において必勝判定と必勝戦略が分かった.

また,一般化した場合のアルゴリズムを予想し3以下のとき成立することを示した.

58

三角形の最小遮光線長に関しての検討

早川　駿　　泉　泰介

名古屋工業大学大学院工学研究科
E-mail:earver.s@gmail.com, t-izumi@nitech.ac.jp

最小遮光線長を求める問題は，1916 年に Mazurkiewicz によって着手された歴史の古い問題である．この問題は，平面上の多角形と交わるようなすべての直線を遮るように衝立を引いたとき，その衝立の長さが最小となる引き方を明らかにする問題である．本研究では，与えられる図形が三角形の場合における最小遮光線長の検討を行う．三角形における最小遮光線長における既知のもっともよい上界は，フェルマー点を用いた作図による手法である．フェルマー点とは三角形の各頂点からの距離の合計が最小になる点であり，フェルマー点から三角形の各頂点へと引いた線分の 3 つの線分の集合は遮光線となる．一方で，一般の多角形に対して，外周総長の半分が遮光線長の下界となることが知られている．三角形に対しては，正三角形に対してのみこれよりもわずかに良い下界が知られているが，一般の三角形に対しては未解決である．本研究では，正三角形の下界向上手法の一般の三角形への拡張可能性について検討する．既存の手法においては，遮光線を構成する線分の角度を制限した場合における，既知の上界の最適性を示すことが証明の鍵となっている．正三角形の場合は，底辺を角度 0 と見た場合，既知の上界は角度 $pi/6$，$pi/2$，$2pi/3$ の線分のみから構成されているが，一般の三角形に対しても，（フェルマー点が三角形の内部にある場合は），既知の上界における遮光線の構成線分がなす角度はある角度 α を用いて $\pi/6+\alpha$，$\pi/2+\alpha$，$2pi/3+\alpha$ の形であらわすことができる．この類似性に注目して，既知の手法の一般の三角形への適用可能性を検討する．

クロストーク回避符号の符号化率と最大クリーク問題の関係

石井　大也　　泉　泰介

名古屋工業大学大学院工学研究科
E-mail:24115009@stn.nitech.ac.jp, t-izumi@nitech.ac.jp

近年，VLSI の小型化が，回路での情報伝達に悪影響をもたらす傾向にある．回路のサイズを縮小化するにはラインをより細くし，ライン同士の幅を狭くする必要があるが，それが結果的にバスでの情報伝達に悪影響をもたらす容量性クロストークを引き起こす．L_1, L_2, L_3 をオンチップの 3 つの配線とし，L_1, L_2, L_3 の間には容量性カップリングがあるとする．この仮定の下，発信元がバスを通じて発信している信号を $(L_1, L_2, L_3) = (0, 1, 0)$ から $(L_1, L_2, L_3) = (1, 0, 1)$ に瞬時に切り替える状況を考える．この時，L_1 と L_3 の急な電圧の上昇に伴い，容量性カップリングを原因とする L_2 上の信号の遅延が発生する．この現象は容量性遅延と呼ばれ，容量性カップリングによって引き起こされる悪影響の 1 つである．容量性遅延を防ぐための方法として，n 本のバスを n ビットの符号語とし，符号語の部分集合のみを送信可能とする方法が挙げられる．このような符号化は (p, q)-禁止符号として定式化されている．形式的には，p, q をそれぞれ長さ k のビット列とした場合，ビット長 n のステートレスな (二元) (p, q)-禁止符号 C とは，$\{0, 1\}^n$ の部分集合であり，C 中の任意の 2 語 w_1，w_2 について，および任意の $i \in [1, n]$ について,w_1 および w_2 の i ビット目から始まる長さ k の部分列が p および q にそれぞれ一致するようなことがないような語集合と定義される．長さ n の (p, q)-禁止符号化を用いてデータ送信を行った場合，一般には 2^n 通り存在する送信パターンのうちの一部分しか符号語として利用することができない．効率的な通信のためには，符号語の集合 C ができるだけ多くの語を含むような符号化を用いることが望ましい．n ビットで表される 2^n 個の語をそれぞれグラフの頂点と捉え，(01.10)-禁止符号において禁止されていない頂点間には辺を引き，禁止されている頂点間には辺を引かないグラフを作成した時，n ビットの符号語に対して $(01, 10)$-禁止符号で表現できる最大の語の数はこのグラフの最大クリークを求める問題に帰着出来る．本研究では，実用上有用な符号化である $(01, 10)$-禁止符号について，それに対応する遷移可能性グラフの最大クリークサイズを導出する式を与える．

リングネットワークにおける メモリ効率のよいモバイルエージェント均一配置アルゴリズム

Masahiro Shibata †, Hirotsugu Kakugawa†, and Toshimitsu Masuzawa†

†Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan
{m-sibata, kakugawa, masuzawa}@ist.osaka-u.ac.jp

Abstract. In this paper, we consider the uniform deployment problem of mobile agents in asynchronous unidirectional ring network. The uniform deployment problem requires agents to uniformly spread in the network. In this paper, we focus on the memory space per agent to solve the uniform deployment problem. We consider two problem settings. First, we consider agents without existence detection of agents at the same node. In this case, we show that each agent requires $\Omega(\log n)$ memory to solve the problem, and propose an algorithm to solve the uniform deployment problem with $O(\log n)$ memory per agent. Next, we consider agents with the existence detection. Then, the proposed algorithm reduces the memory space per agent to $O(\log k + \log\log n)$.

keyword: distributed system, mobile agent, uniform deployment, ring network, token, memory-efficient

1 Introduction

A *distributed system* consists of a set of computers (*nodes*) and communication links. As a promising design paradigm of distributed systems, (mobile) agent systems have attracted a lot of attention [1]. Agents can traverse the system and process tasks on each node, and hence they can simplify design of distributed systems [2]. The *rendezvous problem* (or the *gathering problem*) is a fundamental problem for cooperation of mobile agents. This problem requires all agents to meet at a single node. The rendezvous problem is considered in rings [3, 4], torus [5], trees [6], and arbitrary networks [7]. Some works assume that agents can use whiteboard on each node, and others assume that agents can use only tokens, which are markers that agents can release on nodes.

Another fundamental problem is *uniform deployment* (or *uniform scattering*), which requires all agents to spread uniformly in the networks. From a practical point of view, the uniform deployment is useful for the network management. For instance, if agents that can repair faulty nodes are deployed uniformly, such agents can quickly reach and repair faulty nodes after the faults are detected. If agents with database replicas are deployed uniformly, each node can quickly access the database. Hence, we can regard the uniform deployment problem as a kind of the resource allocation problem. The uniform deployment is interesting to investigate also from a theoretical point of view. The problem exhibits a striking contrast to the rendezvous: the uniform deployment aims to attain the symmetry of agent locations while the rendezvous aims to break the symmetry. It is well known that the symmetry breaking is difficult (and sometimes impossible) to attain in distributed systems. Consequently, it is interesting to clarify how easily the uniform deployment can be attained compared to the rendezvous.

As related works, Flocchini et al. [8] and Elor et al. [9] considered the uniform deployment in the ring networks, while Barriere et al. [10] considered it in the grid network. All of them propose uniform deployment algorithms under the assumption that agents are oblivious (or memoryless) but can observe multiple nodes within its visibility range. This assumption is often called a *Look-Compute-Move* model. On the other hand, Shibata et al. [11] considered the uniform deployment problem in asynchronous unidirectional ring networks for agents that have memory but cannot observe nodes except for their currently visiting nodes. They considered two problem settings: agents with knowledge of k and agents without of knowledge of k, where k is the number of agents. For the first (resp., second) model, they proposed two (resp., one) algorithms to solve the uniform deployment problem from any initial configuration such that all agents are in the initial states and placed at distinct nodes. This

Table 1. Results for agents with knowledge of k

	First result in [11]	Second result in [11]	Model 1	Model 2
Existence detection	Required	Required	Not Required	Required
Agent memory	$O(n \log k)$	$O(\log n)$	$O(\log n)$	$O(\log k + \log \log n)$
Time complexity	$O(n)$	$O(n \log k)$	$O(k^2 n)$	$O(kn^2 + kn^2 \log n)$
Total moves	$O(kn)$	$O(kn)$	$O(k^3 n)$	$O(k^2 n^2 + kn^2 \log n)$

n: the number of nodes, k: the number of agents

shows a striking difference from the rendezvous problem because the rendezvous problem is not solvable from some initial symmetric configurations.

In this paper, we consider the uniform deployment problem in unidirectional asynchronous ring networks. Similarly to [11], we consider agents that have memory but cannot observe nodes except for their currently visiting nodes. Each agent initially has a token and can release it on a visited node. After a token is released at some node, agents cannot remove the token. We assume that agents have knowledge of k. We consider two problem settings and focus on the memory space per agent to solve the problem. At first, we consider agents without existence detection of agents, that is, agents cannot detect whether there exists another agent or not at the current node. In this model, we show that each agent requires $\Omega(\log n)$ memory to solve the problem, and propose an algorithm to solve the uniform deployment problem with $O(\log n)$ memory per agent, $O(k^2 n)$ time, and $O(k^3 n)$ total moves. Hence, the proposed algorithm is asymptotically optimal in terms of memory space per agent. Next, we consider agents with the existence detection, that is, agents can detect whether there exists another agent at the current node or not, but cannot count the exact number of the agents. Then, our proposed algorithm reduces the memory requirement per agent to $O(\log k + \log \log n)$, but allows $O(kn^2 + kn^2 \log n)$ time and $O(k^2 n^2 + kn^2 \log n)$ total moves. To our best knowledge, this is the first research considering the trade-off about the existence detection. In Table 1, we compare our contributions with results for agents with knowledge of k in [11].

Due to limitation of space, we omit several proofs of theorems.

2 Preliminaries

2.1 System model

A *unidirectional ring network* R is defined as 2-tuple $R = (V, E)$, where V is a set of anonymous nodes and E is a set of unidirectional links. We denote by $n\,(= |V|)$ the number of nodes. Then, we define $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and $E = \{e_0, e_1, \ldots, e_{n-1}\}$ $(e_i = (v_i, v_{(i+1) \bmod n}))$. For simplicity, operations to an index of a node assume calculation under modulo n, that is, $v_{(i+1) \bmod n}$ is simply represented by v_{i+1}. We define the direction from v_i to v_{i+1} as the *forward* direction, and the direction from v_{i+1} to v_i as the *backward* direction. In addition, we define the i-th $(i \neq 0)$ forward (resp., backward) agent a'_h of agent a_h as the agent such that there are $i-1$ agents between a_h and $a_{h'}$ in the a_h's forward (resp., backward) direction.

In addition, the *distance* from v_i to v_j $(0 \leq i, j \leq n-1)$ is defined to be $(j - i) \bmod n$.

An agent is a state machine having an *initial state*. Let $A = \{a_0, a_1, \ldots, a_{k-1}\}$ be a set of $k\,(\leq n)$ anonymous agents. For simplicity, operations to an index of an agent assume calculation under modulo k. Since the ring is unidirectional, agents staying at v_i can move only to v_{i+1}. We assume that agents have knowledge of k. In addition, we assume that each agent initially has a *token* and can release it on a node that it is visiting. The token on an agent or a node can be realized in one bit that denotes existence of the token, and thus, the token cannot carry any additional information. Note that if agents are not allowed to have tokens, they cannot mark nodes in any way and this means that the uniform deployment problem cannot be solved. This is because if all agents move in a synchronous manner, they cannot get any information of other agents. After a token is released at some node, agents cannot remove the token. Note that since agents are anonymous, they cannot recognize the owner of each token. Moreover, we assume that agents move through a link in a FIFO manner, that is, when agent a_p leaves v_i after agent a_q leaves v_i, a_p reaches v_{i+1} after a_q reaches v_{i+1}. Note that such

Table 2. Meaning of each element in configuration $C = (S, T, P, Q)$

Element	Meaning and example
$S = (s_0, s_1, \ldots, s_{k-1})$	Set of agent states (s_i: the state of agent a_i)
$T = (t_0, t_1, \ldots, t_{n-1})$	Set of node states (t_i: the state of node v_i)
$P = (p_0, p_1, \ldots, p_{n-1})$	Set of agents staying at nodes (p_i: a sequence of agents staying at node v_i)
$Q = (q_0, q_1, \ldots, q_{n-1})$	Set of agents residing on links (q_i: a sequence of agents in transit from v_{i-1} to v_i)

FIFO assumptions are natural because agents are implemented as messages in practice, and FIFO assumptions of messages are natural and can be easily realized in distributed systems.

We consider two problem settings: agents *without existence detection* and agents *with existence detection*. Agents without existence detection cannot detect whether there exists another agent at the current node or not. On the other hand, agents with existence detection can detect whether there exists at least one agent at the current node or not. However, they cannot count the exact number of the agents. Each agent a_i executes the following three operations in an atomic action: 1) The agent reaches a node v (when a_i is in transit toward v), or it starts operations at v (when a_i is at v), 2) the agent executes local computation, and 3) the agent leaves v if it decides to move. For the case with existence detection, the local computation can depend on whether there exists another agent at v or not. Note that these assumptions of atomic actions are also natural because they can be implemented locally at a node if each node has a *buffer* that stores agents visiting the node and makes them execute processes in a FIFO order. We consider an *asynchronous* system, that is, the time for each agent to transit to the next node and to wait until execution of the next operation (when staying at a node) is finite but unbounded.

A (global) *configuration* C is defined as a 4-tuple $C = (S, T, P, Q)$ and the correspondence table is given in Table 2. The first element S is a k-tuple $S = (s_0, s_1, \ldots, s_{k-1})$, where s_i is the state (including the state to denote whether it holds a token or not) of agent a_i $(0 \leq i \leq k-1)$. The second element T is an n-tuple $T = (t_0, t_1, \ldots, t_{n-1})$, where t_i is the state (i.e., the number of tokens) of node v_i $(0 \leq i \leq n-1)$. The remaining elements P and Q represent the positions of agents. The element P is an n-tuple $P = (p_0, p_1, \ldots, p_{n-1})$, where p_i is a sequence of agents staying at node v_i $(0 \leq i \leq n-1)$. The element Q is an n-tuple $Q = (q_0, q_1, \ldots, q_{n-1})$, where q_i is a sequence of agents residing in the FIFO queue corresponding to link (v_{i-1}, v_i) $(0 \leq i \leq n-1)$. Hence, agents in q_i are those in transit from v_{i-1} to v_i.

We denote by $\mathcal{C}$ the set of all possible configurations. In *initial configuration* $C_0 \in \mathcal{C}$, all agents are in the initial state and placed at distinct nodes, and no node has any token. In addition, in C_0 the node where agent a stays is called the *home node* of a and denoted by $v_{HOME}(a)$. We assume that in C_0 agent a is stored at a buffer of its home node $v_{HOME}(a)$. This assures that agent a starts the algorithm at $v_{HOME}(a)$ before any other agent visits $v_{HOME}(a)$, that is, a is the first agent that takes an action at $v_{HOME}(a)$.

A *schedule* is an infinite sequence of agents. A schedule $X = \rho_0, \rho_1, \ldots$ is fair if every agent appears in X infinitely often. An infinite sequence of configurations $E = C_0, C_1, \ldots$ is called an *execution* from C_0 if there exists a fair schedule $X = \rho_0, \rho_1, \ldots$ that satisfies the following conditions for each h $(h > 0)$:

- If $\rho_{h-1} \in p_i$ holds for some i in a configuration C_{h-1}, the states of ρ_{h-1} and v_i in C_{h-1} are changed to those in C_h by a local computation of ρ_{h-1}. If ρ_{h-1} releases its token at v_i, the value of t_i increases by one. After this if ρ_{h-1} decides to move to v_{i+1}, ρ_{h-1} is removed from p_i and is appended to the tail of sequence q_{i+1}. If ρ_{h-1} decides to stay, ρ_{h-1} is still in p_i. The other elements in C_h are the same as those in C_{h-1}.

- If ρ_{h-1} is at the head of q_i for some i in a configuration C_{h-1}, ρ_{h-1} moves to v_i, that is, ρ_{h-1} is removed from q_i. Then, the states of ρ_{h-1} and v_i in C_{h-1} are changed to those in C_h by a local computation of ρ_{h-1}. If ρ_{h-1} releases its token at v_i, the value of t_i increases by one. After this

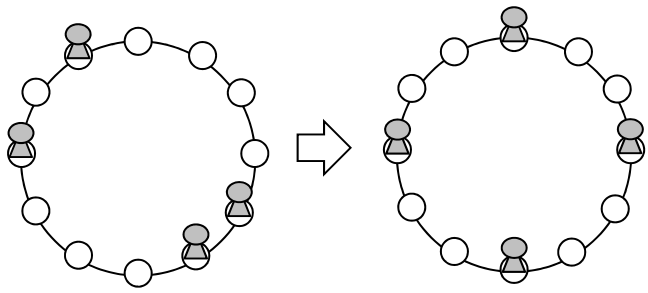


Fig. 1. An example of the uniform deployment ($n = 16, k = 4, d = 3$)

if ρ_{h-1} decides to move to v_{i+1}, ρ_{h-1} is appended to the tail of sequence q_{i+1}. If ρ_{h-1} decides to stay, ρ_{h-1} is inserted in p_i. The other elements in C_h are the same as those in C_{h-1}.

2.2 The uniform deployment problem

The uniform deployment problem in a ring network requires $k\,(\geq 2)$ agents to spread uniformly in the ring, that is, the distance between any two *adjacent agents* should become identical like Fig. 1. Here, we say two agents are adjacent when there exists no agent between them. However, we should consider the case that n is not a multiple of k. In this case, we aim to distribute the agents so that the distance d of any two adjacent agents should be $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$.

We consider the uniform deployment problem *without termination detection*. In this case, *a suspended state* is defined as follows. When agent a_i enters a suspended state, it neither changes its state nor leaves the current node v unless the local configuration of v (i.e., existence of another agent or the number of tokens for agents with existence detection, and the number of tokens for agents without existence detection) changes. The uniform deployment problem without termination detection allows all agents to stop in suspended states, which is also known as communication deadlock.

We define the uniform deployment problem without termination detection as follows.

Definition 1. *An algorithm solves the uniform deployment problem* without *termination detection if any execution satisfies the following conditions.*

- *All agents change their states to the suspended states in finite time.*
- *When all agents are in the suspended states, $q_i = \emptyset$ holds for any $q_i \in Q$ and each distance d of two adjacent agents satisfies $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$.* □

For the uniform deployment problem, we have the following lower bound of total moves.

Theorem 1. *When $k \leq pn$ holds for some constant $p\,(p < 1)$, a lower bound of the total moves to solve the uniform deployment problem (with or without termination detection) is $\Omega(kn)$ even for agents with existence detection.*

Proof. We assume for simplicity that $k \leq n/4$ holds and consider the initial configuration such that all agents stay in a quarter part of the ring like Fig. 2. In this case, the ring is divided into four quarter parts, and in the initial configuration, all agents are in the part a. To achieve the uniform deployment, $k/4$ agents need to move to the part c, the opposite part of a, and each of them must move at least $n/4$ times. Thus the total number of moves is at least $(k/4) \times (n/4) = kn/16$. This argument can be easily extended to any constant p $(p < 1)$ satisfying $k \leq pn$. □

Next, we evaluate the *time complexity* as the time required to achieve the uniform deployment. Since there is no assumption on time in asynchronous systems, it is impossible to measure the exact time. Instead we consider the *ideal time complexity*, which is defined as the execution time under the following assumptions: 1) The time required for an agent to move from a node to its neighboring

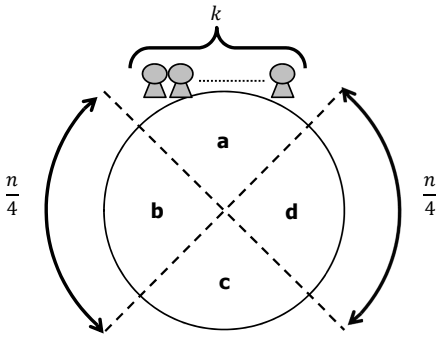


Fig. 2. The initial configuration to derive a lower bound $\Omega(kn)$ of the total moves

node or to wait until execution of the next action is at most one, and 2) the time required for local computation is ignored (i.e., zero)[1]. Note that these assumptions are introduced only to evaluate the time complexity, that is, algorithms are required to work correctly in asynchronous systems. In the following, we simply use terms "time complexity" and "time" instead of "ideal time complexity". Then, we can show the following theorem similarly to Theorem 1.

Theorem 2. *A lower bound of the time complexity to solve the uniform deployment problem (with or without termination detection) is* $\Omega(n)$. □

3 Agents without existence detection

In this section, we consider the uniform deployment problem for agents without existence detection. First, we show the memory space lower bound per agent for this problem, then we propose an algorithm to solve the problem.

3.1 Memory Space Lower Bound

For agents without existence detection, the following lower bound holds.

Theorem 3. *The lower bound of memory requirement per agents to solve the uniform deployment problem for agents without existence detection is* $\Omega(\log n)$. □

3.2 Proposed Algorithm

Next, we propose an algorithm to solve the uniform deployment problem with $O(\log n)$ memory space per agent, $O(k^2 n)$ time, and $O(k^3 n)$ total moves. From Theorem 3, we can show that the proposed algorithm is asymptotically optimal in terms of memory requirement per agent. For simplicity, we assume $n = ck$ for some positive integer c, and we can remove this assumption in Appendix A. The algorithm consists of three phases: selection phase, verification phase, and deployment phase. In the selection phase, agents select several nodes as candidates for *base nodes*, which are the reference nodes for the uniform deployment. In the verification phase, agents check whether the selected candidate nodes can be the base nodes. In the deployment phase, based on the base nodes, each agent determines a *target node* where it should stay and moves there.

3.2.1 Selection Phase In this phase, some of home nodes are selected as candidates of the base nodes, which are used as reference nodes for the uniform deployment. The selected base nodes should satisfy the following condition called the *base node condition*: 1) There exists at least one base node, 2) the distance between every pair of adjacent base nodes is identical, and 3) the number of home nodes

[1] This definition is based on the ideal time complexity for asynchronous message-passing systems [12].

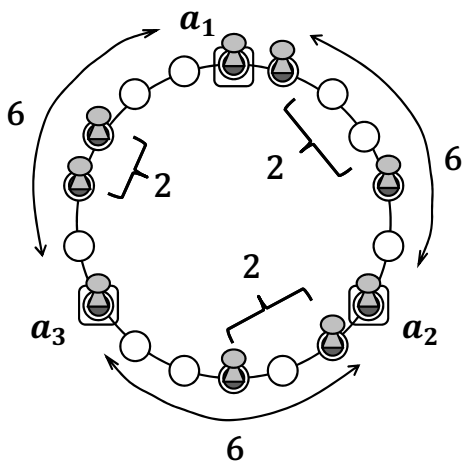


Fig. 3. An example of the base node condition ($n = 18, k = 9, d = 2$)

between every pair of adjacent base nodes is identical. The last condition is introduced to guarantee that the number of the selected base nodes is a divisor of k. For example, in the initial configuration like Fig. 3, distances from $v_{HOME}(a_1)$ to $v_{HOME}(a_2)$, $v_{HOME}(a_2)$ to $v_{HOME}(a_3)$, and $v_{HOME}(a_3)$ to $v_{HOME}(a_1)$ are all 6, and the number of home nodes between $v_{HOME}(a_1)$ and $v_{HOME}(a_2)$, $v_{HOME}(a_2)$ and $v_{HOME}(a_3)$, and $v_{HOME}(a_3)$ and $v_{HOME}(a_1)$ are all 2. Thus, $v_{HOME}(a_1)$, $v_{HOME}(a_2)$, and $v_{HOME}(a_3)$ satisfy the base node condition. Agents select such base nodes with $O(\log n)$ memory space .

The selection phase consists of several *sub-phases*. At the beginning of the selection phase, each agent releases its token at its home node. In the first sub-phase, each agent a_i travels around the ring. For explanation, we define the i-th gap of a token node as the distance to its i-th forward token node from it. During the travel, a_i finds the first gap of each token node and stores the minimum gap to d_{min}. It also stores the number of the token nodes with the gap of d_{min} to $nActive_{now}$. In addition, a_i counts the number of tokens it finds on the way to the nearest token node having the gap of d_{min}, and stores it to *pActive*. After the traversal, a_i determines the next behavior depending on the value of $nActive_{now}$. If $nActive_{now} = k$, distances between any adjacent two token nodes are the same, and this means that the initial configuration is already uniform. Hence, a_i stays at its home node $v_{HOME}(a_i)$. If $nActive_{now} = 1$, this means that only token one node has the first gap of d_{min}, and a_i can determine its target node based on the node. Hence, a_i enters to the deployment phase. Otherwise, a_i executes the next sub-phase. The pseudocode of the first sub-phase in the selection phase is described in Algorithm 1.

In the j-th sub-phase ($j \geq 2$), each agent travels several times around the ring and reduces the value of $nActive_{now}$ using the j-th gaps of the token nodes. Let V^j_{min} be the set of nodes satisfying the followings: 1) the first gap is d_{min}, and 2) the the $(j-1)$-th (resp., j-th) gap is d^{pre2}_{min} (resp., d^{pre1}_{min}). Here, d^{pre2}_{min} (resp., d^{pre1}_{min}) is the minimum $(j-1)$-th (resp., j-th) gap of token nodes in V^{j-1}_{min} (resp., V^j_{min}). Note that all nodes are included in V^0_{min} . In the j-th sub-phase, each agent measures j-th gaps from each node in V^{j-1}_{min}. At first, each agent a_i moves until it observes *pActive* token nodes. Then, a_i reaches the nearest nodes having the minimum $(j-1)$-th gap in the previous sub-phase. After this, a_i moves until it observes j tokens, and stores j-th gap to d_i, and returns to its home node $v_{HOME}(a_i)$. Agent a_i repeats such a behavior and memorizes the minimum j-th gap d^{now}_{min} starting from some node in V^{j-1}_{min} and its number $nActive_{now}$.

After finishing the movement, a_i determines the next behavior depending on the value of $nActive_{now}$. If $nActive_{now} = 1$, a_i can determine its target node based on the node having d^{now}_{min}. Hence, a_i changes to the deployment phase. Otherwise, there are several nodes having d^{now}_{min} and the nodes may be the subset of the base nodes. Thus, a_i changes to the verification phase. The pseudocode in the j-th sub-phase is described in Algorithm 2.

Algorithm 1 The behavior of agent a_i in the former part of the selection phase

Behavior of Agent a_i

/*first sub-phase*/
$phase = 1$, $t = 0$, $nActive_{now} = 1$,
release a token at its home node $v_{HOME}(a_i)$
move to the next token node and get the distance d_i between two token nodes
$d_{min} = d_i$, $t = t + 1$, $pActive = t - 1$
while $t \neq k$ **do**
  move to the next token node and get the distance d_{other} between two token nodes
  $t = t + 1$
  if $d_{other} = d_{min}$ **then** $nActive_{now} = nActive_{now} + 1$
  if $d_{other} < d_{min}$ **then** $d_{min} = d_{other}$, $nActive_{now} = 1$, $pActive = t - 1$
end while
if $nActive_{now} = k$ **then** terminate the algorithm // the initial configuration is already uniform
else if $nActive_{now} = 1$ **then** enter to the deployment phase // only one agent has d_{min}
else $phase = phase\ + 1$, enter to the second phase in Algorithm 2

Algorithm 2 The behavior of agent a_i in the latter part of the selection phase

Behavior of Agent a_i

/*j-th sub-phase ($j \geq 2$)*/
if $j = 2$ **then** $d_{min}^{pre1} = d_{min}$
else $d_{min}^{pre2} = d_{min}^{pre1}$, $d_{min}^{pre1} = d_{min}^{now}$,
$nActive_{pre} = nActive_{now}$, $nActive_{now} = 1$
move until it observes $pActive$ tokens // reach the nearest token node having d_{min}^{pre1}
move until it observes j tokens and get the j-th gap d_i
$d_{min}^{now} = d_i$
return to its home node $v_{HOME}(a_i)$
for $l = 2$ to $nActive_{pre}$ **do**
  if $j = 2$ **then** move to l-th node v_{min}^l such that the first gap from v_{min}^l is d_{min}^{pre1}
  if $j > 2$ **then** move to l-th node v_{min}^l satisfying following three conditions:
  - the first gap from v_{min}^l is d_{min}
  - the $(j-1)$-th gap from v_{min}^l is d_{min}^{pre2}
  - the j-th gap from v_{min}^l is d_{min}^{pre1}
  move to the next token node and get the distance dis
  $d_{other} = d_{min}^{pre1} + dis$ // get the distance between j token nodes
  if $d_{other} = d_{min}^{now}$ **then** $nActive_{now} = nActive_{now} + 1$
  if $d_{other} < d_{min}^{now}$ **then** $d_{min}^{now} = d_{other}$, $nActive_{now} = 1$,
  $pActive =$ (the number of tokens between $v_{HOME}(a_i)$ and v_{j-1}^l)
  return to its home node $v_{HOME}(a_i)$
end for
if $nActive_{now} = 1$ **then** enter to the deployment phase
else enter to the verification phase

3.2.2 Verification Phase In this phase, agents check if the candidate nodes selected in the selection phase can be the subset of the base nodes satisfying the base node condition. At first, each agent a_i moves until it observes $pActive$ token nodes so that a_i reaches the nearest candidate for the base nodes. After this, a_i moves d_{min}^{now} times and checks if 1) there exists a token at the end of the movement, and 2) there exist j tokens during the movement. Agent repeats such a behavior and travels once around the ring. If a_i satisfies the above conditions for each d_{min}^{now} movement, This means that set of the terminal nodes of all the d_{min}^{now} movement satisfies the base node condition. Hence, a_i enters to the deployment phase. Otherwise, the candidates for the base nodes selected in the previous selection phase violate

Algorithm 3 The behavior of agent a_i in the verification phase

Behavior of Agent a_i

1: /*verification phase*/
2: $identical = true$, $nodes = 0$
3: move until it observe $pActive_{now}$ tokens
4: **while** $nodes < n$ **do**
5: move d_{min}^{now} times and reach node v_x
6: $nodes = nodes + d_{min}^{now}$
7: **if** (there does not exist a token at v_x) $\vee$ (there do not exist j tokens during d_{min}^{now} movement) **then**
8: $identical = false$
9: **end if**
10: **end while**
11: return to its home node $v_{HOME}(a_i)$
12: **if** $nodes \neq n$ **then** $identical = false$
13: **if** $identical = false$ **then**
14: $phase = phase + 1$,
15: go to line 2 in Algorithm 2
16: **else**
17: let V_x be the set of nodes of each d_{min}^{now} movement terminal
18: let $v_x^i \in V_x$ be the node existing in the forward direction of $v_{HOME(a_i)}$ and the closet to $v_{HOME}(a_i)$ among V_x
19: $pActive =$ (the number of tokens from $v_{HOME}(a_i)$ to v_x^i)
20: enter to the deployment phase
21: **end if**

Algorithm 4 The behavior of agent a_i in the deployment phase

Behavior of Agent a_i

1: /*deployment phase*/
2: move until it observes $pActive$ tokens // reach its base node
3: move $pActive \times n/k$ times

the base node condition. Thus, to select new candidates, a_i returns to the selection phase and executes the next sub-phase. The pseudocode is described in Algorithm 3.

3.2.3 Deployment Phase In this phase, each agent determines its target node and moves there. From the base node condition, the base nodes are first selected as the target nodes. Hence, if a_i's home node $v_{HOME}(a_i)$ is one of the base nodes, $v_{HOME}(a_i)$ is a_i's target node and a_i stays there. Otherwise, a_i firstly moves until it observes $pActive$ tokens so that a_i reaches the nearest base node. After this, a_i moves $pActive \times n/k$ times and reaches its target node. When all agent move to their target nodes, the final configuration is a solution of the uniform deployment problem.

The pseudocode is described in Algorithm 4. We have the following theorem.

Theorem 4. *For agents without existence detection, our proposed algorithm solves the uniform deployment problem with $O(\log n)$ memory per agent, $O(k^2 n)$ time, and $O(k^3 n)$ total moves.* □

4 Agents with existence detection

In this section, we consider agents with existence detection, and propose an algorithm to solve the uniform deployment problem that reduces the memory requirement per agent to $O(\log k + \log\log n)$, but allows $O(kn^2 + n^2 \log n)$ time and $O(k^2 n^2 + kn^2 \log n)$ total moves. The algorithm consists of three phases: selection phase, collecting phase, and deployment phase. In the selection phase, agents select

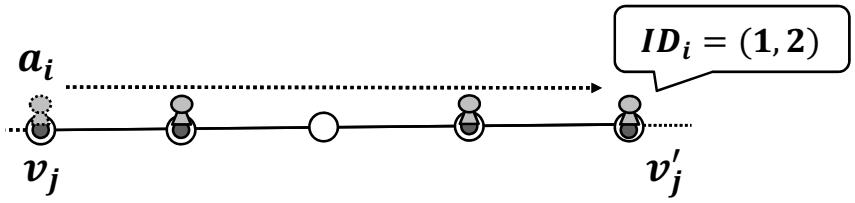


Fig. 4. An ID of an active agent a_i ($p = 3$)

several nodes as base nodes satisfying the base node condition similarly to Section 3.2. In the collecting phase, agents move in the ring so that they stay near one of the base nodes. In the deployment phase, agents determine their target nodes and move there.

4.1 Selection phase

In this phase, some of home nodes are selected as base nodes. When the selection phase is completed, each agent stays at its home node and knows whether its home node is selected as a base node or not. We call an agent a *leader* (but probably not unique) when its home node is selected as a base node, and call it a *follower* otherwise. The state of an agent is *active*, *leader* or *follower*. Active agents are candidates for leaders, and initially all agents are active. Once an agent becomes a leader or a follower, it never changes its state. In the following, we say that a node v is active (resp., a follower) when v is the home node of an active (resp., a follower) agent.

If agents have $O(\log n)$ memory, they can memorize a distance between any two token nodes. However in this section, agents cannot do this because they have only $O(\log k + \log\log n)$ memory. Agents overcome this problem by Chinese Remainder Theorem [13]. The theorem says that for two positive integer n_1 and n_2 ($n_1, n_2 < n$), if the sequence of remainders of the integer division by the sequence of the consecutive prime numbers $2, 3, 5, \ldots \log^2 n$ is the same, then $n_1 = n_2$ holds. Agents use this theorem and compute distance between several token nodes.

At the beginning of the algorithm, each agent a_i releases its token at its home node $v_{HOME}(a_i)$. The selection phase consists of at most $\lceil \log k \rceil$ sub-phases. At the beginning of each sub-phase, each agent stays at its own home node. During the sub-phase, if the agent is a follower, it stays at its home node. If the agent is active, it travels once around the ring and determines the next behavior using *IDs*. Concretely, the ID (not necessarily unique) of an active agent a_i is given as $(d_i^p, fNum_i)$, where d_i^p is remainder of the distance from its home node $v_{HOME}(a_i)$ to the next active node in the sub-phase, say v_{next}, by some prime number p, and $fNum_i$ is the number of follower nodes between $v_{HOME}(a_i)$ and v_{next}. For example in Fig. 4, when agent a_i moves from its home node v_j to the next active node v'_j, it visits four nodes and observes two follower nodes. If $p = 3$, a_i gets its own ID $ID_i = (1, 2)$. We compare two IDs by the lexicographical order: for $ID_1 = (d_1, fNum_1)$ and $ID_2 = (d_2, fNum_2)$, $ID_1 < ID_2$ if $(d_1 < d_2) \vee ((d_1 = d_2) \wedge (fNum_1 < fNum_2))$ holds. Each active agent decides whether it remains active or not using such IDs. Notice that in different sub-phases, the IDs of the same agent are different since the number of active agents is reduced in every sub-phase.

In the following, we explain the implementation of the sub-phase. In the sub-phase, each active agent a_i travels once around the ring. While travelling, a_i executes the followings:

1. Get its own ID $ID_i = (d_i^p, fNum_i)$:
   Agent a_i gets its own ID ID_i by moving from its home node $v_{HOME}(a_i)$ to the next active node v_{next} with counting the numbers of nodes (by modulo p) and follower nodes (Fig. 4). Since all active agents are traversing the ring and all follower agents are staying at their home nodes, a_i can detect its arrival at the next active node when it visits a node with a token but with no agent. Note that this statement holds even in asynchronous systems because active agents do not pass other active agents from the FIFO property of links and the atomicity of the execution.

Algorithm 5 The behavior of active agent a_i in the selection phase

Behavior of Agent a_i

```
1: /*selection phase*/
2: phase = 1, p = 2, identical = true, min = true
3: release a token at its home node v_HOME(a_i)
4: while (phase ≠ ⌈log k⌉) ∨ (p ≠ log^2 n) do
5:    move to the next active node and get its own ID ID_i = (d_i^p, fNum_i)
6:    if a_h is at v_HOME(a_i) then change its state to a leader state // only a_i is active
7:    move to the next active node and get ID ID_next = (d_next^p, fNum_next) of the next active agent
8:    if ID_next ≠ ID_i then  identical = false
9:    if ID_next < ID_i then  min = false // there exists an agent with smaller ID
10:   while a_i is not at v_HOME(a_i) do
11:      move to the next active node and get ID ID_other = (d_other^p, fNum_other) of the next active agent
12:      if ID_other ≠ ID_i then identical =false
13:      if ID_other < ID_i then min = false // there exists an agent with smaller ID
14:   end while
15:   if (identical = true) ∧ (p = log^2 n) then change its state to a leader state
      // all active agents have the same IDs for all target prime numbers
16:   if (identical = true) ∧ (p ≠ log^2 n) then p =(next prime number)
17:   if (identical = false) ∧ (min = false) ∨ (ID_i = ID_next) then change its state to a follower state
18:   else phase = phase + 1, p = 2, identical = true, min = true
19: end while
```

2. Get the ID $ID_{next} = (d^p_{next}, fNum_{next})$ of its next active agent:
   Similarly, with counting the numbers of nodes (by modulo p) and follower nodes, a_i moves from v_{next} to the next active node (i.e., the node with a token but with no agent). Then, a_i gets the ID of $a_i's$ next active agent and stores it to ID_{next}.

3. Compare ID_i with those of all active agents:
   During the traversal of the ring, a_i compares ID_i with IDs of all active agents one by one, and checks 1) whether ID_i is the minimum and 2) whether the IDs of all active agents are identical. To check these, agent a_i keeps boolean variables *min* ($min = true$ means ID_i is the minimum among ever-found IDs) and *identical* ($identical = true$ means that ever-found IDs are identical), and it updates the variables (if necessary) every time it finds an ID of another active agent.

When a_i completes the traversal, it determines its state for the next sub-phase. If $identical = true$ and $p = \log^2 n$ holds, this means that all active agents have the same IDs for all target prime numbers. In this case, a_i (and the other active agents) becomes a leader and completes the selection phase. If $identical = true$ and $p \neq \log^2 n$ holds, a_i travels once around the ring and compares IDs for the next prime number. If $identical = false$ holds, a_i remains active if $min = true$ and $ID_i < ID_{next}$ hold. The second condition means that, when active agents with the minimum ID appear consecutively, only one of them (or the last agent in the consecutive agents) remains active. This guarantees that the number of active agents is at least halved in each sub-phase. If a_i does not satisfy any of the above conditions, it becomes a follower. By repeating such sub-phase at most $\lceil \log k \rceil$ times, all the remaining active agents have the same IDs in some phase and they are selected as leaders so that their home nodes (or the base nodes) should satisfy the base node condition.

The pseudocode is described in Algorithm 5.

