第４回
情報科学ワークショップ

The 4th Workshop on Theoretical Computer Science
Nagahama, Shiga, September 2008
(WTCS2008)

Toshimitsu Masuzawa
Hirotsugu Kakugawa
Fukuhito Ooshita

主担当： 大阪大学

序言

本論文集は 2008 年 9 月 7 日～9 日において滋賀県長浜市の国民宿舎豊公荘で開催された第 4 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(LA)以下のフォーマルさを目指し，お互いを徹底的に切りまくる「ちゃんばら大会」を目的として，広島大学の中野浩嗣先生の呼びかけで始まったものである．第 1 回を 2005 年 9 月に出雲で開催して以来，瀬戸（第 2 回），門司（第 3 回）と毎年 9 月に開催し，今回が 4 回目の開催である．今回は名古屋工業大学，京都工芸繊維大学，奈良先端科学技術大学院大学，広島大学，九州大学，九州工業大学，大阪大学から 46 名の参加者があり，30 件の発表と白熱した議論が行われた．今回の開催場所である国民宿舎豊公荘は豊臣秀吉ゆかりの長浜城跡の豊公園内に位置しており，琵琶湖の眺めもたいへん美しく，また，黒壁スクエアなどの歴史を感じさせる長浜市のたたずまいは，激しい議論の合間に気分をリフレッシュさせてくれるものであった．このおかげか，最後まで矛先が鈍ることなく，３日間にわたって内容の濃い討論が行われた．なお，2009 年は広島大学が主担当となり，広島地区で開催予定である．

最後になりましたが，興味深い研究成果を御発表いただいた講演者の皆様，熱心に御討論いただいた参加者の皆様に，心よりお礼申し上げます．また，今回の開催にあたり，大阪大学の角川，大下の両先生には，会場の選定から始まり，会場および宿泊の手配，会議の運営，さらには，本論文集の表紙デザインを含む全ての編集をしていただきました．また，会場の準備・運営に関しては増澤研究室の学生のみなさんにもお手伝いいただきました．これらの方々の尽力なしには，このような気持のよい環境で白熱した議論を繰り広げることはできなかったと確信しております．ここに，衷心より深く感謝致します．

2008 年 12 月

大阪大学 増澤利光

第4回 情報科学ワークショップ プログラム

9月7日（日）

13:00〜13:10 開会

13:10〜14:00 セッション1（分散アルゴリズム1） 座長：大下福仁

山内由紀子 阪大 L Output stability of self-stabilizing protocols against input changes and transient faults
蔡叔ガイ 名工大 L Space complexity of self-stabilizing leader election in population protocols without oracles

14:15〜14:55 セッション2（画像処理） 座長：中村純哉

三田尚義 広大 L 部品位置同定のための線対称図形マーカー
山本拓明 阪大 S 自己組織化マップを用いたレンジデータ照合の高速化

15:10〜16:25 セッション3（システム設計） 座長：泉泰介

八木貴之 京工繊大 S 目的環境に適合した最小パッケージ構成の自動構成システムについて
山崎雅彦 京工繊大 S 様々な仮想計算機環境で共通使用可能なOSイメージの構成方法について
石橋由子 京工繊大 S 誤った転送設定によるエラーメールを削減する転送メールゲートウェイシステムについて
宅間広大 京工繊大 S 高速二次記憶装置を用いたネットワークブートサーバの高性能化について
小西泰平 京工繊大 S 仮想環境上でのサーバ設定演習における安全な管理者権限の提供方法について

16:40〜18:10 セッション4（自律分散ロボット、エージェント） 座長：伊藤靖郎

橋本圭太 名工大 L 自律分散ロボット群による4台での正方形形成について
羽場康太郎 名工大 L リング上における自律分散ロボット群の一点集合について
山本健太 名工大 L 観測時に一様な誤差が生じるセンサーを持つ自律分散ロボット群の一点収束について
溝口隆 九大 S population protocolを用いた情報収集

20:30 自由討論会

9月8日（月）

08:30〜09:25 セッション5（並列アルゴリズム、PCクラスタ） 座長：小野廣隆

岡本直樹 九工大 S マルチコアにおける並列ソートアルゴリズムの検証と提案
田川博文 九工大 L 膜計算における辞書演算及び最大値計算
大柚智 京工繊大 S PC演習室環境と共生するPCクラスタの構成法とその評価について

09:40〜10:45 セッション6（センサネットワーク） 座長：山内由紀子

楊洋 広大 L Shape Recognition in Sensor Networks
何杏平 広大 L センサネットワークのための自律分散経路集約手法
白石忠士 九工大 S 二連結性を保証する連列センサカバーアルゴリズム

11:00〜12:05 セッション7（教育支援） 座長：大下福仁

中野浩嗣 広大 S 1万円ゲームについて
小川浩平 広大 L 1万円ゲーム優勝プログラム
青木志乃 阪大 L 学習者が多角的な視点から議論するための助言を計算機が自動的に提供する手法の提案

午後 自由討論会

18:00 懇親会，自由討論会

9月9日（火）

08:30〜09:45 セッション8（FPGA） 座長：泉朋子

川上賢介 広大 L A Tiny Processing System for Education and Small Embedded System on the FPGA
重本耕司 広大 L Accelerating Montgomery Modulo Multiplication for Redundant Number System on the FPGA
伊藤靖朗 広大 L 同期ブロックRAMの非同期ブロックRAMへの変換について

10:00〜11:35 セッション9（分散アルゴリズム2） 座長：亀井清華

鈴木健司 奈良先端大 L 共有メモリモデルにおける調停木スキップ相互排除アルゴリズム
長谷川力也 阪大 S モバイルP2Pネットワークにおけるノードの密度を考慮した資源複製法の検討
中村純哉 阪大 L 合意並列化による耐ビザンチン故障レプリケーションの高速化
泉泰介 名工大 S 局所的な辺移動によるリングの構成可能性について
泉朋子 名工大 S 証明書分散配置問題についての一考察

11:35〜11:45 閉会

目次

Output stability of self-stabilizing protocols against input changes and transient faults 1
Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa

Space complexity of self-stabilizing leader election in population protocols without oracles 6
Shukai Cai, Taisuke Izumi, Koichi Wada

部品位置同定のための線対称図形マーカー 17
三田尚義

自己組織化マップを用いたレンジデータ照合の高速化 19
山本拓明

目的環境に適合した最小パッケージ構成の自動構成システムについて 22
八木貴之，桝田秀夫

様々な仮想計算機環境で共通使用可能な OS イメージの構成方法について 26
山崎雅彦，桝田秀夫

誤った転送設定によるエラーメールを削減する転送メールゲートウェイシステムについて 30
石橋由子，桝田秀夫

高速二次記憶装置を用いたネットワークブートサーバの高性能化について 36
宅間広大，桝田秀夫

仮想環境上でのサーバ設定演習における安全な管理者権限の提供方法について 40
小西泰平，桝田秀夫

自律分散ロボット群による４台での正方形形成について 43
橋本圭太，泉泰介，片山喜章，犬塚信博，和田幸一

リング上における自律分散ロボット群の一点集合について …… 52
羽場康太郎，泉泰介，片山喜章，犬塚信博，和田幸一

観測時に一様な誤差が生じるセンサーを持つ自律分散ロボット群の一点収束について …… 60
山本健太，泉泰介，片山喜章，犬塚信博，和田幸一

population protocol を用いた情報収集 …… 66
溝口隆，小野廣隆，定兼邦彦，山下雅史

マルチコアにおける並列ソートアルゴリズムの検証と提案 …… 71
岡本直樹，藤原暁宏

膜計算における辞書演算及び最大値計算 …… 80
田川博文，藤原暁宏

PC 演習室環境と共生する PC クラスタの構成法とその評価について …… 101
大柚智，桝田秀夫

Shape Recognition in Sensor Networks …… 104
Yang Yang, Sayaka Kamei, and Satoshi Fujita

センサネットワークのための自律分散経路集約手法 …… 109
何杏平，亀井清華，藤田聡

二連結性を保証する連列センサカバーアルゴリズム …… 113
白石忠士，藤原暁宏

１万円ゲームについて …… 123
中野浩嗣

１万円ゲーム優勝プログラム …… 126
小川浩平

学習者が多角的な視点から議論するための助言を計算機が自動的に提供する手法の提案 …… 132
青木志乃，長瀧寛之，大下福仁，角川裕次，増澤利光

A Tiny Processing System for Education and Small Embedded System on the FPGA 135
Koji Nakano, Kensuke Kawakami, Koji Shigemoto, Yuki Kamada, and Yasuaki Ito

Accelerating Montgomery Modulo Multiplication for Redundant Number System on the FPGA 143
Koji Shigemoto, Kensuke Kawakami, and Koji Nakano

同期ブロック RAM の非同期ブロック RAM への変換について 151
伊藤靖朗

共有メモリモデルにおける調停木スキップ相互排除アルゴリズム 158
鈴木健司，井上美智子，藤原秀雄

モバイル P2P ネットワークにおけるノードの密度を考慮した資源複製法の検討 168
長谷川力也，山内由紀子，大下福仁，角川裕次，増澤利光

Acceleration of Byzantine Fault Tolerance by Parallelizing Agreements 171
Junya Nakamura, Tadashi Araragi, and Shigeru Masuyama

局所的な辺移動によるリングの構成可能性について 184
泉泰介，泉朋子，大下福仁

A Note on Certificate Dispersal Problems 185
Tomoko Izumi, Taisuke Izumi, Hirotaka Ono, Koichi Wada

Output stability of self-stabilizing protocols against topology changes and transient faults

Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa

Graduate School of Information Science and Technology, Osaka University,
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
Tel: +81-6-6850-6584, Fax: 81-6-6850-6582

{y-yamaut, f-oosita, kakugawa, masuzawa}@ist.osaka-u.ac.jp

Abstract

Self-stabilization provides non-masking fault tolerance against finite number of transient faults and topology changes. However, during the stabilization, self-stabilizing protocols guarantee nothing about the output of the protocol, i.e. the output may change frequently in the entire network. In dynamic networks, one of the most desirable properties for non-masking fault-tolerant protocols is the stability of output. In this paper, we introduce the notion of *output stability* of self-stabilizing protocols and we propose *output stabilizer* that improves the output stability of a particular subclass of self-stabilizing protocols.

Keywords: Distributed system, non-masking fault tolerance, autonomous adaptability, self-stabilization, output stability

1 Introduction

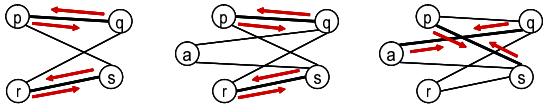
A distributed system is a network where processes communicate with each other by communication links. Recently, large scale and dynamic networks have attracted more and more attention, e.g. the Internet, ad-hoc networks, sensor networks, and inter-vehicle networks. In large scale networks, it is desirable that the system autonomously adapts to faults at processes. In dynamic networks, it is desirable that the system quickly adapts to topology changes. Researchers have tried to add autonomous adaptability to distributed protocols against faults and topology changes.

Self-stabilization is a non-masking approach against transient faults (i.e. memory crash at processes) [2]. A self-stabilizing protocol guarantees that, starting from an arbitrary initial configuration, the system eventually converges to a legitimate configuration where the protocol satisfies its specification. Self-stabilization promises autonomous adaptability against a finite number of transient faults by considering the configuration after the last fault as an initial configuration. In the same way, it also guarantees autonomous adaptability against a finite number of topology changes.

Although self-stabilizing protocols guarantee eventual stabilization, they provide no guarantee during the convergence, i.e. the processes of the *entire network* may change their outputs *repeatedly*. On the other hand, for application users, it is important that processes keep their output unchanged as long as possible, i.e. output stability, during convergence.

Chattopadhyay et al. proposed a self-stabilizing protocol for the maximal matching problem in [1]. A *matching* of a graph is a subset of edges in which no pair of edges are adjacent. A matching M is said to be *maximal* iff no proper superset of M is also a matching. A pair of processes are *matched* iff the link between them is in the matching. In Chattopadhyay's protocol, each process points to a neighboring process that is not matched and has the smallest ID. Thus, even when a process is matched with some neighboring process, if it finds a new process with the smallest ID, the process may cancel the current matching and form a new matching. Figure 1 shows an example of the execution of Chattopadhyay's protocol: in the initial configuration, the protocol outputs a maximal matching (Figure 1 (a)). When process a with the smallest ID joins the network (Figure 1 (b)), the output of the entire network changes (Figure 1 (c)). However, the protocol outputs a maximal matching in the configuration shown in Figure 1 (b). Maximal matching of a network graph is often used to establish disjoint communication paths among processes. Because application users use these communication paths to send some data (possibly) continuously, it is desirable that the valid communication paths continue to exist even when some processes are affected by a topology change or a fault. Thus, when some processes recompute matchings, the number of changes in output matching should be as small as possible.

Related works. Several papers analyze the output stability of self-stabilizing consensus protocols [4, 7, 8]. However, their focus is not on dynamic networks but



(a) (b) (b)

Figure 1: An example of Chattopadhyay's protocol

input changes and faults.

Researchers have tried to investigate self-stabilizing protocols that guarantee useful property during the stabilization. *Fault-containing self-stabilizing* protocols guarantee rapid recovery and small effect against small scale faults [5]. Fault-containment promises the spatial instability while it provides nothing about processes' instability, e.g. the number of times a process changes its output. *Super stabilization* [3] and *safe convergence* [6] promises that the output of the protocol satisfies safety during the stabilization. However, they address nothing about the stability of outputs.

Contributions. In this paper, we propose the notion of *output stability* of self-stabilizing protocols against topology changes and transient faults. Our goal is to add output stability to existing self-stabilizing protocols. We also propose the notion of *output stabilizer* that adds output stability to a subclass of distributed problems. We show an implementation of output stabilizer for the maximal matching problem.

2 Preliminary

A system is a network which is represented by an undirected graph $G = (V, E)$ where the vertex set V is a set of processes and the edge set E is a set of bidirectional communication links. Each process has a unique ID. Process p is a neighbor of process q iff there exists a communication link $(p, q) \in E$. The set of p's neighbors is denoted by N_p. The *k-neighbor* of p is the set of processes within distance k from p (except p) and it is denoted by N_p^k.

Each process p maintains local variables and the values of all local variables at p define the local state of p. Local variables are classified into three classes: input, output, and inner. The input variables indicate the input to the system and they are not changed by the system. The output variables are the output of the system for external observers. The inner variables are internal working variables used to compute output variables. We adopt *locally shared memory model* as a communication model that enables each process to read the memory contents at its neighbors. Each process changes the value of its local variables by executing a protocol. A protocol at each process p consists of finite number of *guarded actions* in the form of $\langle guard \rangle \rightarrow \langle action \rangle$. A $\langle guard \rangle$ is a boolean expression involving the local variables of p and N_p and an $\langle action \rangle$ is a statement that changes the values of p's local variables (except input variables). A process with a guard evaluated *true* is called *enabled*. We adopt *distributed daemon* as a scheduler. In a *computation step*, distributed daemon selects a non-empty subset of enabled processes (if exist) and these processes execute the corresponding actions. The evaluation of guards and the execution of the corresponding action is atomic.

A *configuration* of a system is represented by a tuple of local states of all processes. An *execution* is a maximal sequence of configurations $E = \sigma_0, \sigma_1, \sigma_2, \cdots$ such that σ_{i+1} is obtained by applying one computation step to σ_i, or σ_i is the final configuration. Maximality means that the sequence is either infinite, or it is finite and no process is enabled in the final configuration.

A *problem* (task) $\mathcal{T}(G)$ is defined by a *validity predicate* for any allowed network G of $\mathcal{T}$. (We omit G if it is clear.) The validity predicate is defined over the set of input and output variables of processes in G. A configuration σ is *valid* iff σ satisfies the validity predicate. A *non-reactive* problem is a problem such that no process changes the value of its output variables after the system reaches a valid configuration as long as the input does not changes.

A specification of *protocol* $P(\mathcal{T}(G))$ for problem $\mathcal{T}(G)$ is defined by a *legitimate predicate* on configurations. (We omit $\mathcal{T}(G)$ if it is clear.) The legitimate predicate is defined over the set of input, inner, and output variables of processes in G. A configuration σ is *legitimate* iff σ satisfies the legitimate predicate. Let $LC(P(\mathcal{T}))$ be the set of legitimate configurations of protocol $P(\mathcal{T})$ and let $VC(\mathcal{T})$ be the set of valid configurations of problem $\mathcal{T}$. The legitimate predicate of $(P(\mathcal{T}))$ satisfies $LC(P(\mathcal{T})) \subseteq VC(\mathcal{T})$.

Definition 1 ***Self-stabilization***
Protocol P is self-stabilizing iff it satisfies the following two properties:
Stabilization *: starting from an arbitrary initial configuration, it eventually reaches a legitimate configuration.*
Closure *: once it reaches a legitimate configuration, it remains in legitimate configurations thereafter.*

By a *topology change*, the sets of neighbors at some processes and/or the set of processes in the network are changed. A *transient fault* corrupts some processes by changing the values of their local variables (except input variables) arbitrarily. We say process p is *faulty* iff p is corrupted by a fault and otherwise *correct*. An *external event* consists of a transient fault and a topology change of processes. We say a process is *affected* iff it experiences an external event.

An *f-affected configuration* is obtained by an external event changing the neighbors or the local states

of f processes in a legitimate configuration.

In this paper, we consider the instability in output of a protocol after an external event corrupts some processes or changes the topology in a legitimate configuration. First, we introduce the notion of output instability. The *output instability* of a protocol is measured by *locally temporal instability*, *temporal instability*, and *spatial instability*.

Definition 2 ***Locally temporal instability***
Locally temporal instability is the maximum (worst) number of times a process changes the value of its output variable(s) until the system reaches a legitimate configuration.

Definition 3 ***Temporal instability***
Temporal instability is the maximum (worst) number of times processes change the value of output variables until the system reaches a legitimate configuration.

Definition 4 ***Spatial instability***
Spatial instability is the maximum (worst) number of processes that change the values of their output variables until the system reaches a legitimate configuration.

Locally temporal instability represents the worst local effect of an external event. Temporal instability is the sum of the number of the times each process changes its output. Temporal instability and spacial instability represent the worst global effect of an external event.

Definition 5 ***Output stable self-stabilizing protocol***
A self-stabilizing protocol P is output stable iff for any execution starting from any f-affected configuration, the locally temporal instability, temporal instability, and the spatial instability depend on f (not $|V|$).

Though our goal is to add output stability to self-stabilizing protocols, we cannot add output stability to any protocols. We restrict the target self-stabilizing protocols by limiting target problems. In the following, we show the conditions on target problems.

Let $VO(\mathcal{T}(G))$ be the set of valid outputs for $\mathcal{T}(G)$ such that each element $O \in VO(\mathcal{T}(G))$ is a tuple of output variables at processes in G. Let o_p^O be the value of the output variable at p in O. Let G and G' be allowed networks for $\mathcal{T}$. Consider an external event ϵ that changes the network from G to G' by changing the set of processes and/or the sets of neighbors of some processes, and that changes the configuration from C to C' by corrupting some processes. Let s_p^C be the local state at process p in configuration C. Thus, if process p is not affected by ϵ, $\{(p \in G \cap G') \wedge (s_p^C = s_p^{C'})\}$ holds at p. The *output distance* from configuration C on G to configuration C' on G' by ϵ is defined as follows:

$$\max_{\forall O \in VO(\mathcal{T}(G))} \left\{ \min_{\forall O' \in VO(\mathcal{T}(G'))} \{\Sigma_{p \in G \cup G'} f(p)\} \right\}$$

where

$$f(p) = \begin{cases} 0 & (p \in G \cap G') \wedge (s_p^C = s_p^{C'}) \wedge (o_p^O = o_p^{O'}) \\ 1 & \text{otherwise} \end{cases}$$

We denote the output distance between C on G and C' on G' for $\mathcal{T}$ with $\delta_{\mathcal{T}}((G,C),\epsilon,(G',C'))$. Thus, $\delta_{\mathcal{T}}((G,C),\epsilon,(G',C))$ is the minimum number of output variables that have to change their values for the system to obtain a valid output after ϵ.

Definition 6 ***PSE problem***
A problem $\mathcal{T}$ is partial solution extendable *(PSE) iff for any allowed network G for $\mathcal{T}$ and for any external event ϵ such that it affects f processes in G and the resulting network G' is also an allowed network for $\mathcal{T}$, $\delta_{\mathcal{T}}((G,C),\epsilon,(G',C'))$ is $O(f)$.*

Specification 6 is the necessary condition to provide output stable protocols for $\mathcal{T}$. For example, the maximal matching problem, the vertex coloring problem, and the minimum dominating set problem satisfy Definition 6.

We assume that the validity predicate $L_v(\mathcal{T})$ (the legitimate predicate $L_\ell(P(\mathcal{T}))$, respectively) is denoted by the following form for network $G = (V, E)$:

$$L_v(\mathcal{T}) \equiv \forall p \in V : \ell_{v,p}(\mathcal{T}), L_\ell(P) \equiv \forall p \in V : \ell_{\ell,p}(P)$$

where *local validity predicate* $\ell_{v,p}(\mathcal{T})$ and *local legitimate predicate* $\ell_{\ell,p}(P)$ are evaluated at each process with the local states at processes in N_p^k for some k smaller than the diameter of the network (i.e. these predicates are evaluated locally). We say a process is *locally valid* (*locally legitimate*, respectively) iff it satisfies the local validity predicate (local legitimate predicate, respectively), otherwise *locally invalid* (*locally illegitimate*, respectively). The validity and the legitimacy are defined over the configuration, while the local validity and local legitimacy are locally evaluated at each process.

3 Output stabilizer

The goal of this paper is to add output stability to existing self-stabilizing protocols for PSE problems. In a self-stabilizing system, when some processes are affected by an external event, the entire system starts the stabilization. Self-stabilizing protocols guarantee nothing about output stability during the stabilization. However, for PSE problems, it is possible for the system to reach a valid configuration without changing the output at many processes. The difficulty lies in how to find the set of processes that should keep their outputs unchanged and the set of processes that should change their outputs during the stabilization.

We propose the notion of *output stabilizer* for self-stabilizing protocols that solve PSE problems. Given a self-stabilizing protocol P for a PSE problem $\mathcal{T}$, the output stabilizer provides the following three functions:

1. Determine which processes should keep their output variables unchanged during the stabilization (Let V_{keep} be the set of these processes.) and which processes should change their output variables during the stabilization (Let V_{change} be the set of these processes).
2. Stop the execution of P at process $p \in V_{keep}$.
3. Execute P at process $q \in V_{change}$ without making the processes in V_{keep} locally invalid.

The proposed output stabilizer uses local validity predicate for each process to determine whether it is in V_{keep} or V_{change}. Then, if process p is invalid in the original network, the output stabilizer gives p the input for P that is a subset of the input in the original network.

In the following, we show an example of an implementation of an output stabilizer for the maximal matching problem. We denote the matching problem with $\mathcal{MM}$. Clearly, $\mathcal{MM}$ is a PSE problem.

The input of $\mathcal{MM}$ at each process p is N_p. The output variables at process p are a pointer x_p and a boolean variable s_p: x_p takes one value in the input set (i.e. x_p points to one of the processes in N_p) and s_p takes *true* iff it is matched with one of its neighbors, otherwise *false*. The goal of $\mathcal{MM}$ is to maximize the number of the pairs of matched processes. The local validity predicate of $\mathcal{MM}$ at process p is

$$\ell_{v,p}(\mathcal{MM}) \equiv$$
$$\{s_p = \mathit{true} \wedge x_p = q \wedge x_q = p \wedge q \in N_p\} \vee$$
$$\{s_p = \mathit{false} \wedge (\forall q \in N_p : s_q = \mathit{true} \wedge x_q = r \wedge r \neq p)\}$$

Let *SSMM* be a self-stabilizing protocol for $\mathcal{MM}$. In *SSMM*, each process p selects a target process for matching from processes in N_p, which is given as input to p. We assume that p checks each $q \in N_p$ sequentially in some order. When p executes *SSMM* with a subset of neighbors $N'_p \subseteq N_p$ (thus, p selects a target process from N'_p), we denote it $\mathit{SSMM}(N'_p)$.

In *SSMM*, p may select a neighbor $q \in N_p$ which is matched with other processes $r(\neq p)$, and the matching between q and r may be broken. Such actions may spread in the network, and this is why *SSMM* is not output stable. Our output stabilizer avoids such instability by restricting the input to p in such a way that p selects a process in N_p that is not matched with any process. Thus, selection of target process by p does not cause global change of output variables.

Protocol 3.1 Output stabilizer for *SSMM*

Actions for process $p \in V^i$
S_1: $\neg\ell_{v,p}(\mathcal{MM}(G)) \longrightarrow$
/ *if p is locally invalid in the original network* /
execute $\mathit{SSMM}(\{q \mid q \in N_p \wedge s_q = \mathit{false}\})$

Theorem 7 *For any self-stabilizing maximal matching protocol, Protocol* 3.1 *provides output stable self-stabilizing maximal matching protocol. The locally temporal instability of obtained protocol is Δ, where Δ is the maximum degree in the network. The temporal instability is $2f\Delta + 2f$ and the spatial instability is $4f$.*

Proof. Process p changes the value of x_p at most Δ times to find process $q \in N_p$ with $x_q = \mathit{false}$.

We show the worst case scenario of spatial instability and temporal instability. Consider the case where $f = 1$. In this case, at most one matching is broken by the external event. Let p be the affected process.
(case 1) No matching is broken: after the external event, p may point to one of its neighbors q ($x_p = q$) and then if q is not matched with other process, q may point to p ($x_q = p$). Thus, at most two processes change the value of their output variables.
(case 2) One matching is broken: Let $r \in N_p$ be the process with $x_r = p$ (p was matched with r before the external event.). Then, after the external event, p may point to one of its neighbors q ($s_p = q$) and after that if q is not matched with other process, q may point to p ($x_q = p$). If $q \neq r$, r may point to other process $s \in N_r$ ($x_r = s$) and s may point to r ($x_s = r$). Thus, at most four processes change the value of their output variables.

Consequently, the spatial instability is 4. In both cases, process p changes the value of x_p at most Δ times to find q. In case 2, process r changes the value of x_r at most Δ times to find s. On the other hand, process q (and process s in case 2) changes the value of x_q (and x_s) just once. Thus, the temporal instability is $2\Delta + 2$

When f is bigger than 1, each affected process may have other affected processes in its direct neighbors. Thus, the spatial instability is $4f$ and temporal instability is $2f\Delta + 2f$. □

4 Conclusion

In this paper, we introduced the notion of output stability of self-stabilizing protocols against transient faults and topology changes. Though the output stability of self-stabilizing consensus protocols has been studies before, our goal is to add output stability to existing self-stabilizing protocols. For the application users of non-masking fault tolerant systems, such as self-stabilizing systems, it is critical to know the instability

of the system caused by a fault or topology changes. We focused on PSE problems and proposed the notion of output stabilizer that adds output stability to existing self-stabilizing protocols for PSE problems. Though we used local validity to determine whether each process should update its output or not, there can be another key to determine it.

Future work. One of the most important issues is that whether we can design a universal output stabilizer for self-stabilizing protocols. Our approach is to relax the specification of the protocol and the system just satisfies the validity of the problem. For self-stabilizing maximal matching protocols, we add output stability by restricting the input at each process. Though the same approach can be applied some other problems (e.g. the vertex coloring problem and the minimum dominating set problem), the technique is still not universal. There is also a question whether we can add output stability to any self-stabilizing protocols by closing the gap between the validity of a problem and the legitimacy of its protocols. The goal is to propose a design paradigm for output stable self-stabilizing protocols.

Acknowledgment

This work is supported in part by Global COE (Centers of Excellence) Program of MEXT, Grant-in-Aid for Scientific Research ((B)19300017,(B)17300020, and (B)20300012) of JSPS, Grant-in-Aid for Young Scientists ((B)18700059), Grant-in-Aid for JSPS Fellows (20-1621), and Kayamori Foundation of Informational Science Advancement.

References

[1] S. Chattopadhyay, L. Higham, and K. Seyffarth. Dynamic and self-stabilizing distributed matching. PODC 2002, pp. 290–297. (2002)

[2] E. W. Dijkstra. Self stabilizing systems in spite of distributed control. CACM, vol.17, pp.643–644. (1974)

[3] S. Dolev, and T. Herman. Super stabilizing protocols for dynamic distributed systems. Proc. of WSS. pp.3.1–3.15. (1995)

[4] L. Davidovitch, S. Dolev and S. Rajsbaum. Consensus continue? Stability of multi-valued continuous consensus! GETCO. (2004)

[5] S. Ghosh, A. Gupta, T. Herman and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. PODC 1996, pp.45–54. (1996)

[6] H. Kakugawa, and T. Masuzawa. A self-stabilizing minimal dominating set algorithm with safe convergence. IPDPS APDCM, pp.11–15. (2007)

[7] S. Kutten and T. Masuzawa. Output stability versus time till output. DISC 2007, LNCS 4731, pp. 343–357. (2007)

[8] S. Dolev and S. Rajsbaum. Stability of long-lived consensus. Journal of computer and system science, vol. 67, No. 1. pp. 26–45. (2003)

Space complexity of self-stabilizing leader election in population protocols without oracles

Shukai Cai Taisuke Izumi Koichi Wada Nagoya Institute of Technology
Gokiso-cho, showa-ku, Nagoya, Aichi 466-8555, Japan

Abstract

A population protocol is one of distributed computing models for passively-mobile systems, where a number of agents change their states by pairwise interaction between two agents. In this paper, we investigate the solvability of the self-stabilizing leader election in population protocols without any kind of oracles. We identify the necessary and sufficient conditions to solve the self-stabilizing leader election in population protocols from the aspect of local memory complexity and fairness assumptions. This paper shows that under the assumption of global fairness, no protocol using only $n-1$ states can solve the self-stabilizing leader election in complete graphs, where n is the number of agents in the system. To prove this impossibility, we introduce a novel proof technique, called closed-set argument. In addition, we propose a self-stabilizing leader election protocol using n states that works under the assumption of unfairness. This protocol requires the exact knowledge about the number n of agents. It is also shown that such knowledge is necessary to construct the self-stabilizing leader election protocol.

1 Introduction

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 quite small compared to the region, two agents can communicate only when they are sufficiently close to each other in the region. Passive mobility appears in many real systems. A representative example is a network of smart sensors attached to cars or animals. In addition, a certain kind of natural computing, such as synthesis of chemical materials and complex biosystems, can be included in passively-mobile systems by regarding chemical interactions as communications. While those systems are different in the view of applications, all of them aim to a common goal, that is, how to organize and manipulate computing entities that are uncontrollable in the sense of mobility. Then, it is reasonable to think about some common principle underlying them. Revealing such principle from the aspect of theoretical computer science is an interesting and worthwhile challenge.

Recently, as a model for such passively-mobile systems, *population protocols* are introduced [1, 2, 9]. A population protocol consists of a number of agents to which some program (protocol) is deployed. Following the deployed protocol, each agent changes its state by pairwise interaction to other agents (that is, interactions imply that an agent approaches to the close area of other agents in the region). Typically, the capability of each agent is limited. More precisely, it is often assumed that each agent has only constant-space memory and no identifier. The population protocol is a good abstraction that captures the feature of passively-mobile systems in spite of its mathematical simplicity. Therefore, in the last few years, the interest to it is rapidly growing among the community of the distributed computing [3, 4, 5, 6, 7, 8, 10].

Population protocols are originated by Angluin et al. [1], which investigate the class of predicates that can be computed autonomously over population protocols consisting of weak agents. The primary result of this paper is to show that any predicate in *semilinear* class can

be computed on population protocols by proposing protocols that stably compute any predicate in the class. In the following paper [9], it is also shown that any computable predicate by population protocols belongs to semiliner, that is, semiliner is the necessary and sufficient class of the predicate that can be computed on population protocols.

The protocol computing semilinear predicate, proposed in the above paper, is assumed to start from a properly-formed system configuration. In this sense, it is not a *self-stabilizing* protocol: Self-stabilization is one of the desirable properties of distributed computations, which ensures that the system necessarily converges to the desired behavior regardless of its initial configuration. The primary benefit of self-stabilization is that self-stabilizing protocol require no global initialization. In addition, it also brings another benefit of resilience to any transient faults. Generally, it is not guaranteed that the system correctly come back to its desired behavior after the recovery of transient faults because some effects cause by transient fault, such as memory corruption, remain. However, by the nature of self-stabilization, self-stabilizing protocols necessarily recover their correct behavior.

Self-stabilization on population protocols is considered in several previous papers, [2, 13, 11], which have investigated the solvability of the *self-stabilizing leader election* (SS-LE) problems under some kinds of assumptions. The general model of population protocols introduces an interaction graph specifying the possibility between two agents. The above papers show the solvability and unsolvability of SS-LE for specific classes of interaction graphs such as complete graphs, rings, rooted trees, directed acyclic graphs, etc. Unfortunately, it is easily shown that SS-LE is almost impossible if we assume nothing. Thus, the above papers also consider some additional (but reasonable) assumptions to make SS-LE solvable by introducing several notions extending the computational power of population protocols: *global fairness* and *leader detector* oracle $\Omega?$. A fairness assumption is a constraint for possible executions on population protocols. Intuitively, the global fairness prevents the occurrence of livelock caused by the loop of some illegal execution. The leader detector oracle is an abstracted virtual device that informs the existence and inexistence of the agents having leader states. Both of assumptions give some additional computational powers to population protocols, which are sufficient to solve SS-LE in some cases, but insufficient in some other cases. However, the complete characterization of system assumptions making SS-LE solvable is unknown. Currently, only a few results about the solvablility of SS-LE on the complete interaction graphs are known:

1. Assuming global fairness and the oracle $\Omega?$, there exists a SS-LE protocol where each agent uses only one bit of memory.

2. Under the assumption of unfairness and no oracle, no uniform protocol can solve SS-LE, where a uniform protocol is one that works correctly on the system with arbitrary number of agents (that is, uniform protocols do not use any information about the total number of agents).

3. Without $\Omega?$, any protocol using only one bit of memory cannot solve SS-LE even if we assume global fairness.

In this paper, we also investigate the solvablity of SS-LE on population protocols. In particular, we are interested in self-stabilizing leader election protocols in complete interaction graphs that have no use of oracles. The primary contribution of our work is to identify the necessary and sufficient conditions such that SS-LE becomes solvable from the aspects of local memory space and fairness assumptions. More precisely, this paper shows the following three results:

1. Without oracles, there is no SS-LE protocol using only $\log_2(n-1)$ bits of memory even if we assume the global fairness.

2. There exists an SS-LE protocol that uses $\log_2 n$ bits of memory and correctly works under the unfairness assumption.

3. Even if we assume global fairness, without oracle, there is no uniform SS-LE protocol in strong sense. More precisely, we show that any SS-LE protocol working correctly on n agents does not work on $n + k$ agents ($k > 0$).

The third result implies that even the knowedge about the upper bound for the number of agents is insufficient to design SS-LE protocol, and thus it justifies the fact that the exact value of n is necessary to construct the protocol shown in the second possibility result. It should also be noted that the first impossibility result is quite nontrivial and interesting. The global fairness is reasonable but sufficiently strong so that it can break the essential ideas leading the previous impossibility results. Actually, under the global fairness assumption, many of the existing techniques to prove the impossibility cannot be adapted. In this paper, we resolve such difficulty by introducing a novel proof technique based on *closed sets*. The key idea of it is to identify the set of states that never create the leader state. While this paper utilizes this technique to show the impossibility of SS-LE, we believe that the proposed technique can be applied to more broader cases, including other problems and other graph classes, to prove the impossibility under the global fairness assumption.