4.2 Collecting phase

In this phase, leader agents instruct follower agents so that they stay near one of the base node. Concretely, each leader agent a_i firstly moves to the token node v_j where another agent is staying (i.e., a follower agent). Then, a_i waits at v_j until the follower agent leaves v_j. After this, a_i leaves and moves

Algorithm 6 The behavior of leader or follower agent a_i in the collecting phase (v_j is the current node of a_i)

Behavior of Agent a_i

```
1: /*collecting phase*/
2: // the behavior of leader agents
3: if a_i is in the leader state then
4:    move to the next token node
5:    while there exists another agent at v_j do
6:       wait at v_j until there exists no another agent
7:       move to the next token node
8:    end while
9:    enter to the deployment phase
10: end if
11:
12: // the behavior of follower agents
13: if a_i is in the follower state then
14:    wait at v_j until there exists another agent
15:    move to the token node with no agent
16:    move to the next node
17:    while there exists another agent at v_j do
18:       move to the next node
19:    end while
20:    enter to the deployment phase
21: end if
```

to the next token node. Agent a_i repeats such a behavior until it reaches the next leader node, that is, the node with no agent. On the other hand, each follower agent a_i waits at the current node (i.e., its home node $v_{HOME}(a_i)$) until there exists another agent at v_j. When a_i detects the existence of another agent, a_i firstly moves to the nearest leader node (the token node with no agent). After this, a_i move the the next node and stays there if there does not exist another agent. Otherwise, a_i moves one more time. Agent a_i repeats such a behavior until it reaches a node with no agent. When all agents finish their movement, the agents are divided to groups (possibly only one group) each of which consists of $fNum + 1$ agents, and the agents in a group are deployed at consecutive nodes starting from a base node. The pseudocode is described in Algorithm 6.

4.3 Deployment phase

In this phase, leader agents instruct follower agents which node they should stay, and achieve the uniform deployment. The deployment phase consists of several sub-phases. In the l-th sub-phase, each leader agent a_i firstly moves to the node v_j where the ($fNum$)-th follower agent exists. Then, a_i waits at v_j until the follower agent leaves v_j. When the follower agent leaves v_j, a_i returns to its home node $v_{HOME}(a_i)$. Next, a_i moves to the node v'_j where the ($fNum - 1$)-th follower agent exists. Then, a_i waits at v'_j until the follower agent leaves v'_j. When the follower agent leaves v'_j, a_i returns to its home node $v_{HOME}(a_i)$. Agent a_i repeats such a behavior until it l follower agents move to the next nodes respectively. After this, a_i checks if the locations of agent from $v_{HOME}(a_i)$ to the next leader node are uniform or not using the Chinese Remainder Theorem. If the location is uniform, a_i returns to $v_{HOME}(a_i)$ and enters a suspended state. Otherwise, a_i executes the next sub-phase. We call this procedure *Check* and the pseudocode is described in Algorithm 8. If a_i executes a sub-phase such that $l = fNum$ and the locations are not uniform yet, a_i sets $l = 1$ and executes the next sub-phase. When all agent finish their movement, the final configuration is a solution of the uniform deployment problem.

The pseudocode of the deployment phase is described in Algorithm 7. We have the following theorem in Section 4.

Algorithm 7 The behavior of leader or follower agent a_i in the deployment phase (v_j is the current node of a_i)

Behavior of Agent a_i

```
1: /*collecting phase*/
2: // the behavior of leader agents
3: if a_i is in the leader state then
4:    nLoop = 1
5:    while true do
6:       for l = fNum_i to 1 do
7:          for m = fNum_i to l do
8:             move to the m-th node with another agent
9:             wait at v_j until there does not exist another agent
10:            return to its home node v_HOME(a_i)
11:         end for
12:         check(l,nLoop)
13:      end for
14:      nLoop = nLoop + 1
15:   end while
16: end if
17:
18: // the behavior of follower agents
19: if a_i is in the follower state then
20:   if there exist another agent then move to the next node
21: end if
```

Algorithm 8 Procedure *check*(l: integer, *nLoop*: integer)

Behavior of Agent a_i

```
1: uniform = true,
2: for p = 2, 3, 5, ..., log^2 n do
3:    d_1 = nLoop mod p, d_2 = (nLoop + 1) mod p
4:    move to the l-th node with another agent with counting the number t of tokens
5:    move until it observe (fNum_i + 1) − t tokens and
      get dis = (the distance between the two token nodes) mod p
6:    if (dis ≠ d_1) ∨ (dis ≠ d_2) then identical = false and break
7: end for
8: if uniform =true then
9:    // locations of agents from v_HOME(a_i) to the next leader node is uniform
10:   return to its home node v_HOME(a_i) and enter a suspended state
11: end if
```

Theorem 5. *For agents with existence detection, our proposed algorithm solves the uniform deployment problem with* $O(\log k + \log\log n)$ *memory space per agent,* $O(kn^2 + n^2 \log n)$ *time, and* $O(k^2n^2 + kn^2 \log n)$ *total moves.* □

5 Conclusion

In this paper, we proposed two memory-efficient uniform deployment problem in asynchronous unidirectional ring networks. For agents without existence detection, we showed that each agent requires $\Omega(\log n)$ memory, and proposed an algorithm to solve the uniform deployment problem with $O(\log n)$ memory space per agent, $O(k^2n)$ time, and $O(k^3n)$ total moves. For agents with existence detection, we proposed an algorithm to solve the uniform deployment problem with $O(\log k + \log\log n)$ memory space per agent, $O(kn^2 + n^2 \log n)$ time, and $O(k^2n^2 + kn^2 \log n)$ total moves. As a future

work, we want to analyze the memory lower bound for agents with existence detection. We conjecture that it is $\Omega(\log k + \log\log n)$. If the conjecture is correct, we can show that the second algorithm is asymptotically optimal in terms of memory requirement per agent.

References

1. D. Kotz S. R. Gray, G. Cybenko, A.R. Peterson, and D. Rus. D'agents: Applications and performance of a mobile-agent system. *Software: Practice and Experience*, 32(6):543–573, 2002.
2. D.B. Lange and M. Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, 1999.
3. E. Kranakis, D. Krozanc, and E. Markou. The mobile agent rendezvous problem in the ring, Synthesis Lectures on Distributed Computing Theory, Vol. 1. pages 1–122, 2010.
4. S. Kawai, F. Ooshita, H. Kakugawa, and T. Masuzawa. Randomized rendezvous of mobile agents in anonymous unidirectional ring networks, SIROCCO, LNCS, Vol. 7355. pages 303–314, 2012.
5. E. Kranakis, D. Krizanc, and E. Markou. Mobile agent rendezvous in a synchronous torus, LATIN, LNCS, Vol. 3887. pages 653–664, 2006.
6. P. Fraigniaud and A. Pelc. Deterministic rendezvous in trees with little memory, DISC, LNCS, Vol. 6950. pages 242–256, 2008.
7. A.Kosowski J. Czyzowicz and A. Pelc. How to meet when you forget: Log-space rendezvous in arbitrary graphs. *DISC*, 25(2):165–178, 2012.
8. P. Flocchini, G. Prencipe, and N. Santoro. Self-deployment of mobile sensors on a ring. *Theoretical Computer Science*, 402(1):67–80, 2008.
9. E. Yotam and B. M. Alfred. Uniform multi-agent deployment on a ring. *Theoretical Computer Science*, 412(8):783–795, 2011.
10. E. Mesa-Barrameda L. Barriere, P. Flocchini and N. Santoro. Uniform scattering of autonomous mobile robots in a grid. *International Journal of Foundations of Computer Science*, 22(03):679–697, 2011.
11. M. Shibata, T. Mega, F. Ooshita, H. Kakugawa, and T.Masuzawa. Uniform deployment of mobile agents in asynchronous rings, proceedings of the 2016 acm symposium on principles of distributed computing. pages 415–424. ACM, 2016.
12. G. Tel. Introduction to distributed algorithms. Cambridge university press, 2000.
13. Tom M Apostol. *Introduction to analytic number theory*. Springer Science & Business Media, 2013.

Appendix

A The uniform deployment for the case of $n \neq ck$

To remove the restriction of $n = ck$ imposed in Section 3.2, only the parts for determining the target nodes and for moving to a target node should be modified. In the case that n is not a multiple of k, the distance between some adjacent target nodes should be $\lceil n/k \rceil$ or $\lfloor n/k \rfloor$.

The target nodes should be determined by each agent so that the decisions of different agents should be identical. Since all the agents recognize the same nodes as the base nodes, the common target nodes can be determined using the base node as a reference node: Let b be the number of the base nodes, and $r = n \bmod k$. The distance of every pair of adjacent base nodes is identical even in the case of $n \neq ck$, and is $n/b = (\lfloor n/k \rfloor \times k + r)/b = \lfloor n/k \rfloor \times k/b + r/b$ (notice that k/b and r/b are integers). This implies that we should select $k/b - 1$ target nodes between two adjacent base nodes so that the first r/b intervals between adjacent target nodes should be $\lceil n/k \rceil$ and others should be $\lfloor n/k \rfloor$. With considering the above, each agent can determine its own target node by local computation so that all the agents can spread over the ring to achieve the uniform deployment.

認証機能付き白板を用いたビザンチン故障耐性を持つモバイルエージェント集合アルゴリズム

土田　将司†　大下　福仁†　井上美智子†

† 奈良先端科学技術大学院大学情報科学研究科　〒 630-0192 奈良県生駒市高山町 8916-5
E-mail: †{tsuchida.masashi.td8,f-oosita,kounoe}@is.naist.jp

あらまし　本稿では，ビザンチン環境において，モバイルエージェントを一つのノードに集合させるアルゴリズムを提案する．提案するアルゴリズムは，各エージェントが固有の ID を持ち，各ノードには認証機能付きの白板があり，ビザンチンエージェントが高々 f 個存在するネットワークで，全ての正常エージェントを $O(fm)$ 時間で集合させることができる (m はネットワーク中の辺の数)．従来手法は白板を用いずに $\tilde{O}(n^9\lambda)$ 時間 (n はノード数，λ は最長 ID の長さ) で集合を実現しており，提案手法は白板を用いることで大きく集合に要する時間を削減する．

キーワード　モバイルエージェント，集合問題，ビザンチン故障

Gathering of mobile agents which has Byzantine failure tolerance with authenticated whiteboards

Masashi TSUCHIDA†, Fukuhito OOSHITA†, and Michiko INOUE†

† 8916-5 Takayama-cho, Ikoma, Nara 630-0192, JAPAN
E-mail: †{tsuchida.masashi.td8,f-oosita,kounoe}@is.naist.jp

Abstract　We propose an algorithm for the gathering problem of mobile agents in Byzantine environments. The proposed algorithm can make all correct agents to meet at a single node in $O(fm)$ time (m is the number of edges) under the assumption that each agent has unique ID and behaves synchronously, each node is equipped with an authenticated whiteboard, and at most f Byzantine agents exist. Since the existing algorithm achieves gathering without a whiteboard in $\tilde{O}(n^9\lambda)$ time, where n is the number of nodes and λ is the length of the longest ID, our algorithm shows a whiteboard can significantly reduce the time for the gathering problem in Byzantine environments.

Key words　Mobile Agent, Gathering Problem, Byzantine Fault

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て，ノード間の大量のデータ通信やノードの保守，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，ネットワークを自律的に移動し，さまざまなタスクを行うソフトウェアが注目を集めている [6]．エージェントはそれ自身が情報の収集，解析を行いながら移動するため，ノード自体は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させるための基本的タスクの一つとして集合問題が考えられている．集合問題とは，初期状況においてネットワーク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一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として，これまでに多くの研究が行なわれている [4] [7]．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，また，解決可能な場合は解決に要するコスト（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，匿名性，ノード上のメモリ (以下，白板) の有無，乱択の有無，トポロジなどの異なる様々な環境において，多くの研究がなされている．

ノードに情報を残せない場合，すなわち，白板が存在しない場合，同期的に動作する 2 エージェントに対する決定性アルゴリズムが多数研究されている．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 [2], [3], [8], [9] 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を実現するアルゴリズムを提案している．ノード数を n，エージェントの最小 ID の長さを l，2 エージェントの起動時刻の差を τ とするとき，Dessmark ら [3] は，任意のグラフに対して，n，l，τ に関する多項式時間で集合を実現するアルゴリズムを提案している．Kowalski と Malinowski [8] と Ta-Shma と Zwick [2] は，集合に要する時間を改善し，τ に依存しない時間で集合を実現するアルゴリズムを提案している．また，Miller と Pelc [9] は，集合までに要する移動数と時間について，そのトレードオフを解析している．一方，文献 [10]〜[12] は，エージェントに固有の ID が存在しない場合について研究している．この場合，グラフやエージェントの初期配置によっては集合が不可能であるため，集合可能なグラフ，初期配置に限定してアルゴリズムを提案している．文献 [10], [11] は木に対して，文献 [12] は任意のグラフに対して，少ないメモリで集合を実現するアルゴリズムを提案している．

ノードに白板が存在する場合，集合に要する時間を大きく削減できることが知られている．例えば，エージェントが固有の ID を持つ場合，開始ノードの白板に ID を書き込み，白板を用いてネットワークを一周する（例えば，文献 [14] など）ことで，全てのエージェントが全ての ID を知ることができる．そのため，最小 ID のエージェントの開始ノードに集まることで，集合を実現できる．一方，エージェントが固有の ID を持たない場合は，白板を用いて，かつ，乱数を用いたとしても自明ではない．文献 [15] では，白板が存在するリングネットワークにおいて，終了検知の有無と乱択集合アルゴリズムの実現可能性の関係について議論されている．

また，エージェントはネットワークを移動しながら動作を行うため，クラッキングされる可能性がある．クラッキングされ，本来とは異なる動作をするエージェントをビザンチンエージェントと呼ぶ．ビザンチンエージェントを考慮した集合アルゴリズムは，文献 [1], [13] で研究されている．これらの文献では，固有の ID を持ち，同期的に動作するエージェントを考え，ノードには白板が存在しないとしている．また，ビザンチンエージェントを，弱ビザンチンエージェントと強ビザンチンエージェントに分けて定義している．弱ビザンチンエージェントは，自身の ID を偽ることができないが，その他の任意の動作を行なうことができる．強ビザンチンエージェントは，自身の ID を偽ることも含めて，任意の動作を行なうことができる．文献 [1] では，弱ビザンチンエージェントまたは強ビザンチンエージェントを仮定し，エージェントがノード数 n を既知の場合と未知の場合について，集合を実現するアルゴリズムを提案している．弱ビザンチンエージェントでノード数 n が既知の場合については，正常な 2 エージェントを $P(n,l)$ 時間（l は 2 エージェントの最小 ID の長さ）で集合させるアルゴリズムが存在するとき，$4n^4 \cdot P(n,\lambda)$ 時間（λ は全エージェントの最長 ID の長さ）で全ての正常エージェントを集合させられることを示している．文献 [2] のアルゴリズムを用いると，$P(n,l) = \tilde{O}(n^5 l)$ であることから，$\tilde{O}(n^9\lambda)$ 時間で集合を実現できる．弱ビザンチンエージェントでノード数 n が未知の場合は，正常エージェントが少なくとも $f+2$ 個必要であり，また，正常エージェントが $f+2$ 個存在すれば多項式時間で集合を実現できることが示されている．つまり，弱ビザンチンエージェントに関しては，集合の実現に必要な正常エージェントの数について，タイトな結果が示されている．一方で，強ビザンチンエージェントについては，ノード数 n が既知の場合，未知の場合のそれぞれについて，集合を実現する（多項式時間でない）アルゴリズムが提案されている．しかし，集合の実現に必要な正常エージェントの数についてタイトな結果は示されていない．文献 [2] では，強ビザンチンエージェントについても，集合の実現に必要な正常エージェントの数について，タイトな結果が示されている．

1.3 本研究の成果

本報告では，各ノード上に白板が存在すると仮定し，集合の高速化を目指す．ここで白板とは，ノード上に存在するメモリのことであり，各エージェントはこの領域に情報を残すことができる．しかし，ビザンチンエージェントが白板上の全ての情報を消すことができる場合，正常エージェントは白板の情報を参照することができず，白板の効果を期待できない．そこで，本研究ではシステムに認証機能が存在すると仮定し，白板上に各エージェント専用の領域を設ける．すなわち，各領域に情報を書き込めるエージェントは特定の一個のみとする．一方，読み込みに関しては，白板上の全情報を全エージェントが読み込めるとする．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できるとする．

白板の存在を仮定し，かつ，ビザンチンエージェントを考慮した集合アルゴリズムはこれまでに研究されていない．しかし，本研究で仮定する認証機能付き白板を用いることで，2 エージェントの集合に要する時間を削減できるため，文献 [1] のアルゴリズムをより短い時間で実行できる．すなわち，認証機能付き白板を用いることで，各エージェントは，DFS(後述するアルゴリズム) による $O(m)$ 時間の巡回のあと，一周 $O(n)$ 時間で全ノードを巡回できる [14]．これに Deessmark ら [3] のアルゴリズムを適用することで，2 エージェントの集合を $P(n,l) = O(nl)$ 時間で実現できる．よって，弱ビザンチンエージェントが高々 f 個存在する場合，$O(n^5\lambda)$ 時間で集合を実現できる．

本稿では，より短時間で集合を実現するために，新たなアルゴリズムを提案する．提案アルゴリズムは，同期ネットワークにおいて，エージェント数 k が未知，ノード数 n が未知，エージェントに固有な ID があり，弱ビザンチンエージェントが高々 f 個存在する環境で，白板を用いることで $O(fm)$ 時間で集合をできる．すなわち，文献 [1] のアルゴリズムに認証機能付き白板を用いた場合よりも集合に要する時間を大きく削減できる．

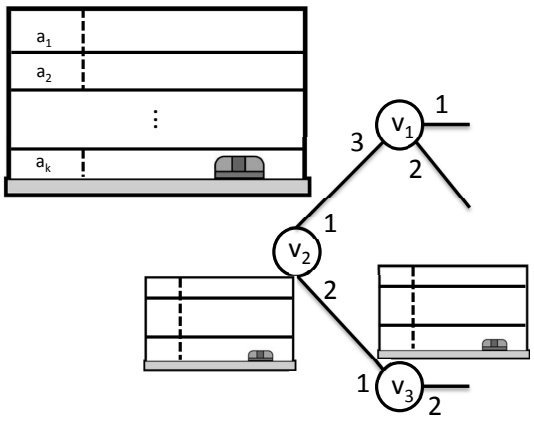


図 1　分散システム

2. 諸　定　義

2.1　分散システム

分散システムは，ノード集合を V，辺集合を E として，グラフ $G=(V,E)$ で表される (図 1)．ネットワークのノード数を $n=|V|$ 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 に接続するリンクは関数 $\lambda_v:\{(v,u):u\in N_v\}\to\{1,2,\cdots,d(v)\}$ によって局所的に一意にラベル付けされており，辺 (v,u) と $(v,w)(u\neq w)$ について，$\lambda_v(v,u)\neq\lambda_v(v,w)$ が成り立つ．$\lambda_v(v,u)$ をノード v における辺 (v,u) のポート番号と呼ぶ．

本稿では，ノードに ID がないシステムを考える．各ノード $v\in V$ には後述するエージェントが情報を残せる白板が存在する．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 を持ち，その長さは $O(\log k)$ ビットである．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同期的に動作する．すなわち，エージェントが隣接するノードに移動する際に要する時間は，全ての正常エージェントで同一である．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．

2.3　署　　名

各エージェントは署名関数を保持しており，自身の ID を用いて値に署名を追加できる．任意の値 x に対してエージェント a_i の署名関数である $Sign_i(x)$ によって出力される値を $\langle x\rangle:ID_i$ と表す．値 x に対してエージェント a_i は $Sign_i(x)$ のみ実行することができ，$Sign_j(x)$ $(i\neq j)$ を実行できない．したがってエージェント a_i が $Sign_i(x)$ にて作成した値 $\langle x\rangle:ID_i$ は，エージェント a_i が作成した値であると保証することができる．

また，エージェントは署名された値にさらに署名を行うことができる．既に署名の行われている値 $x=\langle value\rangle:id_1:id_2:\cdots:id_j$ にエージェント a_i が新たに署名を行ったとき，その出力値を $\langle value\rangle:id_1:id_2:\cdots:id_j:ID_i$ と表す．

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，他エージェントと同期せずに，任意の動作を行うことができる．ただし，自身の ID を変更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値 x に対する $Sign_j(x)$ $(i\neq j)$ を実行することが出来ず，$\langle x\rangle:ID_j$ $(i\neq j)$ を作成することはできない．ネットワークに存在するビザンチンエージェントの個数の上限を f で表す．

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が一つのノードに集合し，終了宣言を行う問題である．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．終了宣言を行ったエージェントは，その後動作することはない．

アルゴリズムの性能を示すため，初期状況から全ての正常エージェントが終了を宣言するまでに要する時間を評価する．本稿ではエージェントがノード間を移動するときに要する時間を 1 単位時間とし，その他の内部計算に要する時間は無視する．

3. アルゴリズム

本アルゴリズムは同期ネットワーク上でビザンチンエージェントの動作にかかわらず，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．

ビザンチンエージェントがない場合，各エージェント a_i はアルゴリズム実行開始時に，ノード v からアルゴリズムを開始したことを表す情報 (以下，スタート地点情報) を v の白板に記述し，ネットワークを深さ優先探索 (以下，DFS) を行いながら

一周する．ネットワークを一周する際，エージェント a_i は他エージェントが残したスタート地点情報を収集する．一周後，最小 ID のエージェントがスタート地点情報を残したノードを集合場所として定め，そのノードへ移動し終了宣言を行う．

しかし，ビザンチンエージェントが存在する場合，ビザンチンエージェント a_b は一部のエージェントにのみ a_b のスタート地点情報が見えるように，書き込み・削除を行うことができる．ビザンチンエージェント a_b の ID_b が最小 ID であり，一部のエージェントにのみ a_b のスタート地点情報が見える場合，それらのエージェントだけが異なるノードを集合場所として算出してしまう．そこで本アルゴリズムではコンセンサスアルゴリズムを応用し，全ての正常エージェント間で同一のスタート地点情報を共有し，同一の集合場所を選択することを目指す．

提案アルゴリズムではグラフ探索アルゴリズムである DFS と，メッセージパッシングシステムにおいて提案されたコンセンサスアルゴリズムを応用して用いる．まず，DFS 及びコンセンサスアルゴリズムのエージェント環境への応用を説明する．

3.1 DFS

DFS(Depth First Search) とは深さ優先探索のことであり，各ノードにポート番号が割り当てられているグラフ中を探索するグラフ理論の手法である．この手法は，分散システムでエージェントが全てのノードを訪れる手法として応用できる．エージェントは，新しいノードに訪れると印を施し，ポート番号の小さい辺から順に探索する．既に印が施されているノードに訪れると，前のノードに戻る．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等しい．そのため，詳細は後述するが，各エージェントが各々の時間差で，任意のノードから DFS を開始したとしても，DFS を終了するまで各々の時間差は引き継がれる．すなわち，DFS を連続して行い，グラフ中を周回することで，各エージェントが任意のタイミングでアルゴリズムを開始したとしても周回単位で同期している状態を作ることができる．

3.2 コンセンサスアルゴリズム

コンセンサスアルゴリズムとは，複数のプロセス間で合意値を算出するアルゴリズムである．文献 [5] は同期的に動作する k プロセスが完全ネットワークを構成している場合に対して，ビザンチン故障に耐性を持つコンセンサスアルゴリズムを提案している．文献 [5] のアルゴリズムでは，高々 f 個のプロセスがビザンチン故障を起こしても，全ての正常なプロセスが同じ値に合意できる．より詳細には，全ての正常なプロセスの提案した値の集合を X_c とすると，正常なプロセス全てが同じ集合 $X \supseteq X_c$ を持つことができ，これにより同じ値 (例えば，$min(X)$ など) に合意できる．このアルゴリズムで合意に要する同期ラウンド数は $f+1$ である．

本アルゴリズムでは，各ノード上に仮想プロセスが存在するとし，それらのプロセスをエージェントがノードに訪れた際にシミュレートすることで，コンセンサスアルゴリズムを実現する．仮想プロセスのイメージを図 2 に示す．各ノード v には各

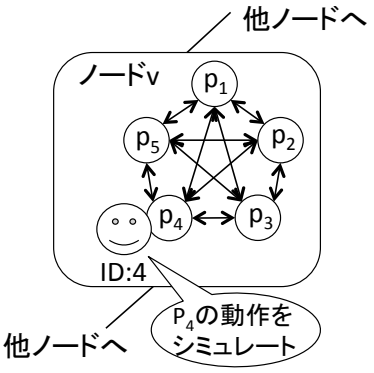


図 2　仮想プロセス

エージェント a_i に対応する仮想プロセス p_i が存在するとし，k 個のプロセス $p_1, p_2, \cdots, p_k$ が完全ネットワークを構成しているとする．エージェント a_i がノード v に訪れた際，仮想プロセス p_i の動作をシミュレートし，コンセンサスアルゴリズムを実現する．

各ノードにおけるコンセンサスアルゴリズムでは，入力を各エージェントがそのノードから集合アルゴリズムを開始したかどうか，出力をそのノードでアルゴリズムを開始したエージェントの集合とする．コンセンサスアルゴリズムを解くため，プロセスを $f+1$ ラウンド分シミュレートする必要がある．提案アルゴリズムでは，DFS を $f+1$ 回繰り返し，各エージェントが各ノードを $f+1$ 回訪れることを実現する．

3.3 提案アルゴリズムの概要

提案アルゴリズムでは，各ノードごとに，スタート地点情報を入力値として 3.2 のコンセンサスアルゴリズムを行う．これにより各ノード v ごとに正常エージェントが同じスタート地点情報の集合 X_v を得ることができ，$X_{all} = \bigcup_{v \in V} X_v$ を得ることができる．さらに X_{all} からビザンチンエージェントが複数ノードに入力値を与えていることを考慮して，重複のしていない，最小 ID のエージェントのスタート地点を集合場所として選択することで，集合が可能となる．

しかしコンセンサスアルゴリズムで合意を取るプロセスは同期している必要がある．エージェントは同期しているが，a_i がノード v を訪れるタイミング，つまり仮想プロセス p_i が動作するタイミングは，同期していない．そのため提案アルゴリズムでは，各仮想プロセスがコンセンサスアルゴリズムを実行できるように同期動作の実現も行う．

3.4 アルゴリズム

i)　main 関数

本アルゴリズムのメイン関数 main() を Algorithm 1 に示す．各エージェントは初期状況で任意のノード $v(\in V)$ に存在する．$v.wb[ID_i]$ はエージェント a_i が書き込める v 上の白板の領域である．初期状況では各エージェントは待機状態であり，任意のタイミングで実行状態に移行する．実行状態に移ったエージェント a_i は，consensus 関数を一度実行し，自身が存在するノー

Algorithm 1 main( )

```
1: —ノード v の白板の変数—
2: var v.wb[ID_i].T
3: var v.wb[ID_i].W
4: var v.wb[ID_i].round
5: var v.wb[ID_i].from_port
6: var v.wb[ID_i].explored_port
7: var v.wb[ID_i].node_num
8: —エージェント a_i の変数—
9: var a_i.node_num = 0                  //ノード数のカウント
10: var a_i.all_edge_num = 0             //辺の数のカウント
11: var a_i.r = 0                        //ラウンド数のカウント
12: var a_i.W = ∅                        //集合地点候補を格納
13: ————————————————
14: consensus()
15: for a_i.r = 1 to f + 1 do
16:     a_i.node_num = 1
17:     a_i.all_edge_num = 0
18:     DFS(null)
19:     wait a_i.all_edge_num × 2
20: end for
21: a_i.W のうち，candidate が重複しているもの全てを削除
22: a_i.W のうち，candidate が最小値のノードへ移動
23: 終了を宣言
```

Algorithm 2 DFS(f_port)

```
1: 待機状態のエージェントが存在する場合，実行状態へ移行させる
2: if v.wb[ID_i].round ≠ a_i.r then
3:     v.wb[ID_i].round = a_i.r
4:     v.wb[ID_i].from_port = f_port
5:     if f_port = null then
6:         v.wb[ID_i].explored_port = ∅
7:     else
8:         v.wb[ID_i].explored_port = {f_port}
9:     end if
10:    v.wb[ID_i].node_num = a_i.node_num
11:    a_i.node_num + +
12:    consensus()
13:    if a_i.r = f + 1 then
14:        for all candidate in v.wb[ID_i].W do
15:            a_i.W = a_i.W ∪ {(candidate, a_i.node_num)}
16:        end for
17:    end if
18:    while {1, ··· , d(v)} \ v.wb[ID_i].explored_port ≠ ∅ do
19:        x = min({1, ··· , d(v)} \ v.wb[ID_i].explored_port)
20:        a_i.all_edge_num + +
21:        v.wb[ID_i].explored_port =
                    v.wb[ID_i].explored_port ∪ {x}
22:        ポート x から次のノードへ移動
23:        DFS(現在のノードに来たポート番号)
24:    end while
25: else
26:    v.wb[ID_i].explored_port =
                v.wb[ID_i].explored_port ∪ {f_port}
27: end if
28: ポート f_port から次のノードへ移動，null なら移動しない
```

ド v の白板にスタート地点情報を記述する (14 行目)．スタート地点情報の詳細は後述する．その後，コンセンサスアルゴリズムを実行し，スタート地点情報を共有する．コンセンサスアルゴリズムは，15 行目から 20 行目にて，引数を $null$ とした DFS を用いてネットワークを周回しながら実行されるが，動作の詳細は後述する．

また，コンセンサスアルゴリズムの実行にはエージェント間で同期をとる必要があるため，DFS でネットワークを一周した後に，それと同じ時間だけ一つのノードで待機する．DFS によりノード間を移動している状態を移動フェーズ，一つのノード上で待機している状態を待機フェーズと呼ぶ．以降では，移動フェーズとそれに続く待機フェーズをまとめて，1 ラウンドと考える．待機フェーズを設けることによって，ある正常エージェントがラウンド r の移動フェーズを行う前に全ての正常エージェントがラウンド $r-1$ までの移動フェーズを終えることを保証できる．$a_i.r = r$ のとき，a_i はラウンド r を実行している．

エージェントがラウンド $f+1$ を終えた後には，コンセンサスアルゴリズムにて全ての正常エージェント間で各ノードのスタート地点情報について共通の認識が得られる．コード中の 21 行目からはコンセンサスアルゴリズム後の動作であり，詳細は後述する．

ii） DFS 関 数

本アルゴリズムの DFS 関数 DFS(f_port) を Algorithm 2 に示す．DFS とは全てのノードを探索するアルゴリズムである．DFS(f_port) は再帰的に定義されており，ノードを訪問するたびに呼び出される．引数の f_port は通ってきたポート番号を意味する．エージェント a_i はノード v を訪れた際，待機状態のエージェント a_j が存在する場合には a_j を実行状態へ移行させる．このとき，a_j が先にアルゴリズムを実行し，後に a_i が動作する．すなわち，a_i は a_j が白板に記した情報を参照することができる．また a_i は，ノード v を訪問する際に通ってきた (ノード v における) ポート番号を局所変数 f_port に記録する．一度訪れたことのあるノードだった場合には，25 行目から 27 行目にて，その f_port を $v.wb[ID_i].explored\_port$ に記録する．この動作を行うことによって，18 行目,19 行目で訪れるポート番号を決定する際に通った辺を除外することができる．

ノード v が実行中のラウンドの DFS 中で初めて訪れたノードであるかを判断するために，$v.wb[ID_i].round$ の値が $a_i.r$ の値と等しいかを判断する．等しければ一度訪れたことのあるノードであり，ポート番号 f_port の辺を通り元のノードへ移動する．異なっていればそのラウンドで初めて訪れたノードであるため，来たポート番号 f_port，エージェント a_i の DFS で訪れたノード数，エージェント a_i の現在のラウンド $a_i.r$ をそれぞれ $v.wb[ID_i].from\_port$ と $v.wb[ID_i].explored\_port$，$v.wb[ID_i].node\_num$，$v.wb[ID_i].round$ に記述する．その後ノード数をカウントするために $a_i.node\_num$ をインクリメン

Algorithm 3 consensus( )

```
1: if a_i.r = 0 then
2:     v.wb[ID_i].T = {Sign_i(ID_i)}
3:     v.wb[ID_i].W = {ID_i}
4: else
5:     for all t such that t ∈ v.wb[j].T for some id do
6:         if (テキスト t の署名数が a_i.r であり,
             それぞれの署名に重複がなく,
             a_i の署名がなく,
             value(t) = initial(t) であり,
             v.wb[ID_i].W に value(t) が入っていない) then
7:             v.wb[ID_i].T = v.wb[ID_i].T ∪ {Sign_i(t)}
8:             v.wb[ID_i].W = v.wb[ID_i].W ∪ {value(t)}
9:         end if
10:    end for
11: end if
```

トし，ノード v におけるラウンド r のコンセンサスアルゴリズムを実行する．

13 行目から 17 行目は，詳細は後述するが，$f+1$ ラウンドを終了し，コンセンサスアルゴリズムにて得た共通認識を a_i の $a_i.W$ に集めるための動作である．$a_i.W$ は a_i の変数である．

コンセンサスアルゴリズム実行後は，$v.wb[ID_i].explored\_port$ に含まれていない未探索の辺を順に訪問し，通ってきたポート番号を引数にし，再帰的に DFS 関数を実行する (18 行目から 24 行目)．この時通過した辺の数を $a_i.all\_edge\_num$ にカウントし，待機フェーズでの時間算出に用いる．

iii） consensus 関数

まず，提案アルゴリズムのコンセンサス関数 consensus() の概要を説明する．consensus() の目的は，各ノード v に対し，v をスタート地点とするエージェントの ID について，全ての正常エージェントが共通の認識を得ることである．これを実現するために，提案アルゴリズムでは，完全ネットワークを構成する k 個の（仮想）プロセス $p_1, p_2, \ldots, p_k$ が各ノード v に存在すると仮定し，ノードごとに k プロセスで文献 [5] のコンセンサスアルゴリズムを実行する．すなわち，各ノード v において，エージェント a_i が仮想プロセス p_i のコンセンサスアルゴリズムの動作をシミュレートすることで，k エージェントで共通の認識を得る．

エージェントは，各ラウンドの移動フェーズにおいて，全ノードで一度ずつ consensus() を実行する．各ノード v において，エージェント a_i がラウンド r で consensus() を実行するとき，a_i はプロセス p_i のラウンド r の動作をシミュレートする．このシミュレーションは，ノード v の白板の変数 $v.wb[ID_i].T$ と $v.wb[ID_i].W$ を用いて行う．$v.wb[ID_i].T$ にはプロセス p_i が他のプロセスに送信する情報を保存し，$v.wb[ID_i].W$ にはプロセス p_i がもつ変数の情報を保存する．よって，a_i はノード v におけるプロセス p_i のラウンド r をシミュレートするために，以下の動作を行なう．

（1） 変数 $v.wb[ID_j].T$ $(1 \leqq j \leqq k)$ を参照して，各プロセスがラウンド $r-1$ で p_i に送信した情報を受信する．
（2） $v.wb[ID_i].W$ に格納されたプロセス p_i の変数の情報と，1 で受信した情報を用いて，p_i のラウンド r の動作を行なう．
（3） ラウンド r で p_i から送信する情報を $v.wb[ID_i].T$ に保存し，p_i の変数の情報を $v.wb[ID_i].W$ に保存する．

先述の通り，エージェント a_i がラウンド r の移動フェーズを実行しているとき，全ての正常エージェントはラウンド $r-1$ の移動フェーズを終えている．つまり，プロセス p_i のラウンド r の動作を，全ての正常なプロセスがラウンド $r-1$ の動作を終えてから実行できる．また，白板上の変数 $v.wb[ID_i].T$ と $v.wb[ID_i].W$ を更新できるエージェントは a_i だけであることから，a_i が正常であれば，プロセス p_i も正常な動作をシミュレートできる．以上より，上記の方法によって，コンセンサスアルゴリズムの動作を正しくシミュレートし，正常エージェント間で共通の認識を得ることができる．

関数 consensus() では，上記の方針に従って，文献 [5] のコンセンサスアルゴリズムをシミュレートする．文献 [5] のコンセンサスアルゴリズムの動作の概要は以下の通りである．まず，第 0 ラウンドにおいて，各プロセス p_i は入力値 x に対して署名を施し $Sign_i(x) = \langle x \rangle : ID_i$ を全プロセスへブロードキャストする．同時に，p_i は変数 W に x を追加する．第 $r(1 \leqq r \leqq f+1)$ ラウンドにおいて，各プロセス p_i は，$r-1$ ラウンドでブロードキャストされたメッセージを全て受け取る．そして，そのうち以下の条件 P を満たすメッセージ $t = \langle x \rangle : id_1 : id_2 : \cdots : id_y$ について，新たに署名を施し $Sign_i(t) = \langle x \rangle : id_1 : id_2 : \cdots : id_y : ID_i$ をブロードキャストする．その際，同時に変数 W に x を追加する．

条件 P
（1） 値 t の署名数 y がラウンド数 r と等しい．
（2） 値 t の署名に重複した ID がない．
（3） 値 t の署名にプロセス p_i の署名がない．
（4） W に x が入っていない．

これを $f+1$ ラウンドまで繰り返すことで，全てのプロセスで W を一致させることができる．

関数 consensus() の擬似コードを Algorithm 3 に示す．まず，ラウンド 0 において，エージェント a_i はスタート地点のノードでのみ，$v.wb[ID_i].T$ にスタート地点情報として $\{Sign_i(ID_i)\} = \{\langle ID_i \rangle : ID_i\}$，$v.wb[ID_i].W$ に $\{ID_i\}$ を書く（consensus() の 1〜3 行目）．コンセンサスアルゴリズムのラウンド 0 では，プロセス p_i は，自身のスタート地点では $\{ID_i\}$ を送信し，他の地点では空集合を送信する．そのため，ラウンド 0 で送信する値の設定は，スタート地点だけで行えば十分である．エージェント a_i のラウンド r (consensus() の 4 行目から 13 行目) では，$\bigcup_{a_j \in A} v.wb[ID_j].T$ のうち，以下の条件 A に当てはまる値 t に署名を施し，$v.wb[ID_i].T$ へ記述する．ここで，署名付きの値 $t = \langle x \rangle : id_1 : id_2 : \cdots : id_y$ に対して，$value(t)$ は x を返す関数，$initial(t)$ は id_1 を返す関数である．

条件 A

（1） 値 t の署名数 y がラウンド数 $a_i.r$ と等しい．

（2） 値 t の署名に重複した ID がない．

（3） 値 t の署名にエージェント a_i の署名がない．

（4） a_i の $v.wb[ID_i].W$ に $value(t)$ が入っていない．

（5） $value(t)$=$initial(t)$ である．

条件 A の 1~4 は，条件 P の 1~4 に対応する．また，正常エージェントが最初に $v.wb[ID_i].T$ に書く値は $\langle ID_i\rangle : ID_i$ であり，常に $value(t) = initial(t)$ が成り立つことから，条件 A に 5 の項目を加えている．条件 A を満たした t を $v.wb[ID_i].T$ へ加える際，$value(t)$ を $v.wb[ID_i].W$ に加える．これらの動作は，プロセス p_i のラウンド r の動作に対応する．

ラウンド $f+1$ では，a_i は各ノード v 上の $v.wb[ID_i].W$ に共通認識を持てた情報を格納する．すなわち，$v.wb[ID_i].W$ は全ての正常エージェントで同一であり，v をスタート地点とするエージェントの ID を格納している．この ID の情報，すなわち $v.wb[ID_i].W$ の情報を，DFS() にて $a_i.W$ に収集する．その後，Algorithm 1 の main() の 17,18 行目にて，$a_i.W$ のうち ID の重複している情報を全て削除し，残った情報のうち最小 ID の情報が書かれたノードを集合地点として移動する．集合地点までの移動は，Algorithm 1 では省略して記述しているが，DFS を用いる．$a_i.W$ の中には，各スタート地点情報と，書き込まれたノードを示す情報が含まれている．そのため，最小 ID のエージェントのスタート地点情報が残されたノードに DFS にて移動できる．移動した後，終了宣言を行う．

4. アルゴリズムの正当性

補題 1. 任意の正常エージェント a_i,a_j に対し，a_i がラウンド r の移動フェーズを開始する前に，a_j はラウンド $r-1$ の移動フェーズを終了する．

(証明) 移動フェーズと待機フェーズの関係を図 3 に示す．正常エージェント a_i は待機状態から実行状態に移行すると，すぐに DFS を開始し，全てのノードを訪問する．その途中で，待機状態のエージェントが存在するノードを訪れた場合，エージェント a_i はそのエージェントを実行状態に移行させる．そのため，全てのエージェントは a_i がラウンド 1 の移動フェーズを終える前に実行状態に移行する．

また，各エージェントの待機フェーズ，移動フェーズの長さは全て $2m$ である．そのため，エージェント a_i のラウンド 1 の待機フェーズが終わる前に全てのエージェントのラウンド 1 の移動フェーズが終了する．ラウンド 2 以降も全ての正常エージェントの移動フェーズと待機フェーズの時間が $2m$ であるため，a_i のラウンド r の移動フェーズが開始する前に a_j のラウンド $r-1$ の移動フェーズが終了する．

補題 2. 任意の正常エージェント a_i，a_j に対し $ID_i \in v.wb[ID_j].W$ が成立するノード v は *1* つだけである．

(証明) 背理法にて証明する．ノード v_1，v_2 で $ID_i \in v_1.wb[ID_j].W$，$ID_i \in v_2.wb[ID_j].W$ であると仮定する．

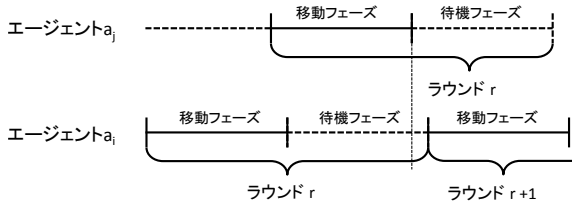


図 3 移動フェーズと待機フェーズ

a_i は分散システム上に 1 つしか存在しないため，v_1，v_2 のどちらかはアルゴリズム開始時に a_i が存在したノードではない．また正常エージェントがスタート地点情報を作成するのは，アルゴリズム開始時のみである．そのため，v_1 を実際に a_i がアルゴリズムを開始したノードだとすると，a_i は v_2 にスタート地点情報を作成することはない．$ID_i \in v_2.wb[ID_j].W$ であるためには，あるエージェント a_x に対する $v_2.wb[ID_x].T$ に条件 A を満たす値 $t = \langle ID_i\rangle : id_1 : id_2 : \cdots : id_r$ が存在し，a_j は $v_2.wb[ID_j].W$ に $value(t)$ を追加したことになる．しかし，条件 A の中に「$value(t) = initial(t)$ である」という条件が存在するため，値 t において $ID_i = id_1$ である必要がある．また，本モデルではエージェント a_i のみが署名関数 $Sign_i()$ を用いてスタート地点情報 $Sign_i(ID_i) = \langle ID_i\rangle : ID_i$ を作成することができ，他のエージェントは $t = \langle ID_i\rangle : ID_i : id_2 : \cdots : id_r$ を v_2 に記述することは出来ない．したがって仮定した状況，$ID_i \in v_1.wb[ID_j].W$，$ID_i \in v_2.wb[ID_j].W$ は矛盾する．よって，補題 2 は成り立つ．

補題 3. 全てのノード v と任意の正常エージェント a_i，a_j について，a_i がラウンド $f+1$ を終えた後の $v.wb[ID_i].W$ と，a_j がラウンド $f+1$ を終えた後の $v.wb[ID_j].W$ は等しい．

(証明) エージェント a_i がラウンド $f+1$ を終えた後の $v.wb[ID_i].W$ を考える．$v.wb[ID_i].W$ に属するある要素 x を考え，a_i が $v.wb[ID_i].W$ に x を追加したラウンドを r とする．すなわち，a_i はラウンド r で，エージェント a_z の $v.wb[ID_z].T$ に含まれる $t = \langle x\rangle : id_1 : id_2 : \cdots : id_r$ を読み込み，t が条件 A を満たすため，x を $v.wb[ID_i].W$ に追加している．このとき，以下の 3 つの場合が考えられる．

Case 1 t 内に a_j の署名が存在するとき

この場合，a_j は署名をする際に x を $v.wb[ID_j].W$ に追加するため，既に $v.wb[ID_j].W$ 内に要素 x が存在する．

Case 2 ラウンド $r \leqq f$ で，t 内に a_j の署名がないとき

この場合ラウンド r で a_i は t に署名を施した $t' = \langle x\rangle : id_1 : id_2 : \cdots : id_r : ID_i$ を $v.wb[ID_i].T$ に追加する．補題 1 より，a_j のラウンド $r+1$ の移動フェーズは，a_i のラウンド r の移動フェーズ以降に始まる．そのため a_j は必ずラウンド $r+1$ でエージェント a_i の $v.wb[ID_i].T$ 内の値 t' を確認する．このとき，t' は条件 A を満たすため，a_j は要素 x を $v.wb[ID_j].W$ に追加する．

Case 3 ラウンド $r = f + 1$ で，t 内に a_j の署名がないとき

t には $f + 1$ 個の署名が付いているため，少なくとも一つの正常エージェント a_z が署名をしている．a_z が署名を行って t を $v.wb[ID_z].T$ に追加したラウンドを r' とする．a_j はラウンド $r' + 1$ で $v.wb[ID_z].T$ 内の値 t を確認する．この t は条件 A を満たすため，a_j は $v.wb[ID_j].W$ に要素 x を追加する．

以上より，a_i がラウンド $f + 1$ を終えた後の $v.wb[ID_i].W$ と，a_j がラウンド $f + 1$ を終えた後の $v.wb[ID_j].W$ は等しい．

補題 4. 全ての正常エージェントはラウンド $f + 1$ 後に同一の集合場所を算出する．

(証明) 正常エージェント a_i と a_j について考える．a_i はラウンド $f + 1$ にて，全てのノード上の $v.wb[ID_i].W$ の要素 $candidate$ を，書かれたノードの場所を含めた二項組 $(candidate, a_i.node\_num)$ として $a_i.W$ に追加する．a_j も同様に $a_j.W$ に追加する．補題 3 より，ノード v における $v.wb[ID_i].W$ と $v.wb[ID_j].W$ の全ての要素は等しい．すなわち，$a_i.W$ と $a_j.W$ に含まれる $candidate$ は等しい．

a_i はラウンド $f + 1$ にて $a_i.W$ の要素で，重複していない $candidate$ のうち，最小値が書かれたノード v を集合場所として算出する．補題 2 より，正常エージェントの ID を $v.wb[ID_i].W$ に含むノードは一つしか存在しないため，$candidate$ の重複していない ID は少なくとも一つ存在する．よって，エージェント a_i は一つのノードを集合場所として選択することができる．a_j も $a_i.W$ と同一の $candidate$ を含んだ $a_j.W$ を保持しているため，同一のノードを集合場所として算出できる．

定理 1. 提案アルゴリズムは $O(fm)$ 単位時間で集合問題を解く．

(証明) 補題 4 より，ラウンド $f + 1$ 後に全ての正常エージェントは同一の集合場所を算出できる．その集合場所に移動し，終了宣言を行うことで集合問題を解くことができる．

本アルゴリズムではコンセンサスアルゴリズムを実行するために，全てのノードを DFS を用いて $f + 1$ 周している．DFS は全ての辺を 2 回通過するアルゴリズムであるため，一周あたり $2m$ 単位時間必要である．また，待機フェーズも一周あたり $2m$ 単位時間必要である．更に，コンセンサスアルゴリズムの実行後，集合場所へ移動するために高々 $2m$ 単位時間を要する．よって，本アルゴリズムでは $(2m + 2m) \times (f + 1) + 2m$ 単位時間必要であり，$O(fm)$ 単位時間で集合問題を解く．

5. ま と め

本稿では同期ネットワークにおいて，認証機能付き白板を用いたビザンチン環境故障耐性を持つモバイルエージェント集合アルゴリズムを提案した．具体的には，アルゴリズムを開始したノードに印（スタート地点情報）を書き込み，各ノード上でスタート地点情報を共通の認識にするためにコンセンサスアルゴリズムを実行する．共通の認識を得ることができれば，集合地点ノードを算出することができる．ビザンチンエージェント数の上限を f，ネットワーク中の辺の数を m とすると $O(fm)$ 単位時間での集合が可能である．

今後の課題としては，非同期ネットワークでの集合を実現するアルゴリズムの提案が考えられる．しかし，本アルゴリズムで用いたコンセンサスアルゴリズムは非同期プロセスであると解が得られないことが証明されているため，異なるアプローチをする必要がある．

文　献

[1] Y. Dieudonné, A. Pelc, D. Peleg, Gathering despite mischief, ACM Transactions on Algorithms(TALG), vol. 11, no. 1, article 1, 2014.

[2] A. Ta-Shma, U. Zwick,Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences, ACM Transactions on Algorithms(TALG), vol. 10, no. 3, article 12, 2014

[3] A. Dessmark, P. Fraigniaud, D.R. Kowalski, A. Pelc, Deterministic rendezvous in graphs, Algorithmica vol. 46, no. 1, pp. 69–96, 2006.

[4] E. Kranakis, D. Krizanc, E. Markou, The mobile agent rendezvous problem in the ring, Synthesis Lectures on Distributed Computing Theory vol. 1, no. 1, pp. 1–122, 2010.

[5] D. Dolev, H.R. Strong, Authenticated algorithms for Byzantine agreement, SIAM Journal on Computing vol. 12, no. 4, pp. 656–666, 1983.

[6] J. Cao, S. K. Das, Mobile Agents in Networking and Distributed Computing, Wiley-Interscience, 2012.

[7] A. Pelc, Deterministic rendezvous in networks: A comprehensive survey, Networks, vol. 59, pp. 331–347, 2012.

[8] D. R. Kowalski and A. Malinowski, How to meet in anonymous network, Theorical Computer Science, 399 (1-2), pp. 141–156, 2008.

[9] A. Miller, A. Pelc, Time versus cost tradeoffs for deterministic rendezvous in networks, Distributed Computing 29(1), pp. 51–64, 2016.

[10] J. Czyzowicz, A. Kosowski, A. Pelc: Time versus space trade-offs for rendezvous in trees, Distributed Computing 27(2), pp. 95–109, 2014.

[11] P. Fraigniaud, A. Pelc, Delays induce an exponential memory gap for rendezvous in trees, ACM Transactions on Algorithms, 9(2):17, 2013.

[12] J. Czyzowicz, A. Kosowski, A. Pelc: How to meet when you forget: log-space rendezvous in arbitrary graphs, Distributed Computing 25(2), pp. 165–178, 2012.

[13] S. Bouchard, Y. Dieudonne, B. Ducourthial, Byzantine Gathering in Networks, SIROCCO, pp. 179–193, 2015.

[14] Y. Sudo, D. Baba, J. Nakamura, F. Ooshita, H. Kakugawa, T. Masuzawa, A Single Agent Exploration in Unknown Undirected Graphs with Whiteboards, IEICE Transactions 98-A(10), pp. 2117–2128, 2015.

[15] F. Ooshita, S. Kawai, H. Kakugawa, T. Masuzawa, Randomized Gathering of Mobile Agents in Anonymous Unidirectional Ring Networks, IEEE Transactions on Parallel and Distributed Systems, 25(5), pp. 1289–1296, 2014.

Randomly Optimized Grid Graph for Low-Latency Interconnection Networks

Koji NAKANO, Daisuke TAKAFUJI, Satoshi FUJITA (Hiroshima Univ.),
Hiroki MATSUTANI (Keio Univ.), Ikki FUJIWARA, and Michihiro KOIBUCHI (NII)

Published in Proc. of ICPP 2016

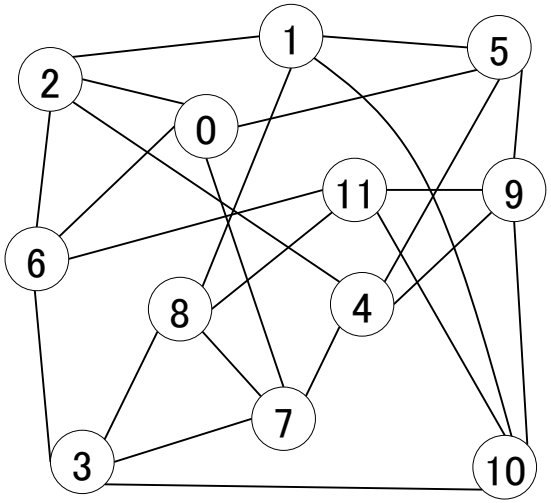
The order/degree problem

Input: N nodes, degree K

Output: a graph of size N and degree $\leq K$ with minimum diameter and minimum average shortest path length (ASPL)

Diameter: the maximum value among shortest path length of all pairs of vertices

$N = 12$ and $K = 4$



Shortest path length of all pairs of vertices

	0	1	2	3	4	5	6	7	8	9	10	11
0		2	1	2	2	1	1	1	2	2	3	2
1	2		1	2	2	1	2	2	1	2	2	1
2	1	1		2	1	2	1	2	2	2	3	2
3	2	2	2		2	3	1	1	1	2	1	2
4	2	2	1	2		1	2	1	2	1	2	2
5	1	1	2	3	1		2	2	2	1	2	2
6	1	2	1	1	2	2		2	2	2	2	1
7	1	2	2	1	1	2	2		1	2	2	3
8	2	1	2	1	2	2	2	1		2	1	2
9	2	2	2	2	1	1	2	2	2		1	1
10	3	2	3	1	2	2	2	2	1	1		1
11	2	1	2	2	2	2	1	3	2	1	1	

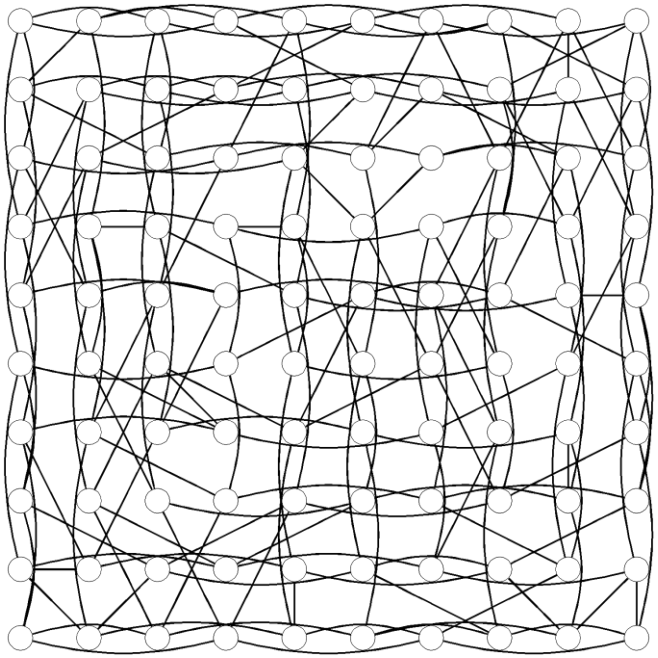
the maximum value
→ Diameter = 3

the average= 1.696
→ ASPL = 1.696

Our Goal

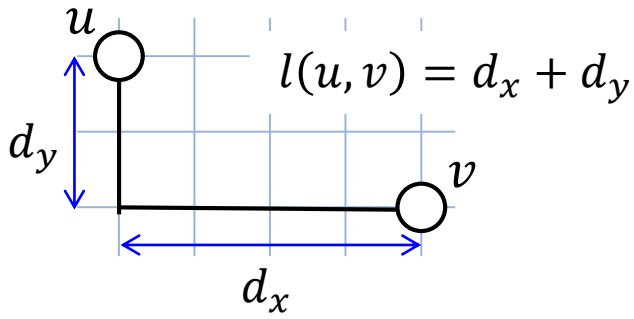
Input: N nodes, degree K, length L

Output: a K-regular grid graph of size N and all edge length $\leq L$ with minimum diameter and minimum average shortest path length (ASPL)



A K-regular L-restricted grid graph with $N = 100$, $K = 4$ and $L = 3$

Length of edge (u, v):
the Manhattan distance of $u, v \in V$.



A grid graph $G = (V, E)$ is L-restricted if $l(u, v) \leq L$ for all edges $(u, v) \in E$.

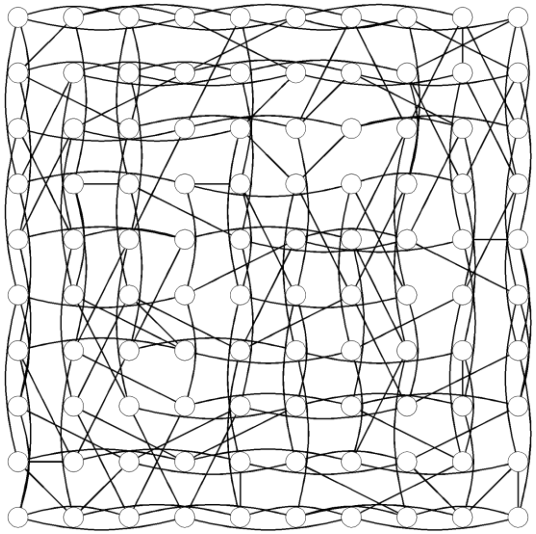
To find a K-regular L-restricted grid graph

Contributions

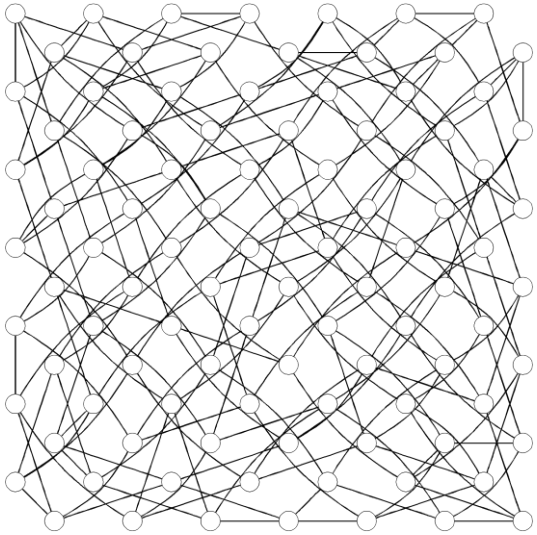
We present

(1) the randomly optimized K-regular L-restricted grid graph
(2) the theoretical lower bounds of the diameter and ASPL
(3) the K-regular L-restricted diagrid graph (on a diagonal grid) with smaller diameter

K-regular L-restricted grid graph



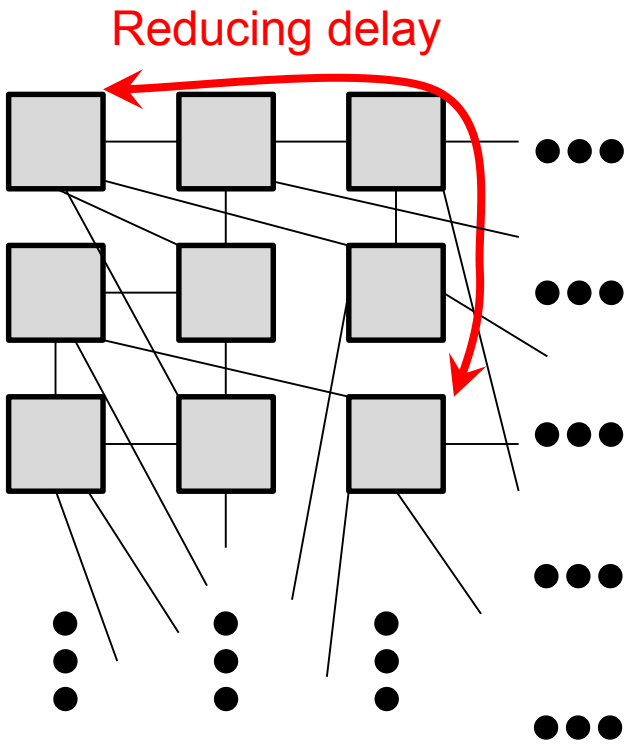
K-regular L-restricted diagrid graph



Background

Design of low-latency networks is important.

We need to
find graphs with minimum diameter or minimum ASPL, and apply to design of the networks.

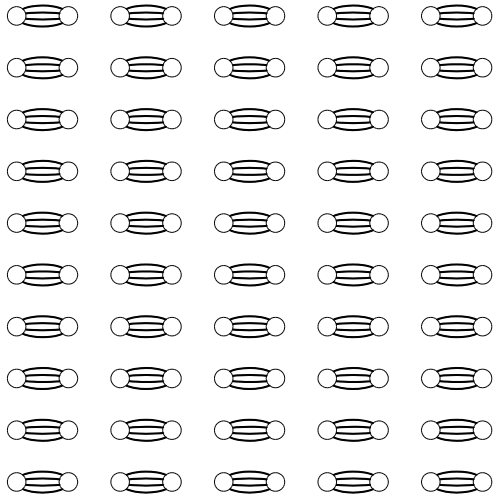


(1) A randomized algorithm for generating an almost optimal K-regular L-restricted grid graph

Step 1: Generate an initial K-regular L-restricted grid graph G.
Step 2: Repeatedly perform random 2-toggle operation to generate a random K-regular L-restricted grid graph.
Step 3: Repeatedly perform random 2-opt operation to find a K-regular L-restricted grid graph as best as possible.

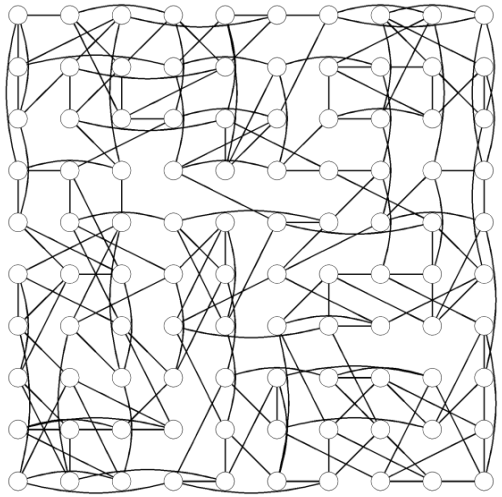
A K-regular L-restricted grid graph of size N
with $N = 100$, $K = 4$ and $L = 3$

Step 1



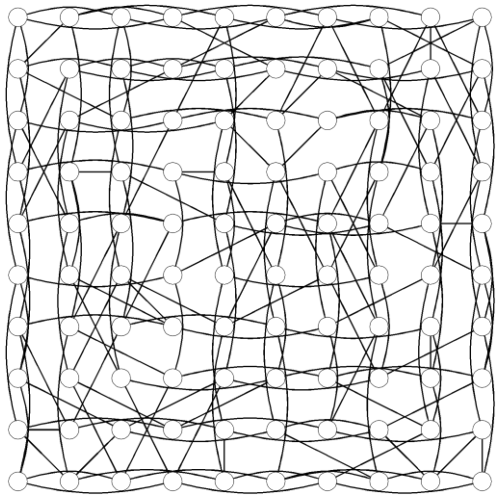
diam $= \infty$
ASPL $= \infty$

Step 2



diam $= 10$
ASPL $= 4.575$

Step 3

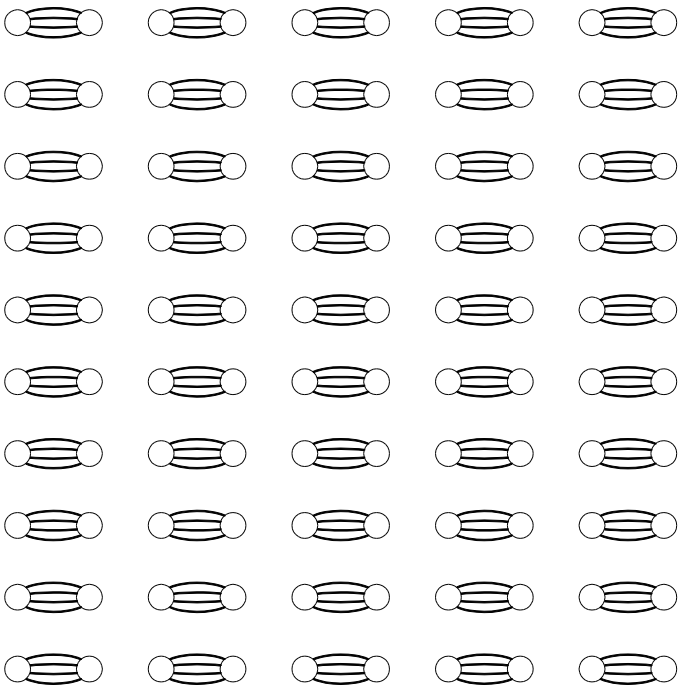


diam $= 6$
ASPL $= 3.443$

A randomized algorithm for generating an almost optimal K-regular L-restricted grid graph

Step 1: Generate an initial K-regular L-restricted grid graph G.

A K-regular L-restricted grid graph of size N
with $N = 100$, $K = 4$ and $L = 3$



diam $= \infty$
ASPL $= \infty$

A randomized algorithm for generating an almost optimal K-regular L-restricted grid graph

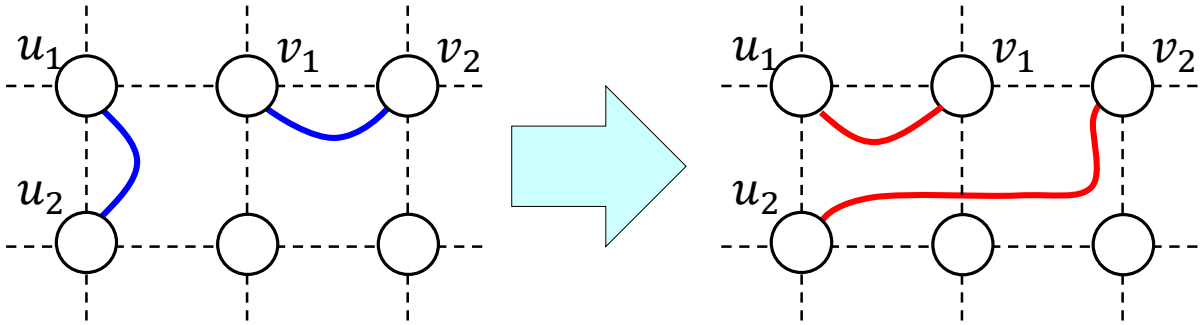
Step 2: Repeatedly perform random 2-toggle operation to generate a random K-regular L-restricted grid graph.

Select two edges $(u_1, u_2), (v_1, v_2)$ randomly.

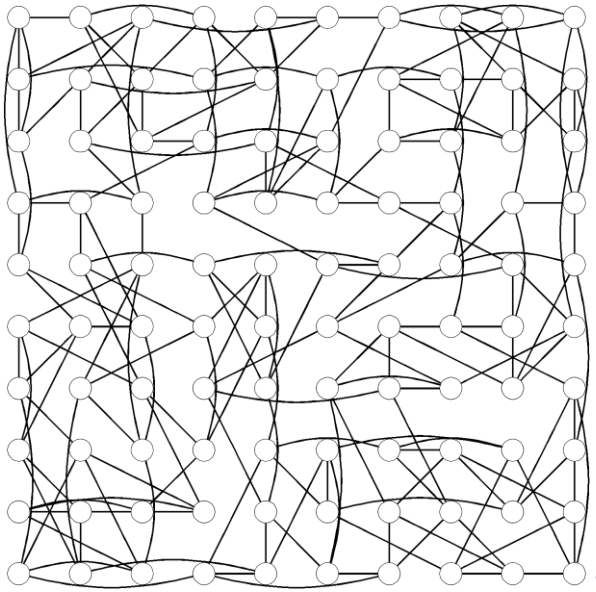
G': replaced $(u_1, u_2), (v_1, v_2)$ to $(u_1, v_1), (u_2, v_2)$

If $l(u_1, v_1) \leq L$ and $l(u_2, v_2) \leq L$, replace G to G'.
(ignore the diameter and the ASPL)

2-toggle operation



A 4-regular 3-restricted grid graph of size 100



diam $= 10$
ASPL $= 4.575$

A randomized algorithm for generating an almost optimal K-regular L-restricted grid graph

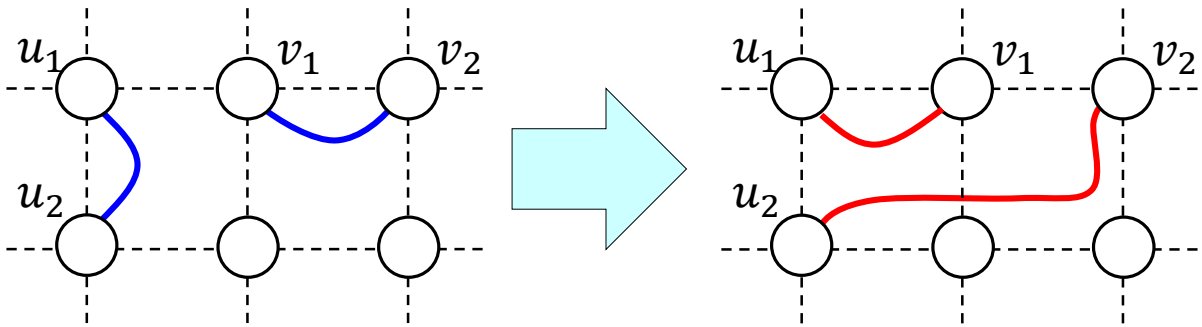
Step 3: Repeatedly perform random 2-opt operation to find a K-regular L-restricted grid graph as best as possible.

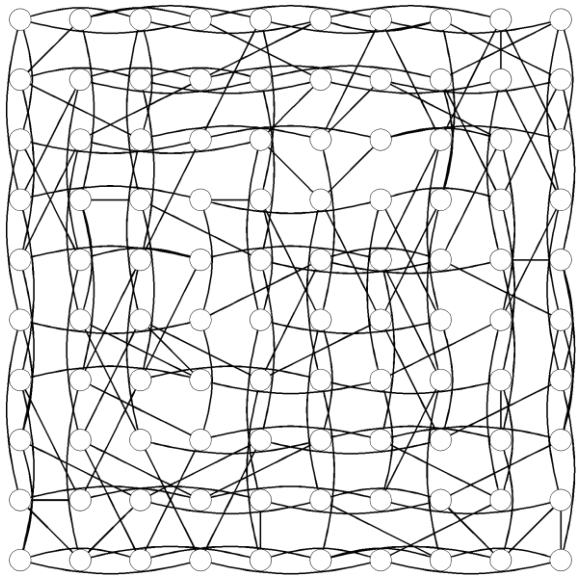
Select two edges $(u_1, u_2), (v_1, v_2)$ randomly.

G': replaced $(u_1, u_2), (v_1, v_2)$ to $(u_1, v_1), (u_2, v_2)$

$D(G)$ or $D(G')$: the diameter of G or G' $A(G)$ or $A(G')$: the ASPL of G or G'

If $(l(u_1, v_1) \leq L$ and $l(u_2, v_2) \leq L)$ and
$(D(G) > D(G')$ or $(D(G) = D(G')$ and $A(G) > A(G')))$, replace G to G'.





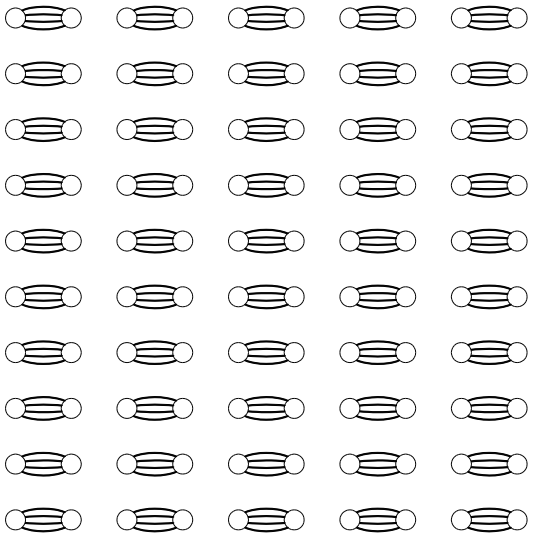
A 4-regular 3-restricted grid graph of size 100

Step 2	Step 3
diam = 10 ASPL = 4.575	diam = 6 ASPL = 3.443

A randomized algorithm for generating an almost optimal K-regular L-restricted grid graph

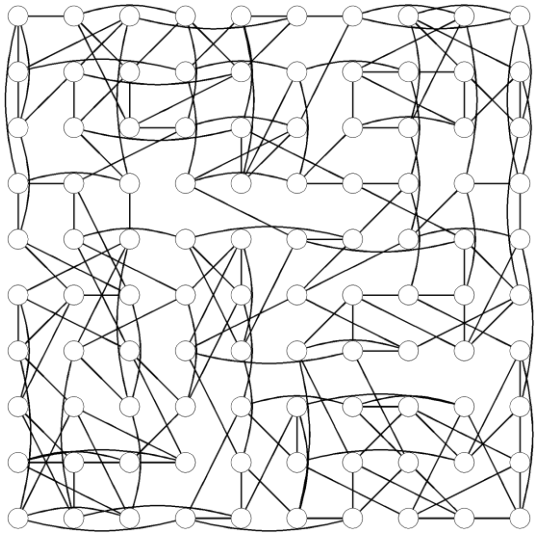
A K-regular L-restricted grid graph of size N with $N = 100$, $K = 4$ and $L = 3$

Step 1



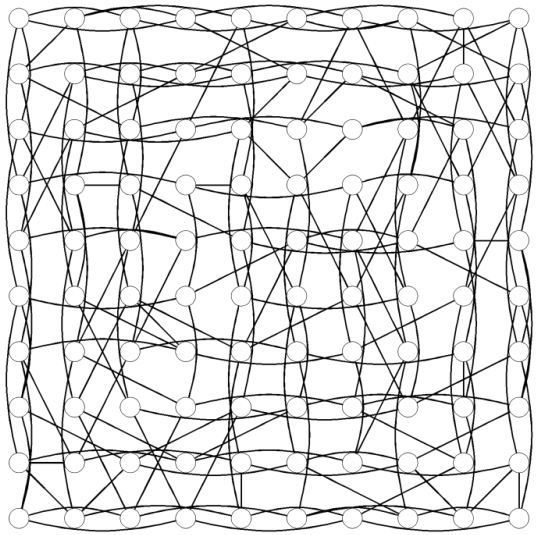
diam $= \infty$
ASPL $= \infty$

Step 2



diam $= 10$
ASPL $= 4.575$

Step 3



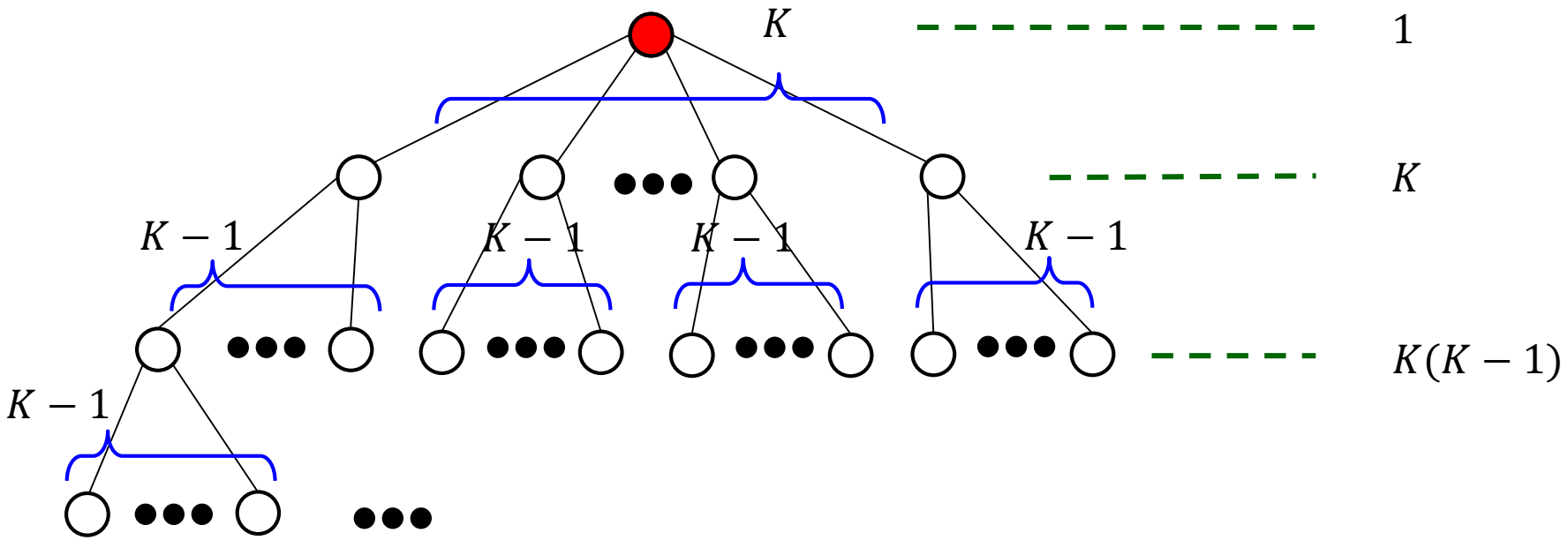
diam $= 6$
ASPL $= 3.443$

(2) Lower Bound of the ASPL

We need to estimate the maximum number of nodes reachable in i hops from a node.