1.1 Related Work

Leader election population protocols are first introduced in [3]. In [2], a non-uniform population protocol is given to solve the self-stabilizing leader election problem in directed rings of odd size under the assumption of global fairness. And the authors think there is no uniform self-stabilizing leader election protocol for all the graphs in the non-simple class. A class C is *simple* if there does not exist a graph in C which contains two disjoint subgraphs that are also in C.

In [13], Fischer and Jiang introduce an *eventual leader detector* $\Omega?$ to realize uniform self-stabilizing leader election protocols for complete graphs and directed rings. They give an uniform self-stabilizing leader election protocol for complete graphs under both global and local fairness using only 1 bit of memory, and give a uniform self-stabilizing leader election protocol for directed rings under the assumption of global fairness using 3 bits of memory. While under the assumption of local fairness, the uniform self-stabilizing leader election protocols are proved to be impossible. All the results of the paper are obtained with the help of $\Omega?$.

Canepa and Maria Gradinariu [11] give an uniform self-stabilizing leader election protocol for rooted trees and acyclic graphs with only one sink-node under the assumption of both global and local fairness. Also they give a probabilistic protocol for arbitrary graphs under local fairness. But the protocol is not silent. Moreover, they prove the necessity of $\Omega?$ to realize a uniform self-stabilizing leader election protocol. All the results in the paper are under the assumption of using 1 bit of memory and with the help of $\Omega?$.

1.2 Organization of Paper

In Section 2. We introduce some formal definitions of population protocols necessary to present the results of this paper. In Section 3. We first discuss about the difficulty of proving the impossibility under the assumption of global fairness. Then we show our first impossibility result. In Section 4. we give a SS-LE protocol without using any oracle but using n states. Section 5 provides our third result that there exists no single protocol that works correctly in complete graphs with two different sizes. Finally, We conclude this paper in Section 6.

2 Model and Definitions

We introduce the formal definitions of population-protocol considered in this paper.

A population consists of n agents, which can change its own state by interacting with each other. In the general model of population protocols, all pairs of agents do not necessarily

have direct interactions. The possibility of direct interactions between two agents is defined by interaction graphs: An interaction graph G=(V, E) is a simple directed graph where each vertex, labeled by $v_1, v_2, v_3, \cdots$, corresponds to each agent. The edge from a node v_i to v_j implies that the agent corresponding to v_1 can interact to the agent for v_2, where v_i is the *initiator* and v_j is the *responder*. Throughout this paper, we assume that the interaction graph is complete. That is, any pair of agents are possible to interact with each other.

A *protocol* $P = (Q, \delta)$ is a pair of a finite set Q of *states* and a *transition function* δ that maps each pair of states $Q \times Q$ to a nonempty subset of $Q \times Q$. The transition function, and the protocol, is *deterministic* if $\delta(p, q)$ always contains just one pair of states. In this paper, we only consider deterministic protocols, and thus we simplify the definition of a transition function to a mapping $\delta : Q \times Q \to Q \times Q$ (i.e., the states after each transition is uniquely determined). For any transition $r : (p, q) \to (p', q')$ of δ, we call p and q the *prestates* of r, p' and q' are called the *poststates* of r. Notice that a transition does not necessarily cause either of the nodes to change its state. That is, a transition $(p, q) \to (p, q)$ is possible. We define *silent* transitions as ones that do not change any state. The transition that is not silent is called *active*.

Formally, a configuration C is an n-tuple $(q_1, q_2, q_3, \cdots, q_n)$ of states where each entry q_k corresponds to the state of the agent v_k. The state of an agent v_k at the configuration C is denoted by $C(v_k)$. Letting C be a configuration, and r be a transition that maps (p, q) to $(p'q')$, we say that r is *enable* in C if there exists an edge (v_i, v_j) such that $C(v_i) = p$ and $C(v_j) = q$. Then, we say that C can go to C' via r, denoted by $C \xrightarrow{r} C'$, if C' is the configuration that are obtained by changing the states of v_i and v_j to p' and q' respectively. We simply say that C can go to C', denoted $C \to C'$, if $C \xrightarrow{r} C'$ holds for some transition r. We define executions in population protocols as follows:

Definition 1 (Execution) Letting $P = (Q, \delta)$ be a protocol, an *execution* of P is an infinite sequence of configurations and transitions $C_0, r_0, C_1, r_1, \cdots$ satisfying

1. for each i, r_i is a transition of δ and $C_i \xrightarrow{r_i} C_{i+1}$, $i = 0, 1, \cdots$ holds , and
2. r_i is active for infinitely many i unless all the enable transitions are silent.

Notice that the second condition ensures the progress of protocols (i.e., it excludes the meaningless executions such that only silent transitions appear in it).

2.1 Fairness Assumption

Fairness is an assumption that restricts the behavior of systems. Formally, it is defined as a constraint for executions. In this paper, we introduce the following two fairness assumptions [13]:

Definition 2 (Global fairness assumption $\mathbb{G}$) An execution $E = C_0, r_0, C_1, r_1, \cdots$ is global fairness, if for every C and C' such that $C \to C'$, if $C = C_i$ for infinitely many i, then $C_i = C$ and $C_{i+1} = C'$ for infinitely many i.

Definition 3 (Local fairness assumption $\mathbb{L}$) An execution $E = C_0, r_0, C_1, r_1, \cdots$ is local fairness, if for every transition r, if r is enable in C_i for infinitely many i, then $C_i \xrightarrow{r} C'_{i+1}$ for many i. Hence, the transition r is taken infinitely many times in E. We say the scheduler $\mathbb{S}$ is local fairness.

In addition to the above, we also define *unfairness assumption* $\mathbb{U}$, which requires no assumption to executions. Given a protocol P and an arbitrary fairness assumption $X \in \{\mathbb{G}, \mathbb{L}, \mathbb{U}\}$, we define $\mathcal{E}_X(P)$ to be the set of all executions of P satisfying the fairness assumption X.

2.2 Self-stabilization, Legitimate Configuration

Self-Stabilizing protocols guarantee the convergence into their desired behavior starting from arbitrary initial configurations. In this paper, we consider the self-stabilizing leader election over populations, which requires that the system eventually reach a legitimate configuration, where exactly one process keeps a special state, called *leader state*, and no other leader is generated in any following executions. Formally, the self-stabilizing leader election problem is defined as follows:

Definition 4 (Self-stabilizing leader election) A protocol P solves the self-stabilizing leader election under fairness assumption X if there is one special state s and any execution E in $\mathcal{E}_X(P)$ satisfies that there exist some i and v_k such that for any $j \geq i$ and $h \neq k$, $C_j(v_k) = s$ and $C_j(v_h) \neq s$ hold.

3 Impossibility of Self-Stabilizing Leader Election Using Only $n-1$ States

In this section, we will show that without the help of $\Omega?$, the self-stabilizing leader election protocol is impossible in a complete network graphs under global fairness only use $n-1$ different states ($\log_2(n-1)$ bits of memory).

3.1 Difficulty of Proving Impossibility under the Global Fairness

In this section, we explain why it is quite nontrivial and difficult task to prove impossibility under the global fairness. We show existing techniques used to prove the impossibility do not work under the global fairness assumption.

Roughly speaking, all of existing impossibility proofs for SS-LE are roughly divided into two types: One is the argument by *illegal loop*, and the other one is that by *partition*. We explain the details for both of them:

Illegal loop argument:

The key idea of Illegal loop argument is to find a looped execution including non-legitimate configuration. The infinite execution repeating the loop never converges to legitimate configurations, which contradicts to the self-stabilization property. This kind of arguments widely used in almost all areas of distributed computation. However, it cannot be applied to prove the impossibility under the global fairness because the global fairness assumption does not allow the system to repeat periodically the same behavior: If the system repeats such looped behavior, any configuration in the loop appears infinitely often. Then, under the global fairness, it is necessarily guaranteed the system could escape from the looped execution.

Partition argument:

It is the technique using the fact that it is difficult to break a certain kind of symmetry. Its basic idea is to divide a given n-node interaction graph into two same subgraphs with size $n/2$ (in general, division to three or more subgraphs can be considered). By their symmetricity, it is possible to show the existence of the execution that converges the configuration where two subgraphs independently and separately elect two leaders respectively. However, it is contradict to the uniqueness of leader. First, this argument can be applied only to the case of uniform protocols because nonuniform protocol does not guarantee to elect one leader in the divided subgraph (that is, it is not guaranteed that the protocol works correctly on $n/2$ agents). In addition, to make an execution where two subgraphs independently elect two leaders, we have to prohibit the interaction between those two subgraphs. However, if some interaction is enabled

on the edge that joints two subgraphs infinitely often, it must occur necessarily under the global fairness. We cannot eliminate the possibility that such interaction breaks the symmetricity, which can result in the union of two leaders.

To circumvent the problems the above two arguments hold, in the following subsection, we newly introduce a proof technique based on *closed sets*. Intuitively, the closed set argument finds a set of states such that the interaction between any pair of two states in the set creates no state out of the set. The key of our proof is to find a closed set excluding the leader state.

3.2 Impossibility Using $n-1$ States

First, we introduce several notions necessary for the following proofs.

Let C be a configuration. A *subconfiguration* C' of C is an n-tuple obtained by replacing several entries in C by $\perp$, where $\perp$ is the special value that masks the state of the corresponding agent. The above definition is simply extended in the case that C is a subconfiguration. That is, if a subconfiguration C' is obtained from another subcofiguration C'', by the replacement of entries, we say that C' is a subconfiguration of C''. The size of a subconfiguration C' is the number of non-$\perp$ values appearing in C', and it is denoted by $|C'|$. For example, letting $C = (a, b, d, e)$ be a configuration, $C'_1 = (a, \perp, d, e)$, $C'_2 = (\perp, \perp, d, \perp)$, and $C'_3 = (\perp, b, d, \perp)$ are subconfigurations of C whose sizes are 3, 2, and 1, respectively. In addition, C'_2 is also a subconfiguration of C'_1 and C'_2 itself, but C'_3 is not a subconfiguration of C'_1.

A *trace* is a sequence of transitions. We say a trace $T = r_0, r_1, \cdots, r_i$ is *applicable* to a (sub)configuration C_0 if there exists a sequence of (sub)configurations $C_0, C_1, \cdots, C_{i+1}$ such that $C_0 \xrightarrow{r_0} C_1 \xrightarrow{r_1} C_2 \xrightarrow{r_2} \cdots \xrightarrow{r_i} C_{i+1}$. For a (sub)configuration C and a trace T applicable to C, we define $\sigma_T(C)$ as the configuration resulted by applying T to C. If $C' = \sigma_T(C)$ holds, we often use the notation $C \xrightarrow{T} C'$.

A (sub)configuration C' is *reachable* from a (sub)configuration C, denoted by $C \xrightarrow{*} C'$, if there exists a trace T such that $C \xrightarrow{T} C'$. We say a (sub)configuration C can *generate* state p, if there is a (sub)configuration C' that is reachable from C and contains p. For a set G of states, if a (sub)configuration cannot generate any state in G, we say C cannot generate G. Letting $P = (Q, \delta)$ be a population protocol, a subset G of Q is called a *closed* set of P if for any transition $r : (p, q) \to (p', q')$ in δ, $p, q \in G$ implies $p', q' \in G$.

We first show three fundamental lemmas obtained from the above definitions.

Lemma 1 Letting C' be a subconfiguration of C, if a trace T is applicable to C', it is also applicable to C, and $\sigma_T(C')$ is a subconfiguration of $\sigma_T(C)$.

Proof Let t be the length of T (i.e., the number of transitions appearing in T). The lemma is proved by the induction on t. (**Basis**) $t = 1$: Let $r : (p, q) \to (r, s)$ be a transition appearing in T. Then, clearly, if r is enabled in C', it is also enabled in C. This implies that T is applicable to C. Let u and v be the initiator and responder of the transition r, since both $\sigma_T(C')$ and $\sigma_T(C)$ are the configurations obtained from C' and C by replacing the states of u and v by r and s respectively. Thus, $\sigma_T(C')$ is a subconfiguration of $\sigma_T(C)$. C. (**Inductive Step**): Suppose that the lemma holds for any trace with length $t-1$ or less. Then, we split the trace T into two traces T_1 and T_2 (T_1 is a prefix of T and T_2 is the remaining part). Then, since T_1 has length less than t, by the induction hypothesis, we can conclude T_1 is applicable to C and $\sigma_{T_1}(C')$ is a subconfiguration of $\sigma_{T_1}(C)$. It is clear that T_2 is applicable to $\sigma_{T_1}(C')$ because T is applicable to C'. Thus, again by the induction hypothesis, T_2 (whose length is less than t) is applicable to $\sigma_{T_1}(C)$ and $\sigma_{T_2}(\sigma_{T_1}(C'))$ is a subconfiguration of $\sigma_{T_2}(\sigma_{T_1}(C))$. This implies the lemma holds.

Lemma 2 If a (sub)configuration C cannot generate a set of states G, then for any (sub)configuration C' such that $C \xrightarrow{*} C'$, C' cannot generate G.

Proof Suppose for contradiction that C' can generate a state p in G. Then, there exists a (sub)configuration D such that $C' \xrightarrow{*} D$ and $p \in D$. Since $C \xrightarrow{*} C'$ holds, we obtain $C \xrightarrow{*} D$, which contradicts the fact that C cannot generate G.

Lemma 3 If a (sub)configuration C cannot generate a set of states G, any subconfiguration of C cannot generate G.

Proof Suppose for contradiction that a subconfiguration C' of C can generate a state p in G. Then, there exists a trace T such that $\sigma_T(C')$ includes the state p. By Lemma 1, T is also applicable to C and $\sigma_T(C')$ is a subconfiguration of $\sigma_T(C)$. This implies p belongs to $\sigma_T(C)$, which is contradiction.

The following lemmas are the key of our impossibility result.

Lemma 4 Let G ($|G| < n-1$) be a set of states, and C ($|C| > 0$) be a subconfiguration that cannot generate G. Then, either of the following conditions holds:

1: The complement of G is closed.

2: There exists a subconfiguration C' and a set of states G' such that $|C| - 1 \leq |C'|$ and $|G| + 1 = |G'|$ hold, and C' cannot generate G'.

Proof We prove this lemma by showing that the condition 2 necessarily holds if the complement of G is not closed. Let $\bar{G}$ be the complement of G. Assuming that $\bar{G}$ is not closed, there exists a transition $r : (p, q) \rightarrow (p', q')$ such that $p, q \notin G$ and at least one of p' and $q' \in G$ (because if such a transition does not exist, any intraction of two states in $\bar{G}$ results in two states in $\bar{G}$, which implies $\bar{G}$ is closed). Then we consider the following cases:

1. One of p and q cannot be generated by C: Without loss of generality, we assume that C cannot genereate p. Then, C cannot generate $\{p\} \cup G$. Therefore, we obtain $C' = C$ and $G' = G \cup \{p\}$ satisfying the condition 2.

2. Both of p and q can be generated by C: Since C can generate p, there exists a subconfiguration D such that $C \xrightarrow{*} D$ and $p \in D$. We consider the subconfiguration D' that is obtained by replacing the entry of p in D by $\perp$. Then, if we can show that D' cannot generate q, the lemma is proved by letting $C' = D'$ and $G = G \cup \{q\}$. In the following, we show it actually holds: Suppose for contradiction that D' can generate q. Then, there exists a trace T that makes D' reach a subconfiguration with q. By Lemma 1, T is also applicable to D, and $\sigma_T(D)$ includes both p and q. This implies that C can reach the configuration $\sigma_T(D)$ that includes both p and q. Then, It is clear that C can generate both p' and q' because the transition r is enable in the configuration $\sigma_T(D)$. However, either of p' or q' belongs to G and thus it is contradiction.

Lemma 5 Any self-stabilizing leader election protocol P has no closed set excluding its leader state.

Proof Suppose for contradiction that P has a closed set H which excludes its leader state in P. Consider an initial configuration C whose states are all in H. Since H is closed, so C can only generate the states in H. Because the leader state is not in H, C cannot generate a leader state. This implies that any executions starting from C cannnot reach configurations with leader. It is contradiction.

By using the above two lemmas, we can show the impossibility of self-stabilizing leader election using only $n-1$ states.

Theorem 1 There is no self-stabilizing leader election protocol that uses only $n-1$ states.

Proof We assume for contradiction that a self-stabilizing leader election protocol P which uses only distinct $n-1$ states. The $n-1$ states of the protocol P are denoted by $Q = \{s_0, s_1, s_2, \cdots, s_{n-2}\}$, where s_0 implies the leader state. The set of all transitions constituting $\mathcal{A}$ is denoted by $\delta_{\mathcal{A}}$.

Letting C be a legitimate configuration, that is, exactly one leader exists in it and another leader is not newly created in any following execution. This implies that the subconfiguration C' which obtained by masking the leader state s_0 in C cannot generate the leader state s_0. Thus, letting $C_0 = C'$ and $G_0 = \{s_0\}$, C_0 cannot generate G_0, and they satisfy $|C_0| = n-1$ and $|G_0| = 1$. By Lemma 5, there is no closed set excluding s_0 in P. Thus, the complement of G_0 is not closed. Then, by Lemma 4, we can obtain a subconfiguration C_1 and a set of states G_1 satisfying that $|C_1| \geq |C_0| - 1 = n-2$, $|G_1| = |G_0| + 1$, and C_1 cannot generate G_1. Similarly, we can also obtain C_{i+1} and G_{i+1} from C_i and G_i by applying Lemma 4 repeatedly. Finally, after applying the lemma $n-2$ times, we have a subconfiguration C_{n-2} and G_{n-2} satisfying $|C_{n-2}| \geq 1$, $|G_{n-2}| = n-1$ and C_{n-2} cannot generate G_{n-2}. Then, G_{n-2} is equivalent to Q, and thus C_{n-2} cannot generate any state. However, C_{n-2} is not empty, which implies a state q in C_{n-2} can be generated by C_{n-2}. This is contradiction.

Remarks 1 It is noteworthy that Lemma 4 and its application in the proof of the main theorem are not so strongly specialized for the impossibility proof of SS-LE. While it is not strictly proved, we expect that it is possible to extract a closed set (if such set exists) from any protocol by using the same technique. Thus, we believe such technique will become a powerful tool to obtain other impossibility results under the global fairness assumption. For example, (while it is a little trivial,) the impossibilty of *mutual exclusion problem* can be easily shown by the same argument as the above proof.

4 Leader Election Protocol Using n States

In this subsection, we will show a self-stabilizing leader election protocol $\mathcal{B}$ which uses distinct n states. The n states of the protocol $\mathcal{B}$ are denoted by $Q = \{s_0, s_1, s_2, \cdots, s_{n-1}\}$, where s_0 implies the leader state. The proposed protocol is quite simple: In the protocol, only two agents with the same state can interact. When they interact, the responder will increment the subscript of its state (modulo n). That is, when the state of the responder is s_i, it will changed to be s_{i+1}, $i = 0, 1, \cdots, n-2$, Exceptionally, s_{n-1} will be changed to s_0.

Protocol 1 $(s_i, s_i) \to (s_i, s_{(i+1) \ mod \ n})$, $(i = 0, 1, \cdots, n-1)$

In what follows, we show that the above protocol correctly elect a unique leader. First, we introduce several notions necessary for the proofs. Throughout this subsection, we use another representation of each configuration $C = (m_0(C), m_1(C), \cdots, m_{n-1}(C))$, where $m_k(C)$ $(0 \leq k < n)$ is the number of the agents whose state is s_k in C. We also define $\#_0(C)$ to be the number of $m_k(C)$ $(k = 0, 1, \cdots, n-1)$ such that $m_k(C) = 0$ holds.

The correctness of the protocol is proved by the argument based on monotonically-decreasing function, which is a standard technique for the proof of self-stabilizing protocols. We first define the *distance* between two states.

Definition 5 (Distance Function) For any configuration C, the distance $d_{k,j}(C)$ between two states s_k and s_j is defined as follows:

$$d_{k,j}(C) = \begin{cases} 0 & (m_j \neq 0 \text{ or } k = j) \\ (k-j)(m_j(C)-1) & (0 \leq k \leq j) \\ (k+n-j)(m_j(C)-1) & (j \leq k \leq n) \end{cases}$$

The *total distance* $d_j(C)$ of state s_j in C is the sum of the distances between any state and s_j, that is, $d_j(C) = \Sigma_{k=0}^{n-1} d_{k,j}(C)$.

From the definition, a pair of states s_k and s_j can have a non-zero distance only if $m_j(C) = 0$ and $m_k(C) > 1$. That is, if the distance between s_k and s_j for a configuration C is non-zero, no agent has the state s_k and two or more agents necessarily have s_j. Then, the value $(k - j)$ implies how many interactions are necessary to create the agent with state s_k from an agent having s_j in C. The distance $d_{k,j}(C)$ is obtained by multiplying the surplus number of agents having the state s_j to such the necessary number of interactions.

The following lemmas show that if the total distance of some state becomes zero, it remains zero in any following executions.

Lemma 6 If $m_k(C) > 0$ $(0 \leq k < n)$ holds in a configuration C, then $m_k(C') > 0$ in C' holds for any configuration C' such that $C \xrightarrow{*} C'$.

Proof Any one active interaction of the Protocol 1 reduces the number of $m_k(C)$ by exactly one for some k. In addition, to enable the interaction that reducing $m_k(C)$, it is nessesary that at least two agents have state s_k. This implies that $m_k(C)$ never becomes zero after it becomes more than zero.

Lemma 7 Let E be any unfair execution of Protocol 1, (i.e., $E \in \mathcal{E}_U(\mathcal{B})$). If $m_j(C) = 0$ holds for some j in a configuration C that appears in E, a configuration C' such that $m_j(C') > 0$ is reachable from C in E.

Proof Clearly, in C', the total distance of the state s_j is zero. In addition, for any configuration C'', if $m_j(C'') = 0$, two or more agents have the same state in C'', which implies that as long as no agent has the state s_j, some active interaction eventually occurs (notice that it holds even under the unfairness assumption). Thus, it is sufficient to show that any active interaction decreases the total distance of s_k by one. Let two agents having state s_k interact at a configuration C_1, and C_2 be the resultant configuration of the interaction. Then, except for $i = k, k+1$, $d_{i,j}(C_1) = d_{i,j}(C_2)$ necesarily holds becuase $m_i(C_1) = m_i(C_2)$ holds for any i other than k and $k + 1$. In addition, by the transition, the number of agents with s_k decreases by one, and the number of agents with s_{k+1} increases by one. Thus, by simple calculation, we can obtain $d_{k,j}(C_1) + d_{k+1,j}(C_1) = d_{k,j}(C_2) + d_{k+1,j}(C_2) + 1$. This implies that $d_j(C_1) = d_j(C_2) + 1$ holds.

By the above two lemmas, we can show the following corollary.

Corollory 1 For any execution $E = C_0, r_0, C_1, r_1, C_2, \cdots \in \mathcal{E}_U(\mathcal{B})$, there exists i such that $\#_0(C_j) = 0$ holds for any $j \geq i$.

The above corollary directly implies the correctness of the protocol.

Theorem 1 Protocol 1 is a self-stabilizing leader election protocol working correctly under the unfairness assumption.

5 No Simple Protocol for Complete Graphs with Difference Sizes

In Section 4, we give a protocol using n states to solve the self-stabilizing leader election in complete graphs of size n. In this section, we will show that there does not exist any single protocol to solve the self-stabilizing leader election in complete graphs with different sizes.

Theorem 2 Letting $\mathcal{A}$ be a protocol which can solve the self-stabilizing leader election in complete graphs with size n, then $\mathcal{A}$ cannot work correctly in complete graphs with size $n-1$.

Proof Consider the legitimate configuration C of a complete graph with size n. Since a new leader will not be created, so the subconfiguration D which obtained by masking the leader state in C where $|D| = n-1$, cannot generate the leader state. So consider an initial configuration $C' = D - \{\perp\}$, from which the leader state will not be generated. Noting that C' is an initial configuration of a complete graph with size $n-1$, and from such the initial configuration, the legitimate configuration will never be reached. See Figure1. Hence, $\mathcal{A}$ cannot elect a leader

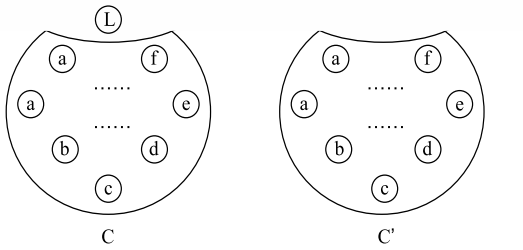


Figure 1: Complete graphs with size n and $n-1$

correctly in complete graphs with size $n-1$

6 Conclusion

In this paper, we showed the necessary and sufficient condition to the solvability of the SS-LE in population protocols having no oracles and complete interection graphs. The condition is characterezed by local memory space and fainess assumption. To prove the impossibility under global fairness, we introduce a new proof technique using closed sets.

References

[1] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, René Peralta. Computation in networks of passively mobile finite-state sensors. *Twenty-Third ACM Symposium on Principles of Distributed Computing.* (2004) 290-299.

[2] Dana Angluin, James Aspnes, Michael J. Fischer, Hong Jiang. Self-stabilizing population protocols. In *Proc. Principles of Distributed Systems, 9th International Conference*, pages 103-117, 2005.

[3] Dana Angluin, James Aspnes, Melody Chan, Michael J. Fischer, Hong Jiang, René Peralta. Stably computable properties of network graphs. In DCOSS, pages 63-74, 2005.

[4] Dana Angluin, James Aspnes, David Eisenstat, Eric Ruppert. On the power of anonymous one-way communication. In *Proc. Principles of Distributed Systems, 9th International Conference*, pages 396-411, 2005.

[5] Dana Angluin, James Aspnes, David Eisenstat. Stably computable predicates are semilinear. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing*, pages 292-299,2006.

[6] Dana Angluin, James Aspnes, David Eisenstat. Fast computation by population protocols with a leader. In *Proc. Distributed Computing, 20th International Symposium*, pages 61-75, September 2006.

[7] Dana Angluin, James Aspnes, David Eisenstat. A simple protocol for fast robust approximate majority. In *Proc. Distributed Computing, 21th International Symposium*, pages 20-32, 2007.

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

[9] James Aspnes, Eric Ruppert. An introduction to population protocols. *Bulletin of the EATCS*, 93, pages 98-117, October 2007.

[10] Joffroy Beauquier, Julien Clement, Stephane Messika, Laurent Rosaz, Brigitte Rozoy. Self-stabilizing counting in mobil sensor networks. Technical Report 1470, LRI, Université Pairs-Sud 11,2007.

[11] Davide Canepa, Maria Gradinariu Potop-Butucaru. Stabilizing leader election in population protocols. Unpublished, 2007.

[12] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *Proc. 2nd IEEE International Conference on Distributed Computing in Sensor Systems*, pages 51-66, 2006.

[13] Michael J. Fischer, Hong Jiang. Self-stabilizing leader election in networks of finite-state anonymous agent. In OPODIS, pages 395-409, 2006.

部品位置同定のための線対称図形マーカーの提案

三田 尚義 (広島大学大学院 工学研究科)

1 はじめに

近年，工場の自動化が進み，人が行っていた様々な作業はロボットが行うようになっている．その中でも部品のビンピッキング作業は自動化が難しい工程である．ビンピッキングというのは箱などにバラバラに積まれた部品から，対象となる部品を取り出してくるという作業である．人間にとって簡単なこの作業も，ロボットに高精度かつ高速に行わせるのは難しい．

本研究ではロボットの視覚センサとなる画像処理部分において対象部品の位置と姿勢を同定する方法について提案する．そこで，部品を加工する際，プレスなどでつけられるようなマーカーを特定の場所に配置しておくことで，それを目印とした対象部品の位置と姿勢の同定を行えるだろうと考えた．具体的には，線対称形状のマーカーを部品に配置し，マーカーの特徴を組合せハフ変換で抽出することで，部品の位置と姿勢の同定を目指した．

今回は楕円形状 (図 1) と半楕円弧形状 (図 2) のマーカーについて考えた．線対称軸を共有するように 2 つのマーカーを配置することで検出精度を向上させている．エッジ点対間の最小距離 ($Dmin$) を設定し，ノイズとの干渉の抑制や処理するエッジ点対の減少による効率化を目指している．エッジ点対間の最大距離 ($Dmax$) を設定し，異なるマーカーとのエッジ点対が選ばれないようにすることでマーカーの検出精度を向上させ，同時に効率化も実現している．

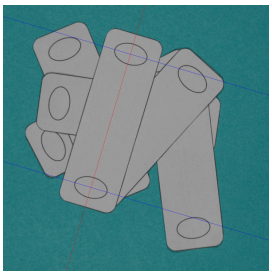


図 1 楕円マーカー

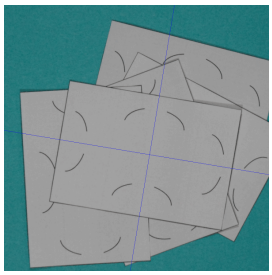


図 2 半楕円弧マーカー

2 組合せハフ変換による線対称軸検出

組合せハフ変換 [1][2] とは，特徴抽出に用いられるハフ変換という手法を拡張したものである．通常のハフ変換ではエッジ画像の 1 点をパラメータに変換する作業を繰り返すことで特徴抽出を行うが，組合せハフ変換ではエッジ点対を用いることで，より価値のある情報を扱うことができ，効率化と検出精度の向上が期待できる．

組合せハフ変換を用いる線対称軸検出方法では，対称軸をはさんで線対称な位置にある 2 点を結んだ線分の垂直二等分線が線対称軸である，という特性を利用している．まずエッジ画像 (図 3) からエッジ点対を選び，その 2 点を結ぶ線分の垂直二等分線を表すパラメータ (θ, s) に変換する．それをパラメータ空間 (図 4) を表す 2 次元累積配列に投票する．θ は対称軸の傾きを表し，s は原点からの距離を表している．全てのエッジ点対について投票を繰り返すことで線対称軸となるパラメータが投票数の多いピークとして検出できる．エッジ点対間の距離が小さすぎる点を投票に用いるとノイズに敏感に反応するので，エッジ点対間の最小距離 $Dmin$ を設定しエッジ点対間の距離が $Dmin$ 以下の点は投票に用いないようにする．また，不要な処理を行わないことで効率化を実現している．

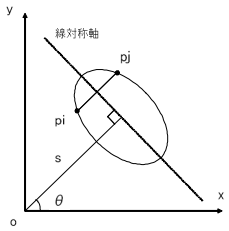


図 3 $x-y$ エッジ画像

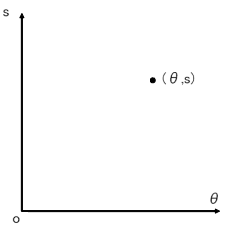


図 4 $\theta-s$ パラメータ空間

3 マーカーを用いた部品位置同定方法

3.1 楕円マーカーを用いた検出方法

まずは楕円形状のマーカーとその検出方法を提案する．楕円の特徴として直交する線対称軸 (長径，短径) とその傾きを持っている．その特徴を利用して部品の位置と姿勢を検出する．

ある程度離れた位置に 1 つの線対称軸を共有するように 2 つの楕円状のマーカーを配置しておく．この 2 つのマーカーから組合せハフ変換を用いて，1 本の投票数の高い軸とそれに直交する 2 本の軸を検出できれば部品の位置と姿勢を同定することが可能である．2 つのマーカーで，ある線対称軸を共有することで共有している軸の検出精度を上げている．さらにそれに直交する 2 つの軸を検出することができれば，2 つのマーカーの位置と姿勢を検出でき，部品の位置と姿勢を同定できたといえる．

エッジ画像に直線部分があると，その直線部分に敏感に反応してしまうので，前処理の段階である程度取り除いておく必要がある．また，2 点間の距離が近いエッジ点対を用いるとノイズの影響を受けやすくなるので 2 点間の最小距離 $Dmin$ を設定し，その距離以下となるエッジ点対は投票に用いない．マーカーのサイズは予め想定できるとして，なるべく同一マーカーを構成しているエッジ点対が選ばれるような 2 点間の最大距離 $Dmax$ を設定する．楕円マーカーにおいては長径を $Dmax$ に設定すればよい．これによって別のマーカーやノイズからの干渉を抑制することができ，高速化にもつながる．

楕円マーカーの問題点として，部品の傾きによってマーカーが変形した場合に，円弧に近づいていくという欠点がある．円弧は直線と同様にあらゆる場所で線対称軸を選ぶ事ができる曲線なので今回の検出方法では検出しづらい．楕円 (図 6) のパラメータ空間では長軸と短軸を表す 2 点に特に投票が集中しているのが分かるが，円に近くなるにつれ (図 8) パラメータ空間の広い範囲に同様の投票が行われてしまっていることがわかる．

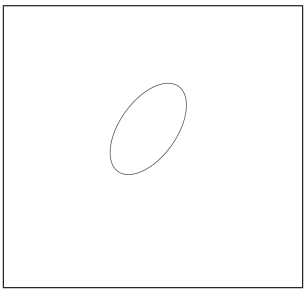


図 5　楕円

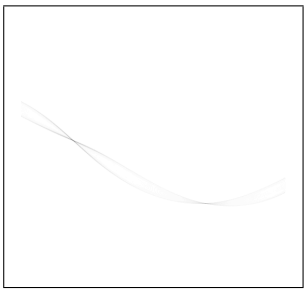


図 6　パラメータ空間

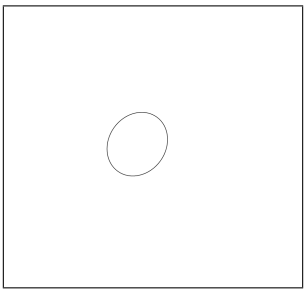


図 7　円に近づいた場合

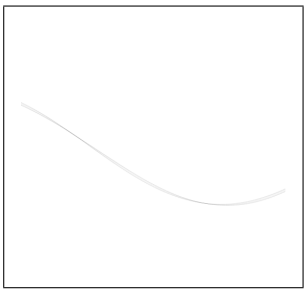


図 8　パラメータ空間

3.2　半楕円弧マーカーを用いた検出方法

次に楕円の輪郭の半分を用いたマーカーとその検出方法を提案する．楕円と違い線対称軸は 1 つしかないのでマーカー単体としての情報量は減ってしまう．しかし部品の傾きなどによるマーカー画像の変形に対する耐性は高い．

半楕円弧の中心部分を取り除いている理由は，線対称軸検出のために用いられるエッジ点対の数を減らす事と，投票の θ 方向の範囲を限定するためである．図 9 のように点線で示される軸を境にして線対称な位置にあるエッジ点対が投票に使われることが期待されるが，それを考慮してエッジ点対間の最小距離 $Dmin$ と最大距離 $Dmax$ を設定し，エッジ点対間の距離が $Dmin$ 以上 $Dmax$ 以下となるエッジ点対のみを投票に用いる．軸を境とした時，同じ領域の 2 点をエッジ点対として選ばないようにするために，同じ領域のエッジ点対が選ばれないような距離 $Dmin$ を設定する．またマーカーの大きさより明らかに大きい距離のエッジ点対も選ぶ必要がないので，マーカーの大きさに応じた距離 $Dmax$ を設定する．提案する半楕円弧マーカー (図 10) の x，y，z に対して $y \leq x < z$ という関係が成り立つ時，$Dmin = x$，$Dmax = z$ とすればよい．このように設定することで，軸をはさんだ 2 つの領域から一点ずつを選ぶことが出来る．

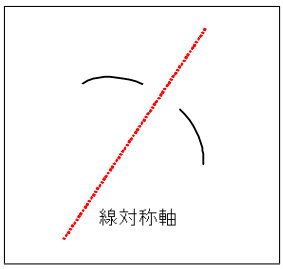


図 9　半楕円弧マーカの線対称軸

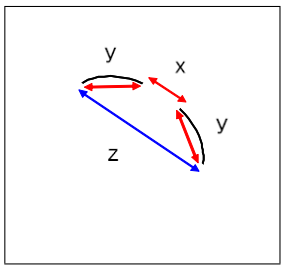


図 10　$Dmin$ 及び $Dmax$ の設定

半楕円弧マーカーの検出の際も，エッジ画像に直線部分があるとその直線部分に敏感に反応してしまうので，前処理の段階である程度取り除いておく必要がある．

図 11〜図 14 は半楕円弧マーカーを配置した金属部品の実験結果である．図 12 では Sobel オペレータを用いたエッジ検出を行い，図 13 では検出の妨げとなる直線成分を直線のハフ変換によってを除去している．図 14 の結果から一番上の部品の線対称軸 2 本を検出できたので，対象部品の位置と姿勢を同定できたといえる．

図 11　原画像 (半楕円弧)

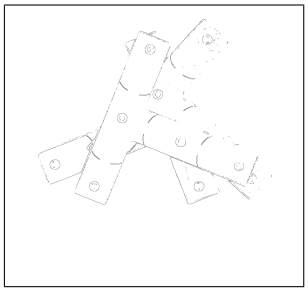


図 12　エッジ画像

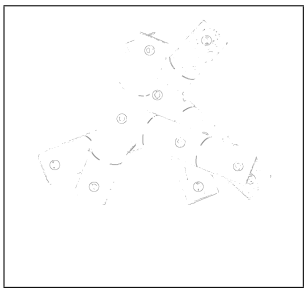


図 13　直線成分削除

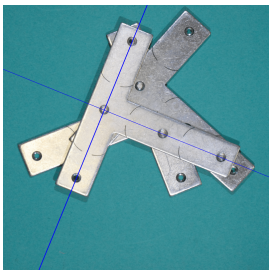


図 14　結果画像

4　まとめ

本研究では部品の位置と姿勢の同定を目指すためにマーカーを用いた画像認識方法を提案した．今回用いた人工画像と撮影したモデル部品についてはマーカーの特徴から部品位置を同定できた．さらに実際の金属部品を撮影した画像についても，同様にうまく検出できた．部品自体にノイズとなるような彫刻や傷がない場合においては，提案手法により部品の位置と姿勢の同定ができると推測できる．

参考文献

[1] Yiwu Lei,Kok Cheong Wong,1999. Ellipse detection Based on symmetry. Pattern Recognition Letters 20,41-47.