A K-regular graph

the number of nodes

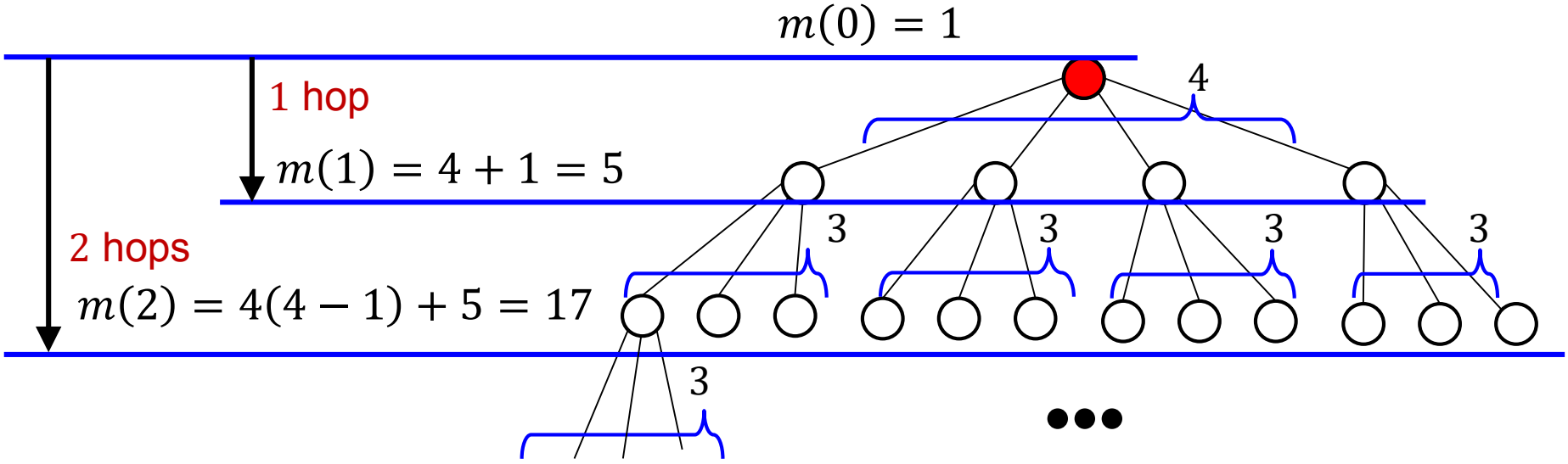


ASPL of a K-regular graph $\geq \dfrac{\sum_{i\geq 1}(K(K-1)^{i-1}\cdot i)}{N-1}$

Lower Bound of the ASPL based on Moore bound (K-regular graph)

$m(i)$: the maximum number of nodes reachable in i hops from a node

A 4-regular graph ($K = 4$)



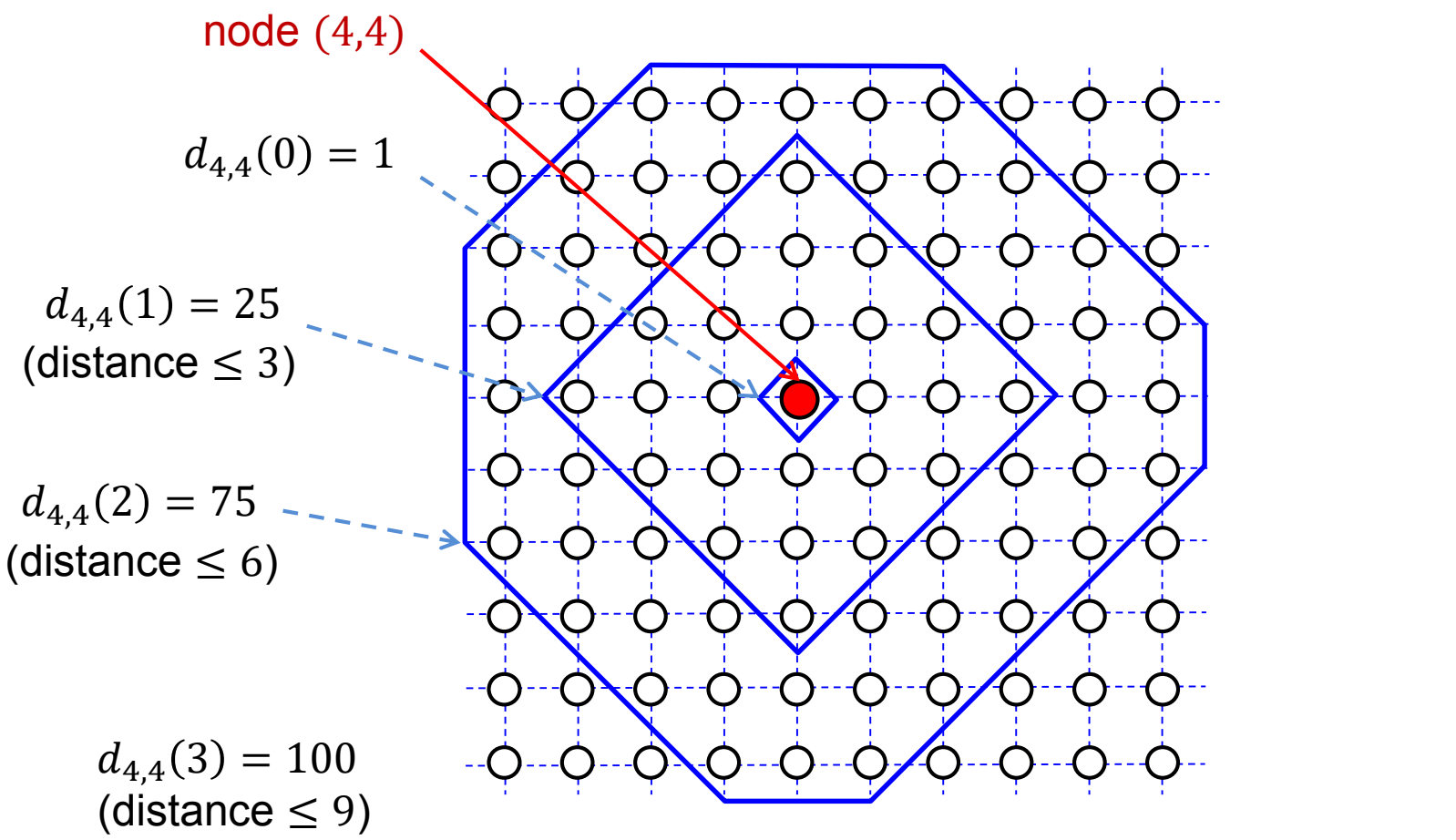
Because a graph is K-regular,

$$m(i) = \min(1 + \sum_{j=1}^{i} K(K-1)^{j-1}, N)$$

Lower Bound of the ASPL based on length L

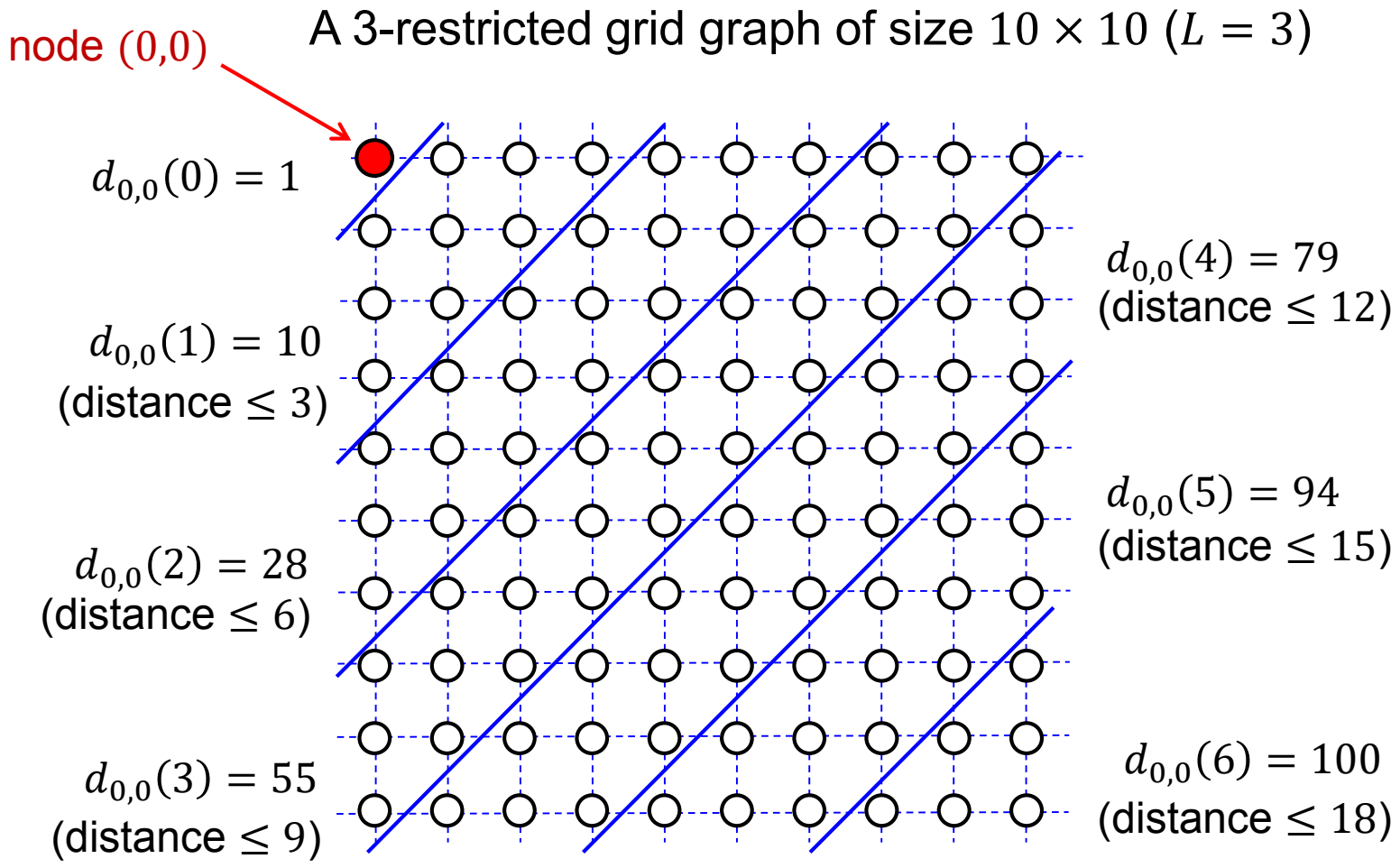
$d_{x,y}(i)$: the number of nodes with the distance $\leq i \times L$ from node (x, y)

A 3-restricted grid graph of size 10×10 $(L = 3)$



Lower Bound of the ASPL based on length L

$d_{x,y}(i)$: the number of nodes with the distance $\leq i \times L$ from node (x, y)



Lower Bound of the ASPL

For a K-regular L-restricted grid graph,

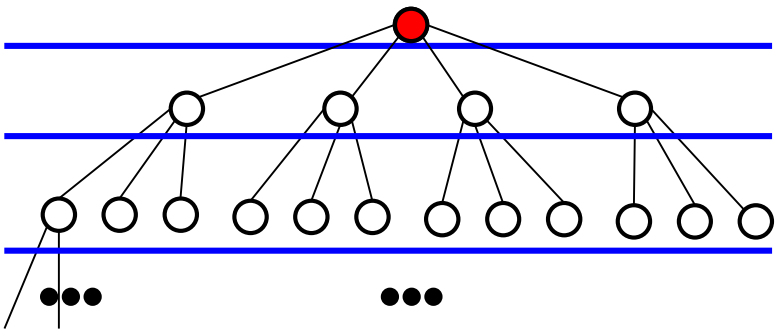
The number of nodes reachable in i hops from node (x, y)

$$\leq md_{x,y}(i) = \min(m(i), d_{x,y}(i))$$

Lower Bound of the ASPL : $A^- = \sum_{(x,y)\in V} \sum_{i\geq 1} ((md_{x,y}(i) - md_{x,y}(i-1)) \cdot i)/N(N-1)$

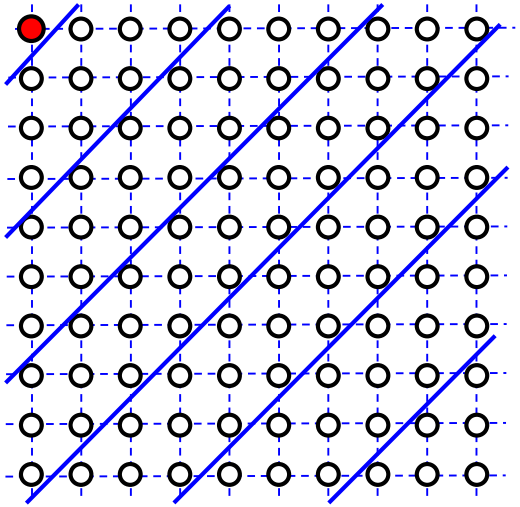
A K-regular graph

$m(i)$: the maximum number of nodes reachable in i hops from a node



An L-restricted grid graph

$d_{x,y}(i)$: the number of nodes with the distance $\leq i \times L$ from node (x, y)



The values of m, $d_{0,0}$ and $md_{0,0}$

A 4-regular 3-restricted grid graph of size 10×10

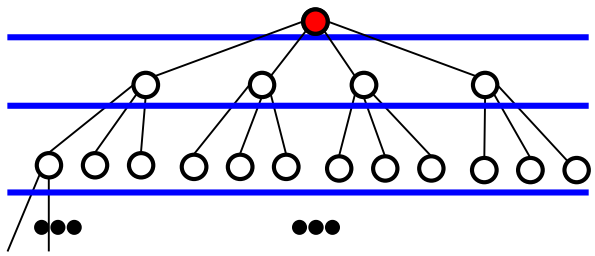
i	0	1	2	3	4	5	6
$m(i)$	1	5	17	53	100	100	100
$d_{0,0}(i)$	1	10	28	55	79	94	100
$md_{0,0}(i)$	1	5	17	53	79	94	100

Lower bound of the diameter

The number of nodes reachable in i hops from node (x, y)
$\leq md_{x,y}(i) = \min(m(i), d_{x,y}(i))$

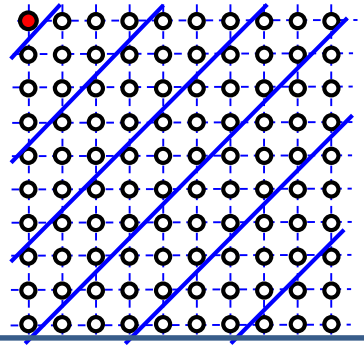
A K-regular graph

$m(i)$: the maximum number of nodes reachable in i hops from a node



An L-restricted grid graph

$d_{x,y}(i)$: the number of nodes with the distance $\leq i \times L$ from node (x, y)



Diameter upper bound D^+ and lower bound D^- of a K-regular L-restricted grid graph of size 30×30

K		L														
		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	D^+	29	20	15	12	12	11	11	11	11	11	11	11	11	11	11
		=	=	=	=	>	>	>	>	>	>	>	>	>	>	>
	D^-	29	20	15	12	10	9	9	9	9	9	9	9	9	9	9
4	D^+	29	20	15	12	10	9	8	8	8	8	8	8	8	8	8
		=	=	=	=	=	=	=	>	>	>	>	>	>	>	>
	D^-	29	20	15	12	10	9	8	6	6	6	6	6	6	6	6
5	D^+	29	20	15	12	10	9	8	7	7	6	6	6	6	6	6
		=	=	=	=	=	=	=	>	>	>	>	>	>	>	>
	D^-	29	20	15	12	10	9	8	6	6	6	5	5	5	5	5

optimal

D^+ : Diameter upper bound (K-regular L-restricted grid graphs generated by our algorithm)

D^-: lower bound

Diameter upper bound D^+ and lower bound D^- of a K-regular L-restricted grid graph of size 30×30

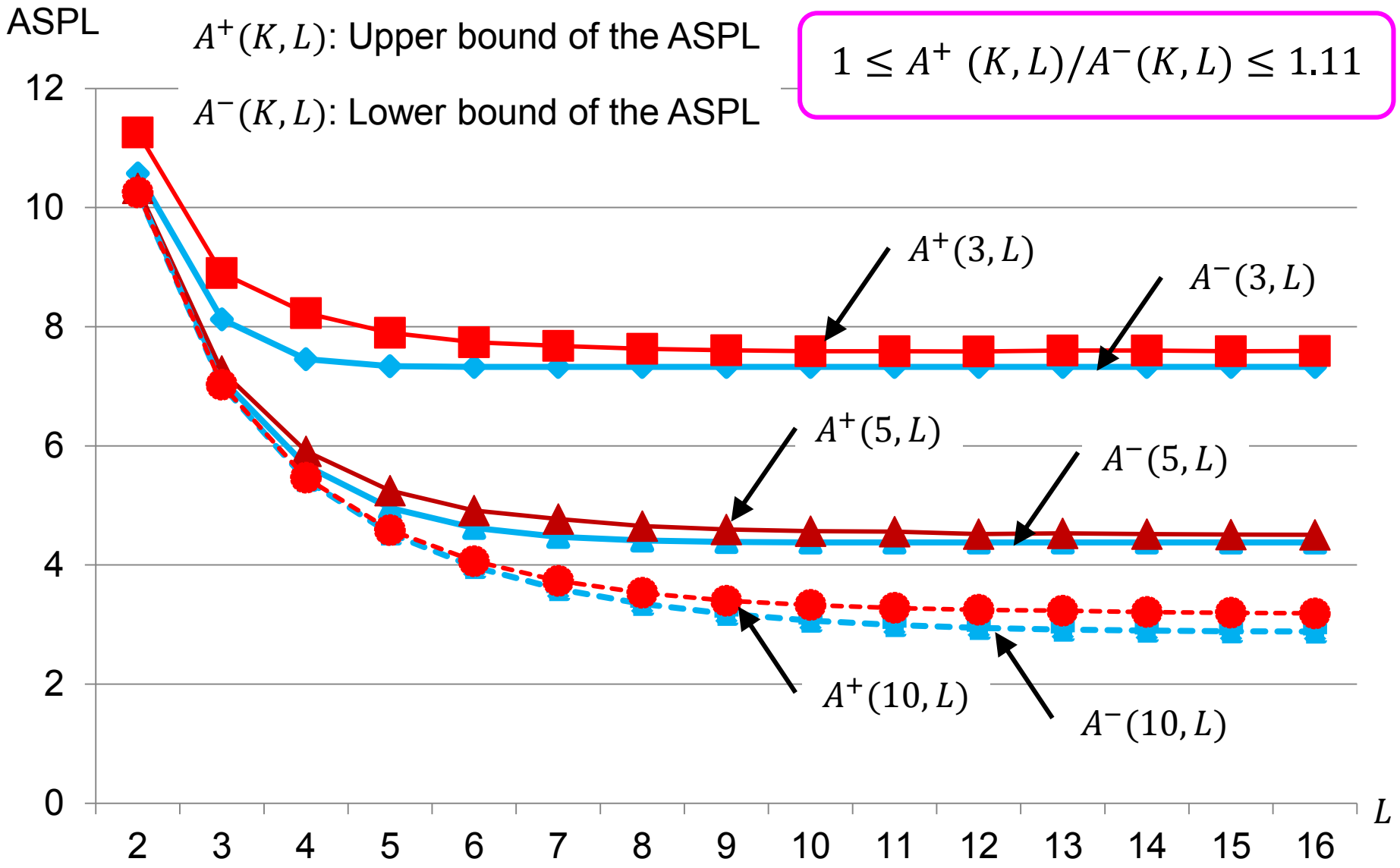
K		L														
		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	D^+	29 II	20 II	15 II	12 II	10 II	9 II	8 II	7 II	6 II	6 II	6 V	6 V	6 V	6 V	6 V
7-9		29 II	20 II	15 II	12 II	10 II	9 II	8 II	7 II	6 II	6 II	5 II	5 II	5 II	5 V	5 V
10-16		29 II	20 II	15 II	12 II	10 II	9 II	8 II	7 II	6 II	6 II	5 II	5 II	5 II	4 II	4 II
6-16	D^-	29	20	15	12	10	9	8	7	6	6	5	5	5	4	4

optimal

D^+ : Diameter upper bound (K-regular L-restricted grid graphs generated by our algorithm)

D^-: lower bound

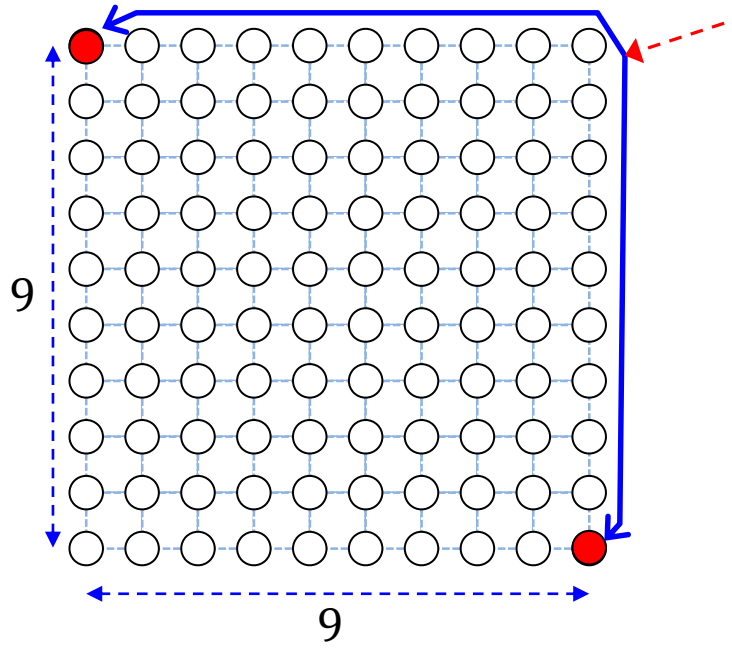
ASPL of grid graphs of size 30×30 for $K = 3, 5, 10$



(3) Diagonal grid (diagrid) graphs with smaller diameter

diameter of L-restricted grid graph $\geq \dfrac{\text{the maximum length of two nodes}}{L}$

A grid graph of size 10×10 $(N = 100)$

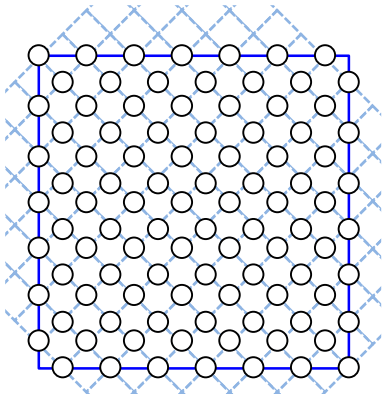


the maximum length of two nodes
$= 9 \times 2 = 18$

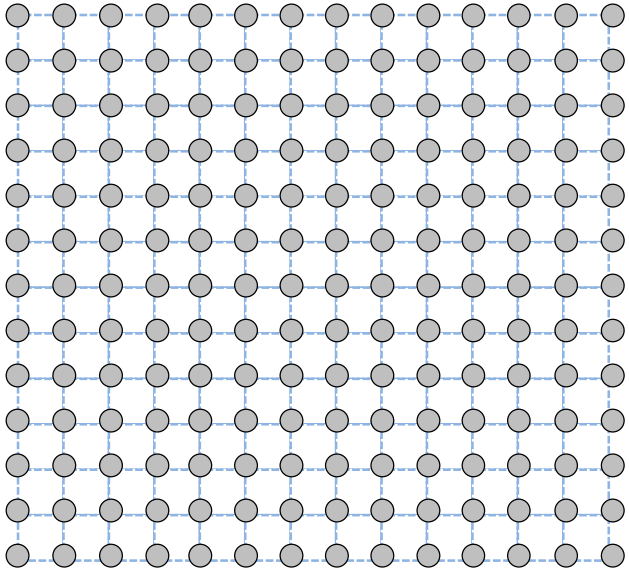
When $L = 3$, diameter $\geq \frac{18}{3} = 6$.

There is no 3-restricted grid graph with diameter ≤ 5.

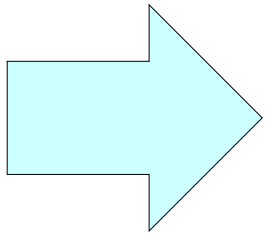
We use a grid layout so as to shorten the maximum length of two nodes.



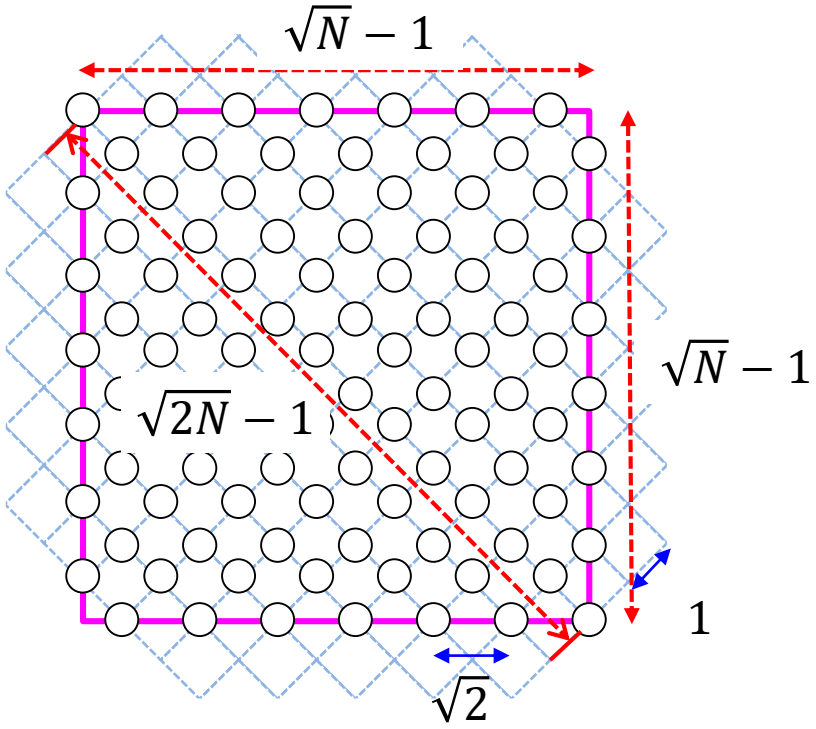
Diagonal grid (diagrid) graphs with smaller diameter



turning by $45°$ and extracting

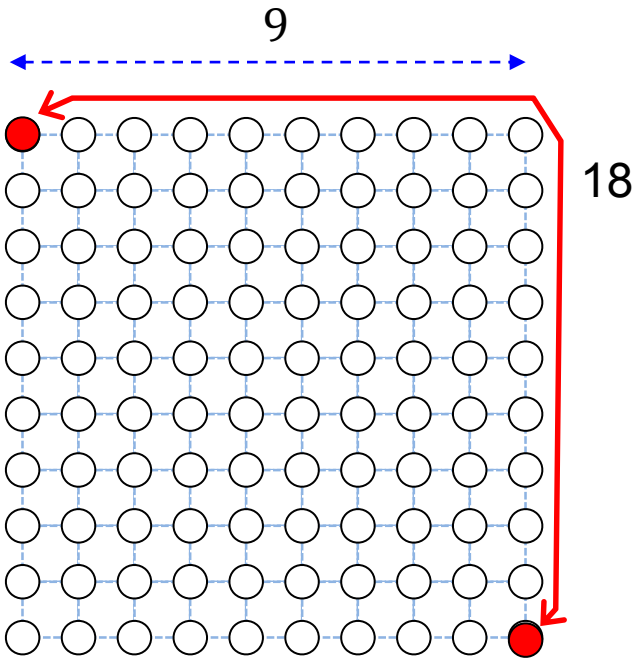


A diagonal grid (diagrid) graph



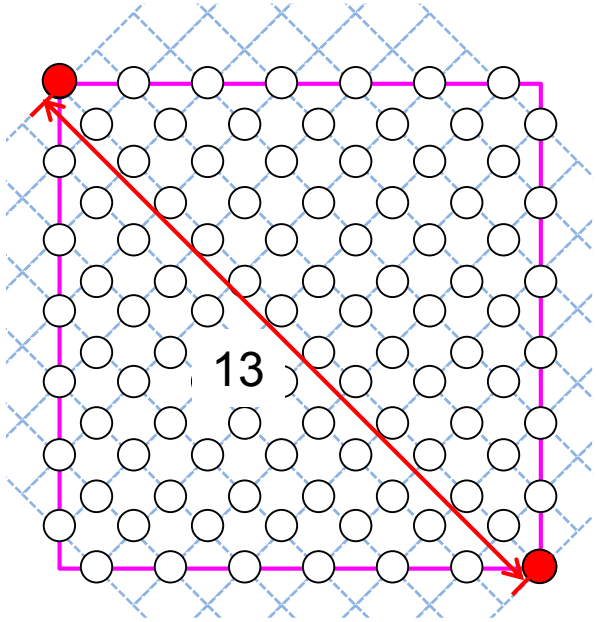
Diagonal grid (diagrid) graphs with smaller diameter

A grid graph of size 10×10 $(N = 100)$



the maximum length of two nodes
$= 9 \times 2 = 18\ (= 2(\sqrt{N} - 1))$

A diagrid graph of size $N = 98$



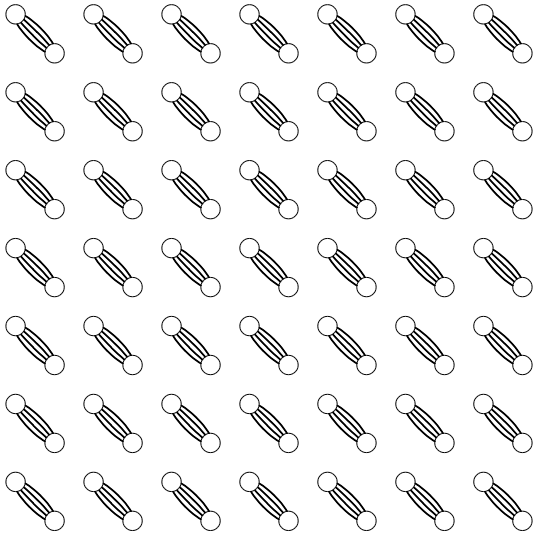
the maximum length of two nodes
$= 13\ (= \sqrt{2N} - 1)$

$$\frac{\text{The maximum length of two nodes in the diagrid graph}}{\text{The maximum length of two nodes in the grid graph}} = \frac{\sqrt{2N} - 1}{2(\sqrt{N} - 1)} \cong \frac{\sqrt{2}}{2} \cong 70.7\%$$

An almost optimal K-regular L-restricted diagrid graph

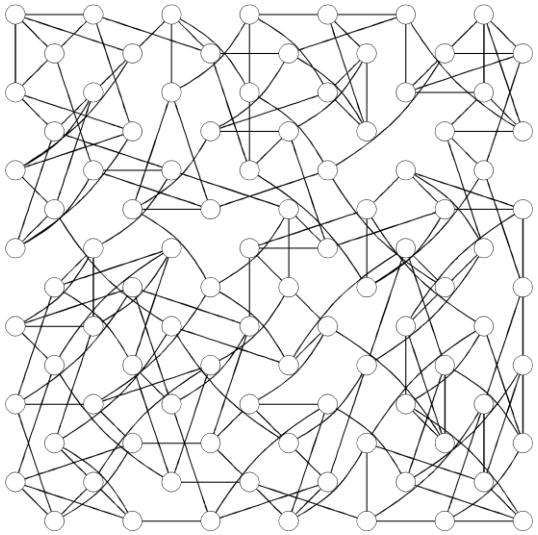
A 4-regular 3-restricted diagrid graph of size $N = 98$

Step 1



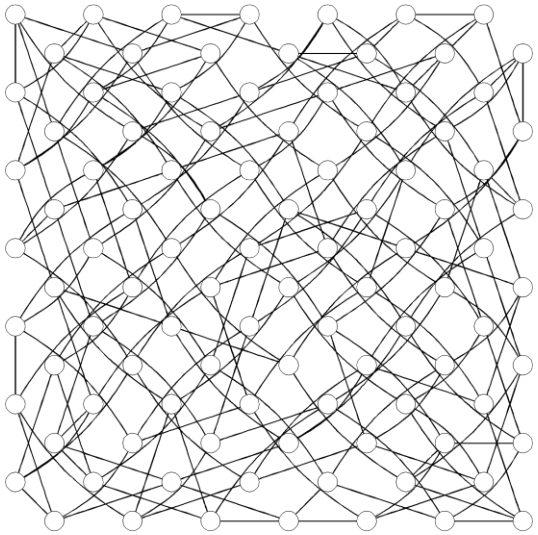
diam $= \infty$
ASPL $= \infty$

Step 2



diam $= 9$
ASPL $= 4.370$

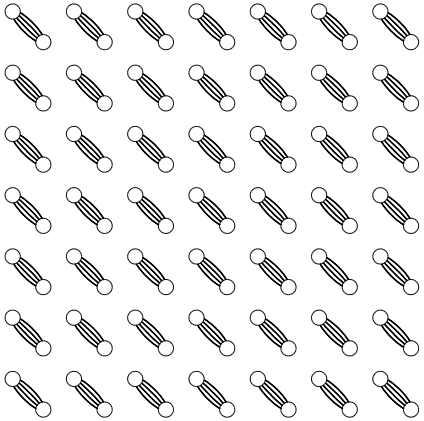
Step 3



diam $= 5$
ASPL $= 3.459$

A 4-regular 3-restricted diagrid graph and grid graph

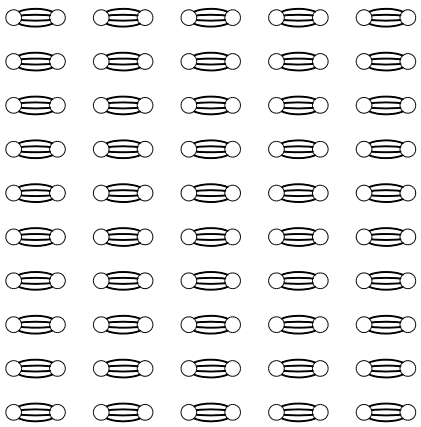
Step 1



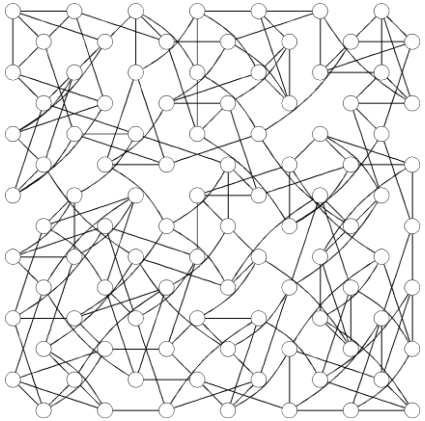
A diagrid graph of size 98

diam = ∞
ASPL = ∞

A grid graph of size 100

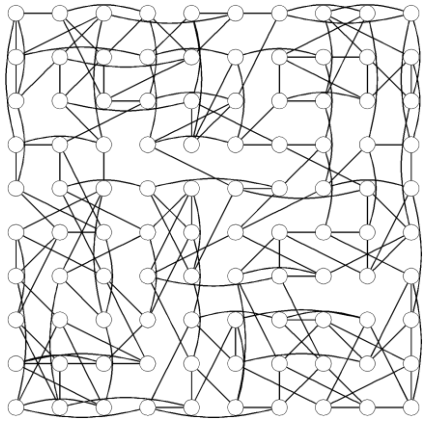


Step 2

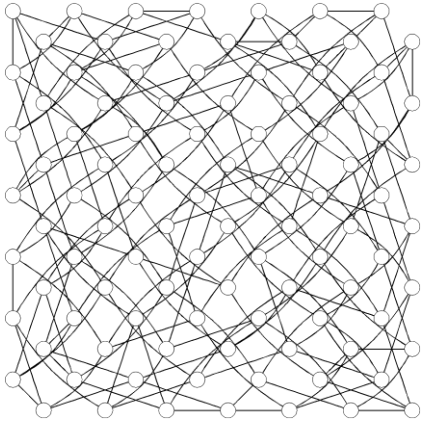


diam = 9
ASPL = 4.370

diam = 10
ASPL = 4.575

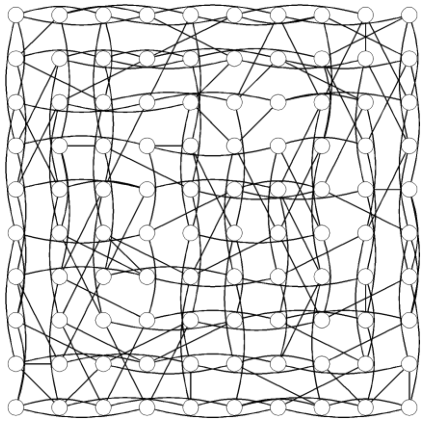


Step 3

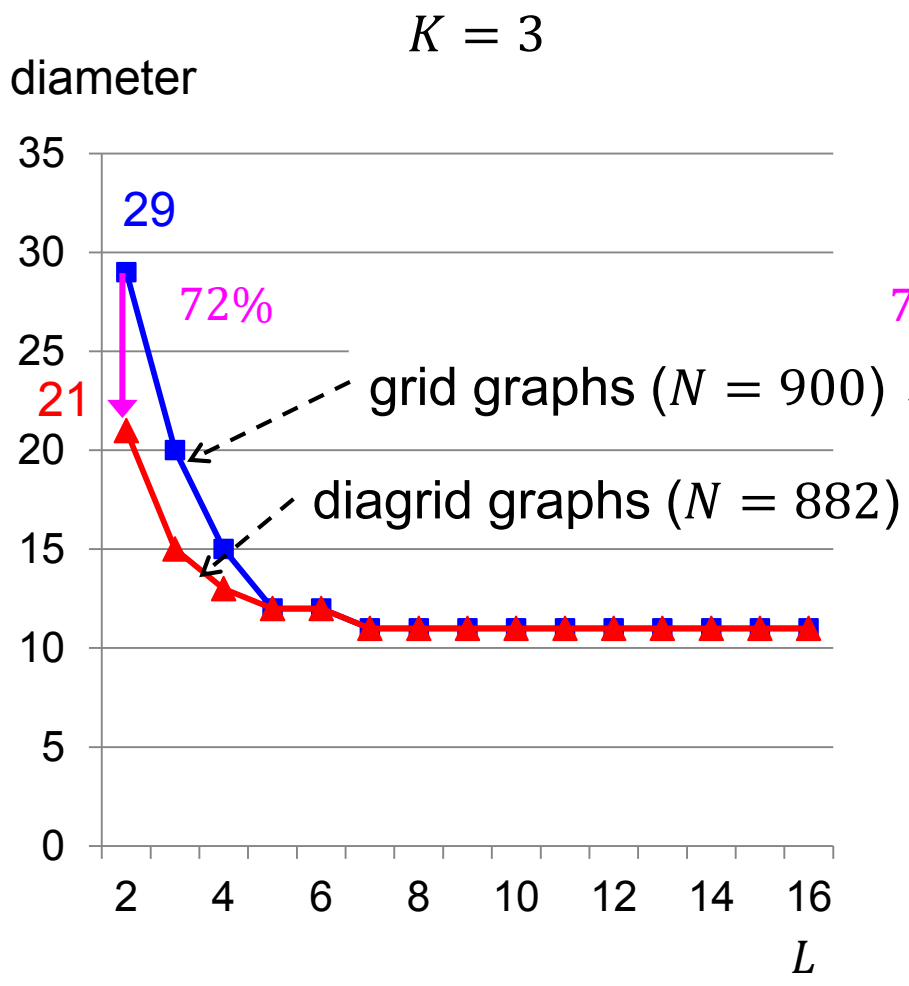


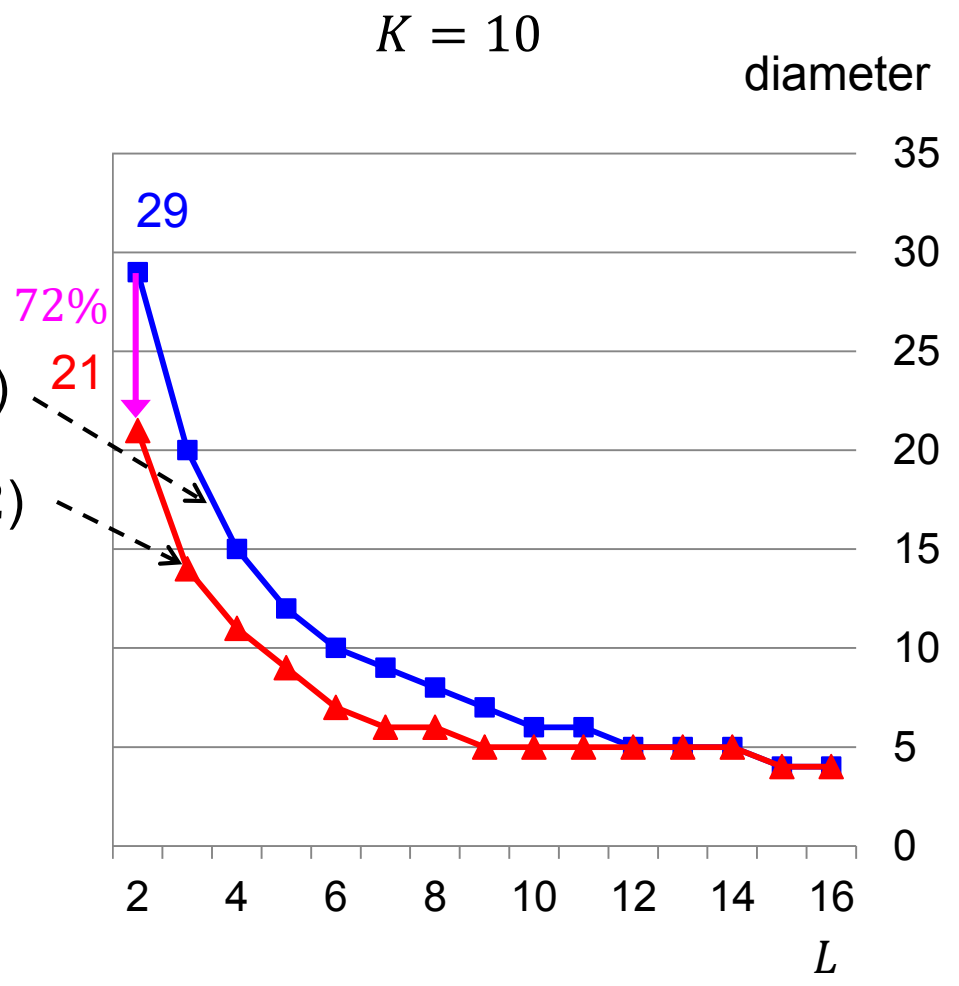
diam = 5
ASPL = 3.459

diam = 6
ASPL = 3.443



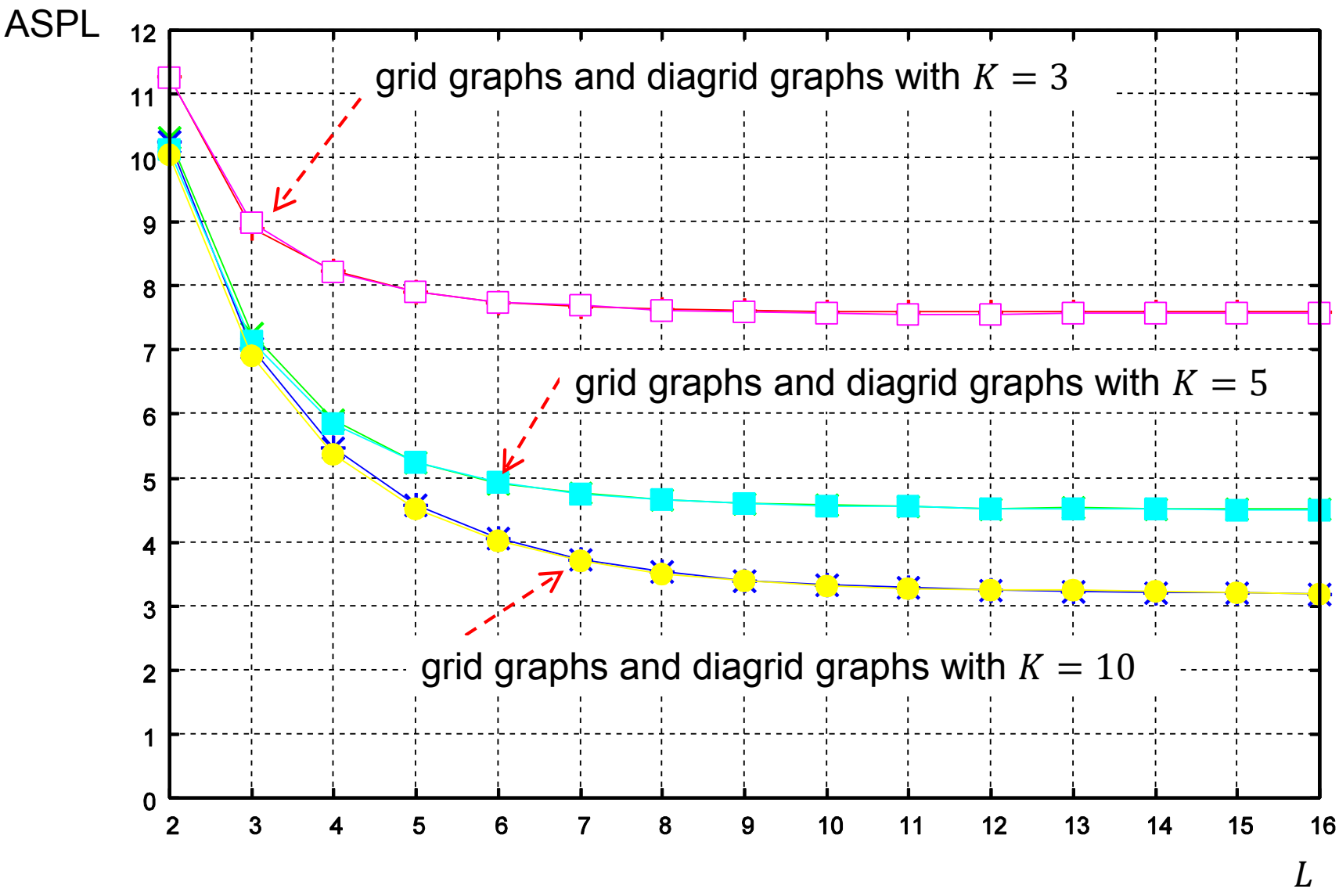
The diameter of K-regular L-restricted grid graphs and diagrid graphs for $K = 3, 10$





$$\frac{\text{The maximum length of two nodes in the diagrid graph}}{\text{The maximum length of two nodes in the grid graph}} = \frac{\sqrt{2N}-1}{2\left(\sqrt{N}-1\right)} \cong \frac{\sqrt{2}}{2} \cong 70.7\%$$