[2] 斉藤文彦, 寺澤仁,2002. 組み合わせ Hough 変換を用いた線対称形状部品の検出手法. 精密工学会誌 68(3),387-391

自己組織化マップを用いたレンジデータ照合の高速化

山本拓明
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

1. はじめに

室内をレーザセンサでスキャンし，特定の物体を検出することを考える．物体探索は，生産ラインでの自動組み立て及び検査，異常の監視，遺失物の探索及び運搬，ロボットの自律移動など，様々なアプリケーションに応用される．リアルタイム性が要求される場合や大量のデータを処理する場合などでは，実用性を高めるためには，探索処理の高速化が課題となる．

屋内をセンシングするために，レンジファインダを用いる．レンジファインダは，レーザを様々な方向に照射することで物体までの距離を細かく計測しする．それらを統合して，屋内空間の３次元形状を表現するレンジデータを得る．物体の３次元形状は通常は変化しないため，レンジデータは探索の安定性に優れる．

物体探索のアプローチとして，モデルベースの手法がある [4, 2]．探索したい物体（以降，目的物体と呼ぶ）の３次元形状（以降，モデルと呼ぶ）を用意しておき，センシングデータ（以降，シーンと呼ぶ）から目的物体を探索する．一般的な探索では，(1) シーンから特徴を抽出，(2) シーン上の点とモデル上の点の対応関係を計算，(3) モデルの空間的な位置を決定，という順に行う．

形状の特徴は，特徴ベクトルと呼ばれる多次元ベクトルで表現する [1]．シーン上の点が，モデル上のどの点に対応しているのかということは，シーンとモデルそれぞれの特徴ベクトルを比較することで判断する．

モデルの空間的な位置を決定するには，シーンとモデルの対応関係を複数用いる．センシング誤差，ノイズ，物体の変形，雑多な物体の存在などの影響のため，シーン上の点とモデル上の点の間に一対一の対応関係を定めることが困難になる．シーンとモデルの間で，多対多の対応関係を考慮し，モデルの位置を決める必要がある．本論文では，クラスタリング手法である自己組織化マップ [3]（以降，ＳＯＭと呼ぶ）を用い，効率的に一対多の対応関係を扱い，物体を探索する手法を提案する．

2. レンジデータからの物体探索

本論文で扱う物体探索問題の設定について述べる．

目的物体のモデルは事前に与えられているとする．モデルは，任意の光線との交点の判定及び計算が可能とする．

物体探索の対象となるシーンは，物体探索処理の実行時に与えられる．シーンは，３次元座標の集合 S として入力される．探索処理では，シーンの点群 S の中からモデルと一致する点群を探索する．探索結果は，S の部分集合 S_A として出力する．

出力の正しさを，次のように定義する．厳密解 S_T を，シーンの点群 S の中からモデルと正しく対応する点を過不足なく抜き出したものとする．厳密解に含まれる点を最低限選び出す割合を f_p，厳密解を網羅している最低限の割合を f_r として，$|S_T \cap S_A|/|S_A| > f_p$ かつ $|S_T \cap S_A|/|S_T| > f_r$ ならば出力は正しいと定義する．

3. 既存手法

既存手法での，シーンとモデルの対応関係からモデルの位置を求めるまでの流れを説明する．

3 1 シーンとモデルの対応関係

モデルは，座標値と特徴ベクトルの組としてデータベース化する．モデル上から N_{model} 個の点を選ぶ．それぞれの座標を $\boldsymbol{q}_\beta$，特徴ベクトルを $\boldsymbol{g}_\beta$ とする．モデル上の点を $m_\beta = (\boldsymbol{q}_\beta, \boldsymbol{g}_\beta)\,(1 \quad \beta \quad N_{model})$ としてデータベース化する．

シーンから N_{scene} 個の点を，サンプリングする．それぞれの点を $s_\alpha = (\boldsymbol{p}_\alpha, \boldsymbol{f}_\alpha)\,(1 \quad \alpha \quad N_{scene})$ とする．$\boldsymbol{p}$ は空間中の座標値，$\boldsymbol{f}$ をその点の特徴ベクトルとする．

次に，シーンとモデルの対応関係（以降，“対応関係” とはシーンのある点とモデルのある点との間の対応関係のことを指す.）を求める．シーン上の点 s_α とモデル上の点 m_β との間の

対応関係を $C_{\alpha\beta} = (s_\alpha, m_\beta)$ とする．$rel(\ ,\ )$ を，２つの特徴ベクトルが似ているほど大きな値を返す類似度関数と定義する．対応関係 $C_{\alpha\beta}$ の信頼度を，$rel(C_{\alpha\beta}) = rel(\boldsymbol{f}_\alpha, \boldsymbol{g}_\beta)$ と定義する．

3 2　対応関係のグルーピング

一組の対応関係だけでは，それが正しい対応なのか容易に判断できないため，複数の対応関係をグルーピングする．

２つの対応関係 $C_{\alpha\beta}$ と $C_{\alpha\prime\beta\prime}$ の一貫性を考える．シーン上の２点 s_α と $s_{\alpha\prime}$ の関係と，モデル上の２点 m_β と $m_{\beta\prime}$ の関係に矛盾がないかを調べる．$al(\ ,\ )$ を２つの３次元座標の位置関係を表す関数とする．シーン上での位置関係 $al(\boldsymbol{p}_\alpha, \boldsymbol{p}_{\alpha\prime})$ とモデル上での位置関係 $al(\boldsymbol{p}_\beta, \boldsymbol{q}_{\beta\prime})$ に矛盾がなければ，$C_{\alpha\beta}$ と $C_{\alpha\prime\beta\prime}$ に一貫性があると定義する．２つの対応関係 $C_{\alpha\beta}$ と $C_{\alpha\prime\beta\prime}$ の一貫性を，$cons(C_{\alpha\beta}, C_{\alpha\prime\beta\prime}) = cons(al(\boldsymbol{p}_\alpha, \boldsymbol{p}_{\alpha\prime}), al(\boldsymbol{q}_\beta, \boldsymbol{q}_{\beta\prime}))$ と定義する．

３組以上のグルーピングは，グラフのクリーク抽出とみなすことができる．グラフの頂点を $C_{\alpha\beta}$ とする．２つの頂点 $C_{\alpha\beta}$ と $C_{\alpha\prime\beta\prime}$ に一貫性があれば，その２頂点間に辺を張る．このグラフから信頼度が一定以上の頂点と，一貫性が一定以上の辺のみを残す．残ったグラフから最大クリークを抽出することで，信頼性の高いシーンとモデルの対応関係を求めることが出来る．

4.　提案手法

4 1　既存手法の問題点と対策

既存手法の問題点として，ノイズなどによる特徴ベクトルの劣化により，計算コストが増大する点がある．シーンから抽出する特徴ベクトルが誤差を含む場合，モデル上の正しい対応点との特徴ベクトルとの類似度が低下し，対応関係の信頼度が下がる．そのため，対応関係のグルーピングの際に，信頼度の低い対応関係も含める必要が生じる．結果として，扱う対応関係の個数が増え，探索処理の計算コストが増大する．

そこで，クラスタリングを用いて類似する特徴量をまとめ，シーンとモデル間の一対多の対応関係を扱うことを考える．モデル上の互いに似た点同士をまとめることで，扱う対応関係の個数を減らす．

提案手法では，クラスタリング手法として自己組織化マップ（以降，ＳＯＭと略記する）を用いる．ＳＯＭには，特徴ベクトルの非線形な分布に適応したクラスタリングが出来る利点がある．

4 2　モデル上の点のクラスタリング

モデルの特徴量 $\boldsymbol{g}_\beta (1 \quad \beta \quad N_{model})$ を SOM を用いクラスタリングし，代表的な特徴ベクトル $\boldsymbol{h}_\gamma (1 \quad \gamma \quad N_{cluster} \ll N_{model})$ に分類する．この代表ベクトルに基づき，すべての m_β を $H_\gamma = \{(\boldsymbol{q}_\beta, \boldsymbol{g}_\beta) | \forall\gamma\prime \ rel(\boldsymbol{g}_\beta, \boldsymbol{h}_\gamma) \quad rel(\boldsymbol{g}_\beta, \boldsymbol{h}_{\gamma\prime})\}$ とクラスタに分ける．

4 3　シーンとクラスタの対応関係

次に，シーンと“モデル”の対応を，シーンと“クラスタ”の対応 $D_{\alpha\gamma} = (s_\alpha, H_\gamma)$ に置き換える．この対応関係の信頼度を代表ベクトルを用いて $rel(D_{\alpha\gamma}) = rel(\boldsymbol{f}_\alpha, \boldsymbol{h}_\gamma)$ と表す．

4 4　対応関係のグルーピング

シーン上の２点 $\boldsymbol{p}_\alpha$ と $\boldsymbol{p}_{\alpha\prime}$ の位置関係は $al(\boldsymbol{p}_\alpha, \boldsymbol{p}_{\alpha\prime})$ として一意に定まる．一方，H_γ と $H_{\gamma\prime}$ は，要素同士で複数の位置関係が存在する．正解の見逃しがないように，これらの中で最も $al(\boldsymbol{p}_\alpha, \boldsymbol{p}_{\alpha\prime})$ に近い位置関係を選ぶ．つまり，

$$cons(D_{\alpha\gamma}, D_{\alpha\prime\gamma\prime}) = \max_{\boldsymbol{q}_\beta \in H_\gamma, \boldsymbol{q}_{\beta\prime} \in H_{\gamma\prime}} cons(al(\boldsymbol{p}_\alpha, \boldsymbol{p}_{\alpha\prime}), al(\boldsymbol{q}_\beta, \boldsymbol{q}_{\beta\prime}))$$

とする．

最後に，信頼度が一定以上の対応関係 $D_{\alpha\gamma}$ を頂点とし，一貫性が一定以上の２つの対応関係の間に辺を張り，グラフを作成する．このグラフから最大のクリークを抽出することで，信頼できるシーンとクラスタ間の対応関係を決定し，物体の位置を出力する．

5.　まとめと今後の課題

モデル上の点をクラスタリングし，物体探索を高速化する手法を提案した．ノイズなどにより，特徴ベクトルが劣化する場合に効果が期待できると考える．

予備実験では，他物体を誤検出する確率が３０%程度あった．正解の見逃しを防ぐために，クラスタ同士の位置関係の計算に最も都合の良い組み合わせを選んでいる．そのため，本来は選ぶべきでない組み合わせも対応関係に含めた事が原因と考える．

今後の課題として，既存手法との比較実験と，誤検出への対策が挙げられる．

謝辞

本稿の成果は，株式会社日立製作所基礎研究所との共同研究により得られたものである．ここに記して謝意を表す．

[1] Richard J. Campbell and Patrick J. Flynn: “A survey of free-form object representation and recognition techniques”, Computer Vision and Image Understanding, vol. 81, no. 2, pp.166-210, 2001.

[2] Andrew E. Johnson and Martial Hebert: “Using Spin Images for E cient Object Recognition in Cluttered 3D Scenes”, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 21, no. 5, pp.433-449, 1999.

[3] Teuvo Kohonen: “Self-Organization and Associative Memory”, Sprinver-Verlag Berlin Heidelberg, 1989.

[4] Chin S. Chua and Ray Jarvis: “Point Signatures: A New Representation for 3D Object Recognition”, International Journal of Computer Vision, vol. 25, no.1, pp.63-85, 1997.

目的環境に適合した最小パッケージ構成の自動構成システムについて

八木　貴之†,　桝田　秀夫‡.
t-yagi07@dsm.cis.kit.ac.jp, h-masuda@kit.ac.jp
† 京都工芸繊維大学大学院　工芸科学研究科　情報工学専攻
‡ 京都工芸繊維大学　情報科学センター

概要　近年，セットトップボックスや家庭用ルータなどの組み込み用途といったリソースの制約の多いシステムに，Linux などの汎用 OS を使用することが増えてきている．通常の Linux ディストリビューションは，ユーザインタフェースの拡充に伴い非常に大がかりとなってきているため，Linux ディストリビューションのインストーラが提供する「最小構成」を選択してもそのままでは組み込み用途には大きすぎるという問題がある．

プロトタイプとして，Linux ディストリビューションのインストーラが提供する「最小構成」から，稼働時のファイルアクセス記録に基づいて必要パッケージを自動的に抽出し，不要パッケージをできる限り削減するツールを実装している．しかし，パッケージ管理システムには，単純な依存関係だけではなく，選択的に依存するパッケージが存在することが判明した．これらのパッケージの選択により，さらに使用ディスク容量を削減することが可能となる．また，ファイルアクセス記録の収集に時間がかかりすぎる問題も残されている．

本稿ではディスク使用量において最小システム構成を構成する際のアルゴリズムとファイルアクセス記録の収集方法の改善法に言及する．

キーワード　Linux，組込み OS，パッケージ管理，自動生成ツール

1　はじめに

従来，セットトップボックスや家庭用ルータなどの組み込み用途といったリソースの制限の多いシステムでは，携帯電話向け OS「Symbian」に代表されるように組み込み専用 OS や独自 OS などがよく使用されている．近年，汎用 OS としての Linux は，Intel の x86 系 CPU だけではなく，PowerPC や ARM，SH4 などの組み込み分野に用いられている CPU でも動作するようになってきているため，組み込み用途への適用も行われるようになってきた．通常の Linux ディストリビューションは，ある目的を持ったファイル群をパッケージという基本単位として管理しており，パッケージ間の依存関係情報を管理したり，セキュリティフィックスやバグフィックスをパッケージの単位で行うことが普通である．

我々は，用途を規定した状況において，Linux のパッケージ単位の管理を維持したまま，最小のシステム構成を自動的に生成する仕組みについて研究を続けている [1]．本稿では，様々なパッケージシステムについて考察し，その上で判明した選択的な依存関係によるさらなるディスク使用量最小化の方策と，目的とする用途に必須となるパッケージの抽出方法の高精度化と高速化について述べる．

2　方針 [1]

最小システム構成に対する要求として以下の 3 つを考える．

- 使用目的は事前に与える．
- パッケージ単位の管理方法は崩さない．
- ディストリビューションのパッケージ構成ポリシーに (極力) 踏み込まずに実現する．

3 改善点

本節では，文献 [1] における手法で浮上した新たな改善点について述べる．

3.1 選択可能なパッケージの存在

現在，ベース OS と使用している debian4.0 のパッケージ管理システムでは，パッケージ間の関係としていくつかのフィールドが定められており，最初に本稿に関係するものを以下に示す．

3.1.1 パッケージ間の関係を表すフィールド [2]

PKG_a *Field* PKG_b

- *Depends:*

  絶対的依存関係を示す．*Depends:*リストに示されたパッケージ (PKG_b) が全てインストールされていない限り PKG_a は正しく構成されないことを示すフィールド．

- *Pre-Depends:*

  PKG_a をインストールするより前に PKG_b のインストールが完了されている必要があることを示すフィールド．

- *Provides:* PKG_a は PKG_b と同等の機能を提供することを示すフィールド．*Depends:*に PKG_b が指定されているとき PKG_a も選択可能であることを示す．

3.1.2 仮想パッケージ [2]

パッケージ管理システムには異なった名称であるが同等の機能を提供するパッケージが存在する．その特定の機能をもつパッケージ群の総称を，その機能を名称にもつ仮想パッケージとする．ゆえにその実体はない．ある特定の機能をもつ PKG_a は，その機能を意味する仮想パッケージ $vPKG$ と以下の関係を持つ．

PKG_a *Provides* $vPKG$

文献 [1] で述べた設計・実装手法では，3.1.1 節におけるフィールド”*Depends*”とフィールド”*Pre-Depends*”に関してのみ考慮していた．フィールド”*Provides*”の存在により，現行の手法ではディスク使用量において真に最小システムを構成できない可能性があることが分かった．

3.2 パッケージ選択によるさらなるシステム縮小化

ここで，パッケージ選択によりさらにシステムを縮小できるという実例を示す．

図 1 は，ベース OS の debian4.0 をディストリビューションのインストーラが提供する”最小構成”でインストールした後，インターネット脅威観測システム WCLSCAN プロジェクト [3] の観測ツール IWR をセットアップした環境に，現行ツール [1] により不要なパッケージを削除した後の全てのパッケージの依存関係を示したものである．

各ノードはパッケージを示す．二重ノードは Essential パッケージと呼ばれる debian システムに必要とされるパッケージ，濃い有色ノードは目的環境のシステム (IWR) の使用したファイルの含まれるパッケージである．無色ノードは目的環境のシステムは使用してはいないが依存関係上必要となるパッケージ，薄い有色ノードは仮想パッケージあるいは古いバージョンのパッケージである．原則として依存関係は上のものが下に依存となっている．また，赤い四角で囲まれたいくつかのノードは選択可能なノードを示し，その都合上システムにインストールされていないパッケージも図に載せており，それらを青い破線ノードで示している．

この図 1 のうち，本稿において重要となる部分を以下に示す．

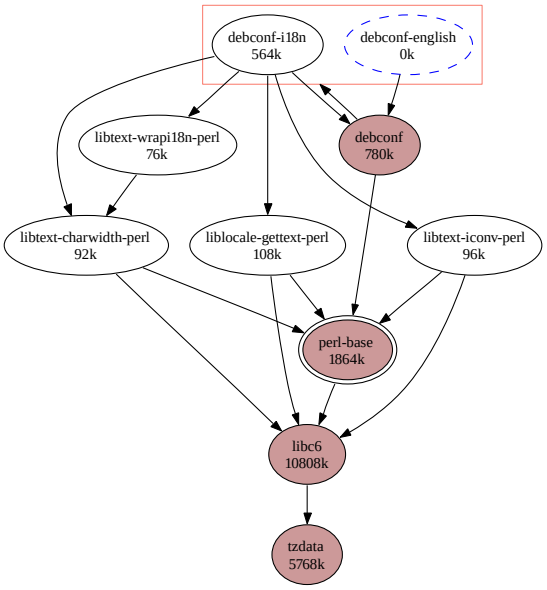


図 2: 選択可能なパッケージ例 1

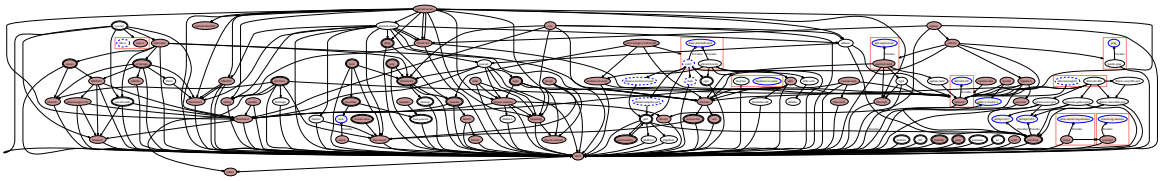


図 1: 現行ツール実施後の全てのパッケージ依存関係

この図 2 は，図 1 の一部で各パッケージのインストールサイズ (Byte) を付加したものである．”perl-base”は Essential パッケージ，”debconf”，”perl-base”，”libc6”，”tzdata”は目的環境のシステムに必要なファイルを含むパッケージを示す．また，”debconf-i18n”と”debconf-english”がいずれか選択可能のあり，現在は”debconf-i18n”がインストールされていることを意味する．ここで，”debconf-i18n”の依存している”libtext-wrapi18n-perl”，”libtext-charwidth-perl”，”liblocale-gettext-perl”，”libtext-iconv-perl”は他のパッケージからは依存されていない．ゆえに，”debconf-i18n”の代わりに”debconf-english”を選択することで，ディスク使用量を 936kByte 削減できることになる．

これに加えて次に仮想パッケージによる選択可能なパッケージの例を示す (図 3)．

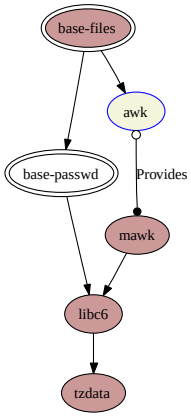


図 3: 選択可能なパッケージ例 2

”awk”は仮想パッケージで，awk という機能を持つパッケージ群の総称である．awk という機能は，図 3 中の”mawk”をはじめ，”gawk”，”original-awk”の 3 つのパッケージが提供している．現在は”mawk”が選択されているが，図 2 のように依存関係如何ではさらにシステムを小さく構築できる可能性を含む．

以上のように選択可能なパッケージを入れ替えることによって，より小さなシステムを構築できる可能性がある．

3.3 必要パッケージ抽出時間の増大の可能性

現行のツールで用いているファイルアクセス記録からシステムに必要となるパッケージを抽出する手法 [1] では，システムに必要なファイルを網羅してアクセスする間，実際にシステムを稼働させる必要がある．現在，このツールの対象システムとしている，観測ツール IWR は，システムに必要なファイル全てをアクセスする時間がたかだか 10 分から 1 時間というものである．しかし，システムに必要なファイルに全てアクセスするのに数ヶ月～数年の時間を要するシステムを対象にする場合，本ツールを用いても，試行錯誤して最小システムを構築するのと変わらない時間がかかる可能性がある．

以上より，現行の手法を改善する必要がある．

4 検討

4.1 選択可能なパッケージ

本稿では，最小システム構成を自動的に生成するツールの実現を目的としている．3.2 節で述べた内容においても自動的に判別･選択できるアルゴリズムを考案する必要がある．そこで本節では，選択可能パッケージの選択アルゴリズム生成を目指し，アルゴリズムのルールを決定する．

1. 目的環境システムに必要なパッケージの依存関係グラフ A を末端まで作成する．
2. 選択パッケージの中に仮想パッケージが含まれる場合，その機能を提供するパッケージ全てを選択パッケージに含める．

3. 選択パッケージのそれぞれにおいて，末端までの依存関係グラフ B を生成する．

4. グラフ A の依存関係を考慮した上で，グラフ B から選択可能パッケージを絞る．

5. 残った選択可能パッケージから，その依存関係も含めたパッケージ群から最もディスク使用量の少ないものを選択する．

ここで問題となるのが，選択パッケージの依存するパッケージ群の中にさらに選択パッケージが存在する場合である．選択パッケージ数の最大値を k，末端までの依存関係の段数を n としたとき，その計算量は，

$$< \text{パッケージ選択アルゴリズムの計算量} >= k^n \quad (1)$$

となり，大規模なシステムになればその計算量は指数関数的に増大することとなる．

ゆえに，このアルゴリズムを少ない計算量で解けるグラフ理論を採用することが必須である．

4.2 必要パッケージ抽出方法

必要パッケージの抽出時間を短時間で安定して行う方法として，現在検討中のものを以下に示す．

1. 現行手法そのものの変更
   - **a** ファイルアクセスのシステムコール監視
   - **b** 目的環境システムに用いるプログラムの静的解析
2. 現行手法でシステムを加速的に稼働
   - **a** システムクロックの定期的更新
   - **b** 仮想マシンにおけるシステムクロックのクロックスピードの変更

まず方法 1.a は，現行のログ収集方法と大差がないと考えられる上，目的環境のシステムを動作させながらシステムコール監視プログラムを走らせる必要があり実施は高負荷となる．

方法 1.b は，プログラムのソースコードを解析したり，そこから呼び出されるプログラムのコードも追跡する必要があり，非常に困難である．また，動的にファイルアクセス先を決定するようなアプリケーションではすべてを網羅することはできない．

次に方法 2.a であるが，目的環境システムの実行周期性を cron プログラムにのみ依存させている場合は有効となる手法と言える．cron の実施間隔に合わせて，システムクロックを断続的に飛ばしていくことにより現行手法よりも早く必要パッケージを抽出できると考えられる．

方法 2.b では，目的環境システムの実行周期性が cron 以外のプログラムに依存していても有効であると考えられる．システムクロックのクロックスピードを増幅させることにより，方法 2.a とは異なり連続的なシステム稼働を実施できると言える．

しかし，方法 2 の二つに共通して考えられる問題が，外部との通信によってなんらかのプログラムが起動する場合には必要なパッケージが得られない可能性が高いと考えられる．

5 まとめと今後の課題

本稿では，パッケージ単位で OS が管理されている Linux ディストリビューションに対して，ある用途に特化した最小システム構成を自動的に生成するために，単純な依存関係と選択的な依存関係を考慮した最小化手法検討した．

また，必須となるパッケージの抽出方法の制度を上げる手法と高速化についても検討した．

今後の課題として，パッケージ選択アルゴリズムの本ツールへの反映，加速実験の検証とその環境への本ツールの適用を目指すことなどが挙げられる．

参考文献

[1] 八木貴之: ”目的環境に対する最小システム構成の自動生成ツールの実現”，第 3 回情報科学ワークショップ (2007)

[2] Debian.org:WEBsite (http://www.debian.org/doc/debian-policy/ch-relationships.html)

[3] WCLSCAN project: (http://www.wclscan.org/)

[4] 八木貴之: ”インターネット脅威検知システムにおける観測ボックスのシステム改良とその評価”， 平成 18 年度京都工芸繊維大学工芸学部電子情報工学科卒業研究 (2007).

様々な仮想計算機環境で共通使用可能なOSイメージの構成方法について

山崎　雅彦†, 桝田　秀夫‡
m-ymzk07@dsm.cis.kit.ac.jp, h-masuda@kit.ac.jp
† 京都工芸繊維大学大学院　工芸科学研究科　情報工学専攻
‡ 京都工芸繊維大学　情報科学センター

概要　計算機ハードウェアの高性能化に伴い，仮想計算機技術が再び注目されてきている．仮想計算機技術を用いることで，複数の計算機を統合でき，それによって各仮想計算機に割り当てるハードウェア資源を柔軟に変更できるため，ハードウェア資源を効率よく利用できる．しかし，一般的には，このような仮想計算機環境上においても，OS インスタンスの数だけ独立した OS イメージを用意し管理する必要があるため，運用管理上の負荷が十分に下がったとは言えない．また，様々な仮想計算機技術が提案され使用可能となってきているが，各技術ごとに機能や性能に得失がある．しかし，仮想計算機技術によって使用できる OS イメージは異なるため，状況の変化に応じて使用する仮想計算機技術を変更したい場合，使用する仮想計算機技術にあわせた OS イメージを毎回構築する必要がある．そこで，本稿では，これらの OS イメージを共通化し，利用環境に応じて容易に仮想計算機技術を変更できる仮想計算機システムを提案する．提案するシステムは単一 OS イメージを実現することにより，異なる仮想計算機技術間におけるサービスのマイグレーションを実現し，柔軟なサービスの運用管理を目指している．

キーワード　仮想計算機（Virtual Machine），union ファイルシステム

1. はじめに

近年，計算機ハードウェアの高性能化に伴い，単一の計算機上で複数の OS インスタンスを稼働させる仮想計算機技術がよく利用されている．仮想計算機技術を用いることにより，物理的に必要な計算機の台数を減らすことができ，運用管理コストを削減できる．また，マイクロプロセッサや I/O 処理における仮想化支援機能 [1, 2, 3, 4] といったハードウェアの対応も進んでおり，仮想化技術の使用によるオーバヘッドも小さくなってきている．それに伴い，それぞれに性能や機能に得失を持つ，様々な仮想計算機技術が提案され使用可能となってきており，サーバファームにおけるサーバの統合だけでなく，テスト環境の構築や共通デスクトップの利用など様々な場面で仮想計算機が利用されている．さらにネットワーク環境の普及によって，グリッドコンピューティングや SaaS（Software as a Service）などのネットワークを介したサービスの提供が注目を集めており，ここでも仮想計算機技術の利用が検討されるなど，仮想計算機技術は幅広く使われつつある．

しかし，仮想計算機技術を用いて物理的な計算機の台数を削減したとしても，論理的な計算機の台数は削減されず仮想計算機自身の運用管理も必要であるため，運用管理上の負荷が十分に下がったとは言えない．例えば，一般的に，仮想計算機環境上においても，OS インスタンスの数だけ独立した OS イメージを用意する必要がある．また，仮想計算機技術によって，使用できる OS イメージは異なるため，使用する仮想計算機技術を変更したい場合，使用する仮想計算機技術にあわせた OS イメージを毎回構築する必要がある．よって，それぞれの OS インスタンスには共通部分が多いにもかかわらず，OS イメージの管理は複雑となってしまう．

そこで，本稿では複数の OS インスタンスにおける OS イメージの共通化，および，仮想計算機技術の変更に対する OS イメージの共通化について検討を行い，様々な仮想計算機技術で共通使用可能な OS イメージの構成方法について提案する．提案するシステムを用いることで，柔軟なサービスの運用管理が可能となる．

2. システムの要件

様々な仮想計算機環境で共通使用可能な OS イメージを構築するにあたり，現状のシステムの問題点について述べる．

2.1 OS イメージの管理の容易化

全ての仮想計算機で同じ仮想計算機技術を用いたとしても，一般的に OS インスタンスごとに OS イメージを作成し，運用管理していく必要がある．しかし，セキュリティパッチの適用などは，どの OS イメージにおいてもほぼ同一の作業であるにもかかわらず OS イメージ毎に行う必要があり，運用管理上の負荷が高い．

2.2 様々な仮想計算機技術での動作

グリッドコンピューティングやクラウドコンピューティングと呼ばれる環境においては，各計算機上で動作させられる仮想計算機技術は様々である．仮想計算機技術において，仮想化の度合いや方法の違いから，必要なドライバや設定ファイルが違うため，OS イメージは仮想計算機技術ごとに異なる．このため，仮想計算機技術を変更すると，OS イメージも構築し直す必要がある．

2.3 マイグレーションの実現

サーバファームのような環境においても，計算機の導入時期の違いなどから物理的な各計算機の環境は均一ではなく，それに伴って異なる仮想計算機技術が用いられている場合がある．この場合，各仮想計算機上で稼働している OS インスタンスの柔軟な配置，移動に制限が生じる．

3. 検討

本節では，まずディスクイメージの単一化の手法について説明し，提案する仮想計算機システムの要素技術について述べる．

3.1 ディスクイメージの単一化

OS イメージの管理の容易化のため，全ての OS インスタンスの基本となる OS イメージを 1 個だけ用意し，各 OS インスタンスはこの基本イメージを元に、差分だけを個別に用意して構成する．この基本イメージを本システムにおける計算機の共通 OS イメージとし，各計算機ごとに用意された個別 OS イメージを union ファイルシステム [5, 6, 7] を用いて重ね合わせることによりディスクイメージの単一化を実現する [8].

3.2 共通 OS イメージ

共通 OS イメージは全ての仮想計算機から参照されるため，各仮想計算機においては読み込み専用でマウントする．しかし，セキュリティパッチの適用など，OS イメージの変更が各仮想計算機の管理外で起こりうる．この時，各仮想計算機上で OS イメージの変更が反映されない可能性があるため，これを回避する必要がある．

本研究では，共通 OS イメージにおいて以下のファイルシステムについて検討する．

- ext3

  Linux の標準ファイルシステムであり扱いが容易である．しかし，単一の OS のみで読み書きすることが想定されているため，本研究の構成では OS イメージの変更が反映されない場合がある．

- NFS（Network File System）

  ネットワークを介したファイル共有のために開発されたファイルシステムであり，複数の計算機から読み書きされることを想定しているため OS イメージの不整合は生じない．ただし，ネットワーク接続が切断されると全く利用できなくなり不安定である．

- GFS（Global File System）

  主にクラスタ環境で用いられ，可用性が高い．複数の計算機で共有するための機構を持ち，OS イメージの不整合が生じないが設定が複雑となる．

3.3 複数の仮想計算機技術で共有できるディスクイメージの構成

一般的には，仮想計算機技術により，仮想化の度合いやカーネルが異なるため，必要となるモジュールやデバイスファイルが異なり，そのままでは，ディスクイメージを共有することができない．一方，カーネル以外のアプリケーションの差はほとんどない．そこで，モジュールは必要なモジュールを全て，共有イメージにあらかじめ用意しておき，デバイスファイルは udev[9] を用いて起動時に自動的に適切なデバイスファイル生成するように設定することで，ディスクイメージの共有を可能とする．

表 1: 利用できるディスクイメージの形式

	ファイル	LVM	パーティション
Xen	○	○	○
VMware	△	○	○
VirtualBox	△	○	○
UML	○	○	○

3.4 マイグレーション機能の実現

本節で説明した単一 OS イメージを用いることで，複数の仮想計算機でディスクイメージを共有することは可能であるが，異なる仮想計算機間でのライブマイグレーションは，メモリ内の情報の違いなどから難しい．

そこで，アプリケーションレベルのマイグレーションの実現を考える．仮想計算機を立ち上げる際に MAC アドレスを指定することが可能であると想定し，OS インスタンスごとに MAC アドレスを指定し，どの仮想計算機で起動しても同一となるように設定する．こうすることで，仮想計算機技術を切り替える際に発生する，サービスの切断を回避することが可能となり，マイグレーション機能が実現できる．

4. 仮想計算機システムの構築

ディスクイメージの作成は基本的に文献 [10] と同様である．現在，OS として Debian 4.0[11] および CentOS 5.2[12] を用い，Xen[13]，VMware Server[14]，VirtualBox[15]，UML（User Mode Linux）[16] の各実装において動作確認を行っている．

4.1 ディスクイメージの認識

現在，共有 OS イメージのファイルシステムとして ext3 と NFS 方式を用いて実装を行っている．共有 OS イメージのファイルシステムに ext3 形式を用いた場合，仮想計算機技術により利用できるディスクイメージの形式が異なる．扱える形式を表 1 に示す．

また，ext3 形式では仮想計算機技術によってディスクイメージの認識が異なるため，差異を隠蔽する機構が必要である．方法としては，udev で全て同じデバイスとみなす方法と，e2label を用いる方法が考えられるが，現在は e2label で実装している．

表 2: ext3 形式のディスクイメージの認識結果

	認識	e2label
Xen	/dev/xvd*, /dev/sd*	○
VMware	/dev/sd*	○
VirtualBox	/dev/sd*	○
UML	/dev/ubd*	×

認識の結果を表 2 に示す．

4.2 ネットワークの設定

各仮想計算機技術において利用されているネットワークデバイスは特に設定を変更しなくても OS からは udev を介して”eth0”として認識される．Debian においては初回起動時の MAC アドレスから変更されると，違うインターフェース”eth1”として認識するように設定されているが，MAC アドレスは OS インスタンスごとに同一のものを使用するため，特に問題は発生しない．

5. 考察

構築する仮想計算機システムについて考慮すべき点を述べる．

5.1 共有 OS イメージ

現時点で共有 OS イメージのファイルシステムとして，ext3，NFS 方式での動作を確認している．ext3 形式を利用した場合，e2label を用い，パーティションごとにラベル付けを行うことで，仮想計算機技術間でのディスクデバイスの認識の違いを隠蔽している．ただし，3.2 節で述べたように，ext3 形式を共有 OS イメージに用いた場合，ファイルの不整合が発生することを確認している．今後は，NFS 方式との性能比較や，GFS，QFS，CXFS といった複数の計算機からの同時アクセスを想定した，SAN 上の共有ファイルシステムの利用の検討が必要である．

5.2 共有 OS イメージ更新時の問題

OS インスタンスが使用するディスクイメージは，union ファイルシステムを用いて重ね合わせている．この方法を用いると，個別 OS イメージ上でファイルが変更された場合，共有 OS イメージの変更が OS インスタンスのディスクイメージに反映されないという問題がある．よって，この差分を管理する仕組みの開発が必要である．

5.3 仮想計算機技術に必要な機能

仮想計算機技術の中には利用するディスクイメージについて性能や安定性の面から，その仮想計算機技術専用のフォーマットを使用する場合も多い．この場合，変換作業無しに単一のディスクイメージを共有することは困難となる．Raw ディスクイメージであれば，”dd”コマンドで簡単に生成でき，仮想計算機技術を用いない場合でも簡単に扱うことが出来るため，仮想計算機技術の実装においては，Raw ディスクイメージを扱えるようにするべきである．本システムにおいても仮想計算機技術の実装に Raw ディスクイメージを扱う機能が含まれていることを想定している．

5.4 マイグレーション機能

サービスレベルのマイグレーションにおいては，稼働中の仮想計算機をシャットダウンし，移動先の仮想計算機環境で再起動させる必要がある．構築した仮想計算機システムでは，全ての仮想化ソフトウェアにおいて，使用する OS イメージは共通化されているので，特別な作業をせずに，仮想計算機技術を切り替えて使用することが可能である．しかし，仮想計算機技術により，切り替えにかかる時間が大きく異なる．特に，物理的に再起動が必要な切り替えでは，オーバヘッドがかなり大きいので，頻繁に切り替えることはできない問題 [10] がある．

Web サービスなど，ある程度サービスが中断することが許容でき，状態をファイルに保存できるサービスであれば，現時点でもマイグレーションが可能であると考えられる．今後の課題としては，サービス停止期間の短いマイグレーション方法の検討や OS インスタンスのライブマイグレーションの実現がある．

6. まとめ

本稿では、様々な仮想計算機環境で共通使用可能な OS イメージの構成方法について検討を行った．現在，union ファイルシステムを用いて統合した OS イメージを作成し，様々な仮想計算機技術を利用することができるシステムを構築している．

今後の課題としては，共有 OS イメージの安全なアップデート手法，サービスマイグレーション機能の適用範囲の検討とその実装，および構築したシステムの性能評価などが挙げられる．

参考文献

[1] Intel Virtualization Technology, http://www.intel.com/technology/virtualization/index.htm.

[2] Intel Virtualization Technology for Directed I/O, http://www.intel.com/technology/itj/2006/v10i3/2-io/1-abstract.htm.

[3] AMD Virtualization, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8826_14287,00.html.

[4] PCI-SIG I/O Virtualization (IOV), http://www.pcisig.com/specifications/iov/.

[5] A Stackable Unification File System, http://www.am-utils.org/project-unionfs.html.

[6] Aufs – Another Unionfs，http://aufs.sf.net.

[7] FunionFS，http://funionfs.apiou.org/.

[8] 桝田 秀夫, 齊藤 明紀:「Xen と unionfs を用いたサーバファーム運用の可能性の検討」, 情報処理学会 DSM 研究会, 2006-DSM-41, pp.85-89, May 11-12, 2006.

[9] udev, http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html.

[10] 山崎 雅彦, 桝田 秀夫:「様々な仮想計算機技術を選択可能な仮想計算機システム構成」第 3 回情報科学ワークショップ, pp. 131—136, 2007.

[11] Debain, http://www.debian.org/.

[12] CentOS, http://www.centos.org/.

[13] Xen, http://www.cl.cam.ac.uk/research/srg/netos/xen/.

[14] VMware, http://www.vmware.com/.

[15] VirtualBox, http://www.virtualbox.org/.

[16] User-mode Linux Kernel, http://user-mode-linux.sourceforge.net/.

誤った転送設定によるエラーメールを削減する転送メールゲートウェイシステムについて

石橋　由子 †, 桝田　秀夫 ‡
y-isbs08@dsm.cis.kit.ac.jp, h-masuda@kit.ac.jp
† 京都工芸繊維大学大学院　工芸科学研究科　情報工学専攻
‡ 京都工芸繊維大学　情報科学センター

概要 電子メールはインターネット上において手軽で最も利用されているコミュニケーションツールである。近年，ひとりで複数のメールアドレスを持ち，その間でメールを転送しているケースも多い。携帯電話のアドレスのように頻繁に変更されやすいアドレスを転送先に指定している場合，アドレスを変更した際に転送先メールアドレスの設定変更を忘れるケースが多い。この場合，転送できないため，元の送信者のもとにエラーメールが届けられる。ところが，エラーメールは転送設定を行った本人の元には届かないため，本人はエラーとなっていることに気がつきにくい。さらに，エラーメールには元のメールにおいて宛先に指定したアドレスが書かれていないため，誰宛のメールが転送不能になったのかを判断することが難しい。そこで，本稿では転送するメールのエラー時の戻り先を転送を行っているメールサーバに変更し，以降のエラーメールを抑制するシステムを提案する。転送エラー時のメールを制御するために，メールエンベロープ From の送信者アドレスの書き換えを行う。また，転送できなかったメールや転送後エラーとなり返送されてきたメールを管理するデータベースを用意して，エラー状態を把握できるような構成とした。

キーワード　電子メール，転送エラーメール

1　はじめに

インターネット上において，電子メールは手軽で最も利用されているコミュニケーションツールである。近年，企業や個人で利用できるプロバイダのメールサービスに加えて，無料で使用可能なフリーメールサービスや，携帯電話でも電子メールの送受信が可能となり，その間でメールの転送も頻繁に行われている。

しかし，フリーメールや携帯電話のメールアドレスは安易に変更する傾向にあるため，メールアドレスを変更した際に，転送先に指定したアドレスの変更を忘れると，転送したメールがエラーとなって元の送信者に返送される。エラーメールは転送設定を行った本人には届かないため，エラーとなっていることに気が付きにくく，無効なアドレスへの転送が止まらないまま放置される。

さらに，エラーメールには，転送先のアドレスとそれが不明であった旨が記されているが，あて先に指定したメールアドレスは明記されていない。このため，あて先に複数のメールアドレスを指定した場合やメーリングリストあてのメールの場合，誰あてのメールが不着となったのかを判断するのは非常に難しい。

本稿では誤った転送設定によるエラーメールを抑制するシステムを提案する。メールを転送する際に，転送したメールがエラーとなった場合に返送されてくるアドレスを，送信者のメールアドレスを転送を行っているマシンが受信できる形に書き換えることができれば，転送エラーが発生した際にエラーメールを受信してエラー状態を把握することができる。これにより，転送エラーが発生した際にはそれを検知してそれ以降の転送を抑制するなどの柔軟な制御が可能となる。

2　電子メール転送の現状と問題点

2.1　電子メール送信時の手順

電子メールを送信するときの手順は SMTP (Simple Mail Transfer Protocol，簡易メール転送プロトコル) と呼ばれる規約で定められており，

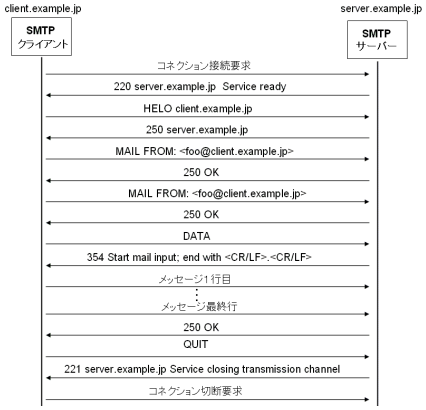


図 1: SMTP コマンドと応答コードによるメール送信手順

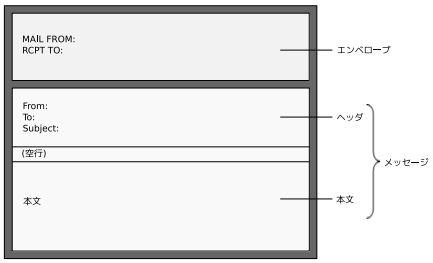


図 2: メッセージフォーマット

RFC2821[1] として定義されている。メールを送信する場合の送信側がクライアント，送られてきたメールを受ける側がサーバとなる。

図 1 に，1 通のメールを送信する際にサーバとクライアントの間でやり取りされる SMTP コマンドと応答コードの例をあげる。

SMTP コネクション確立後，クライアントは HELO コマンドに自ドメインを指定する。次に MAIL FROM コマンドで送信元のメールアドレスを指定し，続いて RCPT TO コマンドで宛先のメールアドレスを指定する。DATA コマンド送信後メッセージを送信し，終了時にピリオドだけの行を送信する。最後に QUIT コマンドでコネクションを切断する。サーバはクライアントから SMTP コマンドを受信するたびに，応答コードを返す。

2.2 メッセージフォーマット

送受信するメールはテキストファイルとして扱われ，RFC2822[2] に規定されている。メッセージフォーマットを図 2 に示す。

エンベロープ 送信する宛先の情報と送信元の情報が MAIL FROM と RCPT TO に設定される。

ヘッダ 宛先や件名，メールサーバが追加する経路情報などから構成されている。

本文 記述したメールの内容が本文に書き込まれる。

2.3 メールアドレスの形式

メールアドレスの形式は，`xxx@yyy.example.jp` となっている。RFC2822 では，`@` の左側をローカルパート，`@` の右側をドメインと呼んでいる。ドメインは受信側のメールサーバを特定する名前が指定され，ローカルパートはメールボックスの名前が指定される。

2.4 エラーメールが発生する条件

図 1 に示す通り，送られてきたコマンドに対してサーバがレスポンスを返す際には数字 3 桁の応答コードを付加する。応答コードは，1 桁目の値により意味が異なる。送信したメールに一時的なエラーが発生した場合は応答コードの 1 桁目が ‘‘4’’ となり，恒久的なエラーの場合は ‘‘5’’ となる。

一時的なエラーの原因は，「受信側メールサーバが停止していた」，「受信者のメールボックスに空き容量がなかった」等が考えられる。この場合，‘‘Return-Path:’’ に指定されたアドレスに送信できなかったことを告げる警告メールを送られ，その後送信側メールサーバが定期的に再送信を試みる。指定された期間内に再送信できなかった場合は，再度 ‘‘Return-Path:’’ に指定されたアドレスにエラーメールを送る。

一方，恒久的なエラーの原因は，「宛先アドレスのドメインに相当するマシンが存在しない」，「宛先アドレスのローカルパートが受信側メールサーバに存在しない」場合が多い。この場合もメールヘッダの ‘‘Return-Path:’’ に指定された送信者にエラーメールを送る。

2.5 メール転送時の問題点

転送先メールアドレスが有効な場合と，無効な場合のメール転送の流れを図 3 および図 4 に示す。

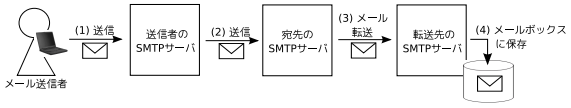
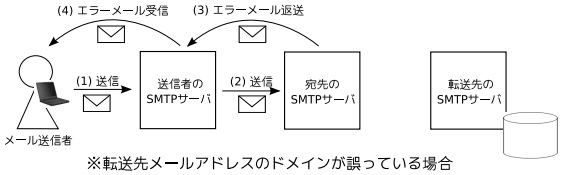


図 3: メール転送先アドレスが有効な場合



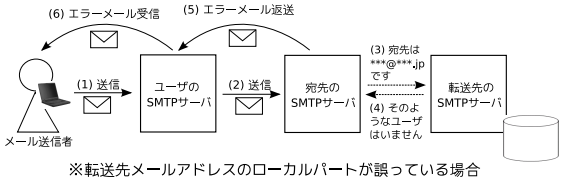


図 4: メール転送先アドレスが無効な場合

転送先メールアドレスのローカルパートが誤っている場合，エラーメールの内容は‘‘User Unknown : **転送先のアドレス**’’となっている。‘‘**転送先アドレス**’’は元の送信者が宛先に指定したアドレスではないため，エラーメールの受信者は何のエラーであるか判断できないことが多い。さらに，宛先に複数のメールアドレスを指定した場合は，誰あてのメールが転送不能となったのかすら不明である。また，転送の設定をした本人にはエラーが通知されないので，エラーが発生していることに気がつきにくいため放置される。

以上の問題点をまとめると次のようになる。

- エラーメールが送信者にのみ返送される
- 送信者は，エラーメールを受信しても誰あてのメールが不着となったのか判断がつきにくい
- 転送先がエラーとなっていることに転送を設定した本人が気がつきにくいため，エラーの状態のまま放置される

3　提案システムの設計

2.5 節であげた問題点を解決するために，次のシステムを提案する。

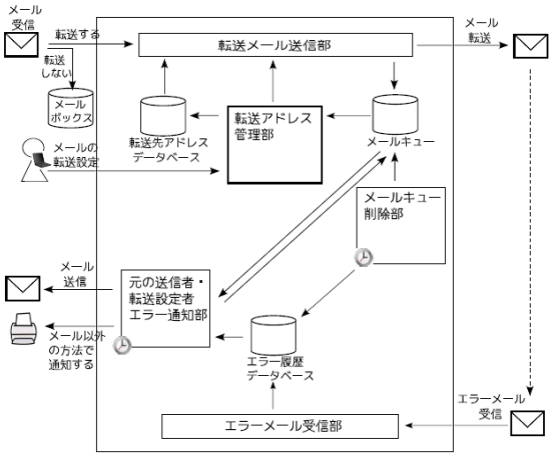


図 5: システム概要図

表 1: 転送先アドレスデータベース

カラム名	説　明
利用者メールアドレス	転送設定する利用者のメールアドレス
転送先メールアドレス 1	転送先のメールアドレス 1
:	:

3.1　システムの概要

システムの概要を図 5 に，本システムで使用するデータベースの情報を表 1 表 2 表 3 に示す。

本システムは，転送メール送信部，エラーメール受信部，転送アドレス管理部，メールキュー削除部，元の送信者・転送設定者エラー通知部の 5 つから構成されている。それぞれの処理手順は 3.2 節から 3.6 節のとおりである。

3.2　転送メール送信部の処理手順

1．転送すべきメールが届く（このアクションがトリガーとなる）

2．メールエンベロープ From のアドレスを転送を行っているマシンのメールアドレスに変更し，転送先アドレスデータベースで指定されたアドレス宛に転送する

3．メールキューデータベースに転送したメールに関する情報を書き込む

3.3　エラーメール受信部

1．転送エラーとなったメールが届く（このアクションがトリガーとなる）

表 2: メールキューデータベース

カラム名	説　明
利用者メールアドレス	利用者のメールアドレス
転送先メールアドレス	転送先メールアドレス
元メールの MAIL FROM	元メールの MAIL FROM のアドレス
元メールの受信日時	元メールを受信した日時
キュー ID	転送したメールに付与した ID

表 3: エラー履歴データベース

カラム名	説　明
利用者メールアドレス	利用者のメールアドレス
転送先メールアドレス	転送先メールアドレス
受信日時	エラーメールを受信した日時
キュー ID	転送設定する利用者のメールアドレス

2．転送エラーとなったメールから，転送設定をした利用者のメールアドレス，転送先メールアドレスを取り出し，エラー履歴データベースに書き込む。

3.4　メールキュー削除部

1．この処理は一定時間ごとに起動される

2．メール転送後一定期間（1 日程度）経過したメールは転送先に届いたとみなし，メールキューデータベースから削除する。

3.5　元の送信者･転送設定者エラー通知部

1．この処理は一定時間ごとに起動される。

2．エラー履歴データベースより，転送エラーとなったメールが届いているか確認する。

3．転送エラーとなったメールが届いている場合，

全ての転送先に送信できなかったならば 送信できなかったことを元メールの送信者に通知する

どこかの転送先に送信できたならば 転送できたアドレス宛に転送エラーが発生していることを通知する

4．エラー履歴データベースより長期間転送エラーが発生していることがわかれば，転送設定した利用者に対してメールやメール以外の手段（API など）で通知する

3.6　転送アドレス管理部

1．利用者が転送先メールアドレスを設定する（このアクションがトリガーとなる）

2．転送先アドレスデータベースに書き込む。

3．転送先アドレスを削除した場合には，メールキューデータベースおよびエラー履歴データベースに削除した転送先に関するデータがあれば削除する

4　提案システムの実装

メール転送時に MAIL FROM を書き換えるため，procmail[3] を使う方法，sendmail の userdb[4] を使う方法，mailman[5] を使う方法，delegate[6] を使う方法の 4 種類について実装を行った。

4.1　procmail を用いた方法

procmail は，受信したメールをレシピと呼ばれるルールにより振り分けやフィルタリングを行うソフトウェアである。

procmail でメールを転送するためには，$HOME/.forward ファイル（$HOME はホームディレクトリを表す）を procmail を使うよう設定した上で$HOME/.procmailrc に procmail に関するルールを記述する。

```
:0
| $SENDMAIL -oi -f fwd@test2.dsm.cis.kit.jp
 u3@test3.dsm.cis.kit.jp
```

(2 行目と 3 行目は実際は 1 行で記述)

この例では，メールエンベロープ From を fwd@test2.dsm.cis.ki.jp に書き換えて，u3@test3.dsm.cis.kit.jp に転送している。複数の転送先がある場合は，カンマで区切ってメールアドレスを列挙すればよい。

4.2　sendmail の userdb を用いた方法

sendmail の userdb は，発信者アドレスや受信者アドレスをデータベースファイルに従って書き換えを行うための B 木タイプのデータベース

である。デフォルトでは /etc/mail/userdb に設定を記述し makemap コマンドでデータベース /etc/mail/userdb.db ファイルを生成する。

userdb はキーと値のペアで構成されている。キーはユーザ名にコロンと maildrop または mailname のいずれかのキーワードをつづけたものである。

```
u1:mailname u1-fwd@test2.dsm.cis.kit.jp
u1-fwd:maildrop u1
```

userdb に上のように設定しておくと，u1 がメールを送信する際に u1-fwd@test2.dsm.cis.kit.jp がメールエンベロープ From に置き換わる。

転送先アドレスを .forward に記述してしまうと userdb を参照しなくなるため，procmail を使って転送先アドレスを記述する。.procmailrc に次のように設定しておくと u3@test3.dsm.cis.kit.jp に転送される。

```
:0
| $SENDMAIL -oi u3@test3.dsm.cis.kit.jp
```

4.3 mailman を用いた方法

mailman はメーリングリストを管理するソフトである。

メーリングリストの管理者のアドレスがメールエンベロープ From の置き換わる。メーリングリストのメンバーを転送したいアドレスに指定する。

4.4 delegate を用いた方法

delegate は，種々の通信を中継・キャッシュするプロキシサーバソフトである。HTTP，FTP，POP3，SMTP，telnet，SSL など多くのプロトコルに対応している。

すでに起動しているメール受信のためのデーモンを停止し delegated -P25 SERVER=smtp コマンドを常時起動しておく。これにより受信するメッセージを delegated が受け取ることができる。

delegated の設定ファイルは，ユーザ名が u2 である利用者を例にあげるとデフォルトでは /etc/smtpgate/users/u2/conf に配置される。

```
INHERIT: sendmail
SERVER-HOST: test2.dsm.cis.kit.jp
SERVER-PORT: 25
CONTROL/RECIPIENT: u3@test3.dsm.cis.kit.jp
CONTROL/SENDER: fwd@test2.dsm.cis.kit.jp
```

設定ファイルをこのように定義すると，u2@test2.dsm.cis.kit.jp 宛のメールは，メールエンベロープ From を fwd@test2.dsm.cis.kit.jp に書き換え，u3@test3.dsm.cis.kit.jp に転送される。

5 評価と考察

5.1 転送遅延時間

測定方法 4章で述べた MAIL FROM を書き換える 4 つの方法に対して，メールの転送時間を測定した。比較のために .forward を使った MAIL FROM を書き換えない従来の転送方法についても測定を行った。

図 6 のテスト環境のもとで，本文なしの 100 通のメールをホスト 1 からホスト 2 に連続して送信し，ホスト 2 からホスト 3 に転送を行った。このとき，ホスト 2 で最初のメールを受信してから最後のメールの転送が終了するまでの時間を，メールのログファイル (/var/log/maillog) から計算し，これを転送にかかる時間とした。

測定結果 測定結果を表 4 に示す。procmail は特に問題がなかった。sendmail の userdb および mailman は 1/3 程度のメールを受信したところで，一旦送信待ちのキュー (/var/spool/mqueue) に保存された後，順次転送が行われた。また表 4 中の備考に記述した通り，転送中に CPU のロードアベレージが 13 および 27 と非常に高い値を示した。また，sendmail の userdb を使用する方法は，userdb を検索する以外は procmail を使う方法と同じであるが，転送時間は procmail の 4 倍も要している。delegate を使う方法は，転送設定するユーザごとに異なるディレクトリにある設定ファイルを読み込む必要があるので，多くの利用者が転送設定をするようになると転送時間が増大することが予想される。

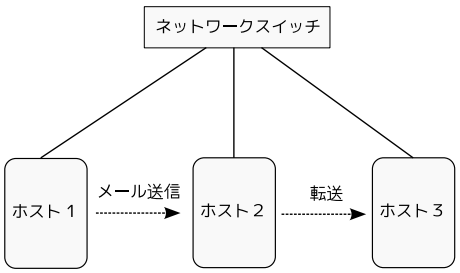


図 6: 転送時間を測定した環境

表 4: 転送時間

転送方法	転送時間	備 考
.forward	22 秒	MAIL FROM の書き換えを行わなかった
procmail	39 秒	
sendmail userdb	163 秒	45 通目受信時にロードアベレージが 13 になった
mailman	196 秒	37 通目受信時にロードアベレージが 27 になった
delegate	34 秒	

5.2 エラーメールの戻り先アドレス

返送されてきたエラーメールより，誰のどの転送先がエラーであったのかを判別する必要がある。そこで次の 2 つが候補として考えられる。

転送先アドレスごとに異なるもの ユーザ名と転送先アドレスを連結したものを，エラーメールの戻り先アドレスのローカルパートとしたいが，RFC2821 にてメールのローカルパートは 64 文字以内と規定されている。そこで ‘‘ユーザ名_転送先アドレスをハッシュしたもの’’ とする。64 文字までまだ余裕があるので，可読性を上げるために転送先アドレスの先頭数文字を埋めておくか，送信時にメールキューデータベースに保存する際のキュー ID を指定してもよい。

システムで共通した 1 つのもの エラーメールの戻り先アドレスを共通の 1 つのアドレスとする。誰のどの転送先がエラーになったのかを判別するために，メール転送時ににメールヘッダにユーザ名と転送先アドレスの情報を付加する。例えば ‘‘X-EFMS-TRANSFER: ユーザ名_転送先アドレス’’ を付加する。

5.3 転送設定の即時性

転送先メールアドレスの追加，削除，変更は随時発生する。4 章であげた 4 つの方法が即座に対応できるか考察してみると，sendmail の userdb を使う方法は，全てのユーザが同一ファイルを更新するため，何らかのファイルのロック処理が必要であるが，この点をのぞけば大きな遅延もなくほぼ即座に対応できる。

5.4 導入の容易性

実際に運用する際に，4 章であげた 4 つの方法について，導入の容易性について考察してみると，mailman は若干時間がかかると予想されるが，この点以外は一時的なサービス停止を伴うが，おおむね容易に導入できる。

6 まとめ

本稿では，誤った転送設定によるエラーメールを削減することを目的として，メールエンベロープ From を書き換えることでエラーメールの返送先を制御し，エラー状態を把握できる手法を提案した。今後の課題としては，提案したシステムのうち実装できていない部分の構築作業を進めていきたい。

参考文献

[1] J. Klensin (ed.): “Simple Mail Transfer Protocol”, RFC2821, IETF (2001)

[2] P. Resnick (ed.): “Internet Message Format”, RFC2822, IETF (2001)

[3] Geoff Mulligan: “SPAM の撃退”, ピアソンエデュケーション, (1999)

[4] Bryan Costales, Eric Allman: “sendmail”, vol.2, O’Reilly, 第 3 版 (2004)

[5] Free Software Foundation: “Mailman, the GNU Mailing List Manager”, http://www.gnu.org/software/mailman/index.html, (参照 2008-09-01)

[6] Yutaka Sato: “DeleGate Home Page”, http://www.delegate.org/delegate/, (参照 2008-09-01)

高速二次記憶によるネットワークブートサーバの高性能化について

宅間　広大[†],　桝田　秀夫[‡].
h-takm08@dsm.cis.kit.ac.jp,h-masuda@kit.ac.jp
† 京都工芸繊維大学大学院　工芸科学研究科　情報工学専攻
‡ 京都工芸繊維大学　情報科学センター

概要　運用管理のコスト削減や情報漏洩対策の一つとして，企業や教育機関でネットワークブートを用いた情報端末の運用が増えてきている．ネットワークブートとは，クライアント PC が OS やデータをネットワーク経由で取得して稼働する技術であり，通常，サーバが複数のクライアントに対する処理を行う．そのため，サーバの性能，とくにネットワーク性能とディスク性能がボトルネックとなると考えられる．このうちディスクの実行性能を向上させる手段として，本稿ではメモリをベースとした製品である GIGABYTE 社の i-RAM を使用し，i-RAM 上に OS イメージを配置する方法を提案する．そのもとで，ハードディスク上に配置した場合と比較して，サーバ性能がどのように変化するかを，クライアント PC の起動にかかる時間を指標として評価した．

キーワード　Linux，ネットワークブート, 高速二次記憶, 高性能化

1.　はじめに

近年，企業や教育機関において，多数の PC を使うようになってきている．多くの PC が利用されるようになると，運用管理にコストがかかる．また最近では，企業などでは情報の漏洩が問題となっている．これらの問題を解決するものとして，ネットワークブートが使われるようになってきている．例えば，東京大学情報基盤センターでは Apple 社の Netboot[1] が，その他にミントウェーブ社の Windows VID[2] や，Ardence 社の Ardence[3] がよく利用されている．

このようなネットワークブートシステムでは，クライアント PC はブートサーバのディスク上にある OS イメージを使いブートする．この OS イメージを共通化することで，多数の計算機を管理するコストの削減につながる．しかし，通常多数のクライアントに対してサーバが処理を行うため，ネットワークブートシステム全体の性能は，ブートサーバの性能がそのまま影響する．システムの性能の向上には，サーバの性能を向上させる必要がある．このサーバに対してかかる負荷は，ネットワークカード (NIC) およびディスクが挙げられる．

本稿では，ディスク性能に着目し，ランダムアクセス性能など，実行性能の高いディスクを使用することによる性能向上の効果について述べる．DDR メモリをハードディスクと同じように扱う GIGABYTE 社の i-RAM[4] を用い，この i-RAM 上に OS イメージを配置することによって，ネットワークブートサーバの性能を向上させることを提案する．

2. 節でネットワークブートシステムの構成について説明し，3. 節で性能向上法として本稿において提案する手法を記述する．そして，4. 節で評価および 5. 節で考察を述べる．

2.　ネットワークブート

本節ではまずネットワークブートの構成に関する要素技術について述べ，次に，今回構成したネットワークブートの構成について述べる．

2.1　ネットワークブートサーバ

ネットワークブートとは，サーバのディスク上に置かれている OS イメージを，NFS などを用いて

ネットワーク経由で取得し，一般的にクライアント PC のディスクを使わずにブートさせる技術である．このため，クライアント PC のローカルディスクを使わずに起動する，PXE という仕組みを用いた．また，本稿ではクライアント PC に PXE 対応の NIC を装備した PC を使用するものとした．

また，PXE を使用するブートローダとして，syslinux の一部である PXELinux を使用した．PXE は特殊な応答を返す DHCP サーバと TFTP サーバを必要とする．

PXE Linux: PXELinux は PXE ブートのブートローダである．

DHCP サーバ: クライアント PC の IP アドレスの他，TFTP サーバの IP アドレスや PXELinux のファイル名や位置，kernel の位置などの情報を提供する．

TFTP サーバ: PXELinux やクライアント OS の kernel を提供するファイルサーバ．

2.2 ブートサーバの構成

本稿において構成したネットワークブートサーバについて述べる．ネットワークブートサーバのスペックを表 1 に示す．

ブートサーバの OS として，CentOS を用いた．使用したバージョンは本稿執筆現在最新版である CentOS 5.2 である．ブートサーバとして機能させるために必要なソフトウェア群として，次に示すものを用いた．

DHCP サーバ: 3.0.5-13.el5

TFTP サーバ: 0.42-3.1.el5.centos

NFS サーバ: NFS version3

表 1: ブートサーバのスペック

CPU	AMD Athlon64 X2 Dual Core 1GHz
メモリ	DDR2 1GB
NIC	Marvell Technology Gigabit Ethernet
DISK	Seagate ST3250620AS or i-RAM(SATA150)

2.3 OS イメージの構成

次に，クライアントであるが，OS として Vine Linux4.2 を利用し，文献 [7] を参考に OS イメージを構成した．

本稿では，NFSRoot という機能を用いて，ルートパーティションを NFS でマウントする方式を採用した．また，本稿に於いて必要のない機能として，次に挙げたサービスを OS 起動時に上がらないように設定した．

Kudzu: 新しいハードウェアがあるかどうかチェックするソフトウェア．

Cups: Unix/Linux の標準的な印刷システム．

iptables: iptables 設定時に一時的にネットワークが途絶するため，この設定を無効にした．

2.3.1 OS イメージの共通化

Linux のルートパーティションには，すべての PC において共通で利用でき，読み込みができれば良い部分という部分がかなりある．しかし，/etc や/var については，起動時に設定を書き込んだり，ログを記録したりするため，書き込みが可能である必要がある．

そこで，スタッカブルファイルシステムの一種である unionfs[6] を用いた．unionfs により，Read Only で export したルートパーティションに，Read/Write で export した別のディレクトリを重ね合わせることで，/etc や/var などのホスト毎に異なる設定を保持できるようになる．

3. ブートサーバの高性能化

本節ではネットワークブートサーバの性能向上法について述べる．

ネットワークブートシステムでは，図 1 に示したように，クライアント PC は OS イメージなどのデータをネットワーク越しに取得し起動する．その性質上，ブートサーバの性能がネットワークブートシステム全体の性能にそのまま影響を与える．ネットワークブートシステム全体の性能向上のためには，ブートサーバの性能を向上させる必要があると言える．

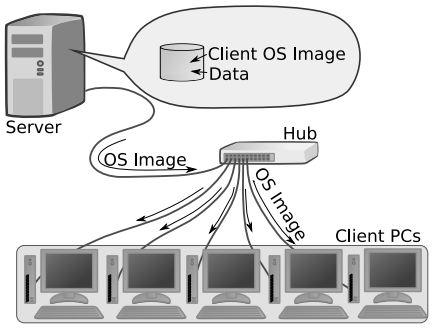


図 1: ネットワークブートシステム概要図

3.1 ボトルネックの解析

クライアント PC の起動は DHCP サーバへの要求から始まる．その後，TFTP サーバから Kernel を取得し，クライアント OS のブートプロセスが始まる．次に，NFS でサーバ上の OS イメージをマウントし，init スクリプト群を実行していく．

このスクリプト群やそれによって起動されるデーモンはすべて NFS で取得される．したがって，多数の PC が同時に起動すると，図 1 からわかるように，サーバの NIC への集中とディスクアクセスが問題になると考えられる．例えば，1000baseT の NIC を使用したとすると，およそ 100MB/s 程度の速度でデータの通信が行われる．また，SATA150 のハードディスクの理論上限値はおよそ 150MB/s である．しかし，ハードディスクの物理的な構造上，ランダムアクセスに対してそこまでの性能が出ることはない．

3.2 ディスク性能の向上方法

前節で述べたように，ディスクの I/O 性能がボトルネックとなる．そこで，本稿では GIGABYTE 社の i-RAM という製品を用いることによって，ディスクの実行性能の向上，ひいてはネットワークブートサーバの性能向上させることを提案する．

i-RAM は通常メインメモリとして使用されている DDR メモリを，あたかもハードディスクのように見せかける製品である．この製品も SATA 規格で PC に接続される．文献 [4] によれば，同製品は多くのアクセスパターンに対して SATA 規格の理論値である 150MBytes/sec に近い性能を発揮する．

4. 測定結果

本節では起動時間，ネットワーク負荷，およびディスク I/O の評価方法について述べ，今回計測した結果を示す．

4.1 クライアント PC の起動時間

今回の計測では，WOL という機能を用い，ほぼ同時にクライアント PC を立ち上げた．開始時刻はマジックパケットを送り始めた時刻，完了時刻として，自動ログイン後ブラウザが自動起動した時刻を用いた．

測定結果をまとめたグラフを図 2 に示す．

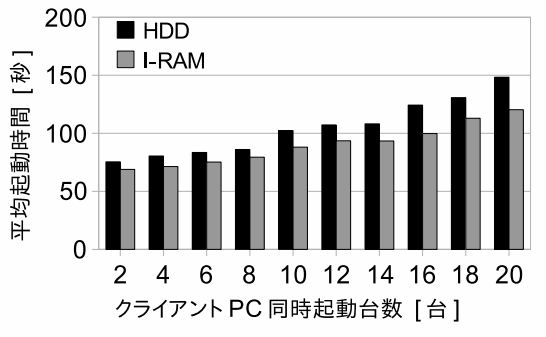


図 2: 起動時間の測定結果

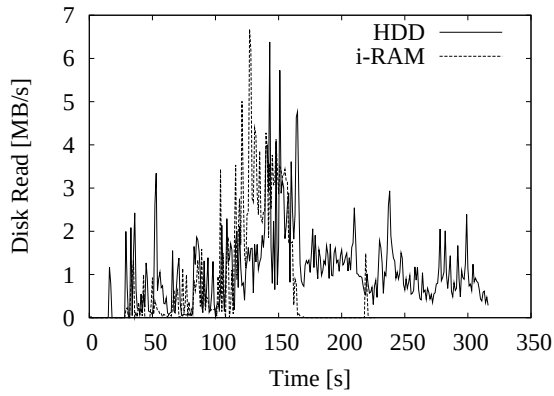


図 3: ディスク読み込みの測定結果

4.2 ディスク I/O

/proc/diskstats を一秒毎に収集し，１秒あたりの読み込み／書き込み速度を測定した

図 3 および図 4 に，クライアントの同時起動台数が 20 台の時に得られたデータを示す．

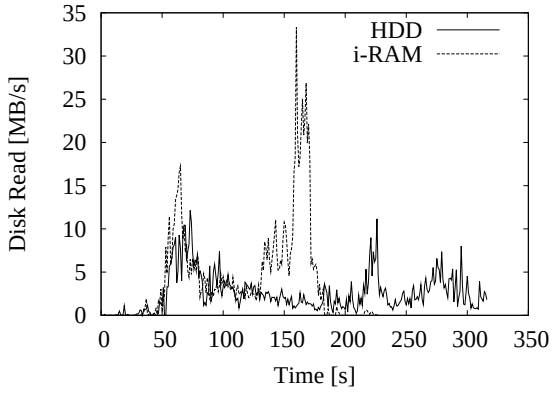


図 4: ディスク書き込みの測定結果

5. 考察

5.1 クライアント PC の起動時間

図 2 より，HDD を用いる場合と比較し，i-RAM を用いた場合には 8〜20%の短縮が見られた．10%程度の短縮は 10 台未満の場合について見られ，10 台以上の場合には 14%〜20%の短縮が見られた．