The ASPL of grid graphs and diagrid graphs for $K = 3, 5, 10$



The ASPL of grid graphs and diagrid graphs

The average Manhattan distance of all pairs of two nodes

A grid graph

$$\iint_D \iint_D (|x-x'|+|y-y'|)dxdydx'dy'$$

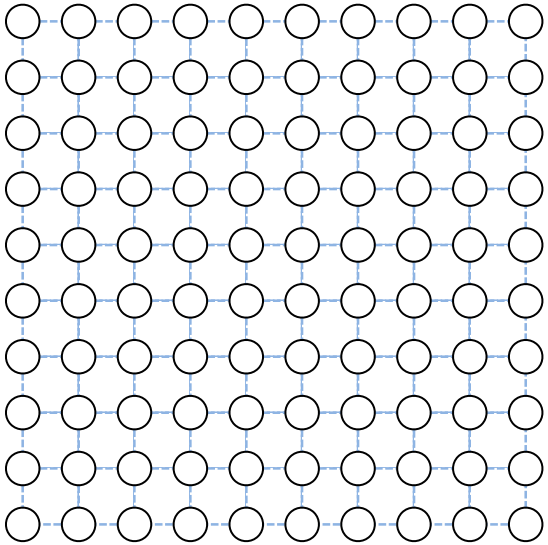
$$= \frac{2}{3}\sqrt{N} \cong 0.667$$

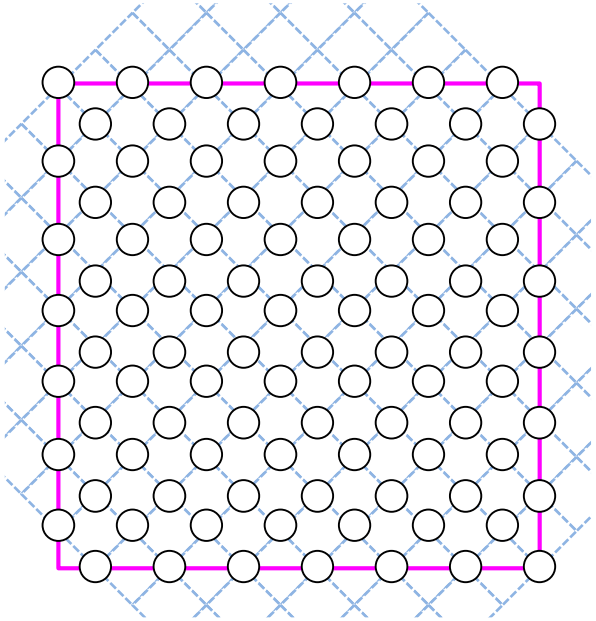
A diagrid graph

$$\iint_D \iint_D \sqrt{2}\max(|x-x'|,|y-y'|)\,dxdydx'dy'$$

$$= \frac{7\sqrt{2}}{15}\sqrt{N} \cong 0.660$$

The average distances of nodes are almost the same.

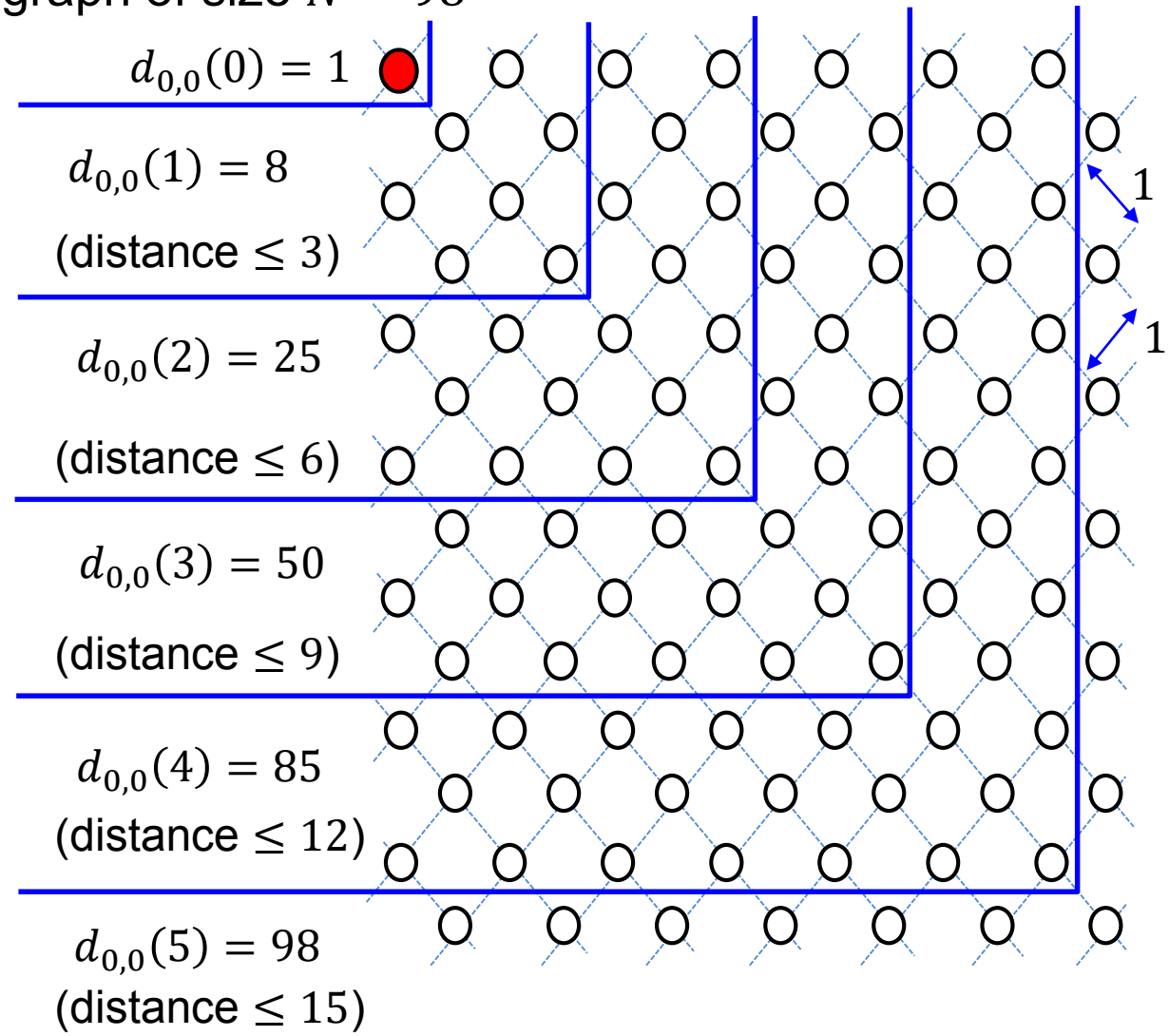




Lower Bound of the ASPL for diagrid graphs based on length L

$d_{x,y}(i)$: the number of nodes with the distance $\leq i \times L$ from node (x, y)

A 3-restricted diagrid graph of size $N = 98$



The values of m, $d_{0,0}$ and $md_{0,0}$ for a diagrid graph

A 4-regular 3-restricted diagrid graph of size $N = 98$

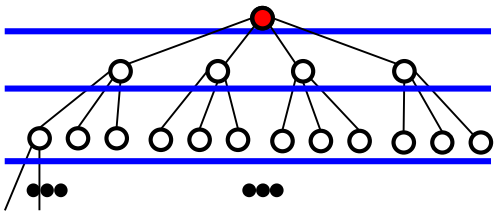
i	0	1	2	3	4	5
$m(i)$	1	5	17	53	98	98
$d_{0,0}(i)$	1	8	25	50	85	98
$md_{0,0}(i)$	1	5	17	50	85	98

Lower bound of the diameter

The number of nodes reachable in i hops from node (x, y)
$\leq md_{x,y}(i) = \min(m(i), d_{x,y}(i))$

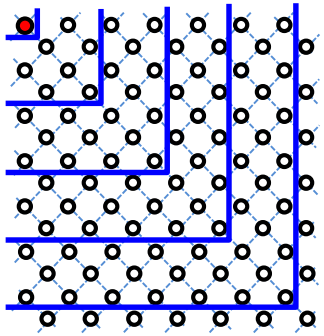
A K-regular graph

$m(i)$: the maximum number of nodes reachable in i hops from a node



An L-restricted diagrid graph

$d_{x,y}(i)$: the number of nodes with the distance $\leq i \times L$ from node (x, y)



The values of m, $d_{0,0}$ and $md_{0,0}$

A 4-regular 3-restricted diagrid graph of size $N = 98$

i	0	1	2	3	4	5
$m(i)$	1	5	17	53	98	98
$d_{0,0}(i)$	1	8	25	50	85	98
$md_{0,0}(i)$	1	5	17	50	85	98

Lower bound of the diameter

The number of nodes reachable in i hops from node (x, y)

$$\leq md_{x,y}(i) = \min(m(i), d_{x,y}(i))$$

A 4-regular 3-restricted grid graph of size 10×10

i	0	1	2	3	4	5	6
$m(i)$	1	5	17	53	100	100	100
$d_{0,0}(i)$	1	10	28	55	79	94	100
$md_{0,0}(i)$	1	5	17	53	79	94	100

Lower bound of the diameter

Conclusions

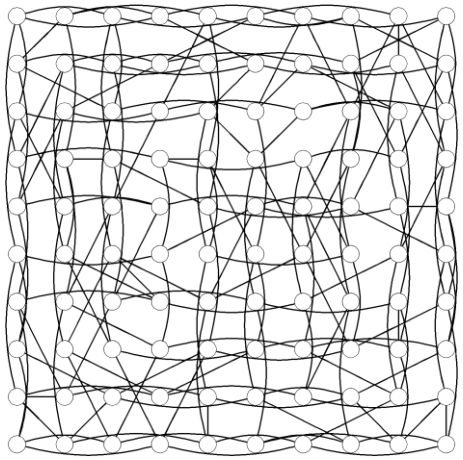
We have presented

(1) the randomly optimized K-regular L-restricted grid graph

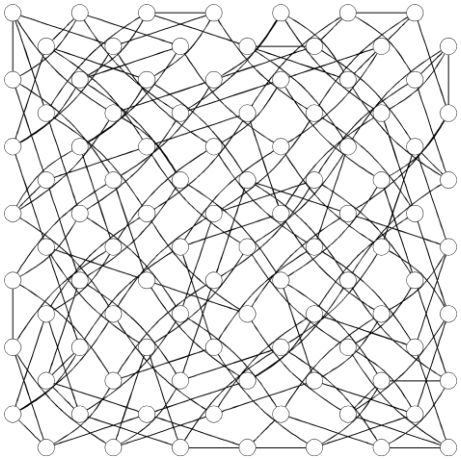
(2) the theoretical lower bounds of the diameter and ASPL

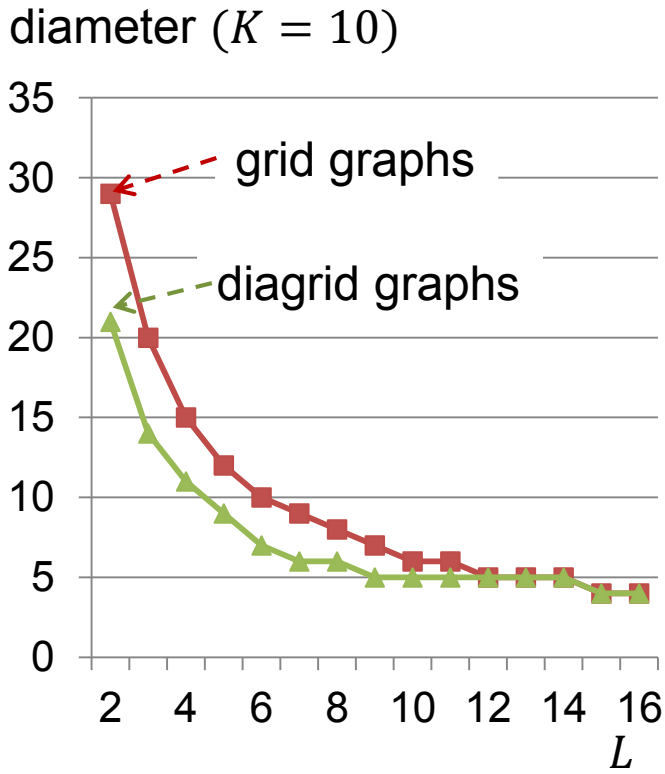
(3) the K-regular L-restricted diagrid graph (on a diagonal grid) with smaller diameter

K-regular L-restricted grid graph



K-regular L-restricted diagrid graph





MANET向けの通信性質を考慮した公平性ルーティングプロトコル

Fair Communication Amount Rounting Protocol for MANET Considering Characteristic of Communication

吉町 優　　真鍋 義文
Masaru Yoshimachi　　Yoshifumi Manabe

工学院大学 工学研究科 情報学専攻
Department of Computer Science, Kogakuin University

1　あらまし

Mobile Ad Hoc Network(MANET)[1] は基地局を必要としない、移動体の無線端末によって構成される自律分散ネットワークである。通信可能範囲外のノードとの通信は他のノードにパケットを中継してもらうマルチホップ通信により可能となる。中継に適した位置にいるノードの中継データ量の負担は大きく、ノードによって中継データ量の負担の不公平性が生じている。MANET向けに多くのプロトコルが提案されており、スループット改善 [2],[3] や、消費電力に関する研究 [4]、安定性に関する研究 [5] 等がされている。しかしながら、MANET における公平性に関する研究 [6][7] は少ない。自ノードが送信元もしくは宛先である自ノード通信量と、中継データ通信量の比を利用した経路選択アルゴリズム [6] があるが、自データの定義を送信元あるいは宛先端末としている。この定義では他ノードからのデータ要求に答えるサーバノードの通信も自ノード通信に含まれる。しかしこの定義では多くの情報を発信するサーバー性質のノードに負担が集中してしまう。故に通信性質を考慮した定義を検討する必要がある。またこのプロトコルはルーティングプロトコルとして AODV をベースとしている。このため AODV の Route Discovery プロセスの過程で公平性を達成するのに適した経路を破棄している場合があり、公平性を達成するために改善した Route Discovery のアルゴリズムが必要である。本提案手法ではデータ分類の定義の変更と、通信の為の経路の構築のアルゴリズムを調整し、通信量に対する公平性を効果的に達成するルーティングプロトコルを提案する。

2　既存のルーティングプロトコル

2.1　AODV (Ad hoc Ondemand Distance Vector)[8]

2.1.1　AODV の特徴

AODV は ReactiveType の代表的なルーティングプロトコルであり、通信が必要な時に経路の構築を行う。AODV は各ノードがパケット転送に関する経路表を持ち、この経路表より次にパケットを転送するノードを決定する。

2.1.2　AODV の Route Discovery

ADOV による Route Discovery アルゴリズムを示す。

1. 送信元ノードは宛先ノードへ通信が必要となった際、探索パケットである RREQ パケットを新たな RREQID で生成し、このパケットを周囲のノードに対してブロードキャストする。
2. RREQ パケットを受け取ったノードは自身の経路表に RREQ の送信元ノードとこのパケットの送信ノードの経路情報を記録し、RREQ パケットの宛先ノードが自分自身かを確認する。もし自身が宛先ノードでなく、初めて受け取る RREQID のパケットであれば、RREQ パケットにノード ID を追記し、ブロードキャストする。もしすでに受け取ったことのある RREQID のパケットであれば、そのパケットはブロードキャストせずに破棄する。
3. RREQ パケットを受け取ったノードが宛先ノードであった場合、ノードは返答のための RREP パケットを生成し、送信元ノードへ送信する。RREP パケットを受け取ったノードは自身の経路表に RREP の送信元ノードとこのパケットの送信ノードの経路情報を記録する。
4. 送信元ノードは最初に到達した RREP パケットの経路を使用経路とし、通信を開始する。

2.1.3　AODV のデータ中継

ノードはパケット転送の為に経路表をもつ。AODV においてデータ転送されるパケットは宛先ノードまでのすべての経由ノードの情報を持たない。AODV の経路表は宛先ノードとそのノードへ送るための次ホップノードを対にして記録する。この経路表に基づいて送信元ノードから宛先ノードへパケットが送信される。

2.2　ISK プロトコル [6]

ISK プロトコルは自己通信量と他通信量の比を使用することで通信量における公平性を達成する方式である。ISK プロトコルは AODV の経路選択のアルゴリズムを変更し、経路選択により通信の公平性を達成する。

2.2.1 ISK プロトコルの特徴

ISK プロトコルは RREP により構築された経路に対して公平度を算出して各経路の評価を行う。経路評価を行うために各ノードは自身の公平値である Dratio を持つ。

$$Dratio = \frac{OwnData}{OtherData} \quad (1)$$

OwnData = 送信元または宛先が自身であるパケット
OtherData = 他ノードへの中継データであるパケット

ISK プロトコルは Dratio という各ノードにおける通信量に対する公平性を示す指標を持つ。OwnData は自己通信の通信量を示している。自己通信の定義はデータパケットの送信元または宛先が自身である場合である。OtherData は他通信の通信量を示している。他通信の定義はデータパケットの送信元または宛先が自身でない場合であり、中継パケットである。

2.2.2 ISK プロトコルの Route Discovery

ISK プロトコルの経路選択アルゴリズムを示す。

1. このプロセスは AODV と同じである。
2. このプロセスは AODV と同じである。
3. このプロセスは Dratio の値を加算する以外は AODV と同じである。もし RREQ パケットを受け取ったノードが宛先ノードであった場合、このノードは RREP パケットを生成する。この RREP パケットには Dratio に関するヘッダーを持ち、生成時の値は 0 である。RREP パケットを受け取ったノードはパケット内の Dratio に自身の Dratio を加算し送信元ノードまで到達するまでブロードキャストを続ける。
4. 送信元ノードは到着した RREP パケットの中から最小の値の Dratio の経路を使用経路とし、通信を開始する。

3 既存プロトコルの問題点

3.1 AODV における公平性を達成するための問題点

宛先ノードを探索するために多くの MANET のルーティングプロトコルはブロードキャストを行うが、ブロードキャストはネットワーク全体の負荷となる。このため AODV は Route Discovery プロセスでブロードキャストによるオーバーヘッドを減らす為に、同一の RREQ を受信したら再送信しない。これにより探索パケットである RREQ はソースノードを中心に環状に広がり、宛先ノードへ到達する。AODV の RREQ を破棄するルールは同一の RREQID を持つパケットを二度目以降受信したパケットを破棄することである。最初に受信した RREQ のみを再送信するため、構築される経路は応答の早い通信ホップ数の少ない経路となりやすい。しかしながら、後から到達した RREQ パケットは破棄されるので、これらの RREQ パケットを用いた経路の構築はされない。AODV をベースとしている ISK プロトコルでは構築された経路の中から経路の公平性を評価し使用経路として選出するため、最適な Dratio の経路を選択できない場合がある。

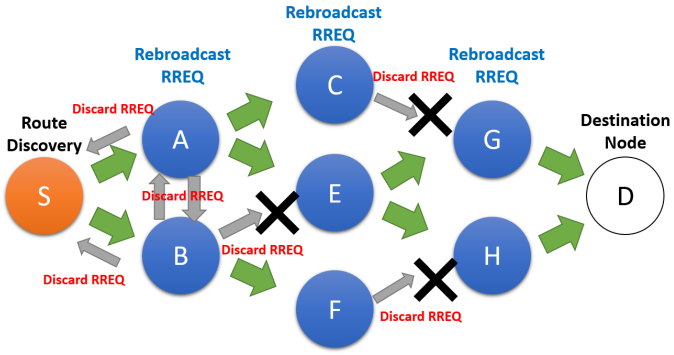


Fig.1 AODV の Route Discovery プロセス

Fig.1 に AODV の Route Discovery プロセスの例を示す。ソースノード S は周囲のノードへ宛先ノードを探索する RREQ パケットをブロードキャストする。周囲のノードは初めて受信した RREQ であれば再送信するが、再度同じ RREQ を受信した場合は破棄する。C → G や B → E、F → H の RREQ が破棄されているが、これは先に他のノードから同一の RREQ をすでに受信しているため破棄されるためである。宛先ノードに届くまでブロードキャストが続き、最終的には Fig.1 に示すように経路の構築が完了する。今回の例では A → E → G → D と A → E → H → D という二つの経路が構築された。これらの 2 つの経路は多くの同じノードを経由する経路となっており、AODV においては独立した経路が構築される可能性が低い。ISK プロトコルのような経路選択アルゴリズムにおいては構築済みの経路から任意の経路を選択することによって公平性を達成しているため、選択可能な経路が非常に似通っており、選択によって得られる効果が薄い。故に、公平性を達成するためには AODV の Route Discovery プロセスを調整する必要がある。

3.2 ISK プロトコルの問題点

先行研究である ISK プロトコルは通信を自データと他データに分類しこれらの通信量の比より公平値を算出し、通信経路のノードの公平値より経路評価を行っている。ISK プロトコルはノード間の公平性の評価に問題があると考える。このプロトコルでは自己データ通信の定義をユニキャスト通信における送信元ノードと宛先ノードが自己の為の通信をしているという定義であるが、実際の通信においてこれらの通信は自己通信とならない場合がある。MANET 上で重要なデータを持ち、多くのノードから通信要求を受け、配信しているノードは自己の為の通信をしているとは考えづらい。しかしながら、ISK プロトコルの定義では他のノードからの通信要求に応答するノードの通信は自己通信となる。故に、多くのノードから通信要求を受けるサーバー性質のノードは自己通信が増加する。このプロトコルにおいて自己通信が他通信より比が多いと、データ中継ノードとして選出されやすい動作をするため、このサーバー性質ノードは中

継ノードとして選出されやすくなる。これは他の通信要求にこたえているにも関わらず、他の通信の中継を行う機会が増えることを意味しており、サーバー性質ノードへの負荷が集中している。これは重要な情報を発信するノードへの不公平性が存在し、ネットワーク運用における懸念点である。

ISK プロトコルの特徴を示すため、C 言語を用い作成したネットワークシミュレータにて各ノードの Dratio と中継データを越ためのシミュレーションを行った。100 ノードを 500m x 500m のスペースにランダムに配置し、100 ノードのうち 25 ノードをサーバー性質ノードとし、残りの 75 ノードをクライアント性質ノードとする。サーバー性質ノードとクライアント性質ノードが対となるようなランダムな通信を生成する。ノードの移動速度は平均 4km/h である。

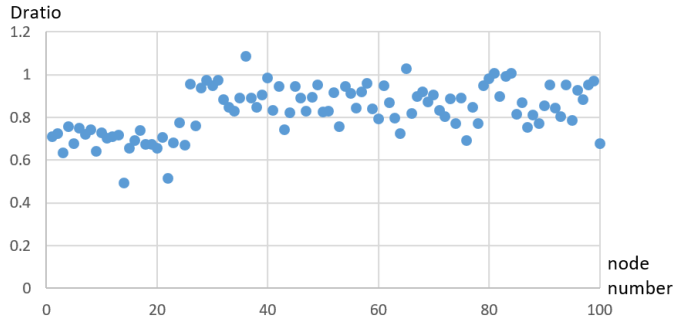


Fig.2 ISK プロトコルによるノードの Dratio 値

Fig.2 はこのシミュレーションにおける各ノードの Dratio を示している。ISK Protocol の Dratio の評価式より、Dratio は値が小さいほど自己通信の比が大きい為、ノードは満足している状態である。したがって、サーバー性質ノードである 1 から 25 ノードはクライアントノード (26 から 100) と比較し、Dratio が小さい為、中継ノードとして選出されやすい状態であり、選出されることでノード全体で Dratio を均一にする働きがある。

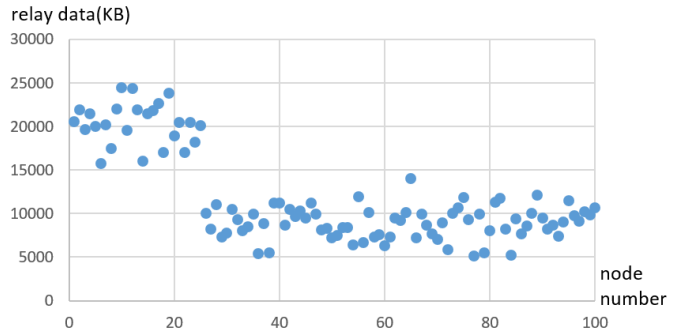


Fig.3 ISK プロトコルによるノードの中継データ量

Fig.3 はこのシミュレーションにおける各ノードの中継データ量を示している。Fig.3 はサーバー性質ノードの中継データ量が他ノードと比較し多いことを示している。これは Dratio により中継ノードの選出が行われたため、サーバー性質ノードが多くデータの中継をしている。Fig.2, Fig.3 より、サーバー性質ノードは他の通信要求に答えているのにも関わらず自己通信として扱われるため、パケット中継において多くの負担をしている。他の通信要求を答えれば答えるほど、さらに中継ノードとして選出されるというケースがあり、通信を考慮した自己通信と他通信の分類をする必要がある。

4 提案手法

我々は通信量における公平性を達成するための AODV をベースとした新しいプロトコル FRAODV(Fair Routing based on AODV) を提案する。MANET において公平性を達成するためには公平性の為に調整された Route Discovery プロセスと適切な使用経路の選択が必要である。これらの問題点を前述してある通り、我々は公平性達成のために 2 つの提案をする。一つ目は AODV の Route Discovery プロセスを変更し、効果的な公平化ルーティングを達成する。二つ目は先行研究の問題点である公平性を考慮した経路制御をする際にサーバー性質のノードへの中継コストの偏りを改善する。これらの改善による、提案手法の FRAODV と既存手法の比較を行う。

4.1 公平性のための Route Discovery アルゴリズムの調整

公平性を達成するために AODV の Route Discovery プロセスを改善する。AODV の問題点で述べたとおり、AODV は Route Discovery プロセスにおけるフラッディングを抑制するために Route Discovery プロセス時にノードは一度受信したことのある RREQID のパケットを受信した際にそれを破棄する。この処理により公平性を達成するために最適な使用経路の構築ができない場合がある。故に我々は公平性を達するために AODV の Route Discovery プロセスを変更する。AODV 改善するためには公平度を調整する効果の高い経路を構築することが必要である。故に Route Discovery プロセス中に経路評価を行い、この評価に基づく RREQ のブロードキャストを行う。ISK プロトコルではノードの公平度を示す値を Dratio として扱っていたが、ISK プロトコルと区別するため提案手法の評価値を FV(Fair Value) とする。

4.1.1 提案手法の Route Discovery アルゴリズム

1. 送信元ノードは宛先ノードへ通信が必要となった際、探索パケットである RREQ パケットを新たな RREQID で生成する。この RREQ パケットには FV というヘッダーを持ち初期値は 0 である。このパケットを周囲のノードに対してブロードキャストする。

2. RREQ パケットを受け取ったノードは自身の経路表に RREQ に関する経路情報を記録し、RREQ の宛先ノードが自分自身かを確認する。もしノードが RREQ パケットの宛先ノードではなく、まだ受け取ったことのない ID の RREQ である場合、このノードは RREQ パケットに自身の ID を追記し、パケットの FV に自身の FV を加算した後にこのパケットをブロードキャストする。もしそのパケットが既にノー

ドが一度受信したことのある同一の RREQID のパケットの場合、以前受信した同じ ID の RREQ の FV と新しい RREQ の FV の値を比較する。もし新しい RREQ の方が値が小さい場合、自身の RREQ の FV に関する記録を新しい値に差し替え、パケットの FV に自身の FV を加算した後にこのパケットをブロードキャストする。

3. もし RREQ パケットを受け取ったノードが宛先ノードであった場合、この RREQ のパケットの FV 値を確認する。もしすでに受け取った FV 値よりも小さい FV である RREQ パケットであった場合、ノードは RREP パケットを生成し、送信元のノードへ送信する。
4. 送信元ノードは最初に到着した RREP パケットの経路を使用経路として通信を開始し、後から到達した RREP の方が FV が小さい場合使用経路を切り替え通信を行う。

4.1.2 公平性のための AODV の Route Discovery アルゴリズムの調整

AODV の Route Discovery プロセスを改善した提案手法を適用した Modified ISK と Original ISK の比較を行う。公平性の為の経路評価に関しては ISK プロトコルである Dratio を共に使用し Route Discovery プロセスによる変化を示す。シミュレーションパラメータは前回のシミュレーションと同じである。

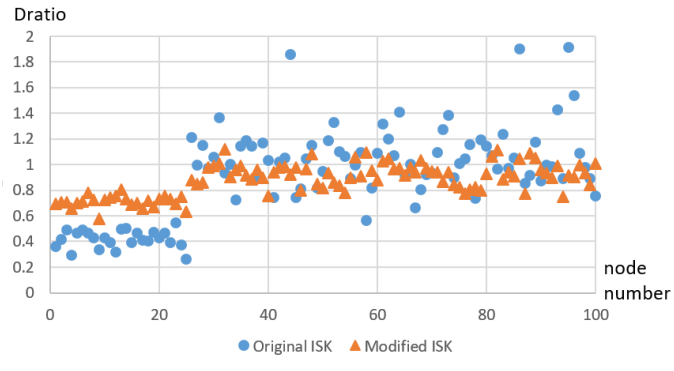


Fig.4 Modified ISK と Original ISK の Dratio 値の比較

Fig.4 は ISK プロトコルの経路選択アルゴリズムを用いて、Route Discovery プロセスが異なる Modified ISK と Original ISK の比較を示している。Fig.4 より Modified ISK は Original ISK と比較し、ノードの Dratio 値の分散が小さいことを示している。経路選択アルゴリズムにより全ノードの Dratio が一定に収束する動作をするが、提案手法の方が効果が高い。Original ISK である AODV は公平性を達成するのに適した経路を Route Discovery プロセスの課程でを調整するのに適した経路を破棄している場合があるが、提案手法である Modified ISK は Route Discovery プロセス時に公平性を達成するのに適した経路を構築するため、Fig.4 のような Dratio の調整効果の違いが得られた。故に Modified ISK である提案手法は公平性を達成するための一定の効果が得られている。

4.2 データ分類の改善

各ノードの公平度を示す値を計算するための ISK プロトコルの定義 Dratio を修正する。公平値は自己通信と他通信の二つから構成される。先行研究では自己通信は送信元または宛先が自分である通信 としており、この定義により配信を多く行うサーバー性質のノードへの負荷の集中が発生している。したがって、この定義を見直す。一般的にデータ配信を行うサーバー性質ノードと通信要求を行うクライアント性質とのデータトラフィックの関係は下記であると仮定する。

サーバーノード：Upload Data ＞ Download Data

クライアントノード :Download Data ＞ Upload Data

Upload と Download のトラフィック量を違いを利用することでサーバー性質ノードへの負荷の軽減を試みる。この性質を使い公平値算出式を下記のように定義する。

$$FV(Node) = \frac{RelayData + UploadData}{DownloadData} \quad (2)$$

$$SelectPath = argmin\{FV(Path)|Path \in A\} \quad (3)$$

式 (2) の FV は FRAODV で用いる公平値で、FV は経路上のノードの FV 値の合計となる。式 (3) はソースノードにおける経路選択を示している。A は RREP によって構築された経路の集合である。自己通信を Download と定義している理由は Download 通信が自己利益の為に行う考えられるためである。また他通信を Upload と Relay と定義している理由はこれらは他ノードへの通信の意味合いが強いためである。この新しい定義により ISK プロトコルと比較し、自己通信と他通信へのより正確な分類が行われると考える。Route Discovery プロセスの変更とこのプロセス内の経路評価の変更の効果を確認するため、FRAODV と ISK プロトコルを提案手法の比較のシミュレーションを行う。

4.3 プロトコル比較のシミュレーション

FRAODV と ISK プロトコルのシミュレーションを行う。この二つのプロトコルにおいて Route Discovery のプロセスが異なっており公平性に関する経路評価式も異なっている。シミュレーションパラメータは前回のシミュレーションと同じである。

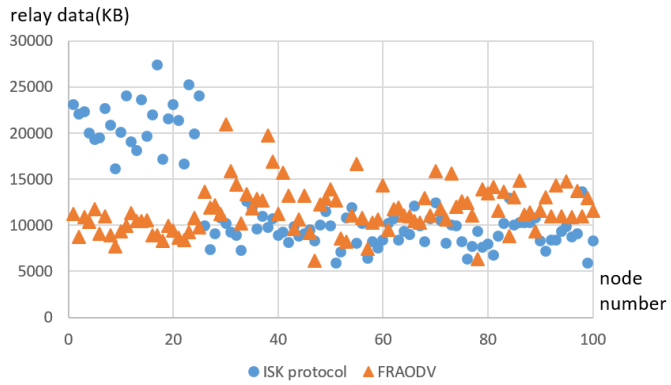


Fig.5 FRAODV と ISK プロトコルの中継データ量の比較

Fig.5 は提案手法と ISK プロトコルの中継データ量の比較を示している。このシミュレーションは Route Discovery プロセスと経路選択アルゴリズムの違いによる差を示している。サーバー性質ノードとクライアント性質ノードの中継データ量の差は提案手法の方が小さい。これは通信の分類の定義を変更したことにより、サーバー性質ノードへの負荷の集中が減少したからである。提案手法は Upload データを他ノード通信と分類したことで、ISK プロトコルの負荷集中問題が改善された。提案手法は公平性を達成するための調整効果が高いことと、自己利益と他への貢献の分類の不公平感の改善が示され、MANET の実運用においてユーザーの懸念点であった、重要な情報を持つノードが情報を多く発信すればするほど自身の負担が増加するという問題点が軽減された。

5 結論

我々は公平性を達成するための新しい AODV プロトコルを提案した。このプロトコルは公平化ルーティングを行う際に特定のノードへ中継コストが集中する問題を改善し、また AODV において公平性を達成するために適した経路の構築を行えるよう Route Discovery プロセスの変更を行った。これにより既存手法と比較しネットワーク全体でノード間の公平性が向上したといえる。今回は公平度の判断基準として Upload と Download を指標としたが、実通信において実態に伴う公平な通信が行われているかの検証と公平性を達成するためのオーバーヘッドの増加がどのように MANET へ影響するかは今後の課題とする。

謝辞

本研究は JSPS 科研費 JP26330019 の助成を受けたものです。

参考文献

[1] E.M. Royer and C-K. Toh: “A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks,” IEEE Personal Communications Magazine, pp. 46-55 (1999).

[2] R. Verma, A. Prakash, R. Tripathi, and N. Tyagi: “Throughput Enhancement of Multi-hop Static Ad-hoc Networks through Concurrent Transmission,” Proc. of IEEE Computational Intelligence, Communication Systems and Networks, CICSYN ’09. pp.482-485 (2009).

[3] S. Islam, N. Hider, T. Haque, and L. miah: “An Extensive Comparison among DSDV, DSR and AODV Protocols in MANET,” International Journal of Computer Applications, Vol. 15, pp. 22-24 (2011).

[4] Q. Dai and J. Wu: “Computation of Minimal Uniform Transmission Power in Ad Hoc Wireless Networks,” IEEE Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on, pp.680-684 (2003).

[5] R. Ramanathan and R. Rosales-Hain: “Topology Control of Multihop Wireless Networks using Transmit Power Adjustment,” IEEE INFOCOM, vol. 2, pp. 404-413 (2000).

[6] J. Ichikawa, S. Sakata, and N. Komuro: “Fair Energy Consumption-Based Routing Control Considering Data Traficc for Mobile Ad hoc Networks,” IEICE Technical Report, IN2013-171, pp.163-168 (2013)(In Japanese).

[7] M. Yoshimachi and Y. Manabe: “Battery Power Management Routing Considering Participation Duration for Mobile Ad Hoc Networks,” JACN 2016 Vol.4, pp. 13-18(2015).

[8] C. Perkins, E. Belding-Royer, and S. Das: “Ad hoc On-Demand Distance Vector (AODV) Routing,” RFC 3561 (2003).

無線センサネットワークにおけるベースステーションからの無線波情報を利用した負荷分散動的ルーティングアルゴリズム

渡部連太郎 角川裕次 増澤利光
大阪大学 大学院情報科学研究科 コンピュータサイエンス専攻

概要 多数の安価な小型無線装置内蔵センサが相互にマルチホップ通信を行うことで構成される無線センサネットワークは, その適用分野の広さから様々なアプリケーションに利用されている．無線センサネットワークにおいてはセンサ間での消費電力を平均化し，ネットワークの存続時間を延ばすことが重要である. 本研究は各センサノードが残存電力やセンシング情報収集施設からの距離を考慮した上で，メッセージの送信先を確率的に決定する手法の考案を目的とする．本稿ではその手法の概要を説明し，予備実験により手法の妥当性を簡易に検証した結果を示す.

1 はじめに

無線センサネットワーク (WSNs) とは，多数の安価な小型無線装置内蔵センサ（ノード）をフィールドに配置し，ノードがセンシングにより得た情報を収集施設（ベースステーション）へ転送することでフィールド全体の情報を収集するネットワークである．WSNs は対象の追跡や環境モニタリングなど様々なアプリケーションに利用され，その適用分野の広さから大きく期待されている．WSNs においてセンサは電池駆動である場合が多く，使用電力に制限がある．よってネットワークの存続時間を高めるために各ノードの消費電力を抑え，かつノード間での消費電力を平均化することが重要である．各ノードは通信可能距離内のノードと相互に通信することが可能であり，取得した情報はマルチホップ通信によりベースステーションに集められる．このようなネットワークの性質上，ベースステーションに近いノードほどデータ転送を行う頻度が高いため，消費電力が非均一となりネットワークの存続時間を短くする原因となる可能性が高い．そこでネットワークの存続時間を延ばすため，ノードの残存電力やベースステーションからの距離によって適切なメッセージ転送経路を決定的に計算する手法が考案されている．

本研究では，各ノードが残存電力やベースステーションからの距離を考慮した上で，メッセージの送信先を確率的に決定する手法の考案を目的とする．本手法では各ノードが確率的かつ局所的にメッセージの送信先を決定することで，特定のノードで大量のメッセージが生成される場合にも高確率で負荷を分散し，ネットワーク存続時間を延ばすことが可能である．またネットワーク内のどのベースステーションやノードもメッセージの転送経路全体を把握せず動作するため，頑健性が高く，適用するアプリケーションのセキュリティ面での安全性向上を見込むことができる．

本稿では提案手法の概要と，手法の妥当性を簡易に検証する予備実験の結果を紹介する．

2 諸定義

WSNs のネットワークやアーキテクチャのモデルは，適用するアプリケーションに応じて様々なものが用いられる. 本稿で提案する手法と予備実験においては以下のモデルを定義する.

2.1 ネットワークモデル

2.1.1 センサノード

全てのセンサノード（以降ノード）の初期配置はランダムで大局的に一意な ID は割り当てられない. センシング可能領域は自身から半径 R_s 以内，他ノードとの通信可能領域は $R_c(\leq R_s)$ 以内とする. またノードは初期配置から移動せず，フィールドにおける自身の座標を認識する GPS などの機能は保有していないと仮定する.

2.1.2 ベースステーション

ランダムに配置された，フィールドに 1 つのベースステーションを仮定する. ベースステーションには安定して電力が供給される. またネットワーク全体に到達する強力な電波を発信することが可能なである.

2.2 電力消費モデル

以下の電力消費モデルを仮定する.

$$\text{送信時：} \quad E_{Tx}(l,d) = lE_{elec} + l\varepsilon_{fs}d^2$$
$$\text{受信時：} \quad E_{Rx}(l) = lE_{elec}$$

l をメッセージの bit 長，d を送受信するノード間の距離とする.E_{elec} は信号の符号化方式や振幅, フィルタ方式などに基づく係数である．ε_{fs} は信号増幅器の電力消費に関わる係数である.

3 提案手法の概要

本章では，各ノードがメッセージの転送先をベースステーションからのホップ数と隣接ノードの残存電力，隣接ノードとの距離に基づいて確率的に決定する手法を提案する．メッセージの転送経路が確率的に変化することで，特にメッセージが特定のエリアで大量に発生する場合に，特定の経路上のノードの集中的な電力消費を緩和することができる．本手法は各ノードが自身と各隣接ノードの，ベースステーションからのホップ数を知っていることを仮定している．ノード i の隣接ノード集合を $\mathbb{N}_i$，ベースステーションからのホップ数を d_i とする. 各ノード i はベースステーションからのホップ数によって

隣接ノード集合を以下の $\mathbb{L}_{d_i-1}, \mathbb{L}_{d_i}, \mathbb{L}_{d_i+1}$ へ分割し，それぞれへの送信確率を設定する．

$$\begin{aligned}\mathbb{L}_{d_i-1} &= \{j \in \mathbb{N}_i | d_j = d_i - 1\} \\ \mathbb{L}_{d_i} &= \{j \in \mathbb{N}_i | d_j = d_i\} \\ \mathbb{L}_{d_i+1} &= \{j \in \mathbb{N}_i | d_j = d_i + 1\}\end{aligned}$$

ノード i はメッセージ送信時，設定された確率に基づき送信する隣接ノード集合を決定する．決定したノード集合中に複数のノードが存在する場合，自身とそれらのノードの距離と残存電力に基づき，送信先のノードを 1 つ決定する．

4 予備実験

メッセージが特定のエリアで集中的に発生した場合，必ずしもメッセージがベースステーションへ近づかないような転送をすることで，ネットワークの存続時間が延びることをシミュレーション実験により検証した．提案手法と比較手法において，各ノードはベースステーションからのホップ数に基づき，メッセージの送信先となる隣接ノード集合を選択する．このとき選択したノード集合が空集合ならばそのメッセージは消失する．アルゴリズム開始から消失したメッセージの合計数が一定値 LL を越えるまでのラウンド数をネットワーク存続時間とみなし，各アルゴリズムを評価する．

4.1 各種定義

今回の実験では簡単のため，ノードがメッセージを送信する隣接ノード集合を決定した後，複数の送信先ノード候補が存在した場合，それらとの距離やそれらの残存電力に依存しない一様ランダムな送信先ノードの選択を行う．またメッセージはランダムに選択されたノードで I 回連続で発生するとする．

4.1.1 比較手法

提案手法と比較する手法は，各ノードが必ずベースステーションへ近づくようにメッセージを送信する手法を用いる．ノードはメッセージを送信する際，自身よりベースステーションからのホップ数が小さい隣接ノード集合を選択する．複数のノードが含まれる場合は，提案手法と同様，一様ランダムにその中から 1 つのノードを選択する．

4.1.2 確率設定

メッセージ送信時に利用する確率の設定について説明する．ノード i が，ベースステーションからのホップ数が $h(d_i - 1 \leq h \leq d_i + 1)$ であるノード集合 $\mathbb{L}_h$ のいずれかにメッセージを送信する確率を $P(d_i, h)$ とする．今回は，ノード i が送信したメッセージがベースステーションへ到達するまでに転送回数の期待値が $kd_i(k$ はパラメータとして入力値を設定) となるように $P(d_i, d_i - 1)$,$P(d_i, d_i)$,$P(d_i, d_i + 1)$ を設定する．具体的には以下の式を満たすよう，ランダムに設定する．提案手法については $k > 1$ とする．

$$\begin{aligned}\frac{1}{k} \leq P(d_i, d_i - 1) &\leq \frac{1}{2} + \frac{1}{2k} \\ P(d_i, d_i) &= 1 + \frac{1}{k} - 2P(d_i, d_i - 1) \\ P(d_i, d_i + 1) &= P(d_i, d_i - 1) - \frac{1}{k}\end{aligned}$$

一方，比較手法においては必ずベースステーションへ近づくようにメッセージを送信するため，$k = 1$ として以下のように各確率を定めたものとみなすことができる．

$$\begin{aligned}P(d_i, d_i - 1) &= 1 \\ P(d_i, d_i) &= 0 \\ P(d_i, d_i + 1) &= 0\end{aligned}$$

4.1.3 パラメータ

実験に使用する各種パラメータの設定値を表 1 に示す．

パラメータ	値
フィールドサイズ	(100×100)m
ノード数	100, 200
初期電力	2.0J
通信可能距離	20m
k	1.0, 2.0 ,...,10.0
I	10, 100, 1000
E_{elec}	50 nJ/bit
ε_{fs}	10 pJ/bit/m^2
l	4000 bit
LL	10000

表 1: 各種パラメータ値

4.2 結果と考察

メッセージがベースステーションへ到達するまでの転送回数期待値の係数 k とメッセージ連続発生数 I によるネットワーク存続時間の変化を記録した．ノード数が 100 の場合の結果を図 1 に，200 の場合の結果を図 2 に示す．

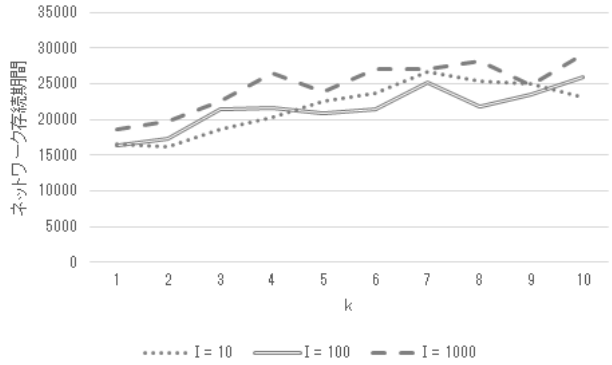


図 1: ノード数 100 での結果

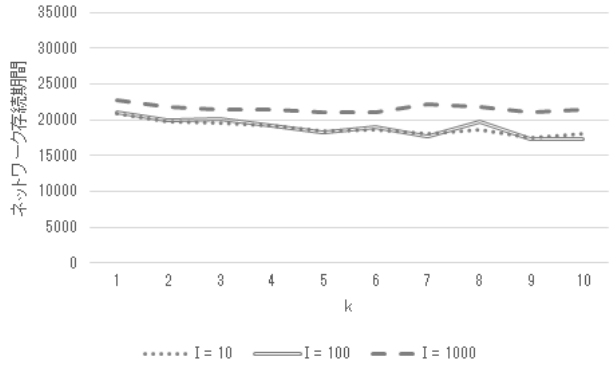


図 2: ノード数 200 での結果

横軸 k の値が 1 のデータが比較手法，2〜10 のデータが提案手法の結果を示している．

ノード数が 100 の場合，いずれの I についても，比較手法に対して提案手法の方がネットワーク存続時間が長くなり，かつ k が増加するにつれてネットワーク存続時間も長くなる傾向が見られる．

一方，ノード数が 200 の場合，k が増加させるにつれてネットワーク存続時間は緩やかに減少している．これはフィールドサイズに対してノード数が多くなったことで，グラフの半径が減少，ノードの平均次数が増加したために比較手法はネットワーク存続時間が増加，提案手法は強みが薄れネットワーク存続時間が減少したためと考えられる．

5 おわりに

本稿では特定のノードで大量のメッセージが生成される場合に，一部のノードに負荷が集中することを回避することで，ネットワークの存続時間を延長する手法の概要を説明した．また予備実験により，特定のケースでは提案手法によりネットワークの存続時間が延長されることを検証した．しかし状況によっては存続時間が短くなる結果を示すケースもあるため，今後はノードの残存電力や自身との距離を考慮したメッセージ送信先の決定法を模索しつつ本手法の性質を詳細に検証する必要がある．さらに，複数のベースステーションへの対応や，強力な伝送波を送信することができるベースステーションの能力を活かした改良を行う予定である．

参考文献

[1] Wendi B. Heinzelman, Anantha P. Chandrakasan and Hari Balakrishnan, “An Application-Specific Protocol Architecture for Wireless Microsensor Networks,”IEEE Transactions on Wireless Communications, Vol.1, No.4, pp.660-670, 2002.

[2] H. Zhou, D. Luo, Y. Gao and D. Zuo, ”Modeling of Node Energy Consumption for Wireless Sensor Networks,” Wireless Sensor Network, Vol.3 No.1, pp.18-23, 2011.

[3] Shashidhar Rao Gandham, Milind Dawande, Ravi Prakash and S. Venkatesan, “Energy Efficient Schemes for Wireless Sensor Networks with Multiple Mobile Base Stations, ”in: Proc. IEEE Globecom 2003, San Francisco, CA December 1-5, vol.1, pp. 377-381, 2003.

[4] J. Cota-Ruiz, P. Rivas-Perea, E. Sifuentes and R. Gonzalez-Landaeta, ”A Recursive Shortest Path Routing Algorithm With Application for Wireless Sensor Network Localization,” in IEEE Sensors Journal, vol.16, no.11, pp.4631-4637, June, 2016.

[5] Fan Ye, A. Chen, Songwu Lu and Lixia Zhang, ”A scalable solution to minimum cost forwarding in large sensor networks,” Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on, Scottsdale, pp.304-309, 2001.

[6] D. b. Zou and Y. B. Wang, ”Adaptive energy-aware routing framework in transmission cost constrained wireless sensor networks,” 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA, pp.534-538, 2013.

[7] P.B.Manoj and Sai Sandeep Baba, ”Random Routing Algorithms for Wireless Sensor Networks,” International Journal of Advanced Research in Computer and Communication Engineering, vol.1, issue 1, March, 2012.

[8] Zhixin Liu, Qingchao Zhenga, Liang Xuea and Xinping Guana, ”A Distributed Energy-efficient Clustering Algorithm with Improved Coverage in Wireless Sensor Networks,” Future Generation Computer Systems 28, vol.28, no.5, pp.780-790, May, 2012.

[9] Y. Liao, H. Qi and W. Li, ”Load-Balanced Clustering Algorithm With Distributed Self-Organization for Wireless Sensor Networks,” in IEEE Sensors Journal, vol.13, no.5, pp.1498-1506, May, 2013.

カエルのサテライト行動に基づく連結支配集合の構築

筒井 黎　　藤原暁宏
九州工業大学大学院
Email: m232052r@mail.kyutech.jp, fujiwara@cse.kyutech.jp

概要

センサネットワークでは，情報の収集を行うために，情報伝達の経路となるセンサ集合を求める必要がある．このセンサネットワークにおける集合の一つとして，全センサに隣接する連結なセンサ集合である連結支配集合がある．

本研究では，この連結支配集合に対して，蛙のサテライト行動に基づく自律分散アルゴリズムの提案を行う．また，提案アルゴリズムをシミュレーション環境に実装し，有用性の評価を行う．

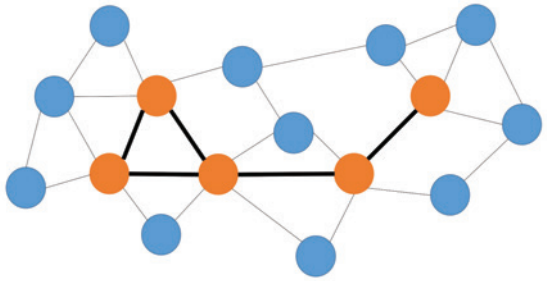


図 1: 連結支配集合

1　はじめに

センサネットワークでは，情報を収集するために，情報伝達の経路となるセンサ集合を求める必要があり，このセンサ集合の一つとして，連結支配集合がある．連結支配集合とは，全センサ集合に隣接するセンサの集合であり，さらに集合の任意のセンサと集合の他のセンサとが連結している場合，この集合を連結支配集合 [1] という．最小の連結支配集合を求める問題は NP 困難であり [2]，多くの近似解を求める方法が考案されている [3]．

本研究では，この連結支配集合に対して，蛙のサテライト行動に基づく自律分散型アルゴリズムの提案を行う．蛙には，周辺に多くの蛙が存在する場合，鳴くことを止め，体力を温存するという習性がある [4]．この習性を用いてセンサの休止，稼動を行うことにより，連結支配集合を構築する．提案アルゴリズムでは，周辺のセンサとのみ通信を行い，連結支配集合を構築することを目的とする．

また，提案アルゴリズムをシミュレーション環境に実装し，有用性の評価を行う．本シミュレーション結果より，蛙のサテライト行動を用いた提案アルゴリズムにより得られる解は，連結支配集合を構築することができた．さらに，提案アルゴリズムを改善することで，より大きい割合で連結支配集合を構築し，また，連結支配集合の数を削減することができた．

2　準備

2.1　センサモデル

本研究の対象であるセンサネットワーク $G = (V, E)$ は，センサ集合 $V = \{I_1, I_2, \cdots, I_n\}$ と通信可能なセンサ間の経路を表すリンク集合 E で定義される．センサ I_i とセンサ I_j が互いに通信可能であるとき，グラフ G 上の I_i, I_j 間に双方向通信リンク $e_{i,j} \in E$ が存在する．なお，このときのセンサ I_i と I_j は隣接しているという．また，この隣接するセンサ I_i と I_j との間のみ，メッセージの送受信が可能であるとする．さらに，通信衝突などによるメッセージの消失は起こらないものとする．

今回のセンサは隣接しているとき，双方向で通信を行えるものとする．また，センサは 2 つのモードを持ち，モード毎に機能が異なる．以下に，センサの各モードを説明する．

アクティブ : 通信領域内のセンサと通信を行い，メッセージを送信する．

スリープ : アクティブなセンサからメッセージを受信する．

2.2　連結支配集合

センサ集合 $V = \{I_1, I_2, \cdots, I_n\}$ とリンク集合 E で定義されるセンサネットワークを表すグラフ $G = (V, E)$ に対する連結支配集合とは，以下の 3 条件を満たす頂点の集合 V' である．

1. V' は V の部分集合である．
2. V' は連結である．
3. V' に含まれない全ての頂点は，V' に含まれる隣接頂点をもつ．

図 1 に連結支配集合の例を示す．図のオレンジ色の円は連結支配集合に含まれるセンサを表しており，青色の円は連結支配集合に含まれない円を表している．

次に，本研究における連結支配集合の評価指標について説明する．本研究では，連結支配集合の構築アルゴリズムを提案するが，評価指標は以下の通りである．なお，本提案アルゴリズムはほとんど通信を行わない分散アルゴリズムであり，必ずしも連結支配集合が得られるわけではないことに注意が必要である．

評価指標：

- 出力として得られた連結支配集合のセンサ数
- 出力されたセンサ集合が連結支配集合の条件を満たす割合

3 蛙のサテライト行動に基づく連結支配集合構築アルゴリズム

3.1 蛙のサテライト行動

まず最初に，提案アルゴリズムの基本アイデアとなる蛙のサテライト行動について説明する．蛙のサテライト行動とは，蛙の発声行動における特徴の一つである．ある種の蛙には周囲に多くの雄が存在する時，鳴くことによって求愛を成功させる確率を上げる利益よりも，鳴くことによって消費する体力の損失のほうが大きいと判断し，鳴くことを止めるという特徴がある．また，他の雄と同時に鳴いては求愛対象の蛙に音を聞き分けてもらえないため，鳴くタイミングをずらすという特徴も持っている．

本研究ではセンサを蛙と見立て，これらの特徴をセンサネットワークに応用する．蛙が発声する場合，センサは周辺に刺激を与えるアクティブモードになり，蛙が鳴かずに待機している場合，センサは休止しスリープモードになる．これら 2 つのモードを蛙の特徴により切り替えることで，連結支配集合を構築できると考えられる．

3.2 蛙のサテライト行動に基づく既存アルゴリズム

本節では，前述の蛙のサテライト行動に基づく既存アルゴリズム [5] を説明する．このアルゴリズムは，初期段階でセンサに発声閾値 v と，ランダムな状態値 c，状態値の進み幅 a を与える．各センサは以下の 3 フェーズを繰り返す．

フェーズ 1: 状態値 c が発声閾値 v に達している場合，センサはアクティブモードに移行し周囲のセンサに刺激を与える．

フェーズ 2: アクティブモードになったセンサは状態値 c をリセットし，センサはスリープモードに移行する．

フェーズ 3: アクティブモードにならなかったセンサは，1 ステップの実行後に，各センサーの状態値 c に進み幅 a を加算する．

図 2 の例を用いて，本アルゴリズムの説明を行う．図 2 の (a) は初期状態の発声閾値，各センサの状態値を表す．このときの各センサの発声閾値 $v = 3$ とする．また，状態値の進み幅 a は v を 2 で割った値の整数部分である $a = 1$ とする．また，センサ A，センサ B，センサ C，センサ D の初期状態値をそれぞれ c_A，c_B，c_C，c_D とする．図 2 の (a) の場合では，c_A=0，c_B=1，c_C=1.5，c_D=2 より，状態値 c が発声閾値 v に達しているセンサが存在しないため，アクティブモードに移行するセンサは存在しない．

図 2 の (b) は 1 ステップ後の状況を表す．このとき，各センサの状態値 c は更新され，c_A=0+1=1，c_B=1+1=2，c_C=1.5+1=2.5，c_D=2+1=3 となる．この時センサ D の状態値 c_D が 3 となり，発声閾値 v=3 に達したためアクティブモードに移行している．

図 2 の (c) はさらに 1 ステップ後の状況を表す．このとき，センサ D の状態値 c_D が 0 にリセットされ，センサ A，センサ B，センサ C の状態値が更新され，c_A=1+1=2，c_B=2+1=3，c_C=2.5+1=3.5 となる．

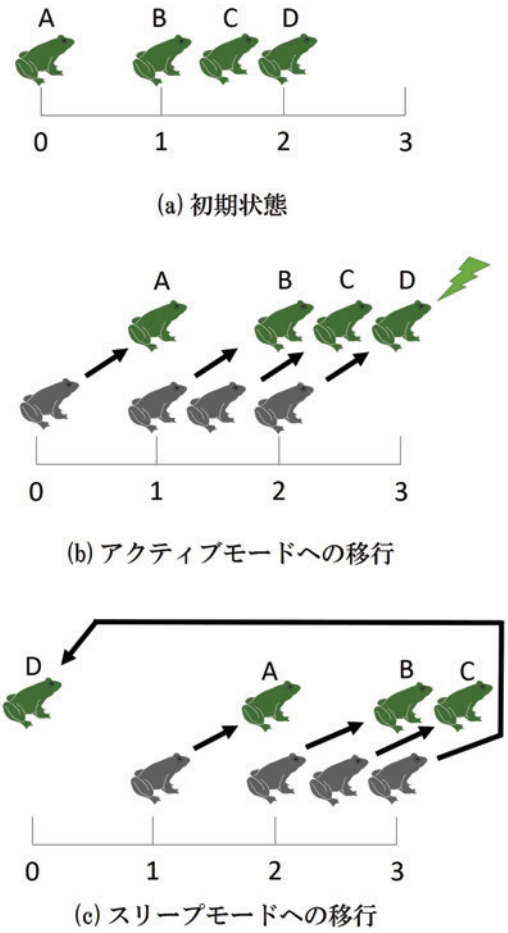


図 2: 既存アルゴリズムの動作例

4 提案アルゴリズム

本節では，提案手法による分散型連結支配集合構築を行うアルゴリズムを紹介する．本節で述べるアルゴリズムは，前節で説明したアルゴリズムに進み幅を調整する概念を組み込んだアルゴリズムである．各センサが，自身の状態値に応じてステップ数が増えた時の状態値の進み幅を調整することで，一定のタイミングでアクティブモードになることを目指す．このアルゴリズムは，以下の 3 つのフェーズで構成される．

フェーズ 1: センサ s_v の状態値 c_v が発声閾値 v_v に達した場合，s_v の隣接センサ群 S_N は各センサ $s_i(\in S_N)$ の状態値に応じた刺激を受け，s_i の状態値の進み幅 a_i を更新する．

フェーズ 2: フェーズ 1 でアクティブモードになったセンサ s_v は，進み幅と状態値をリセットし，スリープモードに移行する．

フェーズ 3: 1 ステップの実行後に，状態値の進み幅 a_i 分，各センサ s_i の状態値 c_i を更新する．

以下では，それぞれのフェーズの詳細について述べる．

フェーズ 1: センサ s_v の状態値 c_v が，発声閾値 v_v に達した場合，隣接センサ群 S_N は各センサ $s_i(\in S_N)$ の状態値に応じた刺激を受け，s_i の進み幅 a_i を更新する．

あるセンサがアクティブモードとなる度に，周辺のセンサは自身の状態値に応じた刺激を受ける．発声閾値 v_v の半分の値を基準値 $\frac{v_v}{2}$，自身の状態値を c_i，カウントの進み幅を a_i とする．刺激の大きさは，$|\frac{v_v}{2} - c_i|$ に比例するものとし，$\frac{v_v}{2} > c_i$ の場合，正の刺激を受け進み幅 a_i は大きくなり，以下の式で与えられるものとする．

$$a_i = a_i + \left|\frac{v_v}{2} - c_i\right|$$

一方，$c_i > \frac{v_v}{2}$ の場合，負の刺激を受け進み幅 a_i は小さくなるものとし，以下の式で与えられるものとする．

$$a_i = a_i - \left|\frac{v_v}{2} - c_i\right|$$

$c_i = \frac{v_v}{2}$ の場合，正と負の刺激が打ち消しあい，進み幅 a_i には影響せず，以下の式で与えられるものとする．

$$a_i = a_i$$

ここで，センサ s_v の通信領域内にある自身を除くセンサの数を N_v とすると，発声閾値 $v_v = N_v$ となる．

フェーズ 2: フェーズ 1 でアクティブモードになったセンサ s_v は，進み幅 a_v と状態値 c_v をリセットし，スリープモードに移行する．

アクティブモードになったセンサ s_v は，以下の式の通り，進み幅 a_v は初期状態の進み幅 a_r にリセットされ，状態値 c_v を 0 にリセットする．

$$\alpha_v = \alpha_r$$

$$c_v = 0$$

ただし，複数のセンサが同時にアクティブになった場合，以下の式に従って状態値をリセットする．状態値 c_v は確率変数 $x = 0$ の時，0 にリセットされ，$x = 1$ の時 1 にリセットされる．確率変数 $x = 0$ になる確率 $P_0 = \frac{1}{2}$ で，$x = 1$ になる確率 $P_1 = \frac{1}{2}$ とする．

$$c_v = \begin{cases} 0 & (x = 0) \\ 1 & (x = 1) \end{cases}$$

フェーズ 3: ステップ数が 1 増えた時に進み幅 a_i 分，状態値を更新する．

進み幅 a_i を状態値 c_i に加算する．更新後の状態値 c_i は，以下の式で与えられる．ただし，アクティブからスリープとなったセンサの状態値は更新しない．

$$c_i = c_i + a_i$$

図 3 の例を用いて，本アイデアの説明を行う．図 3 の (a) は初期状態を示す．このとき，発声閾値 v_v=3 とし，状態値の進み幅 a_i=1 とする．センサ A，センサ B，センサ C，センサ D の初期状態値をそれぞれ c_A，c_B，c_C，c_D とすると，c_A=0，c_B=1，c_C=2.5，c_D=3 である．

図 3 の (b) ではフェーズ 1 での進み幅の更新の様子を示す．青い矢印は負の刺激を，赤い矢印は正の刺激を示し，矢印の長さは刺激の大きさを示す．中央の点線は基準値 $\frac{v_v}{2}$ を示す．状態値 c_D=3 となり，センサ D がアクティブモードとなったので，各センサは刺激を受ける．

図 3 の (c) ではセンサ D のスリープモードへの移行と，各センサの進み幅の更新を示す．各センサは更新された進み幅 a_i 分，状態値を更新する．

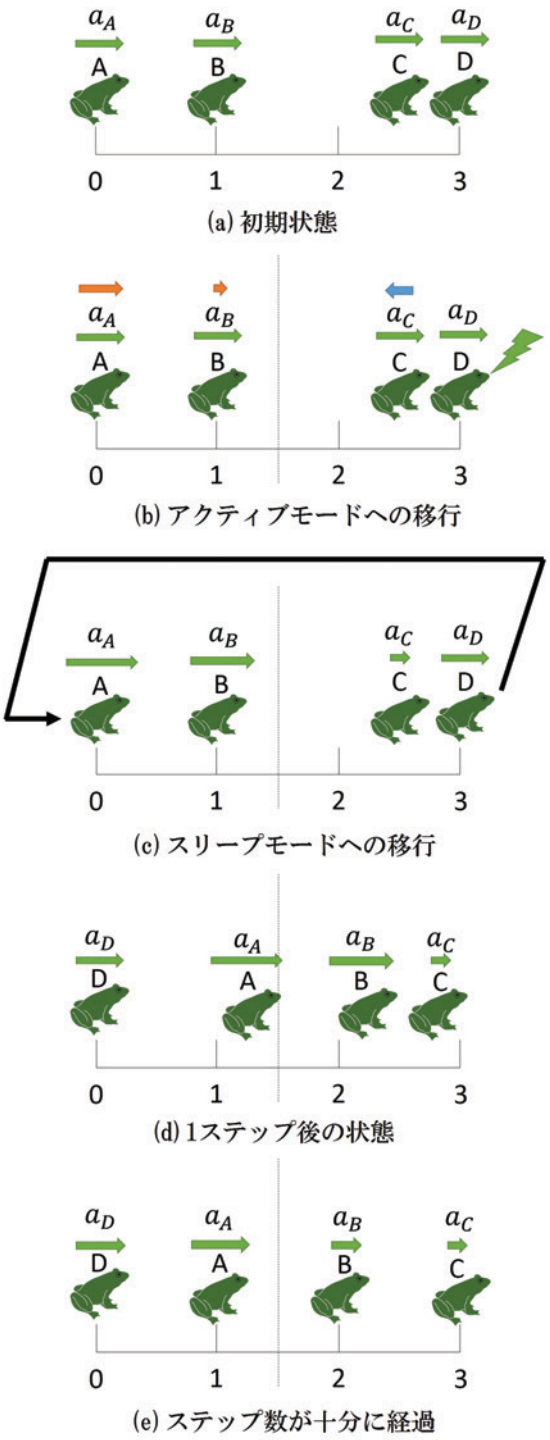


図 3: 提案アルゴリズムの動作例

図 3 の (d) ではセンサ D の進み幅と状態値がリセットされ，各センサが状態値を更新した様子を示す．この時，発声閾値に達したセンサが存在しないため進み幅の更新は行われず，再び進み幅 a_i 分，状態値を更新する．

図 3 の (e) にステップ数が十分経過した状態を示す．進み幅を更新し，正と負の刺激が釣り合った時，センサは効率よくモードを移行できる状態になる．

5　実験結果と考察

5.1　既存手法との比較

提案手法をシミュレーション環境に実装し，評価実験を行った．なお，本実験は，LEDA[6] を用いて実装を行った．また，センサは 100 × 100 の矩型領域にランダムに配置されるものとし，各センサの通信半径は 10 として，実験を行った．図 4 に実験結果を示す．

図 4 より，提案手法が，既存手法よりも連結支配集合を構築する割合が大きいことがわかる．しかし，連結支配集合が必ず構築されるとは限らない．一方，連結支配集合の数は既存アルゴリズムの方が少ない．これは単純に，連結支配集合の数が増えたため，割合が大きくなったためだと考えられる．

連結支配集合が構築されないのは，センサが正方領域にランダムで配置された時に，センサが正方領域の縁に配置されると，その付近のセンサ集合が疎になるためであると考えられる．また，各センサがアクティブモードになる間隔が均等になっていない，ステップ数が少ない段階での割合を計算していたため，割合が小さくなったものと考えられる．

5.2　提案手法の改善

既存手法と提案手法の比較の実験結果より，2 つの問題点があった．ここでは，その 2 つの問題点を改善して提案手法の評価実験を行った．

1 つ目の改善点は，図 5 のように，センサの矩形領域の中にさらに小さい矩形領域を作り，その小さい矩形領域の中でのみ連結支配集合の条件を満たしているかを調べる．これにより，矩形領域の縁で連結支配集合の条件を満たさなくなるのを抑える．

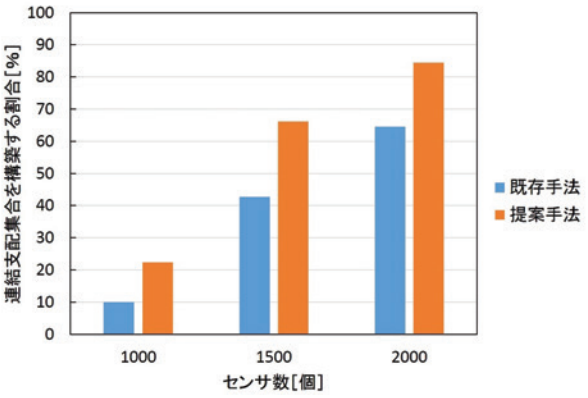
2 つ目は，1 つの試行で 1000 ステップを実行する際の初めの 100 ステップまでを準備期間とし，出力を行わないようにする．これにより，十分にステップ数が経過していない状態での結果を出力しないようにする．

また，提案手法のアルゴリズムにおいて，センサがアクティブになり，状態値が 0 へリセットされるが，この時の状態値を発声閾値に近づける．これにより，センサがよりアクティブになりやすく，連結支配集合を構築する割合が高くなる．

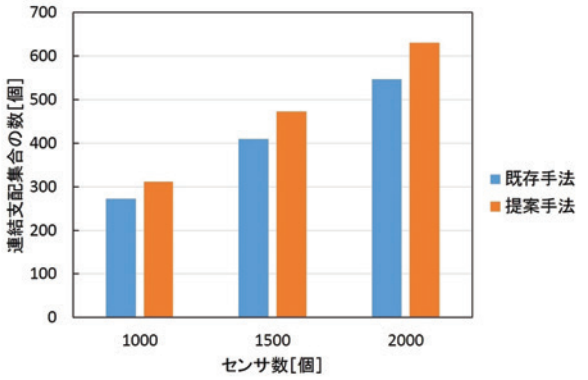
この 3 つの改善点をアルゴリズムに実装し，実験を行った．実験結果を表 1 に示す．なお，ステップ数は 1000 ステップ，試行回数は 100 回とする．また，改善点に従って，1000 ステップ中の初めの 100 ステップは出力しないものとする．さらに，状態値のリセット値 $c_r = \frac{v}{3}$ とする．表 1 の CDS 割合は連結支配集合の条件を満たす割合，CDS 数は連結支配集合の数を示す．

表 1 を見ると，改善前よりも高い数値が出ていることがわかる．また，通信領域の半径が小さくなると，アクティブとなるセンサの数が増えている．これにより，周囲にアクティブなセンサが少ないと，より多くのセンサがアクティブとなることがわかる．まとめると，連結支配集合の条件を満たす割合は全センサ数と通信領域の半径に比例し，連結支配集合の数は，全センサ数に比例するが，通信領域の半径に反比例する．

この結果より，センサ数が少なかったり，通信領域の半径が小さすぎると，連結性を保てなくなることがわかる．そのため，適切なセンサ数と通信領域の半径を選択する必要がある．



(a) センサ数と連結支配集合を構築する割合



(b) センサ数と連結支配集合の数

図 4: 既存手法と提案手法の比較

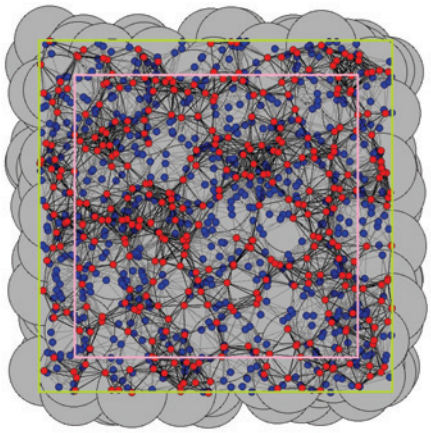


図 5: 2 つの矩形領域の例

表 1: 提案手法の改善結果

		全センサ数		
		1000	1500	2000
CDS 割合 (%)	改善前	22.3810	66.1680	84.5150
	改善後	94.3905	99.5994	99.5943
CDS 数 (個)	改善前	311.589	472.393	630.000
	改善後	282.452	419.422	556.121

6 まとめ

本研究では，蛙の行動特性を用いることにより，連結支配集合を構築するアルゴリズムの提案を行った．既存手法と提案手法の比較実験を行い，見つかった問題点を改善して検証を行った．センサ周辺の環境設定によって結果が変わるため，適切な値を選択する必要がある．

今後の課題として，センサにバッテリーを仮定し，ランダムにアクティブとなるセンサとの寿命比較を行うことや，センサ集合が疎な場合でも連結支配集合が構築可能なアルゴリズムの提案が挙げられる．

参考文献

[1] S. Kamei, H. Kakugawa. A Self-Stabilizing Distributed Approximation Algorithm for the Minimum Connected Dominatind Set *International Journal of Foundations of Computer Science*, pp. 459-476, 2010.

[2] R. Jovanovic, M. Tuba, D. Simian. Ant Colony Optimazation Applied to Minimum Weight Dominating Set Problem *Proc. 12th WSEAS International Conference on AUTOMATIC CONTROL*, 2010.

[3] Chi-Fu Huang, Yu-Chee Tseng. The coverage problem in a wireless sensor network *Mobile Networks and Applications*, Vol. 10, pp,519-528, 2005.

[4] A. Mutazono, M. Sugano, M. Murata. Energy efficient self-organizing control for wireless sensor networks inspired by calling behavior of frogs. *International Journal of Sensor Networks*, 2012.

[5] 横溝 康作, 藤原 暁宏 蛙のサテライト行動に基づくセンサカバーの構築. **卒業論文**, 2015.

[6] 浅野 哲夫, 小保方 幸次. LEDA で始める C/C++ プログラミング―入門からコンピュータ・ジオメトリまで―. **サイエンス社**, 2002.

単位円グラフの最小支配集合問題について

竹中 将成　　泉　泰介

名古屋工業大学大学院工学研究科
E-mail:atelier33media@gmail.com, t-izumi@nitech.ac.jp

最小支配集合問題 (Minimum Dominating Set Problem:MDS) はグラフ上の代表的な NP 完全問題の一つであり，理論計算機科学的な意味での難しさの観点から見ても特に扱いにくい問題の一つである．MDS は近似アルゴリズムを検討した場合においても LOGAPX 完全，すなわち近似率に関して $\Omega(\log n)$ の下界を持つことが強く疑われる問題であり，また，固定パラメタ容易性の文脈においても W[2]-困難，すなわち解の大きさをパラメタとした固定パラメタアルゴリズムの存在が絶望視されている問題でもある．本稿では，入力として与えられるグラフを単位円グラフに制限した場合における最小支配集合問題の効率的なアルゴリズムの検討を行う．最小支配集合問題は入力を単位円グラフに制限したとしても NP 完全であることが知られているが，一方で近似アルゴリズムに関しては PTAS が存在することが知られている．ただし，既知のアルゴリズムにおいて $(1+\epsilon)$-近似を達成するために必要な計算時間は $O(n^{1/\epsilon+O(1)})$ 時間である．また，単位円グラフに入力を制限した場合の固定パラメタアルゴリズムに関しては結果が知られていない．本研究では，単位円グラフに対する解のサイズをパラメタとした固定パラメタアルゴリズムの構成可能性について検討するとともに，それが得られた場合，既存の PTAS アルゴリズムの実行時間が $O(\exp(1/\epsilon)n^{O(1)})$ へと向上させられることを紹介する．

状態を持つ 2 台の自律分散ロボットの計算能力について

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：本稿では,従来の自律分散ロボットの理論モデルに,新たに状態を持つ機能を追加したモデルを考え,ロボットの動作の同期性や自分と周囲のロボットの状態の見え方の異なる各モデルの計算能力の比較を,2 台のロボットの集合問題(ランデブー)などを用いて行う.