5.2 ディスク I/O

図 3，4 を見ると，HDD は双方とも大量のアクセスがあったその後も読み書きを続けている．これは，i-RAM は要求があるとすぐに対応できるのに対し，HDD はすぐには対応できていないためと考えられる．しかし，今回の測定ではサーバ OS と同じパーティションに OS イメージを配置したため，ディスクの Read に関して影響があると考えられる．

6. まとめ

本稿では，ネットワークブートシステムについて，サーバのハードディスク性能の向上による性能向上の効果について評価した．ハードディスクとして，通常のハードディスクとは異なり，ランダムアクセス性能などに優れた i-RAM を用いることで，クライアントの起動時間が 8〜20%削減され，効果があることが実証できた．

今後の課題として，近年入手が容易になってきた SSD(Solid State Disk) を用いた場合の性能評価や，大容量の HDD と高速な (小容量) メモリディスクを効果的に組み合わせる場合のパラメータについての評価が考えられる．

謝辞

本研究は一部，日本学術振興会・科学研究費補助金・基盤研究 (C)（課題番号 19500069）の研究助成による．

参考文献

[1] Netboot; Apple 社:
http://www.apple.com/jp/server/macosx/netboot.html

[2] Windows VID; ミントウェーブ社:
http://www.mintwave.co.jp/product/vidsystem.html

[3] Ardence; Ardence:
http://www.thinclient-net.com/ardence/about/index.html

[4] GIGABYTE GC-RAMDISK i-RAM; GIGABYTE 社:
http://www.gigabyte.co.jp/Products/Storage/Products_Overview.aspx?ProductID=2180&ProductNave=GC-RAMDISK

[5] GIGABYTE GC-RAMDISK i-RAM Rev1.3 Manual; GIGABYTE 社:
http://asia.giga-byte.com/FileList/Manual/desktop_manual_i-ram_1.3_j.pdf

[6] unionfs; project-unionfs:
http://www.fsl.cs.sunysb.edu/project-unionfs.html

[7] ”unionfs を用いたディスクレスシステムの実装とその評価”; 桝田秀夫, 齋藤明紀：2005.12, 分散システム/ インターネット運用技術シンポジウム 2005 年度 論文集

仮想環境上でのサーバ設定演習における安全な管理者権限の提供方法について

小西　泰平 †,　桝田　秀夫 ‡.
t-kons08@dsm.cis.kit.ac.jp, h-masuda@kit.ac.jp
† 京都工芸繊維大学大学院　工芸科学研究科　情報工学専攻
‡ 京都工芸繊維大学　情報科学センター

概要　プログラミングなどに続くより高度な演習として、UNIX 系 OS のサーバ設定を学ばせたいという要求がある。我々の研究室では、ディスクレスの環境でサーバ設定演習を行う仕組みについて研究を進めている。ディスクレスの環境を用いることで、パソコン演習室を使用して演習用の独自環境を提供することが比較的容易に実現できる。また、学習者が演習で追加・更新したファイルがサーバ側に集約化されるため学生の進行状況の把握がしやすくなっている。しかし、ファイルの追加・変更だけでは、サーバプロセスが起動しているかどうかや、どのようなオプションで起動しているのか、といった状況は把握が難しい。本稿では、仮想環境や SELinux を用いて、学習者から影響をうけないように工夫した状況把握モジュールを導入することにより、学習者の学習状況をより詳細に把握できる仕組みの検討を行った。

キーワード　仮想計算機　SELinux 学習状況把握

1　はじめに

大学等のコンピュータリテラシ教育では,「パソコンの使い方」を指導する段階から，より高度な活用方法や情報科学の基礎を教育する段階へ移行しつつある。意欲のある学生に対する次のステップの選択肢の１つとして,「OS の設定やサーバ構築」を体験させれば，コンピュータの動作原理をより深く学ばせることが可能になり，トラブル対応能力も養成できることが経験的に知られている。また、演習において生徒の進行状況を教師の側から取得できるようにしておけば、より的確な指導が可能になる。

そこで、ディスクレスによるサーバ設定演習を想定し、その中で、学生の学習状況を学習者からの影響を受けないようなモジュールを使用することで把握できるようなシステムを設計したい。しかし、サーバ設定演習では、学習者が管理者特権を持つことになるので、管理者特権を持った学生が、自分のネットワーク設定や進行状況のログ取得設定などを書き換えてしまうことが考えられ、その対策を考える必要がある。また、サーバ設定演習を、いろいろな設定がされている PC で行うのは、想定外のエラーが発生してしまう恐れがある。そこで、サーバ設定演習を、仮想化した計算機上の OS で行い、物理的な制約を排除した環境にし、管理者特権もその上で渡すことにする。さらに、仮想計算機上で動いている OS のサーバ設定演習に必要なログを取得するためのアプローチを考える必要がある。

本稿では、学習者用の OS 環境上の進行状況把握モジュールを安全に提供する仕組みを考える。これを実現するために、仮想化技術を用いたアプローチと SELinux を用いたアプローチを検討し、実現可能性について検討を行う。

2　要求

本研究で、実現したい要求は以下の 2 つである。

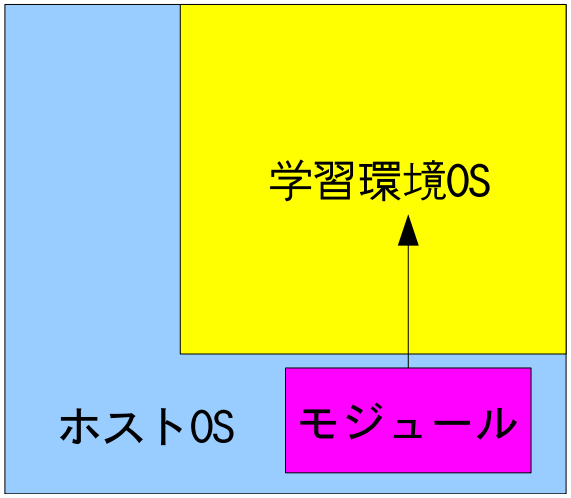


図 1: ゲスト OS 型: 外側からのアプローチ

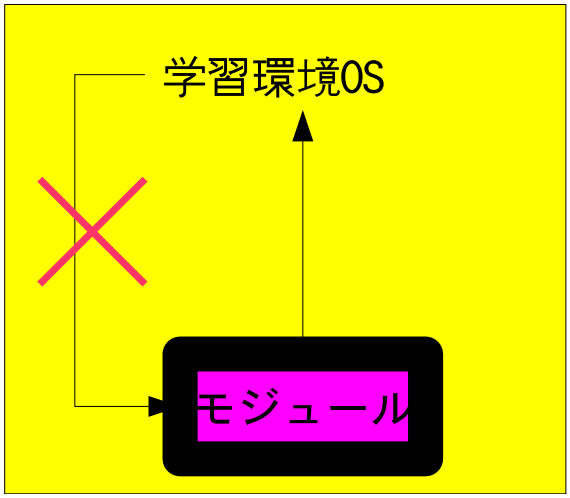


図 2: 権限分離型: 内側からのアプローチ

R1. サーバ設定演習時に管理者権限を持つ。
サーバ設定演習では、学習者が named などのサーバデーモンの設定変更をして、再起動するなどの管理者権限を持った上での作業が可能でなければならない。

R2. 学生の学習状況把握が安全に実施できる。
サーバ設定において、設定ファイルの変更履歴だけではなく、サーバデーモンが正しく起動しているか、といった進行状況が把握できる必要がある。

3 検討

2 節の要求を満たすために、本稿では 2 種類のアプローチを検討する。

3.1 ゲスト OS 型

学習者用の OS 環境を仮想計算機技術のゲスト OS として実現し、ホスト OS 側からゲスト OS の実行状況を監視する方式であり、外側から状況把握をしようとするアプローチである (図 1)。

この方式では、学習者用の OS 環境からはアクセスすることが出来ないホスト OS 側から監視するため、R1 であっても、ホスト OS 側への影響は無視できる。しかし、ホスト OS 側からゲスト OS の詳細な動作がどこまで参照可能となっているかは、仮想化技術によって様々であるため、実現が容易に可能かどうかは調査が必要である。

調査: Xen 環境
本方式の実現可能性を確認するために、仮想化技術のうち Xen[4] について調査した。

Xen では、ホスト OS 的に動作する Domain-0 とゲスト OS 的に動作する Domain-U があり、Domain-0 上でドメイン管理ツールとして `xm` というコマンドが存在する。`xm` コマンドは、ドメインの作成、停止、削除、一覧以外に `top` と呼ばれる各ドメインの CPU 使用率やメモリ使用率、ネットワーク使用量をリアルタイムに表示できる機能を持っているが、R2 としては機能が不足している。また、Domain-U のカーネルデバッグ機能を使用すると、Domain-0 から Domain-U の状況が把握できると考えられるが、実用的に使用できるかどうかは、さらに調査をすすめる必要がある。

3.2 権限分離型

学習者用の OS 環境内に、特別に隔離された領域を設定し、この領域で状況把握モジュールを実行する方式であり、内側から状況把握をしようとするアプローチである (図 2)。。

この方式では、学習者用の OS 環境上でモジュールが動作するため、R2 で必要となる情報の取得には問題がないと考えられる。しかし、ネットワーク接続など隔離することが本質的に困難な部分が

想定されるため、どこまでモジュールの安全性が確保できるかという点で調査が必要である。

調査: SELinux
SELinux (Security Enhanced Linux) [5] は、各プロセスがアクセスできるファイルやディレクトリなどを制限し、本当に必要なリソースだけにアクセスできるようにするための機能である。SELinux 自体は、そもそもセキュリティのための機能であり、ネットワークサービスの脆弱性をついて管理者特権まで得た侵入者が、種々の設定を書き換えてしまうのを防ぐことが期待されている。

まず、SELinux は、「ポリシー」と呼ばれるアクセス制御ルールをもち、各プロセス (デーモン) にはドメインと呼ばれる「ラベル (識別子)」が付与される。例えば、httpd なら/usr/sbin/httpd というプロセスに対して、httpd_t というドメインが付与される。また、各ファイルには「タイプ」と呼ばれるラベルが付与される。例えば、/var/www 以下のファイルには httpd_sys_content_t と言うタイプが付与される。「ポリシー・ファイル」には、どのドメインがどのタイプにアクセスできるのか、というルールが記述されている。このアクセス制御ルールが「ポリシー」となる。さらに、ユーザが属する役割を「ロール (role)」として定義できる。

これにより、システム管理用ユーザというロールと、学習用ユーザというロールを新しく定義し、通常の管理者特権 (uid=0) を区別できるポリシーを作成することで、R1 と R2 を両立させることが可能という感触を得た。

4 まとめ

本稿では、サーバ設定演習時に、学習者に管理者権限を付与しつつ、進行状況監視モジュールを安全に提供するためのシステム構成方法について、仮想計算機技術を用いたアプローチと、SELinux を用いたアプローチの検討を行った。

今後の課題として、実際にサーバ設定演習用の SELinux ポリシー・ファイルの構成を行うことや、状況監視モジュールの実装が残されている。

謝辞

本研究は一部，日本学術振興会・科学研究費補助金・基盤研究 (C)（課題番号 19500069）の研究助成による．

参考文献

[1] Hideo Masuda, Michio Nakanishi and Seigo Yasutome: “Hands-on Training Course for Server Setup and Operations using Diskless Computer System” ,ITHET2007 (2007).

[2] 桝田秀夫, 中西通雄, 安留誠吾:「PC 演習室を使用した持ち込みブートサーバによる OS 設定演習事例」 , 2007PCC, pp.447-450 (2007).

[3] 太田範, 桝田秀夫, 齊藤明紀, 増澤利光 :「個人所有の計算機を想定した計算機環境でのモバイルエージェントを用いた教育支援システム」, FIT2003, LN-006, pp.385-386 (2003).

[4] Xen, http://xen.cl.cam.ac.uk/Research/SRG/netos/xen/.

[5] SELinux, http://www.nsa.gov/selinux/.

自律分散ロボット群による４台での正方形形成について

橋本 圭太 † **泉 泰介** †† **片山 喜章** †† **犬塚 信博** †† **和田 幸一** ††

† †† 名古屋工業大学大 〒 466–8555 愛知県名古屋市昭和区御器所町

† keita@phaser.elcom.nitech.ac.jp, †† {t-izumi,katayama,inuzuka,wada}@nitech.ac.jp

あらまし 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の集合であり，これら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目的を達成するシステムである．自律分散ロボット群に対する協調問題である形状形成問題の中でも，n 台のロボットによって正 n 角形を形成する問題を正多角形形成問題，または円形成問題という．本研究では，今まで未解決であったロボット 4 台の多角形形成問題に対するアルゴリズムを提案する．SYM においては任意の初期配置，CORDA においては制限された初期配置からアルゴリズムを開始しても問題を解く．

キーワード 自律分散ロボット，形状形成問題，正多角形形成問題，円形成問題

1. はじめに

近年，自律分散システムに関する研究が盛んにおこなわれており，自律分散ロボット群の研究もそのひとつである．自律分散ロボット群とは自律的に動作する複数のロボットの集合であり，これら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目的を達成する．深海や宇宙など人間がロボット群を直接管理することが難しいような環境において，人間による管理の必要がない自律分散ロボット群の利用が期待されている．

本研究では自律分散ロボット群を計算論的な観点からとらえ，その協調問題を取り扱う．自律分散ロボット群の二次元平面における協調問題を扱った研究として，鈴木，山下らによる研究 [1] がある．この研究において，自律分散ロボット群の理論モデルが構築され，これをモデルを SYM と呼ぶ．しかし，このモデルにおけるロボットは，行動（観測，計算，移動）を開始するタイミングにおいてのみ非同期であり，行動自体は瞬間的におこなわれることが仮定されている．これに対して，G. Prencipe[2] らによって各ロボットの動作が完全に非同期であるモデル CORDA が提案された．

これらのモデルを用いて，ロボット群による協調問題として一点集合問題，形状形成問題，一点収束問題などについての研究が進められてきた．本研究では形状形成問題，なかでも特に正多角形形成問題について考える．形状形成問題とは，初期配置として任意の場所に配置されたロボット群が協調して動作し，やがて特定の形状を形成する問題である．そして，n 台のロボット群によって正 n 多角形を形成する問題を，正多角形形成問題または円形成問題という．

正多角形問題についての既存研究の結果について表 1 に示す．

文献 [3]，[4] により，SYM で正多角形に収束するアルゴリズムが提案された．これらのアルゴリズムでは，各ロボットは目的状況に無限に近づくが停止できない．一方，文献 [5]–[8] のアルゴリズムは，各ロボットが形状を形成し停止する．文献 [5] では SYM で素数台の場合に，文献 [6] では SYM で 4 台以外の場合に，それぞれ正多角形形成問題が解けることが示されている．文献 [7] では CORDA で奇数台の場合に，文献 [8] では CORDA で 4，8，6 以外の台数で移動に制限がある場合に，それぞれ正多角形を形成するアルゴリズムが提案された．

表 1: 既存研究

	モデル	ロボットの台数	注意点
[3]	SYM	任意の台数	収束
[4]	SYM	任意の台数	収束
[5]	SYM	素数台	
[6]	SYM	4 以外の台数	
[7]	CORDA	奇数台	
[8]	CORDA	4，6，8 以外の台数	途中で停止せず目的地に到着
本研究	SYM	4 台	
	CORDA	4 台	初期配置を制限
未解決	CORDA	6，8 台	

本論文では，SYM，CORDA の両方で未解決であったロボットの台数が 4 台の場合の正多角形形成問題（正方形形成問題）について考える．CORDA では特定の初期配置から，SYM では任意の初期配置から，それぞれ正方形を形成するアルゴリズムを提案する．

2. モデルと問題定義

2.1 ロボットのモデル

本論文では，CORDA と SYM のロボットモデルを扱う．これらのロボットモデルについて説明する．

2.1.1 基本モデル

両ロボットモデルに共通する基本的なモデル示す．

- ロボットは体積を持たない点として扱う．
- ロボットは 2 次元平面を自由に移動できる．
- ロボットは他のロボットを外見で区別できない．
- ロボットは過去の情報を記憶しておくことができない．
- ロボットは通信能力を持っていない．

- すべてのロボットは同じアルゴリズムを実行する．
- ロボットは共通の座標系をもたず，それぞれ独自の x-y 直交座標系を持つ．

2.1.2 実行モデル

ロボットモデルの実行モデルについて CORA と SYM で，共通する部分について述べる．ロボットは観測，計算，移動という動作を一回ずつ順番におこない（これをサイクルという），それを繰り返す．ロボットの 1 サイクルの動作について，以下に詳細を述べる．

1. **観測** 各ロボットの位置を局所座標系上の座標として観測する．また，観測は一瞬でおこなわれる．この観測において，ロボットの視野に制限はなく，他のロボットが視野の邪魔になることはないものとする．以降，r はロボット，またはロボット r がいる点を表す．
2. **計算** アルゴリズムにより目的地の計算をおこなう．ロボットは過去のサイクルにおける情報を記憶しておくことができないので，アルゴリズムの入力は同一サイクルにおける観測結果のみである．出力は次の移動で向かう目的地の局所座標系上の座標点である．以降，ロボット r_i の目的地を d_i と表す．
3. **移動** 計算によって出力された目的地に向かって直進する．また目的地に到達する前に非決定的に停止することがあるが，一度の移動で少なくとも定数 $\delta(>0)$ 以上は移動し，計算した目的地までの距離が δ より短い場合には，その移動で目的地に到達する．目的地に到着する前に停止した場合，次のサイクルではその位置で新たに観測し目的地の計算をおこなう．

サイクルを構成する上記の動作以外に以下の動作も定義する．

(4) **待機** ロボットは何もしない．初期状態ではすべてのロボットがこの状態である．

2.1.3 同期モデル

ロボットの実行モデルである SYM と CORDA について説明する．いずれのモデルでも，各ロボットは同じアルゴリズムにしたがってサイクルを繰り返し実行する．このとき，各ロボットが実行するサイクルの同期の程度によってふたつのモデルが定義される．

SYM では，すべてのロボットが完全に同期した時計を持ち，それに従いサイクルを単位時間で実行すると仮定する．すなわち，サイクルを実行するロボット（active と呼ぶ）は同時に実行を開始し，サイクルを構成する各状態も同期している．ここで，サイクルを実行しないロボット（inactive と呼ぶ）が存在しても良い．ただし，すべてのロボットは，無限にしばしば active になると仮定する．

表 2: 状況の分類

線分	対称	C_{SymL}
	非対称	C_L
三角形	直角二等辺三角形	C_{IR3}
	直角二等辺三角形以外の三角形	C_3
四角形	正方形	C_{Sq}
	正方形以外の長方形	C_{Rect}
	長方形以外の等脚台形	C_{ITrap}
	等脚台形以外の等長対角線四角形	C_{Eq4}
	不等長対角線四角形	C_{UEq4}

CORDA では，同期に関して一切の仮定をおかない．つまり，各ロボットがサイクルを開始する時刻は任意であり，また 1 サイクルにかかる時間は少なくとも定数 $\epsilon(>0)$ 以上で，かつ有限で，非決定的である．さらに，サイクルを構成する各状態にかかる時間も有限であるが非決定的である．

2.2 正多角形形成問題

R をロボットの集合，またはロボットがいる点の集合とする．n 台のロボット群 R に対し，ロボットが重なっていない任意の初期配置から正 n 多角形を形成させロボットを停止させる問題を正多角形形成問題という．特に，n=4 のときの正多角形形成問題を正方形形成問題と呼ぶ．

3. 諸定義

3.1 定義

本論文で用いることがらを次のように定める．

- $|r_i, r_j|$：2 つのロボット r_i と r_j の距離を表す．
- $\angle p_i, p, p_j$：半直線 $[p, p_i)$ と $[p, p_j)$ により作られる角度表す．
- **点集合 P の凸包**：点集合 P が与えられたとき，$\forall p \in P$ を含む面積最小の凸多角形を P の凸包と呼ぶ．
- **対称な線分**：点集合 P が与えられたとき，P の凸包が線分で，かつこの線分に対して垂直なある直線に対して P が対称な配置の場合の形状を対称な線分と呼ぶ．
- **等長対角線四角形**：2 本の対角線の長さが等しい四角形を等長対角線四角形と呼ぶ．
- **不等長対角線四角形** ：2 本の対角線の長さが異なる四角形を不等長対角線四角形と呼ぶ．

3.2 状況の定義

ロボット群 R の配置を状況と呼ぶ．R は 4 点からなるので，その凸包は線分，三角形，四角形のいずれかである．これをさらに次の通りに分類する (表 2)．

R の凸包が線分になる状況を次の 2 通りに分類する．R の凸包が対称な線分を形成している状況のクラスを

```
algorithm  Sq – CORDA :
Input  R
switch
  case R ∈ C_Sq :        動かない
  case R ∈ C_UEq4 :      call  C_UEq4 → C_Sq
  case R ∈ C_Eq4 :       call  C_Eq4 → C_UEq4
  case R ∈ C_IR3 :       call  C_IR3 → C_Sq
  case R ∈ C_3 :         call  C_3 → C_IR3
  case R ∈ C_L :         call  C_L → C_IR3
```

図 1: アルゴリズム Sq-CORDA

```
|r_end1, r_right| < |r_ins, r_end2| となるように，片側から順番に r_end1, r_right, r_ins, r_end2とする
if 自分は r_right
  then {r_end1, d_right, r_end2} が, [r_end1, r_end2] を最長辺とする直角二等辺三角形となる d_right の位置に移動
else  動かない
```

図 2: 手続き $C_L \rightarrow C_{IR3}$

C_{SymL}，R の凸包が対称でない線分を形成している状況のクラスを C_L と表す．

R の凸包が三角形になる状況は，直角二等辺三角形を形成している状況のクラス C_{IR3} と，直角二等辺三角形でない三角形を形成している状況のクラス C_3 に分類する．

R の凸包が四角形になる状況を次の 5 通りに分類する．正方形を形成している状況のクラス C_{Sq}，正方形でない長方形を形成している状況のクラス C_{Rect}，長方形でない等脚台形を形成している状況のクラス C_{ITrap}，等脚台形でない等長対角線四角形を形成している状況のクラス C_{Eq4}，不等長対角線四角形を形成している状況のクラス C_{UEq4} である．

4. CORDA での正方形形成

4.1 アルゴリズムの概要

初期配置が C_{SymL}，C_{Rect}，C_{ITrap} 以外の場合に，CORDA で正方形形成するアルゴリズム Sq-CORDA について説明する（図 1）．アルゴリズム Sq-CORDA では状況により 5 つの手続きを実行する．以降の節でアルゴリズム Sq-CORDA で用いている 5 つの手続きについて説明していく．

4.2 手続き $C_L \rightarrow C_{IR3}$

手続き $C_L \rightarrow C_{IR3}$ は，C_L の状況のときに実行する手続きである（図 2）．移動するロボット r_{right} を 1 台に定め，そのロボットが凸包が直角二等辺三角形になるような位置に向かって移動する（図 3）．

補題 1 初期配置が C_L の状況の場合，手続き $C_L \rightarrow C_{IR3}$ を実行することによって，有限時間内に C_{IR3} または C_3 の状況になり，すべてのロボットが止まる時刻が存在する．

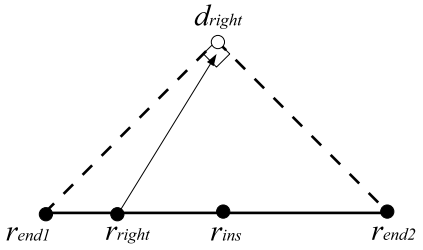


図 3: C_L の状況

```
凸包三角形の頂点でないロボットを r_ins とする
if r_ins が凸包三角形の辺上にいる #図 5 の右の状況
  then r_ins のいる凸包三角形の辺の両端のロボットを {r_end1, r_end2} とし, [r_end1, r_end2] 上にいないロボットを r_right とする
  if 自分は r_right
    then {r_end1, r_end2, d_right} が, [r_end1, r_end2] を最長辺とした r_right 側にできる直角二等辺三角形となる d_right の位置に移動
  else  移動しない
else  #図 5 の左の状況
  if 自分は r_ins
    then r_ins から最も近い凸包三角形の任意の最長辺を l, r_ins から l に垂線を下ろしたときの交点 d_ins の位置に移動
  else  動かない
```

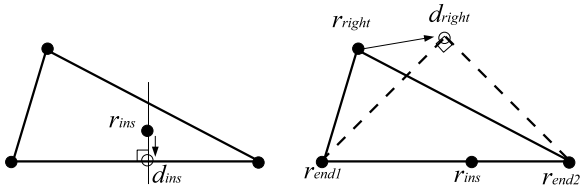
図 4: 手続き $C_3 \rightarrow C_{IR3}$

証明 r_{right} は d_{right} に向かってに移動する．したがって r_{right} は $[r_1, r_3]$ 上から外れ，ロボットが重ならず，r_{right} は必ず有限時間内で再び待機になる．また d_{right} までの距離が 0 でないので，少なくとも δ は移動する．r_{right} 以外のロボットは移動しないので，r_{right} がはじめて止まったとき，C_{IR3} または C_3 の状況になり，すべてのロボットが止まっている． □

4.3 手続き $C_3 \rightarrow C_{IR3}$

手続き $C_3 \rightarrow C_{IR3}$ は，C_3 の状況のときに実行し，直角二等辺三角形を形成する手続きである（図 4）．この手続きは 2 つのフェーズからなる．フェーズ 1 は r_{ins} が凸包三角形の辺上にいない場合であり，このとき r_{ins} が凸包三角形の辺上に向かって移動する．フェーズ 2 は r_{ins} が凸包三角形の辺上にいる場合で，r_{right} が凸包が直角二等辺三角形になるような位置に向かって移動する（図 5）．手続き $C_L \rightarrow C_{IR3}$ の結果 C_3 の状況になったとき，フェーズ 2 に合流する．

補題 2 C_3 の状況ですべてロボットが止まっている場合，手続き $C_3 \rightarrow C_{IR3}$ を実行することによって，有限時間



フェーズ 1　　フェーズ 2

図 5: C_3 の状況

凸包三角形の頂点でないロボットを r_{ins}，凸包三角形の頂点のロボットのうち最長辺の両端のロボットを $\{r_{end1}, r_{end2}\}$，残りの凸包三角形の頂点のロボットを r_{right}とする
if 自分は r_{ins}
　then $\{d_{ins}, r_{end1}, r_{end2}, r_{right}\}$ が，正方形となる d_{ins}の位置に移動
else　動かない

図 6: 手続き $C_{IR3} \rightarrow C_{Sq}$

内に C_{IR3} の状況になり，すべてのロボットが止まる時刻が存在する．

証明　r_{ins} が凸包三角形の辺上にいない場合について考える（図 5 の左図）．r_{ins} は d_{ins} に向かって移動する．d_{ins} は凸包三角形の頂点と一致することがないので，ロボットが重なることはない．また，d_{ins} までの距離は有限なので，必ず到達する時刻が存在する．r_{ins} 以外のロボットは移動しないので，r_{ins} が d_{ins} に到着すると，r_{ins} が凸包三角形の辺上に存在し，すべてのロボット止まっている．

次に，r_{ins} が凸包三角形の辺上にいる場合について考える（図 5 の右図）．r_{right} は d_{right} に向かって移動する．d_{right} は $[r_{end1}, r_{end2}]$ 上や r_{right} の反対側にはないので，$[r_{end1}, r_{end2}]$ と交差せず，ロボットが重なることはない．また，d_{right} までの距離は有限なので，必ず到達する時刻が存在する．r_{right} 以外のロボットは移動しないので，r_{right} が d_{right} に到着して止まったとき，C_{IR3} の状況になり，すべてのロボット止まっている．　□

4.4　手続き $C_{IR3} \rightarrow C_{Sq}$

手続き $C_{IR3} \rightarrow C_{Sq}$ は，C_{IR3} の状況のときに実行する手続きである（図 6）．r_{ins} が凸包が正方形となる位置に向かって移動する（図 7）．

補題 3 C_{IR3} の状況ですべてのロボットが止まっている場合，手続き $C_{IR3} \rightarrow C_{Sq}$ を実行することによって有限時間内に C_{Sq} または C_{UEq4} の状況になり，すべてのロボットが止まる時刻が存在する．

証明　r_{ins} が d_{ins} に向かって移動するとき，凸包三角形の頂点上を通過せず，ロボットは重ならない．d_{ins} までの距離は 0でなく，r_{ins} は少なくとも δ 移動する．r_{ins}

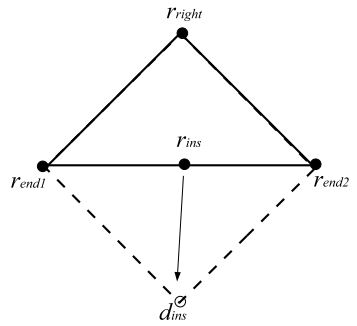


図 7: C_{Ir3} の状況

対角線の交点を O とし，O から最も遠いロボットを r_{outs}，O から最も近いロボットを r_{ins}，残りのロボットを $\{r_{end1}, r_{end2}\}$ とする
if 自分は r_{outs}
　then $[r_{outs}, O]$ と $[r_{end1}, r_{end2}]$ を直径とする円 C との交点 d_{outs}に移動
else　動かない

図 8: 手続き $C_{Eq4} \rightarrow C_{UEq4}$

以外のロボットは移動しないので，r_{ins} がはじめて止まったとき，C_{Sq} または C_{UEq4} の状況になり，すべてのロボットが止まっている．　□

4.5　手続き $C_{Eq4} \rightarrow C_{UEq4}$

手続き $C_{Eq4} \rightarrow C_{UEq4}$ は，C_{Eq4} の状況のときに実行する手続きである（図 8）．r_{outs} が対角線が短くなるように移動する（図 9）．

補題 4 初期配置が C_{Eq4} の状況の場合，手続き $C_{Eq4} \rightarrow C_{UEq4}$ を実行することによって有限時間内に C_{UEq4} の状況になり，すべてのロボットが止まる時刻が存在する．

証明　まず r_{outs} は一意に定まることを証明し，次に r_{outs} が d_{outs} に向かって移動することで，有限時間内に C_{UEq4} の状況になり，すべてのロボットが止まる時刻が存在することを証明する．

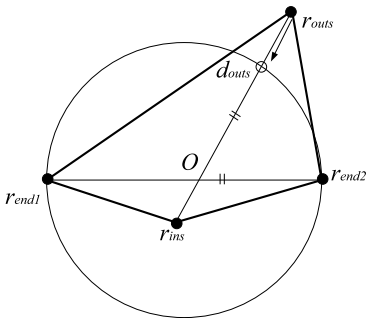


図 9: C_{Eq4} の状況

```
1  短い対角線の両端のロボットを {r_1, r_2} とし，長い
2  対角線の両端のロボットを {r_end1, r_end2},
3  [r_end1, r_end2] を直径とする円を C とする
4  if {r_1, r_2} が C の内側または円周上にいる
5        #図 11 の右の状況
6    then if 自分は r_1, r_2
7      then {d_1, d_2, r_end1, r_end2} が正方形を形成す
8           る {d_1, d_2} の位置のうち近い方に移動
9    else   動かない
10 else   #図 11 の左の状況
11   if 自分は C の内側または円周上にいる
12     then 動かない
13   else   [r_1, r_2] と C との交点のうち近い方に移動
```

図 10: 手続き $C_{UEq4} \rightarrow C_{Sq}$

r_{outs} は一意に定まることを証明する．R の凸包が等長対角線四角形を形成していると仮定する．O から最も遠いロボットが *2* 台存在する場合，この *2* 台のロボットは異なる対角線の端点となる．対角線が等しので，残りの *2* 台のロボットは，O との距離は等しい．このときの R の凸包は等脚台形となり，C_{ITrap} の状況になる．O から最も遠いロボットが *3* 台存在する場合は，R の凸包は等長対角線四角形にならない．O から最も遠いロボットが *4* 台存在する場合，R の凸包は長方形となり，C_{Rect} の状況になる．以上により，O から最も遠いロボットが複数台ある場合は C_{Eq4} の状況にならないので，C_{Eq4} の状況ならば r_{outs} は一意に定まる．

続いて，r_{outs} が d_{outs} に向かって移動することで，有限時間内に C_{UEq4} の状況になり，すべてのロボットが止まる時刻が存在することを証明する．r_{end1}, r_{end2} は C の円周上にいて，r_{ins} は $|r_{end1}, O| > |r_{ins}, O|$，$|r_{end2}, O| > |r_{ins}, O|$ より，C の内側にいる．したがって，r_{outs} のみが円の外側にいて，d_{outs} は C の円周上にあるので，r_{outs} が d_{outs} に向かって移動することで $|r_{outs}, r_{ins}|$ が小さくなり，ロボットが重なることはない．d_{outs} までの距離は *0* でなく，r_{outs} は少なくとも δ 移動する．r_{outs} 以外のロボットは移動しないので，r_{outs} がはじめて止まったとき，C_{UEq4} の状況になり，すべてのロボットが止まっている． □

4.6 手続き $C_{UEq4} \rightarrow C_{Sq}$

手続き $C_{UEq4} \rightarrow C_{Sq}$ は，C_{UEq4} の状況のときに実行する手続きである（図 10）．この手続きは 2 つのフェーズからなる．フェーズ 1 は円の外側にいるロボットが存在する場合であり，このとき外側にいるロボットが円周上に移動する．後フェーズ 2 はすべてのロボットが C の内側または円周上にいる場合で，$\{r_1, r_2\}$ が正方形を形成する位置に向かって移動する（図 11）．手続き $C_{Eq4} \rightarrow C_{UEq4}$ の結果 C_{UEq4} の状況になったときは，フェーズ 1 に，手続き $C_{IR} \rightarrow C_{Sq}$ の結果 C_{UEq4} の状況になったときは，フェーズ 2 に合流する．

補題 5 C_{UEq4} の状況ですべてのロボットが止まっている場合，$C_{UEq4} \rightarrow C_{Sq}$ を実行することによって有限時間内に C_{Sq} の状況になり，すべてのロボットが止まる時刻が存在する．