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[3]を扱ったものが主である.ロボットは平面上で自律的に計算,移動を行う点であり,外見によって個体を識別することはできない.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).後者の状態では休止状態となり,LCM サイクル後に成り,ここから新たな LCM サイクルを開始する.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 に似ているが,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.3 つの計算能力を比較すると,ASYNC が一番弱く,続いて SSYNC,FSYNC の順に強くなることが知られている.例として,2 台のロボットの集合問題(ランデブー問題)は FSYNC では解けるが SSYNC,ASYNC では解くことができない.

本稿ではこの理論モデルに,自身の状態を記録できる定数ビットの記憶領域を搭載したモデル[1,2]について考察した.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.自身の状態のみを観測できるものを internal-light,相手の状態のみを観測できるものを external-light,双方の状態を観測できるものを full-light とする.状態を持つことで可解となる問題があり,例えばランデブー問題は状態を持つ場合は SSYNC,ASYNC でも解くことができる[1,2](表 1).本稿では,ロボットが Move 命令で,目的地に確実に移動できる rigid の場合,full-light, ASYNC でランデブー問題を 3 状態で解くアルゴリズムを提案した.また,5 状態,full-light, ASYNC ロボット群で SSYNC ロボット群が実行するプロトコル P の実行条件を満たすアルゴリズム Protocol-SIM が存在する[1].このアルゴリズムについて考察し,ロボットが 2 台の場合,4 状態 full-light, ASYNC ロボット群でプロトコル P の実行条件を満たすアルゴリ

ズムを提案し, ロボットが 2 台の場合,4 状態 full-light, ASYNC ロボット群は計算能力において SSYNC ロボット群より優れていることを証明した.また external-light の ASYNC ロボット群では 5 状態でプロトコル P を実行するアルゴリズムを提案し, ロボットが 2 台の場合,5 状態 external-light, ASYNC ロボット群は SSYNC ロボット群と計算能力が少なくとも同等であることを証明した.

2.モデルの定義

2.1.理論モデル

モデルに関しては[3]で扱われたものを採用する.ロボットは平面空間上に存在する点として扱い,平面空間上を自由に移動できる.各ロボットを外見で識別することはできず,ロボット群に対する集中制御,ロボット間で通信を行う直接的な手段はないものとする.ロボットは独自の計算能力と記憶領域,また個々に単位距離,原点正負の向きを持つ 2 軸から成る局所座標系を有している.ロボットは周囲の状況を観測できるセンサを持っているものとする.すべてのロボットは等しい性能を持ち,同一のアルゴリズムに従って自律的に動作する.

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元に,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.

inactive の場合,ロボットは休止状態であり,LCM サイクル終了後になるものとし,ここから次の LCM サイクルを開始する.また,ロボットは休止状態を無限時間行うことはできない.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.これを無記憶という.Move 命令を実行した際,計算された目的地へ確実に到達できる場合,その移動は厳密(rigid)であるという.移動が厳密でない場合(non-rigid),ロボットは一度の Move 命令で計算された目的地にたどり着けないことがある.その場合,最小移動距離 δ は必ず移動するものとする.また,この δ の距離を知っているかでも計算能力が変わる(δ 知識有).

ロボットの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)
上記の 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.

ASYNC の場合,他のロボットが Move 命令

を実行しているにも関わらず,Look 命令を実行することなどが考えられる.計算能力としては FSYNC,SSYNC,ASYNC の順に優れていることが知られている[1].

2.2.状態(light)

本稿では,上記で示した基本的な理論モデルに,自身の内部状態を記録できる定数ビットの記憶領域を搭載したモデルを扱う[1].これにより,そのモデルでは非可解な問題に対しても対処することができる.状態は Look 命令で観測し,Compute 命令で更新することができる.また,自身の状態,他のロボット状態の見え方によって,以下の 3 つのモデルが考えられる[2].

- 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.

[2] では ,internal-light は FSTATE, external-light は FCOMM と記述している.

本稿では,k 個の状態を持つ n 台のロボットに対して,スケジュール A,搭載されている light が x のモデルのもとで解くことのできる問題の集合を,$A^n(x,k)$と記述する.

3.各モデルの比較

3.1.ランデブー問題による比較

複数のロボットが任意の初期配置から有限時間内に一点に集合する問題を集合問題という.特に 2 台のロボットによる集合問題をランデブー問題(Rendezvous)という.各モデルについて,ランデブー問題の可解性,状態の使用数で比較を行った.

3.1.1.先行研究

先行研究の結果から得られた定理を以下に示す.状態を持たないロボット群では以下の定理が知られている.

定理 1[1]

Rendezvous $\in FSYNC^2(nothing,1)$

Rendezvous $\notin SSYNC^2(\text{nothing},1)$

Rendezvous $\notin ASYNC^2(nothing,1)$

状態を持つロボット群では以下の定理が知られている.

定理 2[1]

移動が non-rigid の場合,

Rendezvous $\in ASYNC^2(full-light,4)$

定理 3[2]

移動が rigid の場合,

Rendezvous $\in ASYNC^2(external-light,12)$

移動が non-rigid で δ の知識を持っている場合,

Rendezvous $\in ASYNC^2(external-light,3)$

定理 4[2]

移動が non-rigid の場合,

Rendezvous $\in SSYNC^2(external-light,3)$

定理 5[2]

移動が rigid の場合,

Rendezvous $\in SSYNC^2(internal-light,6)$

移動が non-rigid で δ の知識を持っている場合,

Rendezvous $\in SSYNC^2(internal-light,3)$

このようにスケジュールによって非可解である問題も,状態を持つことで可解となる場合がある.

3.1.2.アルゴリズム

本稿では以下のアルゴリズムを提案した.

補題 1
ロボットの移動が rigid である full-light、ASYNC のロボット群によるランデブー問題を,3 状態を用いて解くアルゴリズムが存在する.

アルゴリズム:Rendezvous for rigid ASYNC

2 台のロボットは初期状態 A から始まり,相手が状態 A の場合は状態 B になり 2 台の中点に移動,相手が状態 B の場合は移動しない,そして相手が状態 C の場合は相手の位置に移動し,集合完了となる.自分が状態 B の場合,状態 C になる.自分が状態 C の場合は移動しない.このアルゴリズムにおけるロボットの状態遷移図を図 1 に示す.

```
L[x]：自分の状態
L[y]：相手の状態
Case L[x]
        ・A
          If L[y] = A Then
                L[x] := B
                2 台の中点に移動
          If L[y] =B Then
                移動しない
          If L[y] = C Then
                L[x] := B
                相手の位置に移動
        ・B
          L[x] := C
        ・C
          移動しない
```

証明

全てのロボットが状態 A になった時刻をt_0とする.時刻$t_1 \geq t_0$を,2 台のうち少なくとも 1 台が状態 B になった時刻とする.また時刻 t($t_0 < t < t_1$)に Look 命令を行ったロボットの台数を n とする.これにより次の 2 つの場合が考えられる.

①n=1 の場合

時刻t($t_0 < t < t_1$)に Look 命令を行ったロボットが 1 台の場合を考える.そのロボットを x とし,もう一方のロボットを y とする. x は自分の状態 A と y の状態 A を観測し,自分の状態を B に変える(時刻t_1).時刻t_1以降に Look 命令を行う y は,自分の状態 A と x の状態 B を観測した場合は移動を行わない. x は Move 命令で 2 台の中点に移動する.その後の Look 命令で自分の状態 B を観測し,状態を C に変える.この時の時刻をt_2とする. t_2以降に Look 命令を行なった y は,自分の状態 A と x の状態 C を観測し,自分の状態を B にする.その後,y は Move 命令で相手ロボット(x)の位置へ移動する.y はその後の Look 命令で自分の状態 B を観測し,状態 C となる.状態 C となったロボットはそれ以降動作しない.

②n=2 の場合

時刻t($t_0 < t < t_1$)に 2 台のロボットが Look 命令を行なった場合を考える.2 台のロボットは互いに自分と相手の状態 A を観測し,t_1以降の時刻に状態 B となり,その後の Move 命令で 2 台のロボットの中点に移動する.このとき,2 台のロボットは Look 命令で同じスナップショットを取得しているため 2 台のロボットは同じ点を目的地として計算す

る.中点に移動したロボットはその後の Look 命令で自分の状態 B を観測し,状態 C となる.状態 C となったロボットはそれ以降動作しない. □

定理 6
移動が rigid の場合
Rendezvous ∈ $ASYNC^2(full\text{-}light, 3)$

定理 1,2,3,4,5,6 より,各モデルの状態数を表 1 に示す.表 1 において空欄になっている internal-light, ASYNC のロボット群については,未だランデブー問題を解くことができるアルゴリズムが知られていない.

3.2.シミュレートによる比較

[1]では,5 状態を持つ full-light の ASYNC のロボット群で,SSYNC のロボット群が実行するプロトコル P の実行条件を満たすアルゴリズム,Protocol SIM が紹介されている.これを用いて,ロボットの台数を 2 台に制限した場合,状態を持つ full-light の ASYNC のロボット群でプロトコル P の実行条件を満たすアルゴリズムを提案する.また,external-light の ASYNC ロボット群でプロトコル P の実行所条件を満たすアルゴリズムについても提案する.

補題 2
2 台の full-light,4 状態の ASYNC ロボット群で,2 台の SSYNC ロボット群が実行するプロトコル P の実行条件を満たすことができるアルゴリズムが存在する.

アルゴリズム:Protocol SIM for two robots
使用する 4 状態を T(rying), M(ove), S(topped), F(inished)とする.2 台のロボットは初期状態 T から始まり,相手が状態 T,S の場合は状態 M となりプロトコル P を実行し,相手が状態 M の場合は状態を変えない.自分が状態 M で相手が状態 T,M,S の場合は状態 S となる.自分が状態 S で相手が状態 S,F の場合は状態 F となる.そして自分が状態 F で相手が状態 F,T の場合は再び状態 T となる.このアルゴリズムにおけるロボットの状態遷移図を図 2 に示す.

state *Look*
スナップショット取得
Pos[x]： 自分の位置
L[x]： 自分の状態
L[y]： 相手の状態

state *Compute*
p := Pos[x]
Case L[x]
・T
If L[y]∈{T , S} **Then**
プロトコル P を実行
p:=計算によって求められた位置
L[x] := M
If L[y]∈{M} **Then**
L[x] := T
・M
If L[y]∈{M , T , S} **Then**
L[x] := S
・S
If L[y]∈{S ,F} **Then**
L[x] := F
・F
If L[y]∈{F , T} **Then**
L[x] := T

state *Move*
Move(p)

証明

すべてのロボットが初期状態 T から終了状態 F になるまでのサイクルを Mega-Cycle とする.初期状態 T から始まる 2 台のロボットを x,y とする. x がはじめて Look 命令を行った時刻をt_x, x が状態 T から状態 M に遷移した時刻を$t_x' > t_x$とする.y についても同様に時刻t_y, t_y'を定義する.このとき 2 つの場合が考えられる.

①$t_x \leq t_y < t_x'$

x の Look 命令, y の Look 命令,または 2 台が同時に Look 命令を行い, x の状態遷移の順に操作が行われたとする. x は時刻t_xに y の状態 T を観測する.次に時刻t_yに y が x の状態 T を確認する.互いに状態 T を観測した x,y はそれぞれ時刻t_x', t_y'に状態 T から状態 M に遷移する.移動を行うのは状態 M になったあとのみなので,2 台は同じ観測結果を Look 命令で得ている.その後, x,y は他ロボットが状態 M または S なのを観測し,状態 S に遷移する.最終的に 2 台のロボットは他ロボットが状態 S または F なのを観測し,状態 F になり,Mega-Cycle が終了する.

②$t_x < t_x' \leq t_y$

x の Look 命令, x の状態遷移, y の Look 命令の順に操作が行われたとする. x は時刻t_xに y の状態 T を観測し,時刻t_x'に状態 T から状態 M に遷移する.次に y が時刻t_yに x の状態を確認する.このとき, x が状態 M ならば, y は状態遷移を行わず,状態 T のままとなる. y は, x が状態 S に遷移した以降の時刻に状態 M に遷移し,プロトコル P を実行する.状態 M になった y は, x が状態 S なのを観測し,状態 S になる.最終的に 2 台のロボットは他ロボットが状態 S または F なのを観測し状態 F となり,Mega-Cycle を終了する.

①,②より,一度の Mega-Cycle にすべてのロボットが必ず一度,プロトコル P を実行している. □

補題 2 より以下の定理が成り立つ.

定理 7

$$ASYNC^2(full\text{-}light, 4) \supset SSYNC^2(nothing, 1)$$

証明

定理 1,2 よりランデブー問題は SSYNC のロボットでは解くことができないが,full-light,4 状態を持つ ASYNC のロボットでは解くことができる

$$ASYNC^2(full-light, 4) \setminus SSYNC^2(noting, 1) \neq \emptyset.$$

補題 2 より

$$ASYNC^2(full-light, 4) \supseteq SSYNC^2(nothing, 1)$$

よって定理 1 が成り立つ. □

補題 3

2 台の external-light,5 状態の ASYNC ロボット群で,2 台の SSYNC ロボット群が実行するプロトコル P の実行条件を満たすことができるアルゴリズムが存在する.

アルゴリズム:Protocol SIM for two external-light robots

使用する 5 状態を R(equest), E(xecution), M(ove), W(ait), F(inish)とする.2 台は初期状態 R から始まる.相手が状態 R の場合は状態 E となる.相手が状態 E の場合は状態 M となり,プロトコル P を実行し,移動を行う.相手が状態 M の場合は状態 W となる.相手が状態 W の場合は状態 F となり,1 台でも状

態Fになったらサイクル終了とする.そして相手が状態 F の場合は再び状態 R となる.プロトコルPを実行したロボットが1台のみでサイクル終了を迎えた場合,次サイクルでは少なくともう一方のロボットがプロトコル P を実行する. このアルゴリズムにおけるロボットの状態遷移図を図 3 に示す.

state *Look* スナップショットを取得 Pos[x]： 自分の位置 L[x]： 自分の状態 L[y]： 相手の状態

state *Compute*
p := Pos[x]
If L[y]∈{R} **Then**
L[x] := E
If L[y]∈{E} **Then**
L[x]: = M
プロトコルPを実行
p:=計算によって求められた位置
If L[y]∈{M} **Then**
L[x] := W
If L[y]∈{W} **Then**
L[x] := F
If L[y]∈{F} **Then**
L[x] := R

state *Move* Move(p)

証明

初期状態 R から始まる 2 台のロボットをx,yとする.また,少なくとも1台のロボットが状態 F になるまでのサイクルをSub-Cycleとする. xがはじめてLook命令を行った時刻をt_x, xが状態Rから状態遷移した時刻を$t_x' > t_x$とする.yについても同様に時刻t_y, t_y'を定義する.このとき,2 つの場合が考えられる.

① $t_x < t_x' \leq t_y < t_y'$

このとき, x は時刻t_x'に状態 E, y は時刻t_y'に状態 M となりプロトコル P を実行,移動を行う.その間 y が Look 命令を行う場合は状態Eから状態Wに遷移するので移動しない.x は移動後,y の状態が W ならば,状態 F となり Sub-Cycle は終了する.Sub-Cycle 終了後は新たなSub-Cycleが始まり,移動を行っていないyが状態Rとなり,xが状態Eになる.

② $t_x \leq t_y < t_x' \leq t_y'$

このとき,x,y はそれぞれ時刻t_x', t_y'に状態 E となる.時刻t_y'以降に,状態 E の x がはじめてLook命令を行った時刻をt_x'',状態Eから状態遷移を行った時刻を$t_x''' > t_x''$とする.yについても同様に時刻を定義する.

ⅰ. $t_x'' < t_x''' \leq t_y'' < t_y'''$

このとき,x は時刻t_x'''に状態 M, y は時刻t_y'''に状態Wとなる.以降は①と同様の操作となる.

ⅱ. $t_x'' \leq t_y'' < t_x''' \leq t_y'''$

このとき,x,y は時刻$t_x''' \leq t_y'''$に状態 M となりプロトコル P を実行,移動を行う.移動を行うのは状態Mになったあとのみなので,2 台は同じスナップショットを得ている.また,一方が相手の状態 M を観測し状態 W に遷移した後,もう一方が Look 命令を実行した場合,①と同様の操作となる.

①,②より,2 台のロボットは 2 回の Sub-Cycle 内で確実に一度プロトコル P を実行している. □

定理 8

$ASYNC^2(external\text{-}light, 5)$

$\supseteq SSYNC^2(nothing, 1)$

4.今後の課題

表 1 において空欄になっている internal-light, ASYNC のロボット群におけるランデブー問題アルゴリズムは提案されていないので,その可解性について検討する.また,n(>2)ロボットにおける各モデルの集合問題の可解性について,状態の見え方による計算能力の比較等が挙げられる.

参考文献

[1] ： Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, Masafumi Yamashita: Autonomous mobile robots with lights, Theoretical Computer Science, Volume 609, Part 1, pp.171-184, 2015

[2] ： Paola Flocchini, Nicola Santoro, Giovanni Viglietta, Masafumi Yamashita: Rendezvous of Two Robots with Constant Memory, 20th International Colloquium on Structural Information and Communication Complexity Structural Information and Communication Complexity, Lecture Notes in Computer Science 8179, pp189-200, 2013

[3]：Paola Flocchini, Giuseppe Prencipe, Nicola Santoro: Distributed Computing by Oblivious Mobile Robots, SYNTHESIS LECTURES ON DISTRIBUTED COMPUTING THEORY, pp.1-16, 2012

表 1.ランデブー問題での各モデルの状態数

		full-light	external-light	internal-light	nothing
ASYNC	rigid	3	12		非可解
	non-rigid	4	3 (δ知識有)		非可解
SSYNC	rigid	(2)	(3)	6	非可解
	non-rigid	2	3	3 (δ知識有)	非可解
FSYNC	rigid non-rigid	1	1	1	可解

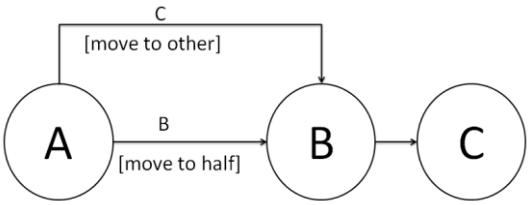


図 1.Rendezvous for rigid ASYNC の状態遷移図(矢印の文字は,相手の状態と自身が移動する位置を示している)

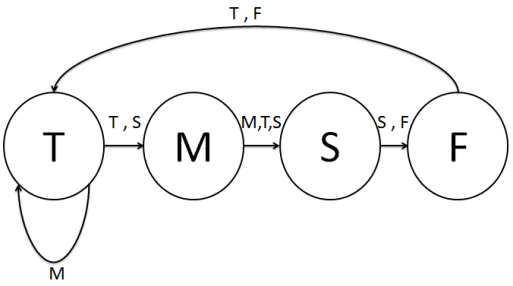


図 2. Protocol SIM for two robots の状態遷移図(矢印の文字は,相手の状態を示している)

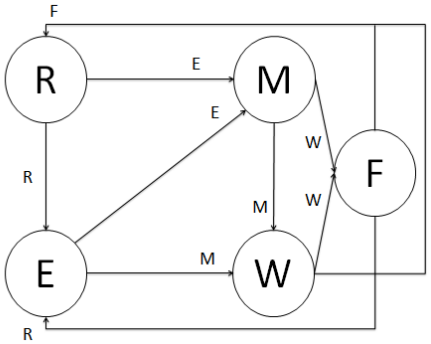


図 3. Protocol SIM for two external-light robots の状態遷移図(矢印の文字は,相手の状態を示している)

自律分散ファットロボットに対する様々なグリッド上における汎用集合アルゴリズムについて

白川遥平 a)　和田幸一 b)　片山喜章 c)

a)法政大学大学院理工学研究科 b)法政大学理工学部 〒184-8584 東京都小金井市梶野町 3-7-2
c)名古屋工業大学大学院 〒466-8555 愛知県名古屋市昭和区御器所町
E-mail: a)yohei.shirakawa.2r@stu.hosei.ac.jp b)wada@hosei.ac.jp c)katayama@nitech.ac.jp

あらまし 本稿ではグリッド上において質量を持つ円または球で表現される自律分散ロボットの集合問題を扱う．グリッドは平面格子，六角格子，立方格子を考える．それぞれのグリッド全てにおいて同様のモデルで集合を達成する汎用的なアルゴリズムを考察する．
キーワード:自律分散ロボット，ファットロボット，集合問題

On General Gathering algrithms in Various Grid for Distributed Autonomous Fat Robots

Yohei SHIRAKAWA[a)]　Koichi WADA[b)]　Yoshiaki KATAYAMA[c)]

a) Hosei University Graduate School of Science and Engineering b) Hosei University Faculty of Science and Engineering 3-7-2 Kajinotyou, Koganei-shi, Tokyo, 184-8584 Japan
c) Nagoya Insitute of Technology gokisotyou, syouwaku, nagoya-si, aichi, 466-8555 Japan
E-mail: a)yohei.shirakawa.2r@stu.hosei.ac.jp b)wada@hosei.ac.jp c)katayama@nitech.ac.jp

Abstract This paper concerns gathering problems in grid for fat robots.We consider a two dimensional orthogonal grid, a hexagonal grid and a three dimensional orthogonal grid. We propose general algorithmse to solve gathering problems in a similar model for each grid.
Keywords Autonomous Mobile Robots, Fat Robots, Gathering Problem

1. はじめに

複数の自律的に動作する計算機群が，互いに通信しあいながら共通の目的を持って動作するようなシステムを，分散システムと呼ぶ．その中で自律分散ロボット群とは自律的に動作する複数のロボットが協調的に動作することにより全体でひとつの問題を達成するというものである．この自律分散ロボット群においては全体を効率的に動作させるアルゴリズムの設計が重要である．

本稿では自律分散ロボット群の中で任意の位置に配置されたロボットを一点に集める問題である一点集合問題を扱っている．ロボットは平面上を自由に移動する(質量を持たない)点で考えられている．この一点集合問題は一見簡単そうに思えて(1)記憶(アルゴリズムが過去の動作履歴を認識できるか)(2)匿名性(各ロボットが固有の識別子を持ち､お互いに異なることを認識できるか)(3)非同期性(ロボットの動作のタイミングにずれが生じるかどうか)(4)観測能力(他のロボットの位置をどれだけ正確かつ一貫して検知できるかどうか)といった少しの要因を緩和するだけで非可解であることが知られている．そして可解となる最弱の件もこれまでに知られておらず，また，非可解となるいくつかの条件が知られている[3]．よって一点集合問題のテーマとして解くべきタスクが与えられた時にそ

表 1†．先行研究でのモデルと提案アルゴリズム

二次元直交座標			small	large
共通座標系有	async	unknown	Spiral	非可解
		known	(Spiral)	SpiralSpread
	ssync	unknown	(Spiral)	PullSlide
		known	(Spiral)	(SpiralSpread) (PullSlide)
共通座標系無	async	unknown	未解決	非可解
		known	未解決	未解決
	ssync	unknown	未解決	未解決
		known	GridPullSpin2	GridPullSpin3

表 2†．本稿でのモデルと提案アルゴリズム 1

正六角形グリッド			small	large
共通座標系有	async	unknown	hexagonSpiral	非可解
		known	(hexagonSpiral)	(視野範囲 1 では非可解)
	ssync	unknown	(hexagonSpiral)	(視野範囲 1 では非可解)
		known	(hexagonSpiral)	視野範囲 1 では非可解
共通座標系無	async	unknown	未解決	非可解
		known	未解決	(視野範囲 1 では非可解)
	ssync	unknown	未解決	(視野範囲 1 では非可解)
		known	hexagonPullSpin	(視野範囲 1 では非可解)

表 3†．本稿でのモデルと提案アルゴリズム 2

三次元直交座標 共通座標系有		small	Large
async	unknown	Spiral3	未解決
	known	(Spiral3)	未解決
ssync	unknown	(Spiral3)	未解決
	known	(Spiral3)	未解決

†表において，括弧つきのアルゴリズム（または非可解性）は，括弧なしのアルゴリズムの可解性(または非可解性)を示すことにより，より強い(または弱い)仮定の場合が示されることを意味している．

れを解くべきアルゴリズムの設計やその問題を解くための必要なロボットの最小能力を明らかにするといった問題がある．文献[1-5]は実際にロボットの観測や移動の能力や知り得る情報と問題に対する関する研究がされている．

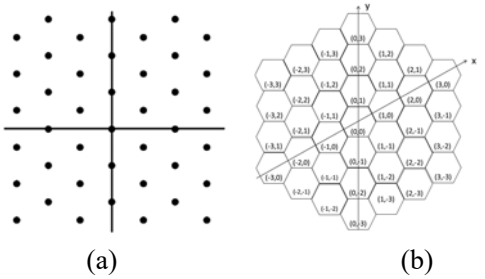
実際のロボットを考えた時ロボットは質量を持ちロボットが他のロボットの観測および移動において障害物となる可能性を考慮すべきである．文献[1]はロボットを点ではなく円盤で表すファットロボットの集合問題を扱った研究である．本稿では，ファットロボットが 2 次元平面の格子点上を移動する場合の集合問題を考える．文献[6-8]が従来の結果であり，実際にファットロボットの 8 通りのモデルで一般的な平面格子の知識有無も考えた．集合問題が考えられておりその結果が表 1 である．本稿ではその先行研究から平面格子と集合した際の形態が異なる六角格子や立方格子上に拡張した集合問題を解くアルゴリズムを設計する．その結果を表 2，3 に示す．六角格子においては，座標系の知識有の場合，非同期，small ロボットで台数未知のとき集合問題を解くアルゴリズム hexagonSpiral を示し, large ロボットの場合は, 非同期, small ロボットで台数を知らないときは正方格子と同様に非可解であること，それ以外のときは視野範囲 1 で非可解となることを示す.また座標系の知識無の場合でも，半同期，small ロボットで台数既知のときに集合問題を解くアルゴリズム hexagonPullSpin を提案する．さらに，立方格子上ではおいて座標系の知識有の場合，非同期，small ロボットで台数未知のとき集合問題を解くアルゴリズム Spiral3 を示す.

2. モデルの定義

2.1 ロボットのモデル

本稿で考える二次元直交座標系 S, (三次元直交座標系 C)は，単位長さ，原点 O，x と y(と z)で識別され互いに直交する 2(3)つの座標軸とその方向，正と負で識別される座標軸の向きで定義される. 平面格子 G_S を S のすべての格子点の集合，立方格子 G_C を C のすべての格子点の集合とする．$u=(u_x,u_y)$および $v=(v_x,v_y)$を G_S 上の点とした時，u と v の距離 $dist_S(u,v)$はマンハッタン距離で定義され $dist(u,v)=|u_x-v_x|+|u_y-v_y|$と定義される．また $u=(u_x,u_y,u_z)$および $v=(v_x,v_y,v_z)$を G_c 上の点とした時，同様にマンハッタン距離で定義され u と v の距離 $dist_S(u,v)$は $dist_c(u,v,z)=|u_x-v_x|+|u_y-v_y|+|u_z-v_z|$となる．六角格子座標系 H は，(図 1(a))に表される様な

$P=\{(\frac{\sqrt{3}}{2}m,n+\frac{1}{2}m)|m,n\in\mathbb{Z}\}$で定義されるの点の集合があり，P に対応する六角格子 $G_H=\{(m,n)|m,n\in\mathbb{Z}\}$を持ち，単位長さ，原点 O、x 軸と y 軸で識別されお互いに 60°で交わる 2 つの座標軸とその方向，正と負で識別される座標軸の向きで定義されるものである(図 1(b))． 点同士の距離は G_S，G_C 同様にマンハッタン距離で定義される．



(a) (b)

図 1．六角格子と六角格子座標系

n 台のロボットの集合 R を R={r_0，…，r_{n-1}}とし，ロボットは格子上の点に存在し，格子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とする．このロボットは区別不可能で同じアルゴリズムを実行し，匿名で，記憶領域を持たない．また，ロボットは共通の座標系を持つ場合と持たない場合を考える．ロボットが共通の座標系を持つ場合は共通の原点，単位距離，座標軸とその方向および向きを知っており，ロボットはその共通した座標軸において自身の位置の座標を認識できる．また ロボットが共通の座標系を持たない場合は共通の原点，単位距離， 右回り方向のみ知っている，ロボットの視野範囲は自身の点を v とし二次元直交座標系では $N_S(v)=\{(u_x,u_y)|(|u_x-v_x|\leqq 1)\wedge(|u_y-v_y|\leqq 1)\}$ の集合であり三次元直交座標系では $N_C(v)=\{(u_x,u_y,u_z)|(|u_x-v_x|\leqq 1)\wedge(|u_y-v_y|\leqq 1)\wedge(|u_z-v_z|\leqq 1)\}$ である．六角格子座標系上では距離 1 となる周囲の 6 点の集合 $N_H(v)=\{(u_x,u_y)|dist(v,u)=1\}$ とする．N_x (v)(x∈{S,C,H})は v にいるロボットが 1 回の移動で移動できる点を表している．ただし，ロボットが同一線上をすれ違う移動と複数のロボットが同一の点に移動することは許されない．また，図 2 に示す様にロボットのサイズでによってはあるロボットは他のロボットが存在する周囲の点へは他のロボットに接触するため移動することができない．ロボットの大きさが十分に小さいロボットが他のロボットが存在しない点に自由に移動できるロボットのサイズを small，周囲のロボットとの接触を避けるため移動に制限が付くロボットのサイズを large と名付ける．平面格子上と六角格子上ではその条件は異なってくる．それぞれの条件を以下に示す．

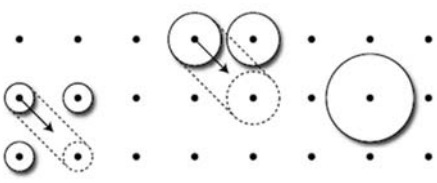


図 2．平面格子におけるロボットの半径 radius と格子サイズの関係

平面格子上では，ロボットの半径を radius として図 2 の例に示すように，ロボットの大きさが十分に小さく斜めに移動したときに他のロボットに接触することがないモデル small とあるロボットが斜めに移動したとき他のロボットに接触する可能性があるモデル large を

・small:：radius < $\frac{1}{4}\sqrt{2}$

・large： $\frac{1}{4}\sqrt{2} \leq$ radius < $\frac{1}{2}$

と定義される．

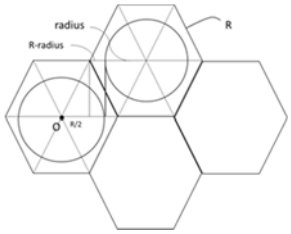


図 3．六角格子上におけるロボットの半径 radius と格子サイズの関係

図 3 に示す様に六角格子上では平面格子上とは条件が異なる．ロボットの半径の大きさを radius とし radius が線を超えた際にあるロボットが移動した時に周辺のロボットと接触する可能性がある．六角格子上では正六角形の一辺の大きさを R とし

・small：radius < $\frac{3R}{4}$

・large：$\frac{3R}{4} \leq$ radius < $\frac{\sqrt{3}R}{2}$

と定義する．

立方格子上では半径 radius の球と変える以外は平面格子上と同様である．

次にロボットの実行モデルについて説明する．各ロボットは以下の 4 つの状態を繰り返し，この 4 つの状態の 1 回の繰り返しをサイクルと呼ぶ．

(1) Wait:ロボットは待機状態．全てのロボットの初期状態はこの状態である．
(2) Look:ロボットは他のロボットの位置を観測する．
(3) Compute:ロボットは観測結果を入力にアルゴリズムに従って行き先を計算する．ロボットが無記憶の場合は入力は直前の Look の情報のみである．
(4) Move:Compute で計算した行き先に移動する

各ロボットが動作する同期の程度によっても図 4 の様な 3 つのモデルが定義される．

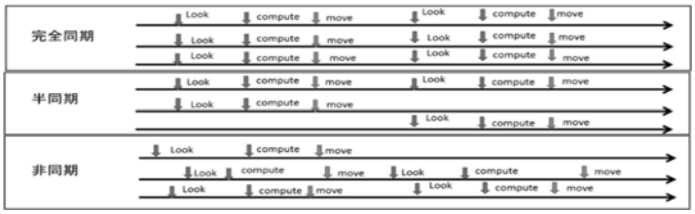


図 4.　同期のモデル

・完全同期(FSYNC):全てのロボットが完全に同期し，全てのロボットが完全に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 を実行する．

・半同期(SSYNC):FSYNC において同じ時刻にのみ Look，Compute，Move が実行されるが，サイクルを実行しないモデルの存在は許すモデルである．

・非同期(ASYNC):FSYNC や SSYNC のようなロボットの実行において同期の仮定を一切置かないモデルである．

これら 3 つのモデルに関して FSYNC のロボットモデルで実現可能な動作は SSYNC のロボットモデルでも実現可能であり，また，SSYNC のロボットモデルで実現可能な動作は ASYNC でも実現可能である．すなわちある問題を ASYNC で解くアルゴリズムが存在する場合は，そのアルゴリズムは SSYNC，FSYNC でも問題を解く．SSYNC と FSYNC の関係も同様である．

動作サイクルの各状態は，仮定するロボットの能力によって細かい定義が異なる．　本稿では，ロボットの視野範囲に制限が存在する．G_x の点 v 上のロボットは Look の結果, ロボットが共通して持つ座標系における $N_x(v)$のうちロボットがいる点の集合 C を得る．Compute で計算される行き先は $N_x(v)$-C のうち現在の座標を除く一点，もしくは移動しないかのいずれかであり，Move における行き先への移動は一瞬であると仮定し移動中のロボットが他のロボットに観測されることはないとする．

ロボットが透明か不透明かの問題についてはここで提案するアルゴリズムにおいてはどちらでもかまわない．

2.2 集合問題の定義

本稿では, あらかじめ決められた原点 O を中心にロボットを集合させる集合問題を扱う．集合時にロボットが構成する形状が問題の可解性に影響し，原点に近い距離からロボットで埋めていき，集合を達成した際に原点からの距離が最も遠いロボットへの距離が最小となるようにロボットを集合させる．

集合の定義は, グリッド x に対し a_i^x を原点からの距離が i である点の数，原点からの距離が j 以下である点の数を S_j^x とし，ロボットが集合を達成したときの原点から最も遠いロボットまでの距離を L_{max}^x とする．

原点 O からの距離が i であるロボットの集合を R(i)={r∈R|dist(O,r)=i}としてグリッド上での集合問題を L_{max} を用いて以下のように定義する．

定義　グリッド上で n 台のロボットが次の条件を満たすときロボットは集合したという．

・$\forall r \in R,\quad dist(O,r) \leqq L_{max}^x$

・$|R_0|=1$

・$0 < i < L_{max}^x$ ，$|R_i| = a_i^x$

本稿で扱う六角格子上での，a_i^H，S_j^H は以下の様になる

・$a_0^H = 1$

・$a_i^H = 6i$ (i>0)

・$S_j^H = 3j(j+1)+1$ (j≧0)

S_j^Hの逆関数を考えることでL_{max}^Hは以下の式で決まる．

$L_{max}^H=\lceil\sqrt{(\frac{n}{3}-\frac{1}{12})}-\frac{1}{2}\rceil$

また立方格子上では

・$a_0^C = 1$

・$a_i^C = 4i^2 + 2$ (i>0)

・$S_j^C = \frac{4j^3}{3}+2j^2+\frac{8j}{3}+1$

となり，L_{max}^C は S_j^C とロボットの台数 n から

$\frac{4j^3}{3}+2j^2+\frac{8j}{3}+1-n=0$ を満たす実数解の小数点以下の切り上げで得られる．この定義を満たす集合問題を解くアルゴリズムの集合を，同期のモデル sync∈{async,ssync,fsync}，ロボットの n を知るか否か num∈{known,unknown}，ロボットの大きさのモデル size∈{small,large}をパラメータとし，GA(sync,num,size)で定義する．

2.3 諸定義

文献[6]では，二次元直交座標系 S において $u=(u_x,u_y)$,$v=(v_x,v_y)$の 2 点を考えた時，$(u_x*v_y - u_y*v_x)<0$ 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していた.本稿では六角格子座標系 H においても，

格子点上の 2 点, u=(u_x,u_y),v=(v_x,v_y)を考えた時, u'=(u_x'=$\frac{\sqrt{3}}{2}u_x$, u_y'=u_x +$\frac{1}{2}u_y$), v'=(v_x'=$\frac{\sqrt{3}}{2}v_x$, v_y'=v_x +$\frac{1}{2}v_y$)として二次元直行座標系の値に適用し，(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する．

ある v'∈H に対して，N_H(v)の点のうち下記の条件を満たす点に対して表記を付ける．

p_v':(dist(O,p_v')=dist(O,v')-1)∧(v_x'p_y'-v_y'p_x'<0)

e_v':(dist(O,e_v')=dist(O,v'))∧(v_x'e_y'-v_y'e_x'<0)

3. 六角形格子状の集合アルゴリズム

3.1 共通座標系知識有の場合

六角格上で共通座標知識有の場合についてのアルゴリズム hexagonSpiral∈GA(async,unknown,small)について述べる．

既存アルゴリズムとその一般化

文献[6]で提案された GA(async,unknown,small)のアルゴリズムが spiral であり，その動きのパターンは図 5 である．

アルゴリズム 1 点 $v = (v_x, v_y)$ 上のロボット r 上のアルゴリズム *Spiral*

1: **Predicate**:
2: $Spin(r) \equiv (v \neq 1 \vee y > 0) \wedge (e_v \notin C)$
3: $Upper(r) \equiv (v = 1) \wedge (y \leq 0) \wedge ((v_x - 1, v_y) \notin C)$
4: $Stay(r) \equiv (v = O) \vee (\neg Spin(r) \wedge \neg Upper(r))$
5: **Actions**:
6: Spin :: $Spin(r)$ → 行き先を e_v に
7: Upper :: $Upper(r)$ → 行き先を $(v_x - 1, v_y)$ に
8: Stay :: $Stay(r)$ → 移動しない

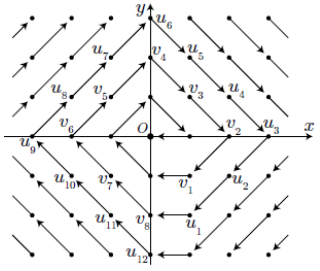


図 5.　アルゴリズム Spiral の移動パターン

正方格子における GA(async,unknown,small)では，ロボットのサイズが small であるため斜めに移動するロボットが他のロボットに接触することはない．しかし，非同期に移動するロボットがすれ違う移動をすることなく，また複数のロボットが同一の点に移動してしまうことなく集合を達成しなくてはならない．このためにアルゴリズム Spiral で取られた戦術はロボットを原点に対して右回りで移動させ特定の位置においてのみ距離を縮めるような一本道を設計することである．このとき，ロボットは移動先にロボットがいないとき移動する．一本道であるので，複数のロボットが同一の点へ移動したり，すれ違うように移動してしまうことはない．

このアルゴリズム Spiral は正方格子上のみでなく一般的な様々な座標軸上に対応する形で同様の戦術で動くように記述できる．共通の座標系，右周りに対する合意を持ち，移動可能な点同士が線で繋がったグラフを考えた際に，距離が同じ点 v_i で誘導される部分グラフがハミルトン閉路を含むといった条件を満たす時，下記の戦略で条件を満たす様々な座標系における集合を達成するアルゴリズムを設計できる．

条件を満たすグラフ上で各原点からの距離iの点の集合V_iにおいて
・原点への距離を縮める移動をする点o_iと
　o_{i+1}からの移動先となっている点e_iの2点を定める(o_{i+1}とe_iの2点は隣接)
・e_iからo_iに向け順序付けを行いハミルトン閉路を作成する
・点vに対するハミルトン閉路上での次の順序の点をnext(v)とする

グラフ上の原点からの距離iの点v上のロボットは
If (v=o_i)∧(移動先にロボットが存在しない)
　→e_{i-1}に移動
Else if (v≠o_i)∧(移動先にロボットが存在しない)
　→next(v)
Else 移動しない

この戦略において特に二次元直交座標系や六角形格子上では e_i と o_i またハミルトン閉路の順序付けを以下の同様のアルゴリズムで作成できる．

ハミルトン閉路の順序付け(next(v))は右回りの方向($\vec{v}$)で行う
原点からの各距離毎に1点ずつ存在する
o_iの集合Oとe_iの集合Eを以下のように再帰的に定義する.