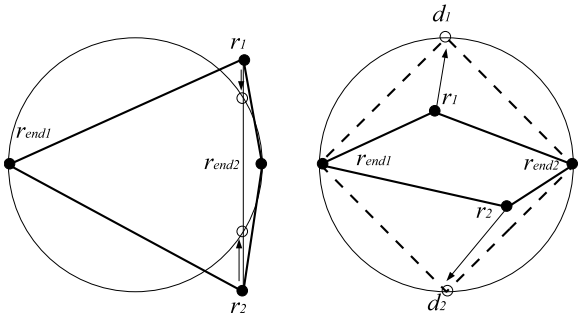


図 11: C_{UEq4} の状況

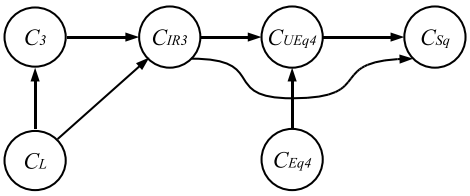


図 12: Sq-CORDA の状態遷移

証明 C の外側にいるロボットが存在する場合について考える（図 *11* の左図）．$\{r_1, r_2\}$ の C の外側にいるロボットが，$[r_1, r_2]$ と C との交点のうち近い方に移動するので，$|r_1, r_2|$ は小さくなり，ロボットが重なることはない．目的地までの距離は有限なので，必ず到達する時刻が存在する．$\{r_{end1}, r_{end2}\}$ と内部または円周上にいるロボットは移動しないので，C の外側にいるロボットが目的地に到着すると，すべてのロボットが C の内側または円周上に存在し，すべてのロボットが止まっている．

次に，すべてのロボットが C の内側または円周上にいる場合について考える（図 *11* の右図）．$\{r_1, r_2\}$ は $\{d_1, d_2\}$ のうち近い方に移動するので移動することでロボットが重なることはない．また，目的地までの距離は有限なので，必ず到達する時刻が存在する．

$\{r_{end1}, r_{end2}\}$ は移動しないので，$\{r_1, r_2\}$ がそれぞれの目的地に到着して止まったとき，C_{Sq} の状況になり，すべてのロボット止まっている． □

4.7 アルゴリズム Sq-CORDA

4.2 節から 4.6 節で，アルゴリズム Sq-CORDA が各状況で実行する手続きについて説明し，これらの手続きの動作の正当性について証明をした．次に，これらの手続きを繋げたアルゴリズム Sq-CORDA の動作の正当性を示す．そのときの状態遷移図を図 12 に示す．

定理 1 C_{SymL}，C_{Rect}，C_{ITrap} 以外の任意の初期配置からアルゴリズム *Sq-CORDA* を実行すると，有限時間内に正方形形成問題を解く．

証明　補題 *1–5* より，C_L となる状況は初期配置しかない．補題 *1* より，有限時間内に C_{IR3} または C_3 の状況になり，すべてのロボットが止まる時刻が存在する．

補題 *1–5* より，C_3 となる状況は，初期配置もしくは C_L からの遷移の *2* 通りである．補題 *1* におけるすべてのロボットが止まる時刻を t_L とする．C_L からの遷移してきて時刻 t_L より前の場合，r_{right} が動いている．手続き $C_L \rightarrow C_{IR3}$ での r_{right} の動きは，手続き $C_3 \rightarrow C_{IR3}$ での r_{ins} が凸包三角形の辺上にいる場合における r_{right} の動きと同じである．また，初期配置もしくは C_L からの遷移で時刻 t_L 以降はすべてのロボットが止まっている．よって補題 *2* より，初期配置もしくは C_L からの遷移の場合，有限時間内に C_{IR3} の状況になり，すべてのロボットが止まる時刻が存在する．

補題 *1–5* より，C_{IR3} となる状況は，初期配置，C_L からの遷移，C_3 からの遷移の *3* 通りである．補題 *2* におけるすべてのロボットが止まる時刻を t_3 とする．手続き $C_3 \rightarrow C_{IR3}$ での時刻 t_3 より前は C_3 の状況である．したがって，C_3 のからの遷移の場合はすべてのロボットが止まっている．また，手続き $C_L \rightarrow C_{IR3}$ での時刻 t_L より前も C_3 の状況なので，C_L のからの遷移の場合はすべてのロボットが止まっている．以上により，初期配置，C_L からの遷移，C_3 のからの遷移の場合，すべてのロボットが止まっている．よって補題 *3* より，初期配置の C_L からの遷移，C_3 のからの遷移場合，有限時間内に C_{Sq} または C_{UEq4} の状況になり，すべてのロボットが止まる時刻が存在する．

補題 *1–5* より，C_{Eq4} となる状況は初期配置しかない．補題 *4* より，有限時間内に C_{UEq4} の状況になり，すべてのロボットが止まる時刻が存在する．

補題 *1–5* より C_{UEq4} となる状況は，初期配置，C_{IR3} からの遷移，C_{Eq4} からの遷移の *3* 通りである．補題 *3* におけるすべてのロボットが止まる時刻を t_{IR3} とする．C_{IR3} からの遷移で時刻 t_{IR3} より前の場合，r_{ins} が動いている．手続き $C_{IR3} \rightarrow C_{Sq}$ での r_{ins} の動きは，手続き $C_{UEq4} \rightarrow C_{Sq}$ での，すべてのロボットが C の内側または円周上にいるの場合における $\{r_1, r_2\}$ の動きと同じである．補題 *4* におけるすべてのロボットが止まる時刻を t_{Eq4} とする．C_{Eq4} からの遷移で時刻 t_{Eq4} より前の場合，r_{outs} が動いている．手続き $C_{Eq4} \rightarrow C_{UEq4}$ での r_{outs} の動きは，手続き $C_{UEq4} \rightarrow C_{Sq}$ での，C の外側にロボットが存在する場合における $\{r_1, r_2\}$ の C の外側にいるロボットの動きと同じである．また，初期配置，C_{IR3} からの遷移で時刻 t_{IR3} 以降，C_{UEq4} からの遷移で時刻 t_{UEq4} 以降はすべてのロボットが止まっている．よって補題 *2* より，初期配置，C_{IR3} からの遷移，C_{Eq4} のからの遷移の場合，有限時間内に C_{Sq} の状況になり，すべてのロボットが止まる時刻が存在する．

補題 *1–5* より，C_{Sq} となる状況は，初期配置，C_{IR3} からの遷移，C_{UEq} からの遷移の *3* 通りである．補題 *5* におけるすべてのロボットが止まる時刻を t_{UEq4} とする．手続き $C_{UEq4} \rightarrow C_{Sq}$ からの遷移で時刻 t_{UEq4} より前は C_{UEq4} の状況である．したがって，C_{UEq4} からの遷移の場合はすべてのロボットが止まっている．また，手続き $C_{IR3} \rightarrow C_{Sq}$ からの遷移で時刻 t_{IR3} より前は C_{UEq4} の状況である．したがって，C_{IR3} からの遷移の場合はすべてのロボットが止まっている．以上により，初期配置，C_{IR3} からの遷移，C_{UEq4} からの遷移の場合，すべてのロボットが止まっている．よって初期配置，C_{IR3} からの遷移，C_{UEq4} からの遷移場合，C_{Sq} の状況ですべてのロボットが止まっている．

以上により，C_{SymL}，C_{Rect}，C_{ITrap} 以外の任意の初期配置からアルゴリズム *Sq-CORDA* を実行すると，有限時間内に正方形形成問題を解く．　□

```
1  algorithm　Sq − SYM : :
2  Input　R
3  switch
4    case R ∈ C_Sq :　　動かない
5    case R ∈ C_UEq4 :　 call　C_UEq4 → C_Sq
6    case R ∈ C_Eq4 :　 call　C_Eq4 → C_UEq4
7    case R ∈ C_ITrap :　call　C_ITrap → C_Sq
8    case R ∈ C_Rect :　 call　C_Rect → C_Sq
9    case R ∈ C_IR3 :　 call　C_IR3 → C_Sq
10   case R ∈ C_3 :　　call　C_3 → C_IR3
11   case R ∈ C_L :　　call　C_L → C_IR3
12   case R ∈ C_SymL :　call　C_SymL → C_ITrap
```

図 13: アルゴリズム Sq-SYM

5. SYM での正方形形成

5.1 アルゴリズムの概要

任意の初期配置から，SYM で正方形形成するアルゴリズム Sq-SYM について説明する（図 13）．アルゴリズム Sq-CORDA では初期配置が限定されていたのに対し，Sq-SYM は任意の初期配置を限定しない．つまり，アルゴリズムで考慮すべき状態として $C_{SymL}, C_{Rect}, C_{ITrap}$ が追加される（図 13 の 7, 8, 12 行目）．この 3 つの手続き以外は，Sq-CORDA と同じものであり，定理 2 より SYM でも動作する．

定理 2 *[2]*
CORDA において，ある問題 P を正しく解く任意のアルゴリズムは，*SYM* においても，P を正しく解く．

したがって，以下では Sq-SYM で新たに追加された 3 つの手続きについてのみ説明する．

5.2 手続き $C_{SymL} \rightarrow C_{ITrap}$

手続き $C_{SymL} \rightarrow C_{ITrap}$ は，C_{SymL} の状況のときに実行する手続きである (図 14)．両端でないロボットが線分上から外れるように移動する (図 9)．

補題 6 C_{SymL} の状況の場合，手続き $C_{SymL} \rightarrow C_{ITrap}$ を実行することによって，C_3，C_{UEq4}，C_{ITrap} のいずれかの状況になる．

証明　$\{r_{short1}, r_{short1}\}$ は $\{d_{short1}, d_{short1}\}$ に移動する．$[r_{long1}, r_{long2}]$ に対して垂直な方向に移動するので，ロボットが重なることはない．$[r_{long1}, r_{long2}]$ から *2* 台のロボットが外れるので，凸包は三角形もしくは四角形になる．$\{r_{short1}, r_{short1}\}$ が同じ側に移動した場合，その

線分の両端のロボットを $\{r_{long1}, r_{long2}\}$，残りのロボットのうち r_{long1} に近いロボットを r_{short1}，r_{long2} に近いロボットを r_{short2} とする
if 自分は $\{r_{short1}, r_{short2}\}$
  then $[r_{long1}, r_{long2}]$ に対して垂直な方向に，
    $|r_{short1}, r_{long1}| = |r_{short1}, d_{short1}|$，
    $|r_{short2}, r_{long2}| = |r_{short2}, d_{short2}|$
    となる $\{d_{short1}, d_{short2}\}$ の位置に移動
else 動かない

図 14: 手続き $C_{SymL} \rightarrow C_{ITrap}$

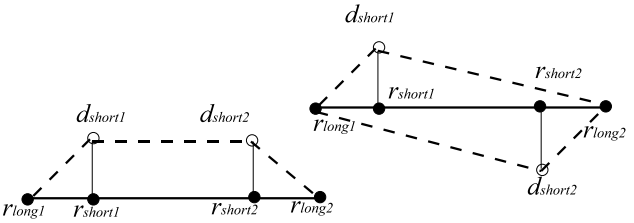


図 15: C_{SymL} の状況

移動距離が等しければ C_{ITrap} の状況になる．移動距離が等しくなく，凸包が四角形になる場合は，対角線の長さも等しくならないので，C_{UEq4} の状況になる‡．移動距離が等しくなく，凸包が三角形になる場合は，C_3 となる．$\{r_{short1}, r_{short1}\}$ が異なる側に移動した場合，C_{UEq4} の状況になる．□

CORDA では，ある状態になったとき，ロボットが停止いているかどうかが重要である．もし停止いていないならば，ある状態になって手続きがはじまっても，その状態を観測しないまま前の動作を続ける可能性がある．補題 1–5 では，これらに注意した議論をおこなった．ここでは SYM であるので，各ラウンド毎に必ず各 active なロボットは観測をおこなうため，そういった議論は不要である．

5.3 手続き $C_{Rect} \rightarrow C_{Sq}$

手続き $C_{Rect} \rightarrow C_{Sq}$ は，C_{Rect} の状況のときに実行する手続きである（図 16）．各ロボットが，対角線によってできる 4 つの角度が等しくなるような位置に向かって移動する（図 17）．

補題 7 C_{Rect} の状況から，手続き $C_{Rect} \rightarrow C_{Sq}$ を実行すると，C_{UEq4}，C_{ITrap}，C_{Sq} のいずれかの状況になる．

証明　任意のロボット r_i に対して，r_i の α 側の隣のロボットを r_α，r_α の目的地を d_α とする．r_i の β 側の隣のロボットを r_β，r_β の目的地を d_β とする．4 台のロボットの移動距離が等しい場合，移動後も C_{Rect} になる．目的地までの距離は有限なので，必ず到達する時刻が存在する．

‡必ず 1 サイクルで目的地に到着するという SYM もあるが，その場合はこの状態にはならなず，省略できる

対角線の交点を O とする．
長方形の長い方の辺を l とする．
2 本の対角線でできる角のうち小さい方をα，大きい方をβとする．
ロボット r_i に対して，l 上の$\angle r_i O d_i = \frac{\pi}{4} - \frac{\alpha}{2}$となる点を d_iとする
$[O, d_i]$ 上の $|O, d'_i| = a|O, d_i|$ (a は $0 < a < 1$ の定数）となる d'_iに移動

図 16: 手続き $C_{Rect} \rightarrow C_{Sq}$

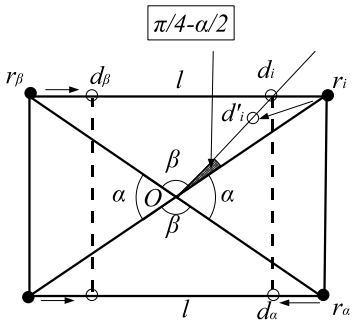


図 17: C_{Rect} の状況

そして，4 台のロボットが目的地に到着したとき，2 本の対角線によってできる角 $\alpha' = \angle d_i O d_\alpha$ と $\beta' = \angle d_i O d_\beta$ は次のようになる．

$$\angle d_i O d_\alpha = \alpha + 2(\frac{\pi}{4} - \frac{\alpha}{2}) = \alpha + \frac{\pi}{2} - \alpha = \frac{\pi}{2}$$

$$\angle d_i O d_\beta = \beta - 2(\frac{\pi}{4} + \frac{\alpha}{2}) = \beta - \frac{\pi}{2} + \alpha = \pi - \frac{\pi}{2} = \frac{\pi}{2}$$

したがって，目的地に到着したとき C_{Sq} の状況になる．

隣り合うロボットが等しい距離を移動し，残り 2 台のロボット同士も等しい距離を移動したとき C_{ITrap} の状況になる．これら以外の場合は，対角線の長さが等しくならず C_{UEq4} の状況になる．

□

5.4 手続き $C_{ITrap} \rightarrow C_{Sq}$

手続き $C_{ITrap} \rightarrow C_{Sq}$ は，C_{ITrap} の状況のときに実行する手続きである（図 16）．2 つの平行辺うち短い方の両端のロボットが凸包が正方形になる位置に向かって移動する（図 17）．

補題 8 C_{ITrap} の状況から，手続き $C_{ITrap} \rightarrow C_{Sq}$ を実行すると，C_3，C_{IR3}，C_{UEq4}，C_{Sq} のいずれかの状況になる．

証明　$\{r_{short1}, r_{short2}\}$ は $\{d_{short1}, d_{short2}\}$ に移動し，$\{r_{long1}, r_{long1}\}$ は移動しない．したがって，R の凸包が線分になることはない．$\{r_{short1}, r_{short2}\}$ の 2 のロボットの移動距離が等しくない場合，等長対角線四角形はないので，C_3，C_{IR3}，C_{UEq4} の形状になる．$\{r_{short1}, r_{short2}\}$

```
2 つの並行辺のうち長い方の両端のロボットを
{r_long1, r_long2}，短い方の両端のロボットを
{r_short1, r_short2} とする
if 自分は {r_short1, r_short2}
   then {r_long1, r_long2, d_short1, d_short2} の
        {d_short1, d_short2} が {r_short1, r_short2}
        に近い正方形となる {d_short1, d_short2} の位
        置に移動
else   動かない
```

図 18: 手続き $C_{ITrap} \rightarrow C_{Sq}$

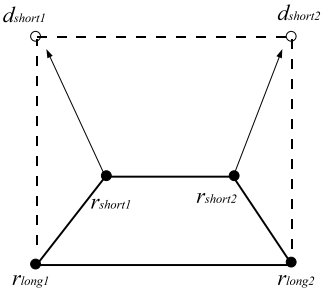


図 19: C_{ITrap} の状況

の *2* のロボットの移動距離が等しい場合，移動後も C_{ITrap} になる．目的地までの距離は有限なので，必ず到達する時刻が存在する．そして，$\{d_{short1}, d_{short2}\}$ に到着したとき C_{Sq} の状況になる． □

5.5 アルゴリズム Sq-SYM

5.2 節から 5.4 節で，アルゴリズム Sq-SYM が各状況のときに実行する手続きのうち，4 章で説明されていない手続き $C_{SymL} \rightarrow C_{ITrap}$，$C_{Rect} \rightarrow C_{Sq}$ $C_{ITrap} \rightarrow C_{Sq}$ について説明した．最後に，これらの手続きを繋げたアルゴリズム Sq-SYM の動作の正当性について証明する．そのときの状態遷移図を図 20 に示す．

定理 3 任意の初期配置から，アルゴリズム *Sq-SYM* を実行すると，有限時間内に正方形形正問題を解く．

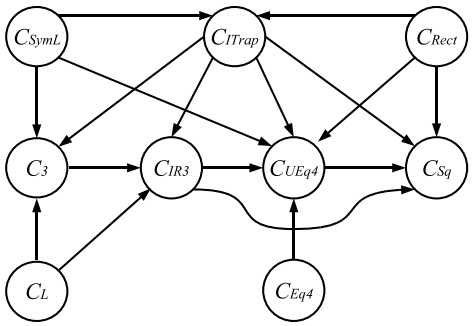


図 20: Sq-SYM の状態遷移

証明 補題 *1–7* より，C_{SymL} となる状況は初期配置しかない．補題 *6* より，有限時間内に C_3，C_{UEq4}，C_{ITrap} のいずれかの状況になる．

補題 *1–7* より，C_{Rect} となる状況は初期配置しかない．補題 *7* より，有限時間内に C_{UEq4}，C_{ITrap}，C_{Sq} のいずれかの状況になる．

補題 *1–7* より，C_{ITrap} となるの状況は，初期配置，C_{SymL} からの遷移，C_{Rect} からの遷移の *3* 通りである．補題 *8* より，初期配置，C_{SymL} からの遷移，C_{Rect} からの遷移の場合，有限時間内に C_3，C_{IR3}，C_{UEq4}，C_{Sq} のいずれかの状況になる．

補題 *1–7* より，C_{SymL} となる状況は初期配置しかない．補題 *1* より，有限時間内に C_3，C_{IR3} のいずれかの状況になる．

補題 *1–7* より，C_3 となる状況は，初期配置，C_L からの遷移，C_{SymL} からの遷移，C_{ITrap} からの遷移の *4* 通りである．*SYM* では，観測をおこなう時刻ではすべてのロボットは止まっている．そして，定義 *1*，*2* より，初期配置，C_L からの遷移，C_{SymL} からの遷移，C_{ITrap} からの遷移場合，有限時間内に C_{IR3} の状況になり，すべてのロボットが止まる時刻が存在する．

補題 *1–7* より，C_{IR3} となる状況は，初期配置，C_L からの遷移，C_3 のからの遷移，C_{ITrap} からの遷移の *4* 通りである．同様の理由により，初期配置，C_L からの遷移，C_3 のからの遷移，C_{ITrap} からの遷移の場合，有限時間内に C_{Sq} または C_{UEq4} の状況になり，すべてのロボットが止まる時刻が存在する．

補題 *1–7* より，C_{Eq4} となる状況は初期配置しかない．補題 *4* より，有限時間内に C_{UEq4} の状況になる．

補題 *1–7* より，C_{UEq4} となる状況は，初期配置，C_{IR3} からの遷移，C_{Eq4} のからの遷移，C_{SymL} からの遷移，C_{ITrap} からの遷移，C_{Rect} からの遷移の *6* 通りである．同様の理由により，初期配置，C_{IR3} からの遷移，C_{Eq4} のからの遷移，C_{SymL} からの遷移，C_{ITrap} からの遷移，C_{Rect} からの遷移の場合，有限時間内に C_{Sq} の状況になり，すべてのロボットが止まる時刻が存在する．

補題 *1–7* より，C_{Sq} となる状況は，初期配置，C_{IR3} からの遷移，C_{UEq} のからの遷移，C_{IR3} のからの遷移，C_{Rect} のからの遷移の *5* 通りである．同様の理由により，初期配置，C_{IR3} からの遷移，C_{UEq} のからの遷移，C_{IR3} のからの遷移，C_{Rect} のからの遷移の場合，C_{Sq} の状況ですべてのロボットが止まる時刻が存在する．

任意の初期配置から，アルゴリズム *Sq-SYM* を実行と，有限時間内に正方形形成問題を解く． □

6. 今後の課題

本研究ではロボットが 4 台の場合の正方形形成問題を CORDA では特定の初期配置から，SYM では任意の初期配置から，それぞれ解くアルゴリズムを提案した．SYM での正方形形成問題に対し，F. Petit らによって同様の結果が得られているようである．

今後の課題として，CORDA で初期配置に制限がない場合で正方形形成問題を解くアルゴリズム研究，もしくは CORDA では $C_{SymL}, C_{Rect}, C_{ITrap}$ の状況から正方形形成問題は解けないという不可能性の証明がある．また，CORDA での 6，8 台の場合の研究なども挙げられる．

参考文献

[1] I. Suzuki and M. Yamashita, "Distributed anonymous mobile robots: formation of geometric patterns," SIAM J.Comput., Vol.28, No.4, pp.1347–1363, 1999.

[2] G. Prencipe, "Distributed coordination of a set of atuonomous mobile robots," PhD thesis, Dipartimento di Informatica, Università di Pisa, 2002.

[3] X. Defago and A. Konagaya, "Circle formation for obrivious anonymous mobile robots with no commonsense of orientation," 2nd ACM International Annual Workshop on Principles of Mobile Computing, pp.97–104, 2002.

[4] I. Chatzigiannakis, M. Markou and S. Nikoletseas, "Distributed circle formation for anonymous oblivious robots," 3rd Workshop on Efficient and Experimental Algorithms, pp.159–174, 2004.

[5] Y. Dieudonne and F. Petit. "Swing word to make circle formation quiesent," SIROCCO 2007, LNCS, vol.4475, pp.166-179, 2007.

[6] Y. Dieudonne and F. Petit. "Circle formation of weak robots and Lyndon words," Information Processing Letters 104(4), pp.156–162. 2007.

[7] B. Katreniak, "Biangular circle fomation by asynchronous mobile robots," SIROCCO 2005, LNCS, vol.3499, pp.185–199, 2005.

[8] Y. Dieudonne, O. Labbani and F. Petit, "Circle formation of weak mobile robots," SSS 2006, LNCS, vol.4280, pp.262-275, 2006.

リング上における自律分散ロボット群の一点集合について

羽場康太郎 †　泉　泰介 ††　片山　喜章 ††　犬塚　信博 ††　和田　幸一 ††

† †† 名古屋工業大学大学院工学研究科情報工学専攻　〒 466–8555 愛知県名古屋市昭和区御器所町
E-mail: †habako@phaser.elcom.nitech.ac.jp, ††{t-izumi,katayama,inuzuka,wada}@nitech.ac.jp

あらまし　本研究では，自律分散ロボット群によるリング上での一点集合問題を扱う．文献 [1] では初期形状が周期的もしくは辺辺対称の場合，またはロボットが 2 台の場合に一点集合不可能なことが示され，それ以外の場合で，一点集合アルゴリズムが示されている．本稿では，形状をさらに分類し，未解決だった偶数台での対称的な初期形状について考察する．準同期モデルの場合，ロボット台数 4 台について可解であるための必要十分条件を与えた．非同期モデルの場合，ロボット台数 4 台で点点対称な形状に対する一点集合アルゴリズムと，ロボット台数 $2n(n \geqq 3)$ 台で点点ロボット対称な形状に対する一点集合アルゴリズムを与えた．

1. はじめに

近年，自律分散システムにより制御されている自律分散ロボット群に関する研究が盛んにおこなわれている．自律分散ロボット群とは，複数のロボットがそれぞれ自律的かつ協調的に動作することによって，ロボット群全体として目的を達成するシステムである．自律分散ロボット群は耐故障性・柔軟性において優れ，人間などが直接管理する必要がないため，宇宙などでの利用が期待されている．

自律分散ロボット群による協調問題を扱った研究として，鈴木，山下らによる研究 [2] がある．この研究によって，初めて自律分散ロボット関する理論的モデルが構築された．このモデルは準同期モデルと呼ばれ，ロボットの動作に制限がある．文献 [3] では [2] における同期の制限をとり除いた非同期モデル (CORDA) が提案されている．これらのモデルに基づく研究として，一点集合問題 [4]〜[6] や形状形成問題 [4] の可解性に関する研究がある．

一点集合問題とは，任意の位置に配置されたロボット群があらかじめ決められていないある一点に集合する問題である．本稿では, [1] で研究されたリング上での自律分散ロボットの一点集合問題を解くための決定性アルゴリズムについて考える．リング上での一点集合問題とは，n 個のノードが均一に配置されたリング上において，いくつかのノードに任意に配置されたロボット群を，ノード間を移動させることで，あらかじめ決められていないある一つのノードに集合させる問題である．同様の問題を扱ったものに, [7]〜[10] がある．本稿では, [1] と同様のモデルを扱う．

この問題の難しさは，もし初期状況におけるロボットの配置が対称的な形状だった場合に，対称性を崩さなければいけないところにある．しかしながら，本質的に対称性を崩すことが不可能な初期形状が存在することが既に知られている. [1] では，一点集合が可能であるための初期形状の必要十分条件について考察されている．この研究では，リング上のロボット群の初期形状を，周期性，対称性と呼ばれる特徴により分類している．対称性はさらに，辺辺対称，点点ロボット対称，点点ノード対称，点辺対称等に分類される (図 1 参照)．既存の結果 [1] として，ロボットが奇数台の場合には，形状が周期的なときは一点集合不可能，非周期的なときは可能なことが示されている．偶数台の場合には，2 台のとき，または辺辺対称のときは，それぞれ集合不可能なことが示され，非周期的かつ非対称的な場合における一点集合アルゴリズムが示されている．これらのアルゴリズムはいずれも非同期モデルで動作する．

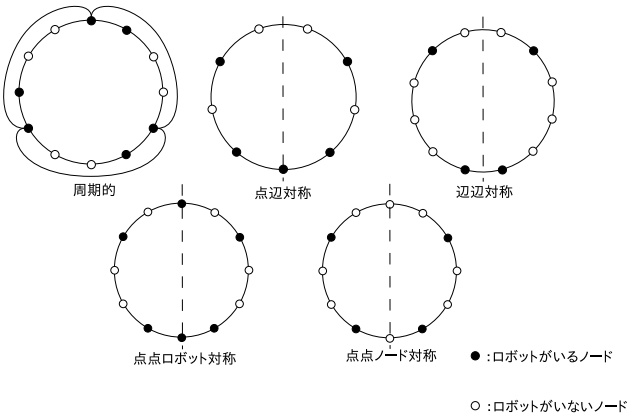


図 1　周期的な形状と対称的な形状

表 1　既存の結果と本稿の成果

周期性	対称性	ロボット台数	ノード数	一点集合可否	文献
在	任意	任意	任意	不可	[1]
無	任意	奇数	任意	可（非同期）	
	無	4 以上の偶数	任意	可（非同期）	
	無	2	任意	不可	
	辺辺	4 以上の偶数	偶数	不可	
	任意	4	5	不可	本稿
	点点ロボット	4 以上の偶数	偶数	可（非同期）	
	点点ノード	4	偶数	可（非同期）	
	点辺	4	奇数	可（準同期）	

本稿では，ロボットが 4 台のときの，点点対称，点辺対称な形状から準同期で動作する一点集合アルゴリズムを示す．これにより，準同期モデルにおいて 4 台の場合は完全に解決する．また，任意の偶数台のとき，点点ロボット対称な形状からの非同期で動作する一点集合アルゴリズムを示す（表 1）．

本稿の構成を以下に示す．**2.** 節では，本研究で扱うモデルについて述べる．**3.** 節では，既存の研究で示されたアルゴリズムを示す．**4.** 節では，ロボットの台数が 4 台での一点集合問題の非可解性を示し，それ以外の場合での，一点集合アルゴリズムを提案し，その証明を行う．**5.** 節では，ロボットが任意の偶数台のときの，点点ロボット対称な形状からの一点集合問題アルゴリズムを提案し，その証明を行う．**6.** 節では，結論と今後の課題について述べる．

2. 問題定義とモデル

この章では，リング上での一点集合問題と，本研究で扱うロボットのモデルについて述べる．ロボットのモデルは，基本的には [1] で扱われたモデルと同様だが，**2.4** 節のロボットの同期性では，制限を設けている．また，リング上でのロボット群の形状について，本研究での表現について述べる．

2.1 リング上での一点集合問題

一点集合問題とは，任意の位置に配置されたロボット群をあらかじめ決められていないある一点に集合させる問題である．ロボットの初期配置，移動を，n 個のノードが均一に配置されたリング上に制限する．いくつかのノードに分散配置されたロボット群を，ノード間を移動させることで，あらかじめきめられていないある一つのノードに集合させるのが，リング上での一点集合問題である．

2.2 ロボットのモデル

本研究で扱うロボットのモデルを以下に示す．

- ロボットは観測，計算，移動を 1 サイクルとして，これを繰り返す．詳細は **2.3** 節で述べる．
- ロボットは重複感知能力を持っている．本稿では，重複しているかどうかは感知できるが，重複しているノードに何台いるかは分からないとする．
- ロボットの移動は瞬間的に行われ，リング上での隣接ノードにのみ移動する．リングについての詳細は **2.5** 節で述べる．
- ロボットはほかのロボットを外見から区別することはできない．
- ロボットは過去のサイクルにおける情報を記憶しておくことができない．
- ロボットは通信能力を持っていない．
- すべてのロボットは同じ決定性アルゴリズムを実行する．

2.3 サイクルについて

ロボットの 1 サイクルにおける動作は次の状態からなる．

（1） **観測**

現在のロボット群の形状を観測し，アルゴリズムの入力とする．本論文のロボットの観測の詳細は 2.7 節で述べる．

（2） **計算**

ロボットは決定性アルゴリズムにしたがって計算する．以前のサイクルで得られたことを記憶することができないので，同一サイクルの観測で得られた結果のみを入力とし，隣接ノードに移動するかしないかを出力する．

（3） **移動**

アルゴリズムの出力に従い，隣接ノードに移動するか，同じノードにとどまる．

2.4 ロボットの同期性

非同期とは，1 サイクルにかかる時間は有限で各ロボットの観測，計算，移動が起きるタイミングに一切の仮定を置かない場合を言う．準同期で動作する各ロボットは，観測時刻が同期しており，観測時，ロボットが active ならその観測から始まるサイクルを行い，inactive ならそのサイクルを行わない．また，各時刻でロボットが active か inactive かは予測不能だが，無限に inactive が続くことはないと仮定する．

2.5 リングについて

リングについて以下に述べる．

- リングには n 個のノードが均一に配置されている．
- 各ノードに番号などは付いておらず，外見で区別できない．
- 各ロボットはリングの向きに関する知識が一致していない．
- 隣接するノード間の距離を 1 する．
- 初期状態では，各ノード上に高々1 台のロボットがいる．

2.6 形状の表現

リング上でのロボット群の形状は，整数の列の三つ組 $((a_1,...,a_r),(b_1,...,b_s)(c_1,...,c_s))$ で表す．整数 $a_i, b_j, c_{,k}$ は以下の意味を持つ．

a_i： ロボットが少なくとも 1 台いる任意のノードを選び u_1 とする．u_1 から時計回りに，少なくとも 1 台のロボットがいるノードの連続した列 $u_1, u_2, u_3, ..., u_r$ を考える．整数 $i < r$ について，a_i はリング上での，ノード u_i と u_{i+1} との時計回りでの距離をあらわし，整数 a_r は，ノード u_r と u_1 との時計回りでの距離を表す．

b_j： ノードに 2 台以上のロボットがいるとき，ロボットが重複しているという．$u_1, u_2, u_3, ..., u_r$ の中から，重複しているノードの列を考える．この列を $u_{v_1}, ..., u_{v_s}$ とする．整数 b_j は，ノード u_1 と u_{v_j} との時計回りでの距離を表す．

c_k： 重複しているノード u_{v_k} にいるロボットの数を表す．

異なる u_1 を選べば，異なる列の組ができる．この異なる u_1 により生じる各組は，ロボット群の形状は等しい．例えば，列の組 $((1,2,1,1,3),(1,3)(2,2))$ と，$((2,1,1,3,1),(0,2),(2,2))$ は，同じ形状を表している．本研究では簡単のため，任意のノードを u_1 とした列の組のみで表す．また，重複のない場合は，二つ目以降の列を除き，列 $(a_1,...,a_r)$ で表す．

2.7 ロボットの観測

ロボット R がいるノードから始まる形状を $((a_1,...,a_r),(b_1,...,b_s),(c_1,...,c_s))$ とする．R の観測結果は，二つの列のペア $\{((a_1,...,a_r),(b_1,...,b_s)),((a_r,a_{r-1},...,a_1),(n-b_s,...,n-b_1))\}$ となる．また，重複しているノードが無い場合は，観測結果は，二つ目の列を除いて，$\{(a_1,...,a_r),(a_r,...,a_1)\}$ と書く．各ロボットは自分がいるノードの左右に関する知識が一致していないためこのように表現する．例えば，ノード 1 から 9 の 9 ノードのリングで，3 台のロボットがノード $1,2,4$ にいるとする．このときノード 1 にいるロボット R の観測結果は $\{(1,2,6),(6,2,1)\}$ となる．

2.8 形状の定義

重複の無いロボット群の形状 $C=(a_1,...,a_r)$ について，その上の周期性，対称性を定義する（図 1）．

- 周期性

$(a_1,...,a_r)$ が，部分列 p の少なくとも二つの連結によって構成されているとき，C を周期的と呼ぶ．

- 対称性

ロボット群の配置が対称的になるような対称軸がリングにあるとき，C を対称的と呼ぶ．

– 辺辺対称

対称軸上にノードが無い場合の対称的な形状．ノード数，ロボット数がともに偶数のときのみこの形状になる可能性がある．

– 点点ロボット対称

点点対称で，対称軸上の二つのノードの両方にロボットがいる場合．

– 点点ノード対称

点点対称で，対称軸上の二つのノードの両方にロボットがいない場合．

– 点辺対称

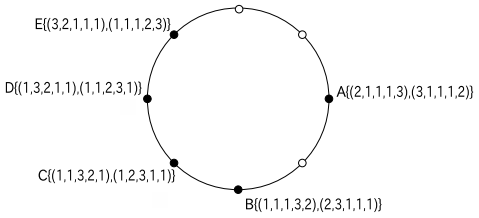


図 2　決定的な形状と，各ロボットの観測.

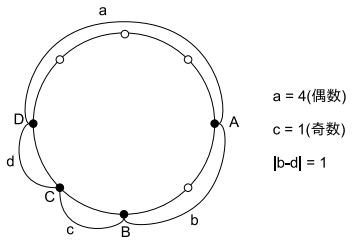


図 3　条件付き決定的な形状.

ノード数が奇数のときの対称的な形状．対称軸の一方にはノードがあるが，もう一方にはノードが無い．今回はロボットが偶数台の場合のみ考えるので，初期形状で点辺対称な場合は，軸上ノードにロボットはいない.

- 決定的

重複が無く，全てのロボットの観測結果が異なるとき，C は決定的である（図 2）．全てのロボットの観測結果が異なることから，ロボットに全順序をつけることができ，1 台を一意に決定することができる.

- 条件付き決定的

ロボットが 4 台，ノード数が偶数のときの決定的な形状 $C = (a, b, c, d)$ において，一般性を失わずに a が偶数，c が奇数で，$a - c > 2$ とする．このとき $|b - d| = 1$ となるなら，C を条件付き決定的と呼ぶ（図 3）.

上記の形状について，以下の補題が成り立つ [1].

［補題 1］ 重複の無い形状において，周期的でも対称的でもないとき，そのときに限り決定的である.

［補題 2］ 重複の無い形状において，非周期的かつ，非決定的であるとき，ただ一つの対称軸を持つ.

3. 既存の結果

以下の定理 1~4 は [1] で示された定理である.

［定理 1］ 任意の周期的な形状から，一点集合不可能である.

［定理 2］ ロボットが 2 台のとき，任意の形状から一点集合不可能である.

［定理 3］ 任意の辺辺対称な形状から，一点集合不可能である.

［定理 4］ 周期的でないとき，ロボット台数が奇数または，4 以上の偶数で対称性がないなら一点集合可能である.

本稿では，ロボット台数が 4 以上の偶数の場合のアルゴリズムを示すが, [1] で用いられるアルゴリズム Single-Multiplicity-Gathering（SMG）と Rigid-Gathering（RG）を利用する.

[1] のアルゴリズムのアイデアは，重複のない初期形状から，ちょうど一つ重複を作り出す．形状が周期的でも対称的でもなければ，補題 1 より形状は決定的になる．決定的な形状から重複を一つ作り出すアルゴリズムが RG である．一つの重複から一点集合させるアルゴリズムが SMG である.

3.1 SMG と RG

SMG と RG は非同期で動作し，同時に移動するロボットの台数が一台であるようなアルゴリズムになっている.

3.1.1 SMG

形状に重複がひとつある場合は，SMG を用いて一点集合する．このアルゴリズムでは，重複に 1 番近いロボットのうち一台を決定し，そのロボットのみが重複に向かって移動していく．こうすることで，重複の単一性をたもったまま，各ロボットを重複点に向かわせ，一点集合することができる.

3.1.2 RG

形状が決定的な場合は，RG を用いて一点集合する．このアルゴリズムでは，最大距離で隣り合っているロボットのペアを，一意に決定することが可能である．そのペアのうち，もう一方の隣のロボットとの距離が短い方の 1 台が，最大距離を伸ばすように移動していく．このとき，移動途中の任意の形状が周期的，対称的にならないことが保証されるため，決定的な形状を崩さずに重複を一つ作ることができる．重複ができた後は，SMG を用いて一点集合を達成できる.

4. ロボット 4 台での一点集合問題の非可解性と可解性

ロボットが 4 台でノード数が 5 のときの非可解性を示し，その後，それ以外の場合で，形状別に一点集合可能なアルゴリズムとその正当性を示す．最後に，ロボットが 4 台で一点集合可能な場合の，一点集合アルゴリズムを示す．また，以下では，一点集合問題を $P(S, R, N, C)$ で表すこととする．S はロボットが非同期か準同期か，R はロボットの台数，N はノード数，C は形状をそれぞれ表す．例えば，非同期で動作するロボット 8 台，ノード数 20，決定的形状からの一点集合問題は，P(非同期, 8, 20, 決定的) と表す.

4.1 ロボット 4 台での一点集合問題の非可解性

ロボット 4 台でノード数 5 の場合，初期形状では，ロボットは重複していないため，形状は常に点辺対称で，ロボットは対称軸上のノード以外の 4 つのノードにいることになる．以下，一点集合問題 P(準同期, 4, 5, 任意)(= P(準同期, 4, 5, 点辺対称)) の非可解性を示す．そのために，まず次の補題を示す.

［定理 5］ 一点集合問題 P(準同期, 任意, 任意, 任意) について，ロボットがいるノードが二つのみになり，かつその 2 ノードがともに重複している場合，いかなるアルゴリズムも一点集合不可能である.

［証明 ］ ある一つのノードに複数のロボットが重複している場合，どのようなアルゴリズムを用いてもその複数のロボットが同じ動きをするスケジュールが考えられる．従って，それらのロボットは 1 台のロボットと同一とみなすことができ，ロボットがいるノードが二つのみの場合は，ロボットが 2 台の場合と同一視できる．補題 2 より，ロボットが 2 台のときは，任意の形状から一点集合不可能なため，ロボットがいるノードが二つのみの場合も，一点集合不可能となる． □

次に，一点集合問題 P(準同期, 4, 5, 点辺対称) の非可解性を示す.

［補題 3］ 一点集合問題 P(準同期, 4, 5, 点辺対称) は非可解である.

［証明 ］ まず，軸上ノードに隣接する 2 台のロボットが軸上ノードへ移動するアルゴリズムを考える．この場合，2 台が同時に移動すれば重複が一つできて集合可能となるが，常に片方

if R が軸上
then 移動しない
else if 軸上ノードとの距離が，軸上でないロボットと比べて最小
　　then 軸上ノードとの最小距離が縮まる様に, 軸上
　　ノードに向かって移動
　　else 移動しない

図 4　アルゴリズム　4 台点点 Gathering

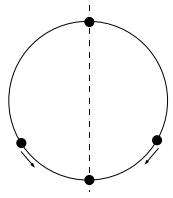


図 5　4 台点点 Gathering の動作例（点点ロボット対称）

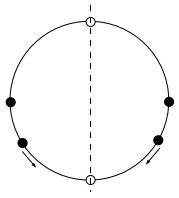


図 6　4 台点点 Gathering の動作例（点点ノード対称）

のみが移動するスケジュールも考えられる．その場合，永遠に点辺対称が続くことになり，一点集合は不可能となる．従って，初期状態からの移動で，この動きを含んだアルゴリズムは使えない．次に，対称軸をまたいで隣接しているロボットが，対称軸をまたぐように移動するアルゴリズムを考える．この場合 2 台が同時に移動しつづけて，場所を交換し続けるスケジュールが存在するため，一点集合不可能である．従って，この動きを含んだアルゴリズムも使えない．残るのは，対称軸をまたいで隣接している 2 台が軸上ノードに近づく，もしくは，軸上ノードに隣接している 2 台が軸上ノードから離れる，またはこの両方を実行するアルゴリズムのみである．どちらかのみの場合は，2 台が同時に移動すると二つの重複のみになってしまい，一点集合不可能となる．両方の場合は，全てが同時に移動すると場所を交換し続けるのみになってしまい，これも一点集合不可能となる．以上の議論より，どのようなアルゴリズムを用いても，一点集合できないスケジュールが存在する．従って一点集合問題 P(非同期, 4, 5, 点辺対称) はいかなるアルゴリズムでも解くことができない．　□

4.2　ロボット 4 台での一点集合問題の可解性

ロボット 4 台での一点集合問題の可解性を形状別に示す．以降に示すアルゴリズムにおいて，そのアルゴリズムを実行しているロボットを R とする．

4.2.1　点点対称時のアルゴリズムとその正当性

一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点対称) に対して実行するアルゴリズム 4 台点点 Gathering を図 4 に示す．このアルゴリズムでは，軸上にいるロボットは移動せず，それ以外のロボットで軸に一番近いロボットが軸に近づくように移動する．図 5,6 に，点点ロボット対称，点点ノード対称それぞれにこのアルゴリズムを実行したときの動作例を示す．以下，アルゴリズム 4 台点点 Gathering に関する補題，定理を示す．

［補題 4］　一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点対称) に対して 4 台点点 Gathering を実行すると，移動するロボットは高々2 台である．

［証明 ］　まず，点点ロボット対称の場合を考える．この場合，2 台のロボットは軸上ノードにいるので移動しない．よって，移動する可能性があるのは，軸上ノードにいない 2 台のロボットのみである．次に，点点ノード対称の場合を考える．この場合，軸上ノードにいないロボットは 4 台である．このとき，この 4 台全ての軸上ノードとの最小距離が等しいと仮定する．するとある距離 a_1, a_2 について，形状は $C = (a_1, a_2, a_1, a_2)$ と表され，周期的になるため，非周期的という前提に矛盾する．よって，4 台全ての軸上ノードとの最小距離が等しいことはなく，軸上ノードとの距離が最小となるのは 2 台のみとなる．

以上より，移動するロボットは高々2 台である．　□

点点ロボット対称時における補題を以下に示す．

［補題 5］　一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点ロボット対称) に対して 4 台点点 Gathering は，重複を一つ作るか，点点ロボット対称または決定的な形状に遷移する．

［証明 ］　点点ロボット対称時の形状を $C = (a_1, a_2, a_2, a_1)$ とし，一般性を失わずに $a_1 > a_2$ とする．補題 4 より，このとき移動するのは最大 2 台である．まず，$a_2 = 1$ の場合を考える．このとき，移動する可能性のある 2 台のロボットのうち，1 台でも動けば重複が一つできる．次に，$a_2 > 1$ の場合を考える．2 台のロボットが同時に移動すると，形状は $C' = (a_1 + 1, a_2 - 1, a_2 - 1, a_1 + 1)$ となり，対称軸の位置は等しい点点ロボット対称である．2 台のロボットのうち片方のみが移動した場合，形状は $C' = (a_1 + 1, a_2 - 1, a_2, a_1)$ となり，これは決定的である．　□

ロボットが非同期で動作する場合に注意して，以下の補題を示す．

［補題 6］　一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点ロボット対称) を解くアルゴリズムが存在する．

［証明 ］　補題 5 より，一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点ロボット対称) に対して 4 台点点 Gathering を用いると，重複が一つできるか，点点ロボット対称になるか，決定的になる．まず，重複が一つできた場合を考える．移動する可能性のある 2 台が同時に移動して重複した場合は問題ないが，1 台のみが移動して重複した場合，移動していない 1 台 (仮に α とする) がすでに計算を終えており，移動する前かもしれない．しかし，SMG を用いると，次に移動するのは，重複しているノードに隣接している α である．α は重複ノードに移動し，重複は一つのみとなり，SMG を使って集合可能となる．次に，点点ロボット対称になった場合を考える．この場合，移動した 2 台のロボットは，対称軸との距離をより縮めたことになり，再び 4 台点点 Gathering を用いれば，また同じ 2 台のロボットのみが移動することになる．よって，点点ロボット対称が続けば，最終的に軸上で重複が一つでき，集合可能になる．最後に，決定的になった場合を考える．もとの点点ロボット対称な形状を $C = (a_1, a_2, a_2, a_1)$ とすると，このときの形状は一般性を失わず $C' = (a_1 + 1, a_2 - 1, a_2, a_1)$ とできる．この場合も，1 台が移動して重複ができた場合と同様，移動していない 1 台 (仮に β とする) がすでに計算を終えており，移動する前かもしれない．このとき RG を用いると，最大距離 $a_1 + 1$ を伸ばすように同じロボット再びが移動していき，元の対称軸上で重複が一つできる．もし RG での移動の前に β が移動したとしても，対称軸までの距離が縮まった点点ロボット対称となり，再び 4 台点点 Gathering を用いていけば集合可能となる．　□

点点ノード対称における補題を以下より示す．

［補題 7］　一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点ノード対称) に対して 4 台点点 Gathering を実行すると，重複が 1 つできるか，点点ノード対称または条件付き決定的な形状となる．

［証明 ］　点点ノード対称時の形状を $C = (a_1, a_2, a_3, a_2)$ とし，一般性を失わずに $a_1 > a_3$ とする．補題 4 より，このとき移動

条件付き決定的な形状 $C = (a, b, c, d)$ において，
一般性を失わずに a が偶数，c が奇数で
$a - c > 2, b - d = 1$ とする．
N = 隣のロボットとの距離が d と c のロボット．

if R = N
then 距離 d を伸ばすように移動
else 移動しない

図 7　アルゴリズム　条件付き決定的 Gathering

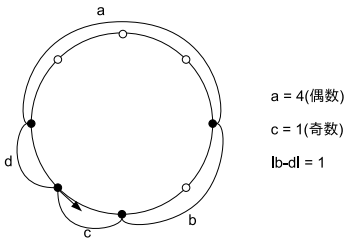


図 8　条件付き決定的 Gathering の動作例.

するのは最大 2 台である．まず，$a_3 = 2$ の場合を考える．このとき，移動する可能性のある 2 台のロボットが同時に移動すると，重複が一つできる．2 台のうち片方のみが移動した場合，形状は $C' = (a, a_2 + 1, 1, a_2)$ となり，これは条件付き決定的な形状である．次に，$a_3 > 2$ の場合を考える．2 台が同時に移動すると，形状は $C' = (a, a_2 + 1, a_3 - 2, a_2 + 1)$ で，これは点点ノード対称である．2 台のうち片方のみが移動した場合，形状は $C' = (a, a_2 + 1, a_3 - 1, a_2)$ で，これは条件付き決定的な形状である．　□

補題 7 より，一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点ノード対称) に対してアルゴリズム 4 台点点 Gathering を用いたときに条件付き決定的な形状になりうる．このとき，アルゴリズム条件付き決定的 Gathering（図 7）を用いて集合する．ロボット群の非同期性による集合不可能なスケジュールを避けるために，4 台点点 Gathering を実行したときに移動する可能性はあったが，移動しなかったロボットのみに，4 台点点 Gathering と同様の移動をさせるのがアルゴリズム条件付き決定的 Gathering である．動作例を 8 に示す．ここで，アルゴリズム条件付き決定的 Gathering に関する補題を示す．

［補題 8］　任意の条件付き決定的な形状に対して条件付き決定的 Gathering を実行すると，点点ノード対称になるか重複が一つできる．

［証明 ］　条件付き決定的な形状を (a, b, c, d) とし，一般性を失わずに a が偶数，c が奇数で $a - c > 2$，$b - d = 1$ とできる．これに条件付き決定的 Gathering を実行すると，形状は $(a, b, c - 1, d + 1)$ となる．このとき，a は偶数，$c - 1$ は偶数で，$a \neq (c - 1)$，$b = d + 1$ となる．ここで，$c - 1 > 0$ ならば，対称軸が距離 a，$c - 1$ の部分を通る点点ノード対称である．$c - 1 = 0$ ならば，重複が一つできたことになる．　□

ロボットが非同期で動作する場合の，4 台点点 Gathering と条件付き決定的 1Gathering のつなぎ部分に注意して，以下の補題を示す．

［補題 9］　一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点ノード対称) を解くアルゴリズムが存在する．

［証明 ］　補題 7 より，一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点ノード対称) に対して 4 台点点 Gathering を用いると，重複が 1 つできるか，点点ノード対称になるか，条件付き決定的な形状になる．まず，重複がひとつできた場合を考える．これは，移動する可能性のある 2 台が同時に移動したときで，SMG を用いれば集合可能である．次に，点点ノード対称になった場合を考える．この場合は，移動した 2 台のロボットは，対称軸との最小距離を縮めたことになり，再び 4 台点点 Gathering を用いれば，また同じ 2 台のロボットのみが移動することになる．よって，点点ノード対称が続けば，最終的に軸上で重複が一つでき，集合可能になる．最後に，条件付き決定的な形状になった場合を考える．これは，移動する可能性のある 2 台のロボットのうち 1 台のみ移動した場合で，移動していない 1 台 (仮に γ とする) がすでに計算していて，移動する前かもしれない．このとき条件付き決定的 Gathering を用いると，γ が移動することになり，どちらにしても 2 台のロボットと対称軸との最小距離が元よりも縮まった点点ノード対称となり，これを繰り返していけば，軸上で重複が一つでき，集合可能になる．　□

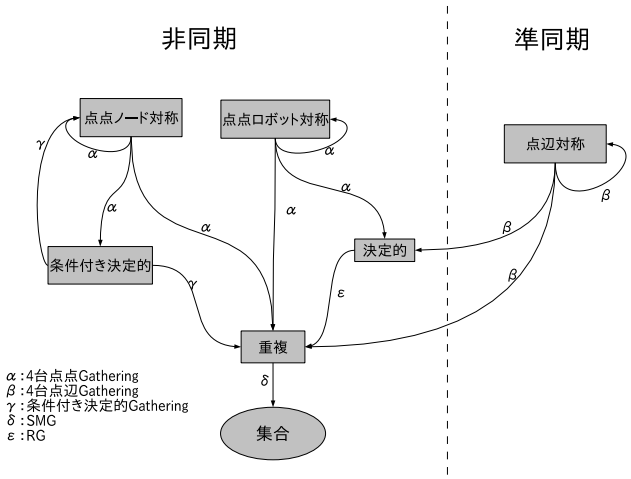


図 9　アルゴリズムの遷移図.

補題 6，9 より，以下の定理が得られる．

［定理 6］　一点集合問題 P(非同期, 4, 偶数, 非周期的かつ点点対称) を解くアルゴリズムが存在する．

点点対称な形状から一点集合するまでの遷移図を図 9 に示す．

4.2.2　点辺対称時のアルゴリズムとその正当性

点点対称な形状の場合は非同期のアルゴリズムを示したが，点辺対称な形状に対しては，準同期のアルゴリズムを示す．一点集合問題 P(準同期, 4, 6 以上, 非周期的かつ点辺対称) に対して実行するアルゴリズムを図 10 に示す．このアルゴリズムでは，同時に移動するのは高々2 台で，各ロボットは軸上のノードに向かって移動する．2 台が同時に移動する場合は対称性を保ったまま重複を一つ作るが，1 台のみ移動する場合は最終的に決定的な形状になるように移動する．このアルゴリズムを実行したときの動作例を図 11 に示す．また，点辺対称な形状から一点集合するまでの遷移図を図 9 に示す．以下に，アルゴリズム 4 台点辺 Gathering に関する定理を示す．

［定理 7］　一点集合問題 P(準同期, 4, 6 以上, 非周期的かつ点辺対称) を解くアルゴリズムが存在する．

［証明 ］　一点集合問題 P(準同期, 4, 6 以上, 非周期的かつ点辺対称) に対して 4 台点辺対称 Gathering を実行したときを考える．点辺対称時の形状を $C = (a_1, a_2, a_3, a_2)$ とし，一般性を失

点辺対称な形状 $C = (a_1, a_2, a_3, a_2)$ において，
軸は a_1, a_3 部分を通り，a_1 は奇数，a_3 は偶数とする．
P = 隣のロボットまでの距離が a_1，a_2 のロボット (2 台).
Q = 隣のロボットまでの距離が a_3，a_2 のロボット (2 台).

if ノード数が 5 **then** 出力：集合不可能
else if (a_2 が奇数)$\wedge$($a_2 \neq 1$)
　then if R=P
　　then a_1 を伸ばす方向に移動
　　else 移動しない
　else if R=Q
　　then a_3 を縮める方向に移動
　　else 移動しない

図 10　アルゴリズム　4 台点辺 Gathering

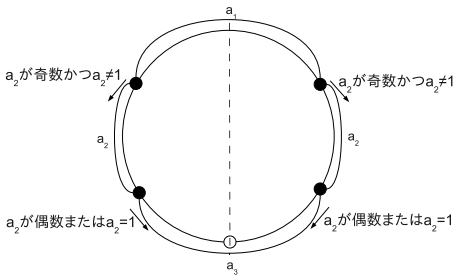


図 11　4 台点辺 Gathering の動作例.

わずに a_1 が奇数，a_3 が偶数とする．まず，距離 a_1 を伸ばすようにロボットが移動するのは，$a_2 > 1$ のときのみのため，重複ができないことがわかる．よって，重複ができるのは，$a_3 = 2$ で，距離 a_3 を縮めるように 2 台が同時に移動するときのみである．この場合，SMG を用いれば集合可能となる．これ以外の場合について，1. 2 台が同時に移動する場合と，2. 1 台のみが移動する場合の二通りに分けて考える．

（1） 2 台が同時に移動する場合.

a_2 が奇数かつ $a_2 > 1$ のとき，距離 a_1 を伸ばすように 2 台が移動し，形状は $C' = (a_1 + 2, a_2 - 1, a_3, a_2 - 1)$ となり，点辺対称となる．それ以外のときは距離 a_3 を縮めるように 2 台が移動し，形状は $C' = (a_1, a_2 + 1, a_3 - 2, a_2 + 1)$ となり，点辺対称となる．これらの場合，ロボットがより対称軸上のノードに近づいた点辺対称となっており，これが続けば，対称軸上で重複ができ，SMG で集合可能となる．

（2） 1 台のみが移動する場合.

（a） $a_1 = a_3 - 1$ のとき.

a_2 が奇数かつ $a_2 > 1$ のとき，距離 a_1 を伸ばすように 1 台が移動し，形状は一般性を失わず $C' = (a_1 + 1, a_2 - 1, a_3, a_2)$ となる．これは，対称軸が距離 $a_2 - 1$ 距離 a_2 の部分を通る点辺対称である．この形状は，次に述べる (b) の場合に含まれる．それ以外のときは，距離 a_3 を縮めるように 1 台が移動し，形状は一般性を失わず $C' = (a_1, a_2 + 1, a_3 - 1, a_2)$ となる．この時 a_2 が偶数だったら，(b) の場合に含まれ，$a_2 = 1$ だったら，$a_2 + 1$ が対称軸をまたぐ偶数距離の点辺対称となり，同じく (a) の場合に含まれるが，a_1 が奇数かつ $a_1 > 1$ のため，次は距離 a_2 を伸ばすように移動し，(b) に含まれる．

（b） $a \neq a_3 - 1$ のとき.

a_2 が奇数かつ $a_2 > 1$ のとき，距離 a_1 を伸ばすように 1 台が移動し，形状は一般性を失わず $C' = (a_1 + 1, a_2 - 1, a_3, a_2)$ となり，決定的となる．それ以外のときは距離 a_3 を縮めるように 1 台が移動し，形状は一般性を失わず $C' = (a_1, a_2 + 1, a_3 - 1, a_2)$ となり，決定的となる．これらの場合は，ともに RG を用いて集合可能となる．

以上の議論より，一点集合問題 P(準同期, 4, 6 以上, 非周期的かつ点辺対称) に対して 4 台点辺 Gathering を実行すると，集合可能となる．　□

if 周期的または辺辺対称 **then** 出力：集合不可能
else if 重複を一つ持つ
　then SMG
　else if 条件付き決定的
　　then 条件付き決定的 Gathering
　　else if 決定的 **then** RG
　　　else if 点点対称
　　　　then 4 台点点 Gathering
　　　　else 4 台点辺 Gathering

図 12　アルゴリズム　4 台 Gathering

4.2.3　ロボットが 4 台のときのアルゴリズムとその正当性

一点集合問題 P(準同期, 4, 6 以上, 非周期的かつ非辺辺対称) に対して実行するアルゴリズム 4 台 Gathering を図 12 に示す．このアルゴリズムは，これまでの形状別のアルゴリズムを使い，準同期で動作するロボット 4 台での一点集合問題を解く．

［定理 8］ 一点集合問題 P(準同期, 4, 6 以上, 非周期的かつ非辺辺対称) に対して，アルゴリズム 4 台 Gathering を実行すると一点集合する．

［証明 ］ まず，周期的なときは不可能．重複を一つ持つ場合は SMG を用いて集合できる．条件付き決定的な形状のときは，補題 8 より，条件付き決定的 Gathering を用いて点点ノード対称になる．その他の決定的のときは RG を用いて集合できる．ここで，補題 2 より，残りは対称的な形状のみとなる．点点対称のときは定理 6 より，4 台点点 Gathering を用いて集合できる．最後に，点辺対称のときは定理 7 より，4 台辺 Gathering を用いて集合できる．以上より，一点集合問題 P(準同期, 4, 6 以上) において，非周期的な任意の形状に対して，アルゴリズム 4 台 Gathering を実行すると一点集合する．　□

5. 6 台以上のロボットに対する一点集合可解性

一点集合問題 P(非同期, 6 以上の偶数, 偶数, 非周期的かつ点点ロボット対称) に対するアルゴリズム点点ロボット Gathering を図 13 に示す．このアルゴリズムでは，軸上のロボットを移動させ対称性を崩すが，そのとき，周期的な形状または，対称的な形状にならないことを以下の補題で示す．その際，ある形状 $C = (a_1, ..., a_r)$ について，a_i の重みとは，$(a_1, ..., a_r)$ 中の a_i の数とする．

［補題 10］ ロボット台数偶数で，ノード数偶数の非周期的かつ，点点ロボット対称な形状に対して，軸上のロボットのうち，どちらの 1 台を移動させても，周期的な形状にはならない．

［証明 ］ まず，軸上ノードの隣のノードにロボットがいる場合は，軸上のロボットが移動すれば重複が一つできるため，この場合は明らかに周期的でない．これ以外の場合について背理法により示す．軸上のロボットのうちどちらか 1 台を動かした結果，周期的になると仮定する．このとき，ロボット R は反時計回りに一つ移動したと仮定し，移動した方の軸上のノードを q

W_A ：軸上のロボット A の，隣のロボットとの距離の重み
$D(R1, R2)$：ロボット $R1, R2$ 間の距離

if ロボットが 2 台　**then** 出力：集合不可能
else if R が軸上
　then M =R
　　N = もう一方の軸上ロボット
　　if $W_M = W_N$
　　then $j = 1$
　　$M_1 = M$ の隣のロボット
　　$N_1 = N$ の隣のロボット，
　　$M_0 = M, N_0 = N$
　　while $(D(M_j, M) = D(N_j, N))$
　　$j = j + 1$
　　$M_j = M_{j-2}$とは異なる
　　M_{j-1} の隣のロボット
　　$N_j = N_{j-2}$ とは異なる
　　N_{j-1} の隣のロボット
　　if $D(M_j, M) > D(N_j, N)$
　　then 移動
　　else if $W_{R1} > W_{R2}$
　　then 隣接ノードに移動
　　else 移動しない

図 13　アルゴリズム　点点ロボット Gathering

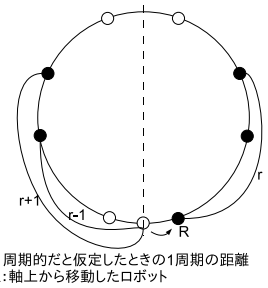


図 14　点点ロボット対称から，軸上のロボットが移動したときの形状

とし，1 周期の距離を r とする．このとき，ノード q から，時計回りに距離 $r-1$ のノードと，反時計回りに距離 $r+1$ のノードにロボットがいるはずである．この時の状況を図示すると 14 のようになる．元々，点点ロボット対称だったため，ノード q から時計回りに距離 $r+1$ のノードと，反時計回りに $r-1$ のノードにもロボットがいるはずである．周期的なので，ロボット R から時計回りに距離 2 のノードにロボットがいるはずである．しかし，そのノードは軸上ノードの隣であり，前提よりこのノードにはロボットはいない．よって矛盾し，周期的にはならない．　□

［補題 11］　ロボット台数偶数で，ノード数偶数の非周期的かつ，点点ロボット対称な形状に対して，軸上のロボットのうち，どちらの 1 台を移動させても，対称的な形状にはならない．

［証明 ］　点点ロボット対称な形状では，ロボットがいるノードを通る軸に対して対称的なため，ロボット間の距離の重みはすべて偶数のはずである．これを利用してまず，点点ロボット対称な形状にならないことを示す．点点ロボット対称な形状において，軸上のロボット R の，隣のロボットとの距離を a とする，ロボット R が隣へ移動したとすると，R と隣のロボットとの距離は，$a+1$ と $a-1$ となる．つまり，形状全体としては，距離 a が二つ減り，距離 $a+1$ と $a-1$ が一つずつ増えたことになる．このとき，$a+1$ と $a-1$ の重みは奇数となるため，点点ロボット対称にはならない．次に，点点ノード対称にならないことを示す．もし点点ノード対称になるとしたら，$a+1$ と $a-1$ の重みが奇数なため，対称軸は必ずこの二つの距離をとるロボット間を通らなければならない．しかし，アルゴリズムに従うと，隣のロボットとの距離の重みが大きい方の軸上のロボットが移動する．すると，移動したロボットがいたノードではなく，もう一方の軸上のロボットとその隣のロボットとの距離を考えると，元の対称軸以外の軸を持つ対称的な形状にはなり得ない．よって，点点ノード対称にはならない．これは，辺辺対称に対しても同様に言えるため，辺辺対称にもならない．　□

以上の補題より，次の定理が成り立つ．

［定理 9］　一点集合問題 P(非同期, 偶数, 偶数, 非周期的かつ点点ロボット対称) を解くアルゴリズムが存在する．

［証明 ］　補題 10 と補題 11 より，アルゴリズム点点ロボット Gathering を用いて軸上のロボットが移動すると，非周期的かつ非対称的な形状になる．すなわち，決定的な形状もしくは，ただ一つの重複ができる．よって，RG もしくは SMG を用いて一点集合可能となる．　□

6. おわりに

本稿では，ロボットが 4 台のときの，点点対称，点辺対称な形状から準同期で動作する一点集合アルゴリズムを示した．これにより，準同期で動作する 4 台の場合は完全に解決した．また，任意の偶数台のときの，点点ロボット対称な形状からの非同期で動作する一点集合アルゴリズムを示した．

4 台で点辺対称の場合は，準同期のときのアルゴリズムしか示していないので，今後の課題としては，4 台で非同期のときの完全な可解性，アルゴリズムの提案が挙げられる．また，6 台以上の偶数台については点点ロボット対称についてしか示していないので，任意の形状に対する可解性の証明，アルゴリズムの提案などが挙げられる．

文　献

[1] R. Klasing, E. Markou and A. Pelc: “Gathering asynchronous oblivious mobile robots in a ring”, Lecture Notes in Computer Science, **4288**, No. 3, pp. 744–753 (2006).

[2] I. Suzuki and M. Yamashita: “Distributed Anonymous Mobile Robots: Formation of Geometric Patterns”, SIAM Journal on Computing, **28**, 4, pp. 1347–1363 (1999).

[3] G. Prencipe: “Corda: Distributed Coordination of a Set of Autonomous Mobile Robots”, 4th European Research Seminar on Advances in Distributed Systems (ERSADS 2001), pp. 185–190 (2001).

[4] G. Prencipe: “Distributed coordination of a set of autonomous mobile robots”, PhD thesis,Universita di Pisa (2002).

[5] P. Flocchini, G. Prencipe, N. Santoro and P. Widmayer: “Gathering of asynchronous oblivious robots with limited visility”, STACS 2001, pp. 247–258 (2001).

[6] M. Cieliebak and G. Prencipe: “Gathering autonomous mobile robots”, SIROCCO 2002, pp. 57–72 (2002).

[7] G. D. Marco, L. Gargano, E. Kranakis, D. Krizanc and U. V. A. Pelc: “Asynchronous deterministic rendezvous in graphs”, Theoretical Computer Science, **355**, pp. 315–326 (2006).

[8] A. Dessmark, P. Fraigniaud, D. Kowalski and A. Pelc: “Deterministic rendezvous in graphs”, Algorithmica, **46**, pp.

69–96 (2006).

[9] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro and C. Sawchunk: "Multiple mobile agent rendezvous in a ring", Lecture Notes in Computer Science, **2976**, pp. 599–608 (2004).

[10] D. Kowalski and A. Pelc: "Polynomial deterministic rendezvous in arbitrary graphs", Lecture Notes in Computer Science, **3341**, pp. 644–656 (2005).

観測時に一様な誤差が生じるセンサーを持つ自律分散ロボット群の一点収束について

山本 健太 † 泉 泰介 †† 片山 喜章 †† 犬塚 信博 †† 和田 幸一 ††

† †† 名古屋工業大学大学院工学研究科情報工学専攻

〒 466-8555 名古屋市昭和区御器所町

† kenken@phaser.elcom.nitech.ac.jp †† {t-izumi,katayama,inuzuka,wada}@nitech.ac.jp

あらまし 一点収束問題は，平面上に配置されたロボット群をある決められていない一点へ自律的に収束させる問題であり，自律分散ロボット群の代表的な協調問題である．近年，ロボットの観測能力を弱めたモデルの一つである，誤差を含む観測モデル上での一点収束問題の可解性が研究されている [1]．本研究では，論文 [1] で提案されたモデルに新たな仮定を加えた一様誤差モデルを定義し，その上での一点収束問題の可解性について考察する．論文 [1] のモデルでは，最大角度誤差が 60° 以上のときは一点収束不可能であることに対し，本研究で提案する一様誤差モデルでは，最大角度誤差が 90° 未満の場合に一点収束可能である．また，最大角度誤差が 90° 以上の場合に一点収束不可能であることを併せて証明する．このことは，本研究の結果が，許容角度誤差の点において最適であることを意味している．

1. まえがき

近年，自律分散システムの一つとして，自律分散ロボット群の研究が盛んに行わなわれている．自律分散ロボット群は高い耐故障性，柔軟性を持つ．そのことから，深海や宇宙空間，危険地帯などの，人による制御が困難な場所での利用が考えられている．

自律分散ロボット群の二次元空間における協調問題を扱った研究として，鈴木，山下らによる研究 [2], [3] がある．この研究で自律分散ロボット群の理論モデルが提案された．このモデルの上で様々な協調問題の可解性が研究されている．

協調問題の一つに一点集合問題がある．これはロボットをあらかじめ決められていない一点に集合させる問題である．一般的なロボットの理論モデルでは，各ロボットは自分の現在位置を原点とする独自の座標系を持ち，観測により他のロボットの位置を知ることが可能である．しかし，各ロボットの座標系の方向が一致しないモデルでは，一点集合が不可能であることが証明されている [4]〜[6]．そこで，条件を緩めた一点収束問題が考えられている．一点集合問題がすべてのロボットを正確に一点に集める必要があるのに対し，一点収束問題では任意の二台のロボット間の距離を 0 に収束させることのみが要求される．一点収束問題は各ロボットが持つ座標系の方向が一致しないモデルにおいて可解であることが知られている [7]．そのため，より弱いモデルでの可解性についての研究が行われている．文献 [1] では，ロボットの観測が誤差を含むモデルを定義し，その上での一点収束問題の可解性について議論している．このモデルでは，観測誤差を距離誤差率と角度誤差の 2 つのパラメータによりモデル化している．それぞれの誤差パラメータには上限値 ϵ_0 と θ_0 が定義されており，この上限値が大きいほど，各ロボットの観測時に生じうる誤差の範囲が大きくなる．論文 [1] では $\theta_0 \geqq 60°$，ロボット数 3 台以上の場合における一点収束の不可能性，および，$\sqrt{2(1-\epsilon_0)(1-\cos\theta_0+\epsilon_0^2)}$ が 0.2 未満の場合での一点収束の可能性が示されている．

表 1　既存結果と本論文の結果

モデル	距離誤差	角度誤差	一点収束
一様でない誤差モデル (論文 [1])	任意の ϵ_0	$\theta_0 \geqq 60°$	不可能
	$0.2 > \sqrt{2(1-\epsilon_0)(1-\cos\theta_0+\epsilon_0^2)}$		可能
	その他		未解決
一様な誤差モデル	$\epsilon_0 < 1$	$\theta_0 < 90°$	可能 (本研究)
	$\epsilon_0 \geqq 1$	$\theta_0 < 90°$	未解決
	任意の ϵ_0	$\theta_0 \geqq 90°$	不可能 (本研究)

論文 [1] のモデルでは，あるロボットが他のロボットを観測するとき，観測対象となるそれぞれのロボットに対して，異なる度合いの誤差が生じうる．しかしながら，現実のシステムにおいては，観測誤差は観測対象としている個々のロボットに依存して決まるのではなく，観測するロボットが備えている観測デバイスのエラーにより生じることが多い．この意味において，観測誤差の度合いが観測対象によって異なることを許容する [1] のモデルは必ずしも現実的ではない．上記の理由から，本研究では新たに，一様な誤差モデルを提案し，その上での一点収束の可解性について考察する．提案するモデルは論文 [1] と同様に誤差上限 ϵ_0，θ_0 が導入されているが，1 回の観測における誤差の度合いはすべての観測対象について等しいことが仮定されている．これは，論文 [1] で提案されたモデルの観測誤差に関する仮定を強めたモデルである．本論文では，このモデル上で，$\theta_0 \geqq 90°$ である場合の一点収束の不可能性，および，$0 \leqq \epsilon_0 < 1, \theta_0 < 90°$ における一点収束の可能性を示す．

本論文の構成を以下に示す．第 2 章では，本論文で扱われるロボットのモデルについて述べる．第 3 章では，$\theta_0 \geqq 90°$ の場合での不可能性，第 4 章では，$0 \leqq \epsilon_0 < 1, \theta_0 < 90°$ の場合での，一点収束アルゴリズムと，そのアルゴリズムの正当性ついて述べる．

2. ロボットモデル

本論文で仮定するロボットの動作モデルは [2], [3] のモデルを基とし，その上に観測誤差の振るまいを導入している．本研究で扱うロボットのモデルの概要を示す．

- ロボットは点として扱い体積は持たない．
- ロボットは二次元平面上を自由に移動できる．

- ロボットは外見で他のロボットを区別できない.
- ロボット同士で通信できない.
- ロボットはそれぞれ独自の座標系, 単位距離を持つ.
- ロボットは過去の情報を覚えておくことができない.
- すべてのロボットは同じアルゴリズムを実行する.
- すべてのロボットは誤差の情報として ϵ と θ_0 を持つ.