(i)原点をe_0と定め,原点からの距離1の点から任意でe_1を1つ定める
$e_0, e_1 \in E$
(ii)If $\overleftarrow{eiv} \wedge N(e_i) \wedge dist(e_i)=dist(v)$
　　o_i=v∈O
　If $\overleftarrow{eiv} \wedge N(e_i) \wedge dist(e_i)=dist(v)+1$
　　o_{i+1}=v∈O
　If $\overrightarrow{oiv} \wedge N(o_i) \wedge dist(o_i)=dist(v)$
　　e_i=v∈E

GeneralAlgorithm1
座標系上の原点からの距離iの点v=(vx,vy)のロボットrのアルゴリズム
If (v∈O∧(移動先にロボットが存在しない))
　→e_{i-1}に移動する
Else if (v∉O∧(移動先にロボットが存在しない))
　→同距離内で$\vec{v}$の方向に移動する
Else 移動しない

六角格子への適用

二次元直交座標系上で GeneralAlgorithm1 において e_1=(0,-1)と定めることで全てのハミルトン閉路が一意に決まり，図 5 の移動をするアルゴリズム 1 を設計できる.そして六角格子上でも GeneralAlgorithm1 を適用し e_1=(0,-1)と定めることで図 6 のような移動パターンのハミルトン閉路を一意に作成するアルゴリズム hexagonSpiral を作成できる.

アルゴリズム hexagonSpiral

1: Predicate:
2: Up(r) ≡ (vx≦0 ∧ (vy=-1)) ∧ (p_v∉C)
3: Spin(r) ≡ ¬Up(r) ∧ (e_v∉C)
4: Stay(r) ≡ (v=O) ∨ (¬Up(r) ∧ ¬Spin(r))
5: Actions:
6: Up::Up(r) →行き先をp_vに
7: Spin:Spin(r) →行き先をe_vに
8: Stay::Stay(r) →移動しない

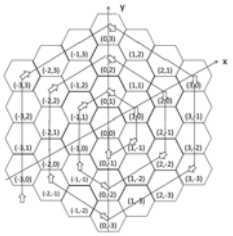


図 6. hexagonSpiral の移動パターン

ロボットが hexagonSpiral に従って移動すると，原点からの距離が L_{max-1} 以下である点と L_{max} の距離で(v_x≦0∧v_y=-1)の点から順次すべての点にロボットが移動する．以降移動を行うロボットが存在せずにこのときロボットは集合を達成している．集合を達成するまでの間少なくとも 1 台のロボットは移動可能であり，複数のロボットが同時に同一の点に存在することはない.

よって 次の定理が成り立つ.

定理 1　hexagonSpiral∈GA(async,unknown,small)

Large ロボットの六角格子上での非可解性

ロボットサイズ large の場合での非可解性について示す.

視野範囲 1 でサイズ large のロボットにおいて集合問題を解くことができないことを証明する.

補題 1　六角格子において直線状にいる距離が 2 つ離れたロボット同士において片方が距離を縮める移動をする可能性がある場合，もう片方は原点から遠ざかる移動はできない

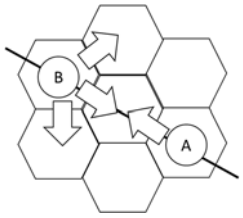
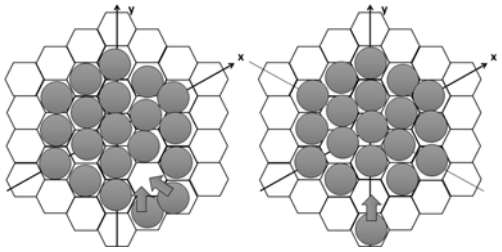


図 7.　設定できない移動パターン

(証明)　図 7 において A と B がお互いに矢印の方向に動いたときどのパターンでもロボット同士が接触する. この時 B のロボットは視野範囲 1 のため A のロボットを認識することができない. よって A のロボットが矢印に移動するかどうかを判断できずにロボット同士の接触を避けるためには A への 3 方向への移動を設定することができない.

補題 2　原点からの距離が i である全ての点の中で必ず 1 つは，軸上または$|v_x|=|v_y|$の位置において原点に距離を縮める移動が存在する.



(a)　　(b)

図 8.　軸上とそれ以外での移動

(証明)　ロボットは集合を達成するためには残り 1 つのロボットを除いて集合を達成済みという状況になる必要がある．このとき図 8(a)の状態のように，軸上または$|v_x|=|v_y|$となる点以外の点 1 つを除いて L_{max-1} と L_{max} における全ての点が集合を達成しているとする時，その点にロボットが入ることができず集合が達成できない．よって図 8(b)の状態のように L_{max-1} と L_{max} における全ての点が集合を達成しているとしても残りの位置に入ることができる軸上または$|v_x|=|v_y|$の点で距離を原点に近づける必要がある.

定理 2　視野範囲 1 かつサイズ large のロボットで集合問題を解くアルゴリズムは存在しない.

(証明)　補題 2 より必ず存在する軸または$|v_x|=|v_y|$の点からの移動を 1 つ考える．そこから直線状の距離 2 離れた位置にいるロボットは補題 2 より図 9(a)のように原点に距離を縮める 3 方向にしか移動できない．ロボットが集合を達成するためにはこのロボットが移動する必要があり 3 方向どこに移動するときでも直線上で原点に対して距離 1 近い点にロボットがいる場合は移動できない．より図 9(b)のようにその点のロボットは原点に対し距離が 1 遠い点にロボットが存在する状態で移動する必要があるがその時この点のロボットは原点に近づく 3 方向にしか移動できない．またその点に対する直線状で原点に 1 近いロボットについても同様でありこれが繰り返される．

原点のロボットが動かない場合，図 9(c)のようにロボットが直線状にいる時どのロボットも移動できず集合が達成できない．また原点のロボットが動く場合は直線状にいるそれ以降のロボットは原点から離れる移動しかできず永遠に原点に近づくことができないため集合を達成することができない

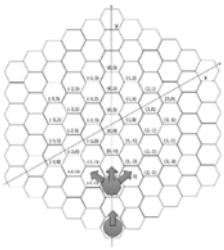


図 9(a)

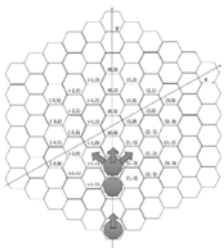


図 9(b)

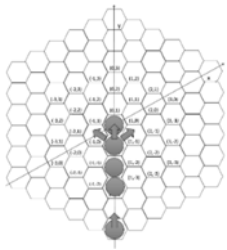


図 9(c)

3.2 共通座標系なしの場合

共通の座標系なしでのアルゴリズム hexagonPullSpin∈GA<ssync,known,small>の説明を行う．文献[7,8]では二次元直交座標系で共通の座標系なしの場合は，共通の原点，単位距離，識別されない互いに直交する 2 つの座標軸の方向，右回り方向を知っており，ロボットらはそれぞれが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といった定義であった．六角格子座標系では時計回りの回転を正として y→x に+60 度でありそのときの X, Y 軸それぞれの向きを固定する 6 パターンの状態とする．

既存アルゴリズム

文献[7]で提案された GA(async,unknown,small)の場合の集合問題を解くアルゴリズムが PullSpin であり，その動きのパターンは図 10 である．Pull は原点に対し右回りで距離を縮める移動であり，Spin は原点に対し右回りで同距離の点に動く移動である．複数のロボットが同一に移動させないために Pull を Spin より優先させ，軸上のロボットは Pull しないことで実現した．また永遠に移動し続けるロボットを無くすことを L_{max} に存在するロボットの Spin を禁止することで実現した．

Algorithm 1 点 $v=(v_x,v_y)$ 上のロボット r のアルゴリズム

1: **Predicate:**
2: $PullBlocked(r) \equiv p_v \in \mathcal{C}$
3: $SpinBlocked(r) \equiv s_v \in \mathcal{C}$
4: $SpinStop(r) \equiv e_v \in \mathcal{C}$
5: $Onthe(r) \equiv (v_x = 0) \vee (v_y = 0)$
6: $StopDistance(r) \equiv dist(O, v) = L_{max}$
7: $GatheringStop(r) \equiv StopDistance(r) \wedge ((dist(O, v) = 1) \vee \neg OntheAxes(r))$
8: $Pull(r) \equiv \neg PullBlocked(r) \wedge \neg OntheAxes(r)$
9: $Spin(r) \equiv \neg Pull(r) \wedge \neg SpinBlocked(r) \wedge \neg SpinStop(r) \wedge \neg GatheringStop(r)$
10: $Stay(r) \equiv \neg Pull(r) \wedge \neg Spin(r)$
11: **Actions:**
12: Pull::$Pull(r)$ → 行き先を p_v に
13: Spin::$Spin(r)$ → 行き先を s_v に
14: Stay::$Stay(r)$ → 移動しない

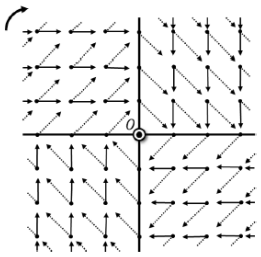


図 10.　アルゴリズム PullSpin の移動パターン

六角格子への適用

先行研究の方法を六角格子座標系上に当てはめ作成する．まず原点に対する右回りと特定の位置を実現するために六角格子座標系を図 11 のような 6 つのエリアに分ける．

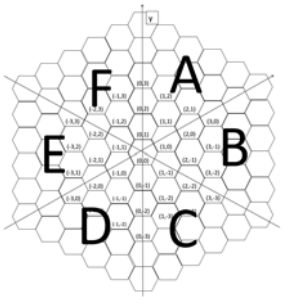


図 11.　6 つのエリア

これにより Pull と Spin を実現し実際にそれを実現したアルゴリズムが hexagonPullSpin であり, そのアルゴリズムの移動パターンが図 12 である.

hexagonPullSpin

Predicate:
PullBlocked(r)≡rightup(r) ∨ rightdown(r) ∨ down(r) ∨ leftdown(r) ∨ leftup(r) ∨up(r)
down1(r)≡{A∧(vy-1) ∉C}
leftdown1(r)≡{B∧(vx-1) ∉C}
leftup1(r)≡{C∧(vx-1,vy+1) ∉C}
up1(r)≡{D∧(vy+1) ∉C}
rightup1(r) ≡{E∧(vx+1) ∉C}
rightdown1(r) ≡{F∧(vx-1,vy-1) ∉C}
SpinBlocked(r)≡rightdown(r) ∨ down(r) ∨ leftdown(r) ∨ leftup(r) ∨ up(r) ∨rightup(r)
rightdown2(r)≡{A∧{(vx+1,vy-1) ∉C}
down2(r)≡{B∧{(vy-1) ∉C}
leftdown2(r)≡{C∧{(vx-1) ∉C}
leftup2(r)≡{D∧{(vx-1,vy+1) ∉C}
up2(r) ≡{E∧{(vy+1) ∉C}
rightup2(r) ≡{F∧{(vx+1) ∉C}
SpinStop(r)≡rightup(r) ∨ rightdown(r) ∨ down(r) ∨ leftdown(r) ∨ leftup(r) ∨up(r)
rightup3(r)≡{A∧{(vx+1) ∉C}
rightdown3(r)≡{B∧{(vx+1,vy-1) ∉C}
down3(r)≡{C∧{(vy-1) ∉C}
leftdown3(r)≡{D∧{(vx-1) ∉C}
leftup3(r) ≡{E∧{(vx-1,vy+1) ∉C}
up3(r) ≡{F∧{(vy+1) ∉C}
ontheAxes(r) ≡(vx=0)∨(vy=0)∨(|vx|=|vy|)
StopDistance(r)≡dist(O,v) = Lmax

Pull(r)≡¬PullBlocked(r)∧¬ontheAxes(r)
Spin(r)≡¬Pull(r)∧¬SpinBlocked(r)¬SpinStop(r)∧¬StopDistance(r)
Stay(r)≡¬Pull(r)∧¬Spin(r)

Actions:
Pull::Pull(r)
Pull(r)=down1(r)→行き先を(vx,vy-1)に
Pull(r)≡leftdown1(r)→行き先を(vx-1,vy)に
Pull(r)≡leftup1(r)→行き先を(vx-1,vy+1) に
Pull(r)≡up1(r)→行き先を(vx,vy+1)に
Pull(r)≡rightup1(r)→行き先を(vx+1,vy)に
Pull(r)≡rightdown1(r)→行き先を(vx-1,vy-1)に
Spin::Spin(r)
Spin(r)≡rightdown2(r)→(vx+1,vy-1)
Spin(r)≡down2(r)→行き先を(vx,vy-1)に
Spin(r)≡leftdown2(r)→行き先を(vx-1,vy に
Spin(r)≡leftup2(r)→行き先を(vx-1,vy+1)に
Spin(r)≡up2(r)→行き先を(vx,vy+1)に
Spin(r)≡rightup2(r)→行き先を(vx+1,vy)に

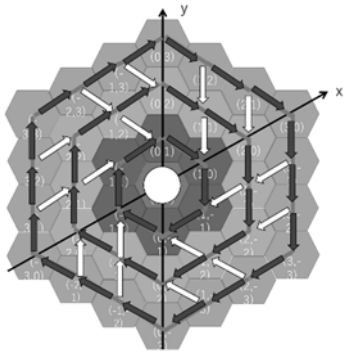


図 12.　hexagonPullSpin の移動パターン

図 12 の各エリアがどのエリアに重なるように回転させても移動方向は一致しており共通の座標系を持たない場合でも集合を達成できる. 文献[7]の二次元直交座標系と同様の方法で正当性の証明も行うことができる. より, 次の定理が成り立つ

定理 3　hexagonPullSpin∈GA(ssync,known,small)

4. 立方格子上での集合問題

3.1 の汎用アルゴリズムを実際に立方格子上に適用する.

アルゴリズム上で使用する移動パターンとして平面格子上では $u=(u_x,u_y)$, $v=(v_x,v_y)$を二次元直交座標系 S 上の 2 点とし, $(u_xv_y-u_yv_x)<0$ を満たす時, 原点を中心に v は u の右回りに位置するとしており,

・e_v:$(dist(O,p_v)=dist(O,v)-1)\wedge(v_xp_y-v_yp_x\leqq 0)$が存在した.

ここでは右回りの移動を考える際三次元上で 2 軸を固定し, 平面上での右回りを考える.

$u=(u_x,u_y,u_z)$, $v=(v_x,v_y,v_z)$を三次元直交座標系 C 上の 2 点とした時, xz 平面上での右回りの移動を

・e_{vxz}: $(dist(O,e_{vxz})=dist(O,v)-1)\wedge(e_{vpz}-e_{ypx}<0)$とし他の平面も同様の定義とする.

さらにそれと逆方向の移動を

・Le_{vxz}: $(dist(O,e_{vxz})=dist(O,v)-1)\wedge(v_{xez}-v_{yex}\geqq 0)$ とし他の平面も同様の定義とする.

4.1 汎用アルゴリズムの三次元への適用

3.1 の GeneralAlgorithm1 は二次元座標系上での汎用アルゴリズムである. 同様の条件を持ち, このアルゴリズムに用いた戦略を再帰的に書くことで三次元上に適用することができる.

各原点からの距離iの点の集合V_iにおいて
・原点への距離を縮める移動をする点o_iと
o_{i+1}からの移動先となっている点e_iの2点を定める(o_{i+1}とe_iの2点は隣接)
・e_iからo_iに向け順序付けを行いハミルトン閉路を作成する
・V_iにおいてo_iが存在しない平面全てにo_iが存在する平面に距離を縮める移動o_i'とo_{i+1}'からの移動先の点e_i'がそれぞれ2i個存在する
(o_{i+1}'とe_i'の2点は隣接)
・$o_{2i}'=e_i$, $o_i=e_1'$ である.(o_i'からはe_{i-1}'に移動する)
・それぞれの平面でei'からoi'まで右回りの移動を行うことでハミルトン閉路を設計する

If ($v=o_i\wedge$(移動先にロボットが存在しない))
→e_{i-1}に移動する
Else if
If ($v=o_i'\wedge$(移動先にロボットが存在しない))
→e_{i-1}'に移動する
Else if (($v\neq o_i'$)∧(移動先にロボットが存在しない))
→平面上で右回りに移動する

4.2 アルゴリズム Spiral3

4.1 の戦略を立方格子に適用する.

三次元としての only reduce point が一つ二次元平面の同距離内に 1 つ only reduce point という形で図 13 のような移動パターンを作成することで一方通行の経路

が得られる．実際にそれを実現したアルゴリズムが Spiral3 である．

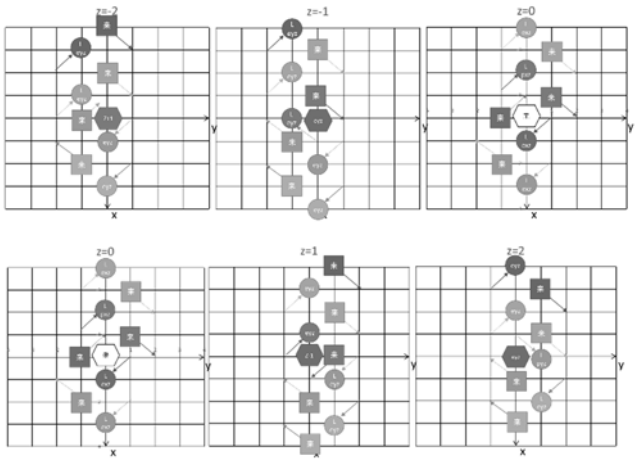


図 13．三次元上での一方通行の経路

Spiral3

```
//If v=o_i
 If (vx=0∧vy=0)∧dist(O,r)=odd∧(vz>0)∧((vx,vy,vz-1)∉C)
  →(vx,vy,vz-1)
 If
(vx=0∧vy=0)∧dist(O,r)=even∧(vz<0)∧((vx,vy,vz+1)∉C)
  →(vx,vy,vz+1)

//Else v=¬o_i
 //If   v=o_i'
  If (vz>0){
   If (vz=odd){
    If (dist(O,r)=even)∧(vy=0∧vx<0)∧(pyz∉C)
     →pyz
    If (dist(O,r)=odd)∧(vy=1∧vx>0)∧(Lpyz∉C)
     →Lpyz }
   If (vz=even){
    If (dist(O,r)=even)∧(vy=0∧vx≦0)∧(pyz∉C)
     →pyz
    If (dist(O,r)=odd)∧(vy=1∧vx≧0)∧(Lpyz∉C)
   →Lpyz }
  }
  If (vz<0){
   If (vz=odd){
    If (dist(O,r)=even)∧(vy=-1∧vx≦0)∧(Lpyz∉C)
     →Lpyz
    If (dist(O,r)=odd)∧(vy=0∧vx≧0)∧(pyz∉C)
     →pyz }
   If (vz=even){
    If (dist(O,r)=even)∧(vy=-1∧vx<0)∧(Lpyz∉C)
     →Lpyz
    If (dist(O,r)=odd)∧(vy=0∧vx>0)∧(pyz∉C)
     →pyz }
 }
 If vz=0{
   If (dist(O,r)=even)∧(vy=0∧vx<0)∧(Lpxz ∉C)
    →Lpxz
   If (dist(O,r)=odd)∧(vy=0∧vx>0)∧(Lpxz∉C)
    →Lpxz
 }
 //Else If v=¬o_i'
  Else (pxy∉C)
   →pxy
```

図 13 での移動パターンは基本的に右回りの点に移動し丸の位置で平面移動を行い四角の位置に移動するパターンである．ロボットがそれを実現した Spiral3 に従って移動すると，原点からの距離が L_{max-1} 以下である点と L_{max} の距離では各距離の二次平面で次に o_i' に移動する点から順次移動し o_i'からは，各距離の三次元上で o_i に近い面に移動する．同距離内ですべての点に移動しており，o_i では三次元上で距離を縮める移動をしているので順次すべての点にロボットが移動する．以降移動を行うロボットが存在せずにこのときロボットは集合を達成している．集合を達成するまでの間少なくとも 1 台のロボットは移動可能であり，複数のロボットが同時に同一の点に存在することはない．よって 次の定理が成り立つ

定理 4‡　Spiral3∈GA(async,unknown,small)

5. むすびに

本稿では，共通の座標系を持つ場合において六角格子座標系上ではロボットサイズが小さい時に集合問題を解くアルゴリズムの提案と，ロボットサイズが大きい時に集合達成不可能となる非可解性を示した．また共通の座標系を持たない場合において六角格子座標系上でロボットサイズが小さくロボットの総数を知っており半同期の時集合問題を解くアルゴリズムも提案した．三次元直交座標系上ではロボットサイズが小さいときに集合問題を解くアルゴリズムを提案した．今後の研究課題としては六角格子座標系上でロボットサイズが大きい時に視野範囲を拡大して集合問題を解くアルゴリズムについての考察や三次元直交座標系上での他のモデルにおける集合問題を解くアルゴリズムの考察があげられる．

謝辞

本研究の一部は JSPS 科研費基盤研究（C)（課題番号 26330020，15K00011）の助成を受けて行われた．

文献

[1] Bandettini, A.,Luporini, F. and Viglietta, G.: A Survey on Open Problems for Mobile Robots, ArXiv e-prints (2011).

[2] Cieliebak, M., Flocchini, P., Prencipe, G. and Santoro, N.: Solving the Robots Gathering Problem, Automata,

‡ [10]で同様の結果が得られている．また，[9]では立方体に集合させる問題を解いている

Languages and Programming (Baeten, J., Lenstra, J., Parrow, J. and Woeginger, G., eds.), Lecture Notes in Computer Science, Vol. 2719, Springer Berlin Heidelberg, pp. 1181–1196 (online), DOI: 10.1007/3-540-45061-0 90 (2003).

[3] X. Defago, M. Gradinariu, S. Messika and P. Raipin-Parvedy, Fault-tolerant and self-stabilizing mobile robots gathering, Proc. of 20th International Symposium on Distributed Computing, pp.46-60, 2006.

[4] Flocchini, P., Prencipe, G., Santoro, N. and Widmayer, P.: Gathering of asynchronous robots with limited visibility, Theoretical Computer Science, Vol. 337, No. 1-3, pp. 147 – 168 (online), DOI: 10.1016/j.tcs.2005.01.001 (2005).

[5] Prencipe, G.: On the Feasibility of Gathering by Autonomous Mobile Robots, Structural Information and Communication Complexity (Pelc, A. and Raynal, M., eds.), Lecture Notes in Computer Science, Vol. 3499, Springer Berlin Heidelberg, pp. 246–261 (online), DOI: 10.1007/11429647 20 (2005).

[6] 伊藤公一， 片山喜章，和田幸一：”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)*．

[7] 伊藤佳進， 片山喜章，和田幸一：”共通の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).

[8] 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”, コンピュテーション研究会 講演論文集，pp．1-8，Mar. 2015.

[9]長尾英剛 : 三次元グリッド空間における自律分散ロボットの集合に関する研究，名古屋工業大学情報工学科論文，2016-02.

[10] 長尾秀剛， 片山喜章， 金鎔煥， 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，電子情報通信学会総合大会，D-22-2, 2016-03.

自律分散ロボット群における捕獲アルゴリズム

小玉 悠人　　藤原 暁宏
九州工業大学大学院 先端情報工学専攻
Email: q676113y@mail.kyutech.jp, fujiwara@cse.kyutech.jp

概要

自律分散ロボット群とは，複数のロボットが自律的，協調的に動作することによって全体として一つの問題を解決するロボット群である．この自律分散ロボット群に関して，ロボット群が特定の形状を形成する形状形成問題を解くアルゴリズムや，自律分散ロボット群を用いて移動する目標の追跡を行うアルゴリズムなどが既に提案されている．

本研究では，複数の自律分散ロボット群が，逃走する 1 台のロボットを追跡し，捕獲するアルゴリズムを提案する．また，提案アルゴリズムをシミュレーション環境において実装し，提案アルゴリズムの追跡性能を検証する．

1　はじめに

自律分散ロボット群 [1] とは，複数のロボットが自律的，協調的に動作することによって全体として一つの問題を解決するロボット群であり，耐故障性，拡張性に優れ，今後さらに発展していくと予測される研究テーマの一つである．

この自律分散ロボット群について，文献 [2] では自律分散ロボット群による通信ができない状況での追跡を行うアルゴリズムが提案されており，また，文献 [3] ではロボット群が特定の形状を形成する形状形成問題を解くアルゴリズムが提案されている．

そこで本研究では，複数の自律分散ロボット群が，逃走する 1 台のロボットを追跡し，捕獲するアルゴリズムを提案する．本研究の提案アルゴリズムは，それぞれの追跡ロボットが逃走するロボットに向かって移動する「単純アルゴリズム」，他の追跡ロボットの位置を参考にしながらターゲットを端に追い込んで捕獲を目指す「境界包囲アルゴリズム」，そしてこの 2 つのアルゴリズムを合わせた「ハイブリッドアルゴリズム」の 3 つである．

また，これらの提案手法をシミュレーション環境において実装し，自律分散ロボット群による追跡性能を検証する．

2　自律分散ロボット群

2.1　問題の定義

自律分散ロボット群とは，ロボット群全体を管理する機構を持たず，各ロボットがそれぞれ自律的に動作することにより，全体として目的を達成するロボット群のことである．

本研究，自律分散ロボットの集合 $R = \{r_0, r_1, \cdots r_n - 1\}$ が，それぞれのアルゴリズムに従って，1 台のターゲット r_t を追跡し，逃走ロボットに接触することを目的とする．なお，各自律分散ロボット（追跡ロボットとも呼ぶ）は後述のアルゴリズムより動作し，ターゲット（逃走ロボットとも呼ぶ）は人的に操作するものとする．

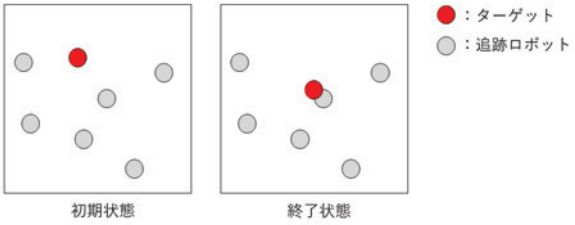


図 1: 問題の例

図 1 に問題の入出力の例を示す．追跡ロボット群は，2 次元平面上のランダムな位置から逃走ロボットの追跡を始める．1 台以上追跡ロボットがターゲットに接触していれば，追跡終了とする．

本研究ではターゲットは追跡ロボットよりも高速に移動できるものとし，その速度は，追跡ロボットの速度の α 倍であるとする（$\alpha > 1$）．また，追跡ロボット群，及び，ターゲットの可動範囲は xy 座標系に平行な矩型領域であるとし，追跡ロボット群は，可動範囲の境界を認識できるものとする．

2.2　すべてのロボットに共通する定義

本研究におけるすべてのロボットは，2 次元平面上で動作するものとし，共通の xy 座標系を持つものとする．また，2 次元平面上の 2 点 $(x_1, y_1), (x_2, y_2)$ の距離 D を，以下のユークリッド距離を用いて定義する．

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

2.3　追跡ロボット群の定義

本研究における追跡ロボット群は，すべてのロボットに共通する定義に加えて，以下に示す条件を満たすものと仮定する．

- 追跡ロボットは互いに通信する能力はない．
- すべての追跡ロボットは同じアルゴリズムを実行する．
- 各追跡ロボットは，「待機」，「観測」，「計算」，「移動」という動作を 1 サイクルとして，サイクルを繰り返す．
- 各追跡ロボットは，以前のステップで得られた情報を記憶しておくことができない．
- 各追跡ロボットは，任意の観測において，自分の座標，全ての追跡ロボットの座標，ターゲットの座標，及び可動領域を知ることができる．

また，各ロボット $r_i (0 \leq i \leq n - 1)$ の座標を点 $P_i = (x_i, y_i)$ とする．

2.4　ターゲットの定義

本研究ではターゲットの操作は人的に行われるものと仮定し，逃走時は常に人間の指定する場所を目的地として動作する．またターゲット r_t は 1 台であり，座標を点 $P_t = (x_t, y_t)$ とする．

2.5　ロボットの移動

各ステップにおけるロボットの移動方法を示す．追跡ロボットの場合，目的地は各ステップ毎に計算される座標であり，ターゲットの場合は，目的地は人間により指定される座標である．ただし，目的地への移動は 1 ステップで高々距離 d だけ移動できるものとする．

距離 d を x 方向， y 方向それぞれに分解したものをそれぞれ dx, dy とする． 座標 (x, y) にいる追跡ロボット r が，目的地 (X, Y) へ移動する場合の移動距離 dx, dy の導出方法を図 2 に示す．各追跡ロボットが 1 サイクルで移動できる距離は d なので，図 2 より，dx, 及び，dy は以下の式で計算できる．

$$dx = \frac{d \times (X - x)}{\sqrt{(X - x)^2 + (Y - y)^2}}$$

$$dy = \frac{d \times (Y - y)}{\sqrt{(X - x)^2 + (Y - y)^2}}$$

これより，(x, y) にいるロボットが目的地 (X, Y) に向かって移動する場合，1 サイクル後の位置 (x', y') は，目的地とのロボット間の距離を w とすると，以下の式で表される．

- $d < w$ の場合:

$$x' = x + dx$$

$$y' = y + dy$$

- $d \geq w$ の場合:

$$x' = X$$

$$y' = Y$$

3　追跡アルゴリズム

本論文において，追跡ロボットは以下のサイクルを実行する．

待機: 追跡ロボット群は何もしない．

観測: 追跡ロボット群は，自分の座標，全ての追跡ロボットの座標，ターゲットの座標，及び可動領域の情報を入手する．

計算: 観測で得た情報をもとに計算を行い，目的地を設定する．

移動: 計算した目的地に向かって 2.5 節で説明の移動方法で移動する．

以下では，提案する 3 つのアルゴリズム（単純アルゴリズム，境界方位アルゴリズム，及び，ハイブリッドアルゴリズム）の概要を示す．

3.1　単純アルゴリズム

単純アルゴリズムは，全ての追跡ロボットが，ターゲットを目的地として追跡するアルゴリズムである．各サイクルにおいて，以下のように目的地を設定することにより実行される．

(単純アルゴリズム)

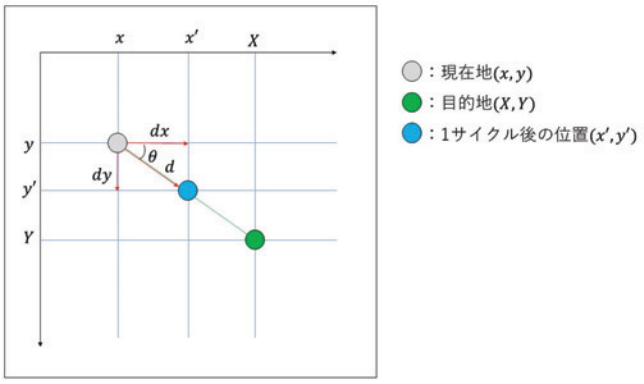


図 2: 移動距離の算出

観測において得られたターゲットの座標を (x_t, y_t) とする．このとき，目的地の座標 (X, Y) について，$X = x_t, Y = y_t$ とする．

3.2　境界包囲アルゴリズム

境界方位アルゴリズムは，追跡ロボット群により可動領域の端にターゲットを追い込むことを目的とするアルゴリズムである．本アルゴリズムは，以下のステップにより構成される．

(境界包囲アルゴリズム)
Step 1: 追跡ロボットの集合をターゲットを中心とする偏角順にソートする，このとき，偏角順にソートされた追跡ロボットの集合を $R_s = \{r_0, r_1, \cdots, r_{n-1}\}$ とし，アルゴリズムを実行する追跡ロボットを r_i $(0 \leq i \leq n-1)$ とする．
Step 2: 各ロボット r_i において，ターゲットと r_i を通る直線を引く．この直線により 2 つの半平面 P_L, P_R が得られる．(簡単のため，P_L, P_R をそれぞれ左半平面，右半平面と呼ぶ.) 次に，P_L, P_R に存在する追跡ロボットの数を計算し，それぞれ n_L, n_R とする．この n_L と n_R により，以下のような場合分けを行う．

- $n_L = 0$ の場合: このとき，半平面 P_L に存在するロボットは存在しない．そこで，ターゲットが可動領域の端から逃走することを防ぐために，以下のように仮目的地を設定する．

  (1) ターゲットと自身を通る直線に対して，左方向に角度 90°をなし，自身を端点とする半直線 L を考える．

  (2) 半直線 L において，自身から距離 d だけ離れた点を仮目的地 $P_s = (X_s, Y_s)$ とする．

  上記の仮目的地の設定方法の概念図を図 3 に示す．これによりこの追跡ロボットを可動域の端まで移動させる．

  (3) 仮目的地 P_s が可動領域内の場合は，目的地の座標 (X, Y) について，そのまま $X = X_s, Y = Y_s$ とする．仮目的地 P_s が可動領域外の場合は，以下の手順に従い，目的地の設定を行う．

  (3-1) ターゲットと自身を通る直線に対して，左方向に角度 θ をなし，自身を端点とする半直線 L_2 を考える．

  (3-2) 半直線 L_2 において，自身から距離 d だけ離れた点を目的地とする．

上記の目的地の設定方法の概念図を図 4 に示す．

このアルゴリズムにより，目的地を再設定することで，可動領域境界に到達した追跡ロボットはターゲットに向かって徐々に近づくようになる．

また，θ の数値を大きくすると，ターゲットに向かって接近する移動距離が大きくなり，反対に数値を小さくすると，接近する移動距離が小さくなる．

- $n_R = 0$ **の場合：** $n_L = 0$ の場合と同様の方法で，半平面 P_R に目的地を設定する．
- **それ以外の場合：** ロボット r_i は，追跡ロボットの間隔を均等に保つことを目的として，以下の式のように，偏角順において前後のロボット r_{i-1},r_{i+1} との中点を目的地 (X, Y) として設定する．

$$X = \frac{x_{i-1} + x_{i+1}}{2}, Y = \frac{y_{i-1} + y_{i+1}}{2}$$

なお，図 5 に本ステップの概念図を示す．

3.3 ハイブリッドアルゴリズム

本アルゴリズムは，単純アルゴリズムと境界包囲アルゴリズムを合成したアルゴリズムである．ターゲットに最も近い 1 台のロボットのみ単純アルゴリズムを実行することにより，ターゲットの逃走が困難になることを目的とする．

(ハイブリッドアルゴリズム)

ターゲットに最も近い追跡ロボットは単純アルゴリズムを実行し，それ以外の追跡ロボットは，境界包囲アルゴリズムを実行する．なお，境界包囲アルゴリズムの実行においては，ターゲットに最も近い追跡ロボットは，アルゴリズムの実行対象から除かれるものとする．

4 実験結果と考察

提案アルゴリズムを Java アプレットとして実装し，被験者 3 名に対して，追跡ロボットがターゲットを捕獲するまでの時間を計測した．なお，計測結果の値は各被験者に 5 回ずつ施行を行ってもらった結果の平均値である．

図 6 に実験結果の一部を示す．図 6 は各アルゴリズムにおいてターゲットを捕獲するまでの逃走時間を表すグラフである．この結果が示すように，単純アルゴリズムより，境界包囲アルゴリズムやハイブリッドアルゴリズムの方が捕獲時間が短くなっていることがわかる．各々のロボットでターゲットだけを見て目的地を決定するよりも，他のロボットを見て協力しながらターゲットの捕獲を目指す方が，捕獲にかかる時間を減らすことができると考えられる．

また，θ の値については，境界包囲アルゴリズムは θ の値によって捕獲までの時間が多少変化しているが，ハイブリッドアルゴリズムの方は θ の値に関わらず捕獲までの時間が安定している．境界包囲アルゴリズムの方は角度を小さくすることで包囲網を狭める時間が短くなるので，捕獲までの時間に変化が出るが，ハイブリッドアルゴリズムの方は，最終的にロボットを捕獲するのは単純アルゴリズムのロボットなので包囲網を作るまでの時間にそれほど依存しなかったためではないかと考えられる．

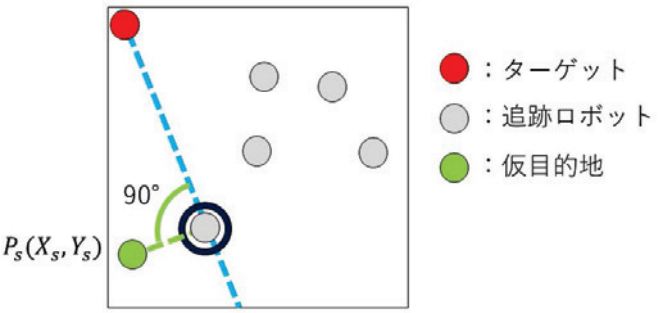


図 3: 仮目的地の設定

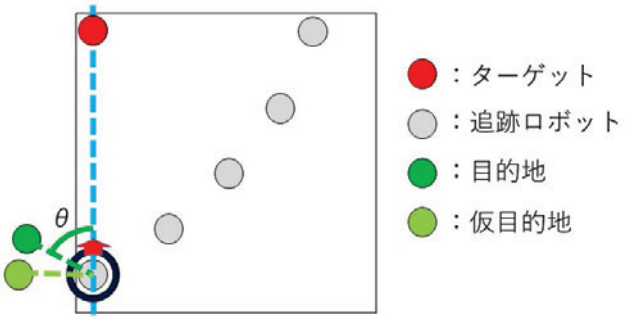


図 4: 目的地の再設定

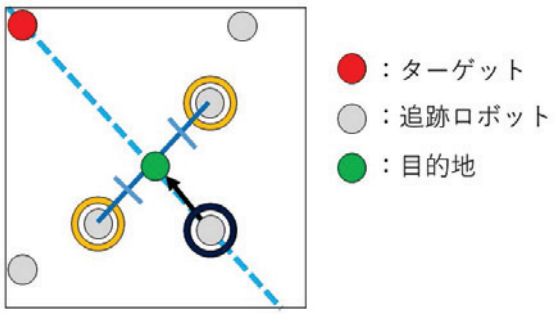


図 5: 中点を目的地とするロボットの例

図 6: アルゴリズムの実験結果

5 まとめと今後の課題

本研究では，自律分散ロボット群において，ターゲットの捕獲を目標とする捕獲アルゴリズムを提案した．中でもハイブリッドアルゴリズムはターゲットの捕獲に有用であるということを示すことができた．

今後の課題として，より実環境にあわせたシミュレーションの構築として，障害物の導入や，ロボットの視界範囲の設定などを行い，その上でのアルゴリズムの考案することなどが挙げられる．

参考文献

[1] I. Suzuki and M. Yamashita: ”A Theory of Distributed Anonymous Mobile Robots –Formation and Agreement Problems,” SIAM J. Computing 28, 4, 1347–1363(1999).

[2] 宮本淳史．視界制限状況における自律分散ロボット群による追跡アルゴリズム．九州工業大学卒業論文，2008

[3] 山中，伊藤，片山，犬塚，和田．”軸の方向に関する共有知識をもたない自律分散ロボット群に対する形状形成アルゴリズム，”電子情報通信学会論文誌，Vo1．J88-D1，No．4, pp．739-750，2005．