以下にモデルの詳細について述べる.

2.1 システムモデル

n 台で構成されるロボットシステム $S = \{s_0, s_1, \cdots, s_{n-1}\}$ を考える．本論文では，ロボットシステムを，平面上を離散的に移動する点の集合としてモデル化する．システムは離散時間 $0, 1, 2\cdots$ に基づき動作する．ロボットの位置をシステム全体で一貫して定義するために，平面上における大域的な座標系を仮定する．また，この座標系における各ロボットの座標を，大域座標と呼び，個々のロボット独自の座標系における座標と区別する．なお，大域座標は論文における説明のために導入している概念であり，各ロボットはその値を知ることはできない．各時刻において，ロボットは行動，停止のいずれかの状態を取る．ある時刻 t において，行動状態にあるロボットは他のロボットの位置を観測し，自身の時刻 $t+1$ における位置 (移動先) を決定する．各時刻におけるロボットの行動/停止については仮想的なスケジューラにより決定されるものと考える．このスケジューラによって，各時刻でそれぞれのロボットがサイクルを実行するか，実行しないかが決定される．仮想スケジューラは公平であると仮定する．すなわち，あるロボットが永久に停止し続ける事はない.

各ロボットは他のロボット群の配置を自身の持つ座標系のもとで観測する．観測結果は，観測者の座標系における，他のロボット座標の集合により表現される．なお，各ロボットが有する座標系の方向，単位距離は一致しておらず，また時間とともに変化しうる．視野に制限はなく，他のロボットが観測の邪魔になることはないとする．本論文では，観測結果が誤差を含むモデルを扱っているため，実際のロボットの位置と観測結果の間にはずれがある．誤差に関しては 2.2 節において述べる.

ロボットは観測結果を元に移動先を決定する．このとき，ロボットは過去の状況を記憶することができないため，次の目的地は観測結果のみから決定される．また，各ロボットで動作するアルゴリズムはすべて同一であると仮定する．形式的には，アルゴリズムは座標の集合 (観測結果) を入力とし，座標 (移動先) を出力とする決定性関数 f として定義される．なお，入力と出力の座標はいずれも，大域座標ではなく，計算しているロボット自身の座標系における座標である．時刻 t に行動するロボットの時刻 $t+1$ における位置は，時刻 $t+1$ に計算した目的地と必ず一致する．つまり，移動は必ず目的地に到達すると保証されており，移動経路の途中で止まることはない.

2.2 観 測 誤 差

ロボットが観測時に生じる誤差は距離誤差率と角度誤差に分けることができる．それぞれの誤差に対しては上限値が仮定される．以下，ϵ_0 は距離誤差率の最大値，θ_0 は角度誤差の最大の大きさとし，距離誤差率，角度誤差が実際に観測にあたえる影響について説明する．(図 1)

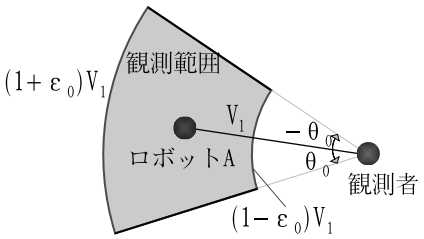


図 1　観測誤差の範囲

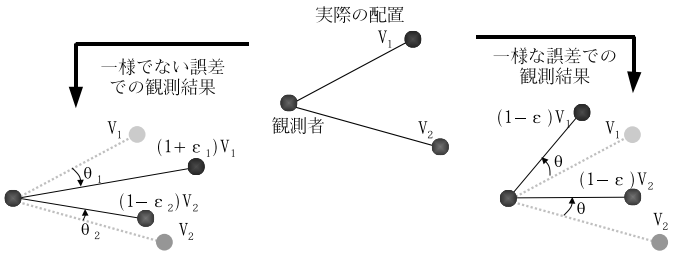


図 2　一様誤差，一様でない誤差の観測例

- 距離誤差

観測対象への正しい距離を V とすると，観測結果における対象までの距離が $(1-\epsilon_0)V$ から $(1+\epsilon_0)V$ の間のある値になる.

- 角度誤差

観測者の座標系における観測対象の偏角を θ とすると，観測結果における対象の偏角は $\theta-\theta_0$ から，$\theta+\theta_0$ の間の値をとる．ここでは，反時計周りの誤差を＋，時計周りの誤差を－とする.

2.3 誤差モデル

ここでは，一様な誤差モデル [1] と一様ではない誤差モデルを定義する．(図 2)

- 一様でない誤差モデル [1]

一回のロボットの観測においては，観測対象となる各ロボットに対して異なる距離誤差 ϵ_i と角度誤差 θ_i が生じる．例えば，ロボット s_0 が観測を行ったとき，ロボット s_1 に対しては距離誤差 ϵ_1，角度誤差が時計回りに θ_1 発生し，ロボット s_2 に対しては距離誤差 ϵ_2，角度誤差が反時計回りに θ_2 発生する可能性がある.

- 一様な誤差モデル

あるロボットの観測では，観測対象となるすべてのロボットに対して同じ距離誤差 ϵ，角度誤差 θ が生じる．ロボット s_0 が観測したとき，すべてのロボットに対して同じ大きさの距離誤差 ϵ と同じ向きの角度誤差 θ が発生する.

どちらのモデルでも，異なる 2 回の観測においては，たとえそれが同一ロボットによる観測であったとしても，異なる度合いの誤差が生じうる．定義から一様な誤差モデルでは一様でない誤差モデルより強い観測に関する仮定が導入されている．つまり，一様な誤差モデルで一点収束可能であれば，一様でない誤差モデルでも一点収束可能である.

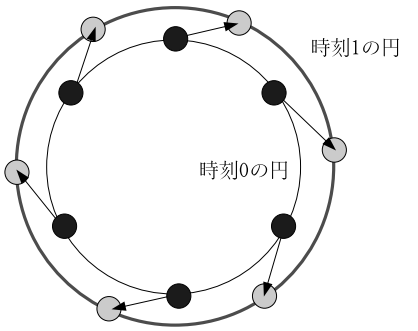


図 3　各ロボットが円の中心に対して対象的に，外側へと動く例

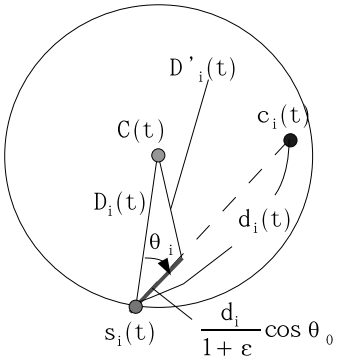


図 4　証明で使用する記号

3. 一点収束不可能性の証明

この章では一様な誤差モデルにおける，最大角度誤差 $\theta_0 \geqq 0$ の場合についての一点収束の不可能性を証明する．

［定理 1］ $\theta_0 \geqq 90°$ である一様な誤差モデルにおいて，2 台以上のロボットを一点収束を実現するアルゴリズムは存在しない．

［ 証明 ］ 証明は背理法による．一点収束するアルゴリズム A を仮定し，時刻 0 において，n 台のロボットが任意の半径 R の円周上に等間隔に配置されている状況を考える．このとき，時刻 1 において n 台のロボットが R 以上の円の円周上に均等に配置されるような A の実行が存在することを示す．このような実行が存在する場合，円を繰り返し拡大させることで，n 台のロボット間の距離が 0 に収束しないような A の実行が構成でき，矛盾を導くことができる．

以下では，すべてのロボットの座標系において単位距離はそれぞれ等しいものとする．上記の状況において，時刻 0 ですべてのロボットが動作し，観測誤差なしで観測を行ったとする．このとき，アルゴリズム A が一点収束を達成することより，少なくとも 1 台のロボットが移動する．移動するロボットを s_i とし，時刻 0 における s_i の座標系もとでの，円の中心の偏角を ϕ とする．このときの s_i の振る舞いにより場合分けを行う．

（1） s_i の移動先が円の外側 (円の境界を含む) の場合．

この場合，すべてのロボットの座標系を，円の中心が偏角 ϕ を向くように回転させると，すべてのロボットは s_i と同様の動作をする．このとき，すべてのロボットが行動状態であれば，各ロボットは円の中心に対して対象的に，外側へと動く．(図 3)

（2） s_i が円の内側へと移動する場合．

このとき，時刻 0 において，s_i の座標系を 90° 回転させて観測を行わせる．このとき，観測結果が角度誤差 90° を含む場合，座標軸と観測対象がいずれも 90° 回転しているため，アルゴリズムへの入力は座標軸が回転する前と等しい．それゆえ，出力である目的地も回転前と一致する．しかし，座標軸が 90° 回転していることから，実際の目的地は回転前に対して 90° 回転している事になり，これは s_i が円の外へと移動していることを意味する．以降は (1) と同様に証明できる．　□

4. 一点収束アルゴリズム

4.1 一点収束アルゴリズムの概要

本説では，一様な誤差モデルにおいて $\theta_0 < 90°$，$\epsilon_0 < 1$ の仮定のものロボット群が一点収束できるアルゴリズムを述べる．

提案するアルゴリズムでは，ロボットは観測結果をもとにして，最小包含円とその中心の座標を計算する．包含円とは，すべてのロボットを円周上あるいは内部に含むような円であり，その中で半径が最も小さいものが最小包含円である．次にロボットは，自分が最小包含円の円周上に存在するならば，円の中心に向かって移動する．しかし，観測結果は誤差を含むため，観測結果から計算した最小包含円は，実際の最小包含円とは異なっている可能性がある．提案するアルゴリズムは移動距離を制限することで，実際の最小包含円の外へとロボットが出ていかないことを保証する．

以下に，証明で使用する記法を説明する．なお，以下の説明において，位置，距離はすべて大域座標系での位置と距離を意味する．時刻 t でのロボット s_i の位置を $s_i(t)$，時刻 t で s_i の観測結果から計算した (誤差を含む) 最小包含円の中心の位置を $c_i(t)$，$s_i(t)$ と $c_i(t)$ の距離を $d_i(t)$，時刻 t における実際の最小包含円の中心を $C(t)$，$s_i(t)$ と $C(t)$ の距離を $D_i(t)$，$s_i(t+1)$ と $C(t)$ との距離を $D_i'(T)$，時刻 t の最小包含円の半径を $R(t)$，$C(t)$ と $C(t+1)$ の距離を $x(t)$ とおく．(図 4)．また，dis(点,点) を二点間の距離を表す関数とする．すべての観測において一様な誤差が生じるので，観測結果は実際のロボットの配置のないものと相似関係にあることが保証される．このことより，$(1-\epsilon_0)\,D_i(t) \leqq d_i(t) \leqq (1+\epsilon_0)\,D_i(t)$ が成り立つことに注意する．以降，ここで提案する一点収束アルゴリズムを包含円収束アルゴリズムと呼ぶことする．ロボット s_i に対する包含円収束アルゴリズムの擬似コードを次に示す．

```
1  観測結果から独自の座標上に最小包含円と
2  その中心の向きπとそこまでの距離 d を計算する．
3  If    計算した最小包含円の円周上にいる．
4          現在値から角度πの方向に向かって
5          距離 d/(1+ε0) cos θ0 だけ移動する．
6  Else  動かない．
```

4.2 アルゴリズムの正当性の証明

証明の流れは以下の通りである．1. 時刻 t でロボット i が移動するならば，ロボット i は時刻 t での実際の最小包含円の中心に近づくことを示す．2. 最小包含円の中心が時刻 t におけるロボット群の行動により移動した場合，時刻 $t+1$ では最小包含円の半径が時刻 t の最小包含円の半径より小さいことを示す．3. 最小包含円の半径，最小包含円の中心と各ロボットの距離の総和のいずれかが，任意の動作について，少なくとも定数の割合減少することを示す．4.〜3. により，明らかに最小包含円の半径が 0 に収束することが示す．

［補題 1］ ロボット s_i が移動するならば，任意の $t \geqq 0$ に対して，$D_i'(t) \leqq \alpha D_i(t)$ となる α $(0 \leqq \alpha < 1)$ が存在する．

［ 証明 ］ アルゴリズムからロボット s_i の移動距離は大域座標上では $\frac{d_i(t)}{1+\epsilon_0}\cos\theta_0$ である．ロボット s_i の観測の角度誤差が θ_i であるとする．このとき，$D_i'^2(t) = D_i^2(t) + \left(\frac{d_i(t)}{1+\epsilon_0}\right)^2 \cos^2\theta_0 - 2D_i(t)\frac{d_i(t)}{1+\epsilon_0}\cos\theta_0\cos\theta_i$ が成り立つ (詳細は付録参照)．$(1-\epsilon_0)\,D_i(t) \leqq d_i(t) \leqq (1+\epsilon_0)\,D_i(t)$ であるから，$\theta_i = \theta_0$ ，$d_i(t) = (1-\epsilon_0)\,D_i(t)$ のとき，$D_i'(t)$ は最大となる．よって $D_i'(t) \leqq D_i(t)\sqrt{1 + \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}\cos^2\theta_0}$ となる．ϵ_0, θ_0 はいずれも定数であり，$\epsilon_0 < 1$ を仮定しているため，上式の平方根部分は 1 未満の定数となる．これより，$\sqrt{1 + \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}\cos^2\theta_0} = \alpha$ とおくことが可能であり，$D_i'(t) \leqq \alpha D_i(t)$ が成立する． □

次に示す補題は，時刻 $t+1$ の最小包含円の半径と，時刻 t の最小包含円の半径の間で常に成り立つ不等式を示している．

［補題 2］ 任意の $t \geqq 0$ について $R(t+1) \leqq \sqrt{R^2(t) - x^2(t)}$ が成り立つ．

［ 証明 ］ 補題 1 から，時刻 t に行動するロボットの移動先は必ず時刻 t の最小包含円の内部にある．よって，時刻 $t+1$ で最小包含円が大きくなることはない．また，時刻 t の最小包含円と時刻 $t+1$ の最小包含円の共通部分は存在することは明らかである．このとき，二つの円の関係を場合分けして考える．

（1） 時刻 $t+1$ での最小包含円が時刻 t での最小包含円に外接する場合．

ロボットは接している点に集合していることになる．そのとき，最小包含円の中心は外接している点であり，半径は 0 となるため，明らかに $R(t+1) \leqq \sqrt{R^2(t) - x^2(t)}$ 成り立つ．

（2） 時刻 t での最小包含円と時刻 $t+1$ での最小包含円が一致するとき．

$x(t) = 0$ であるので $0 = R(t+1) = \sqrt{R^2(t) - x^2(t)} = R(t)$ である．

（3） 時刻 t での最小包含円の境界と時刻 $t+1$ での最小包含円の境界が二点で交わる．

二つの円の交点をそれぞれ X_1, X_2，線分 $C(t)C(t+1)$ と線分 X_1X_2 の交点を Z，線分 $C(t)C(t+1)$ に垂直で，$C(t+1)$ を通る直線を引き，その直線と時刻 $t+1$ での最小包含円との交点をそれぞれ Y_1, Y_2 とする．このとき，線分 Y_1Y_2 は時刻 $t+1$ の最小包含円の中心を通る弦であるため，その長さは直径に一致する．よって，線分 $C(t)C(t+1)$ と $dis(Y_1, Y_2) = 2R(t+1)$ である．ここで $dis(Z, C(t))$ と $x(t)$ の大小関係で場合分けを行う．

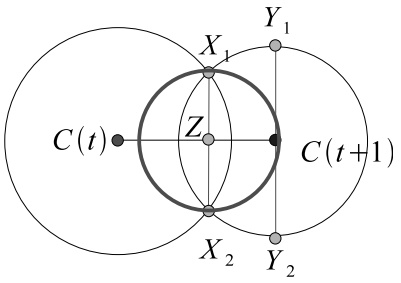


図 5 $dis(Z, C(t)) < x(t)$ での状態

- $dis(Z, C(t)) < x(t)$ の場合．(図 5)

このようなケースが生じないことを，背理法により示す．今，$dis(Z, C(t)) < x(t)$ が成り立つと仮定する．このとき，$dis(X_1, X_2)$ を直径とする円は 2 つの最小包含円の共通部分を包含する円となっている．また，$dis(X_1X_2)$ はいずれの最小包含円の直径よりも短い．ここで，最小包含円の半径が単調非減少であることより，$R(t) \leqq R(t+1)$ であり，$dis(X_1, X_2) \leqq 2R(t+1)$ が成り立つ．このとき，$dis(X_1, X_2) = 2R(t+1)$ となるのは，$dis(Z, C(t)) = x(t)$ のときである．これより，$dis(Z, C(t)) < x(t)$ の場合，$dis(X_1, X_2) < 2R(t+1)$ となる．このときは $dis(X_1, X_2)$ を直径とする円が最小包含円となり，$C(t+1)$ が最小包含円の中心という事実に矛盾する．よって $dis(Z, C(t)) < x(t)$ であることはない．

- $dis(Z, C(t)) \geqq x(t)$ の場合．(図 6)

直線 Y_1Y_2 と時刻 t の最小包含円との交点をそれぞれ Y_1', Y_2' とする．このとき, 明らかに $dis(Y_1Y_2) \leqq dis(Y_1', Y_2')$ である．直角三角形 $C(t)C(t+1)Y_1'$ に対して三平方の定理を適応すると，$dis(Y_1', C(t+1)) = \sqrt{R^2(t) - x^2(t)}$．同様に，$dis(Y_2', C(t+1)) = \sqrt{R^2(t) - x^2(t)}$．これより，$dis(Y_2', Y_2') = dis(Y_1', C(t+1)) + dis(Y_2', C(t+1) = 2\sqrt{R^2(t) - x^2(t)}$ であり，$dis(Y_1, Y_2) \leqq 2\sqrt{R^2(t) - x^2(t)}$ である．また，$dis(Y_1, Y_2) = 2R(t+1)$ であるから，$R(t+1) \leqq \sqrt{R^2(t) - x^2(t)}$.

（4） 時刻 $t+1$ での最小包含円が時刻 t での最小包含円の内部に存在する場合．(図 7)

$dis(Z, C(t)) \geqq x(t)$ の場合と同様の議論により $R(t+1) \leqq \sqrt{R^2(t) - x^2(t)}$ を得る．

以上より $R(t+1) \leqq \sqrt{R^2(t) - x^2(t)}$ が成り立つ． □

次に最小包含円の半径と，最小包含円の中心と各ロボットの距離の和の二つをを考え，時刻 t でのロボットが一台でも動くならば，どちらかが減少することを示す．

［補題 3］ 任意の時刻 $t \geqq 0$ においてロボットが一台でも移動するならば，時刻 $t+1$ で以下のいずれかが成り立つ．

- 時刻に依存しない定数 $\beta (0 \leqq \beta < 1)$ が存在し，

$$R(t+1) \leqq \beta R(t)$$

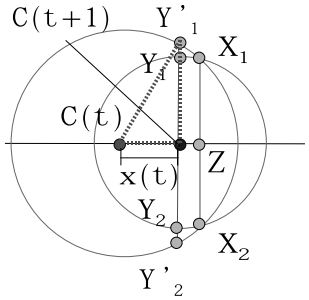


図 6　$dis(Z, C(t)) \geqq x(t)$ での状態

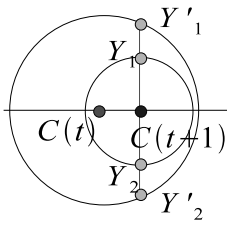


図 7　時刻 t+1 での最小包含円が時刻 t での円の内部に存在

- 時刻に依存しない定数 $\gamma(0 \leqq \gamma < 1)$ が存在し，
$$\sum_i D_i(t+1) \leqq \gamma \sum_i D_i(t)$$

［ 証明 ］　$R(t+1)$ は補題 2 から $R(t+1) \leqq \sqrt{R^2(t) - x^2(t)}$ が成り立つ．また，三角不等式から，任意の点 P について，$dis(P, C(t)) + x(t) \geqq dis(P, C(t+1))$ が成り立つ．そのため，ロボット s_i が時刻 t で移動したとすると，$D_i'(t) + x(t) \geqq D_i(t+1)$ であり，このことから $\alpha D_i(t) + x(t) \geqq D_i(t+1)$ が成り立つ．また，時刻 t で移動していないロボット s_i に関しては $D_i(t+1) \leqq D_i(t) + x(t)$ が成り立つ．よって，時刻 t で動いたロボットの集合を $S'(t)$，その数を $|S'(t)|$ とすると，以下が成り立つ．

$$\begin{aligned}
&\sum_i D_i(t+1) \\
\leqq &\sum_{i \notin S'(t)} (D_i(t) + x(t)) + \sum_{i \in S'(t)} (\alpha D_i(t) + x(t)) \\
= &\sum_{i \notin S(t)} D_i(t) + nx(t) - (1-\alpha) \sum_{i \in S'(t)} D_i(t)
\end{aligned}$$

動くロボットは円周上のロボットだけであるので $\sum_{i \in S'(t)} D_i(t) = |S'(t)| R(t)$ であり，これより $\sum_i D_i(t+1) \leqq \sum_i D_i(t) + nx(t) - (1-\alpha)|S'(t)|R(t)$ となる．

ここで，$nx(t) - (1-\alpha)|S'(t)|R(t)$ と $\frac{\alpha-1}{2n} \sum_i D_i(t)$ の大小関係で場合分けをする．

$nx(t) - (1-\alpha)|S'(t)|R(t) > \frac{\alpha-1}{2n} \sum_i D_i(t)$ の場合．

式を変形すると，$x(t) > \frac{2(1-\alpha)|S'(t)|R(t) + \frac{(\alpha-1)}{n} \sum_i D_i(t)}{2n}$ となる．$\sum_i D_i(t) \leqq nR(t)$ であるから，

$$\begin{aligned}
x(t) &> (1-\alpha) \frac{2|S'(t)|R(t) - R(t)}{2n} \\
&= (1-\alpha) R(t) \frac{2|S'(t)| - 1}{2n}
\end{aligned}$$

この結果を $R(t+1) \leqq \sqrt{R^2(t) - x^2(t)}$ に代入して，以下の不等式を得る．

$$\begin{aligned}
R(t+1) &< \sqrt{R^2(t) - \left((1-\alpha) R(t) \frac{2|S'(t)| - 1}{2n} \right)^2} \\
&= R(t) \sqrt{1 - \left(\frac{(2|S'(t)| - 1)(1-\alpha)}{2n} \right)^2}
\end{aligned}$$

$1 \leqq |S'(t)| \leqq n$ であるから，$R(t+1) < R(t)\sqrt{1 - \left(\frac{1-\alpha}{2n}\right)^2}$ となる．$0 \leqq \alpha < 1$ より，上式の平方根部分は 1 未満の定数である．よって，$\beta = \sqrt{1 - \left(\frac{1-\alpha}{2n}\right)^2}$ とおくことで，補題は示された．

（1）　$nx(t) - (1-\alpha)|S'(t)|R(t) \leqq \frac{\alpha-1}{2n} \sum_i D_i(t)$ の場合．

$\sum_i D_i(t+1) \leqq \sum_i D_i(t) + nx(t) - (1-\alpha)|S'(t)|R(t)$ に代入する．

$$\begin{aligned}
\sum_i D_i(t+1) &\leqq \sum_i D_i(t) + \frac{\alpha-1}{2n} \sum_i D_i(t) \\
&= \frac{1+\alpha}{2n} \sum_i D_i(t)
\end{aligned}$$

このとき，$0 \leqq \alpha < 1$ より，$\frac{1+\alpha}{2n} < 1$ となる，よって，$\gamma = (1+\alpha)/2n$ とおくことで，補題が示された．　□

最後にアルゴリズムが一点収束を達成することを示す．

［定理 2］　一様な誤差モデル，$\theta_0 < 90°$，$0 \leqq \epsilon_0 < 1$ であり，包含円収束アルゴリズムで動作する自律分散ロボット群は一点収束することができる．

［ 証明 ］　一般性を失うことなく，各時刻において，少なくとも 1 台のロボットが移動すると仮定する．補題 3 から一回のサイクルで最小包含円の半径の減少か，各ロボットと最小包含円の中心との距離の和の減少のどちらかが起こる無限長の実行を考えた場合，いずれかの値の減少は無限回生じる．

（1）　半径減少が無限回おきる場合．

半径減少がおきる時刻 t を早い順に $t_0, t_1, \cdots t_j$ とおく．このとき，t_{j+1} と $t_j + 1$ を比べると半径は明らかに $R(t_{j+1}) \leqq R(t_j + 1)$ となる．このとき，最小包含円の半径は 0 に収束するため，任意の 2 台のロボット間の距離も 0 に収束する．

（2）　各ロボットと最小包含円の中心との距離の和の減少が無限回おきる場合．

各ロボットと最小包含円の中心との距離の和の減少がおきる時刻 t を早い順に $t_0, t_1, \cdots t_k$ とおく．(1) と同様に考えると，補題 3 から $\sum_i D_i(t_{k+1}) \leqq \sum_i \gamma D_i(t_k + 1)$ $(0 \leqq \gamma < 1)$ が成り立つ．このとき，任意の 2 台のロボット間の距離は明らかに 0 に収束する．　□

5. 結　　論

本論文では一様な観測誤差モデルで動作する自律分散ロボット群は角度誤差の最大が 90° 以上での一点収束不可能，90° 未満は一点収束可能であることを示した．今後の課題として，一様でない観測誤差モデルにおける未解決部分の解明が挙げられる．

文　献

[1] R. Cohen and D. Peleg. Convergence of Autonomous Mobile Robots With Inaccurate Sensors and Movement. Lecture Notes in Computer Science, 3884, p549-560, 2006.

[2] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots –Formation and Agreement Problems. 3rd International Colloquium on Structural Information and Communication Complexity(SIROCCO 96), p313-330, 1996.

[3] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. SIAM Journal on Computing, Vol.28, No.4, p1347-1363, 1999.

[4] S. Souissi, X. Defago and M. Yamashita. Gathering Asynchronous Mobile Robots with Inaccurate Compasses. Lecture Notes in Computer Science, 4305, p333-349, 2006.

[5] T. Izumi, Y. Katayama, N. Inuzuka and K. Wada. Gathering Autonomous Mobile Robots with Dynamic Compasses: An Optimal Result. Lecture Notes in Computer Science, 4731, p298-312, 2007.

[6] Y. Katayama, Y. Tomida, H. Imazu, N. Inuzuka and K. Wada. Dynamic Compass Models and Gathering Algorithms for Autonomous Mobile Robots. Lecture Notes in Computer Science, 4474, p274-288, 2007.

[7] R. Cohen and D. Peleg. Convergence of Autonomous Mobile Robots of the Gravitational Algorithm in Asynchronous Robot System. In Proc. 12th European Symp. on Algorithm, 2004. SIAM Journal on Computing archive, Volume 34, p1516-1528, 2005

付　録

補題 2 の証明において省略した式変形:

$$
\begin{aligned}
& D_i'^2(t) \\
=\ & D_i^2(t) + \left(\frac{d_i(t)}{1+\epsilon_0}\right)^2 \cos^2\theta_0 - 2D_i(t)\frac{d_i(t)}{1+\epsilon_0}\cos\theta_0\cos\theta_i \\
\leqq\ & D_i^2(t) + \left(\frac{d_i(t)}{1+\epsilon_0}\right)^2 \cos^2\theta_0 - 2D_i(t)\frac{d_i(t)}{1+\epsilon_0}\cos^2\theta_0 \\
=\ & D_i^2(t) + \left(\left(\frac{d_i(t)}{1+\epsilon_0}\right)^2 - 2D_i(t)\left(\frac{d_i(t)}{1+\epsilon_0}\right)\right)\cos^2\theta_0 \\
=\ & D_i^2(t) + \left(\left(\frac{d_i(t)}{1+\epsilon_0}\right) - D_i(t)\right)^2 \cos^2\theta_0 - D_i^2(t)\cos^2\theta_0 \\
\leqq\ & D_i^2(t) + \left(\frac{1-\epsilon_0}{1+\epsilon_0}D_i(t) - D_i(t)\right)^2 \cos^2\theta_0 - D_i^2(t)\cos^2\theta_0 \\
=\ & D_i^2(t)\left(1 + \left(\frac{1-\epsilon_0}{1+\epsilon_0} - 1\right)^2 \cos^2\theta_0 - \cos^2\theta_0\right) \\
=\ & D_i^2(t)\left(1 + \left(\frac{1-\epsilon_0}{1+\epsilon_0} - \frac{1+\epsilon_0}{1+\epsilon_0}\right)^2 \cos^2\theta_0 - \cos^2\theta_0\right) \\
=\ & D_i^2(t)\left(1 + \frac{4\epsilon_0^2}{(1+\epsilon_0)^2}\cos^2\theta_0 - \frac{\epsilon_0^2+2\epsilon_0+1}{(1+\epsilon_0)^2}\cos^2\theta_0\right) \\
=\ & D_i^2(t)\left(1 + \frac{3\epsilon_0^2-2\epsilon_0-1}{(1+\epsilon_0)^2}\cos^2\theta_0\right) \\
=\ & D_i^2(t)\left(1 + \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}\cos^2\theta_0\right)
\end{aligned}
$$

□

補題 2 における，$0 \leqq \sqrt{1+\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)}\cos^2\theta_0} < 1$ の証明.

$0 \leqq \cos^2\theta_0 \leqq 1$ であるので，$\sqrt{1+\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}\cos^2\theta_0}$ は $-1 \leqq \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2} < 0$ であればよい．これを調べるために $\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}$ を ϵ_0 で微分する

$$
\frac{(6\epsilon_0-2)(1+\epsilon_0)^2 - \left(3\epsilon_0^2-2\epsilon_0-1\right)2(\epsilon_0+1)}{(1+\epsilon_0)^4} = \frac{8\epsilon_0}{(1+\epsilon_0)^3}
$$

$0 \leqq \epsilon_0 < 1$ の範囲で微分した式が 0 となるのは $\epsilon_0 = 0$ のときのみ．よって，微分した式は $0 < \epsilon_0 < 1$ では常に正となる．このことから $\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2}$ は単調増加であるとわかる．$\epsilon_0 = 0$ のとき $\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2} = -1$ であり，$\epsilon_0 = 1$ のとき $\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2} = 0$ であるから $-1 \leqq \frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)^2} < 0$ となる．よって，$0 \leqq \sqrt{1+\frac{(3\epsilon_0+1)(\epsilon_0-1)}{(1+\epsilon_0)}\cos^2\theta_0} < 1$ である．　□

Population Protocol Model を用いた情報収集

溝口 隆 * 小野 廣隆 † 定兼 邦彦 † 山下 雅史 †
* 九州大学大学工学部電気情報工学科 † 九州大学大学院システム情報科学研究院

1 はじめに

Population Protocol Model とは, 相互作用する移動エージェント集合による計算モデルのことをいう [1]. エージェントは, 有限状態機械でプログラムされ, 2 つのエージェントが相互作用することでそれらの状態を書き換える. 相互作用は任意のパターンで起こりうる. このため, 通常, 相互作用は仮想的な敵対者がスケジュールするものと考える. ただし, ある 2 エージェント間での相互作用がいつまでも起こらないなどの極端なスケジュールを排除するため公平性と呼ばれる性質を仮定する. Population Protocol においては, はじめエージェントには適当な入力値が割り振られ, 最終的には正しい出力値に収束する.

Population Protocol Model は, 小型装置によるアドホックモバイルネットワークにおける計算や化学反応により状態を変える分子の集合による計算を抽象化したものととらえることができる. このため, 近年, その計算能力の解明に関する研究が注目を浴びている.

本稿では, あるエージェントの属性が過半数以上であるかどうかを判定するプロトコル [4] を紹介し, その応用としてある属性がエージェント全体に対して任意の割合より多いかどうかを判定するプロトコルを提案する.

本稿の構成は, 以下のようになっている. 2 章で Population Protocol Model の基本モデルとその説明を行う. 3 章でその例を紹介し, 4 章では提案手法の説明を行う. 最後に 5 章でまとめと今後の課題について述べる.

2 準備

2.1 基本モデル

Population protocol model は, 自分自身で自由に動くことができない能力が極めて制限された移動エージェントから成るセンサーネットワークのモデル化のために提案された. また, それは理論化学の相互作用する分子のモデル [2, 3] と多くの類似点をもつ.

基本モデルを決定づける特徴は, 以下のようにまとめられる:

1. 名前のない有限状態エージェント.
システムは識別子を持たないエージェントの集合から成る. 各エージェントは有限状態しか持つことができない.

2. 直接相互作用することによる計算.
エージェントはメッセージの送信やメモリの共有をしない. そのかわり, 2 エージェント間で相互作用することで共通の遷移表に従ってそれぞれの状態を書き換える.

3. 予測のつかない相互作用のパターン.
どのエージェント同士が相互作用するのかは敵対者によって選ばれる. エージェント自身はどのエージェントと相互作用するのか選べない. 敵対者は, 起こりうる相互作用の関係を表した相互作用グラフの隣接状況に応じて相互作用を起こす 2 つのエージェントを選ぶ. 通常, 相互作用グラフは位置関係などから定義される. 敵対者にはプロトコルが進捗するように全体的に公平な条件が課せられる (後述).

4. 入力と出力が割り振られる.
Population protocol に対する入力は集合全体の初期状態として割り当てられるものと考える. 同様に, 出力はすべてのエージェントの状態値の集合として考える.

5. 終了するというよりも収束する.
Population protocol において, 各エージェントは通常, いつ「計算」が終了したのか気付かない. そのかわり, エージェントの出力は必ず有限時間後に共通の正確な値に収束する.

2.2 プロトコルの定義

プロトコルは以下により記述される:

Q : エージェントのとることができる状態の有限集合.

Σ : 有限な入力アルファベット.

ι : Σ から Q への入力写像. ここで $\iota(\sigma)$ は入力が σ のエージェントの初期状態を表す.

ω : Q から出力幅 Y への出力写像. ここで $\omega(q)$ は状態 q のエージェントの出力値を表す.

$\delta \subseteq Q^4$: エージェントの組がどのように遷移できるかを表している遷移関係.

計算は n 個のエージェント間で行われる, ここで $n \geq 2$ である. 各エージェントは Σ から入力値が与えられる. 各エージェントの初期状態はその入力値に対する ι を適用することで決定される. これによりある実行に対する初期配置が決まる. システムの構造はすべての状態の 1 次配列で表すことができる. なぜなら, 同じ状態のエージェントは区別がつかず, 各配列はある非順序な状態の集合としてまとめることができるためである.

2.3 プロトコルの実行

プロトコルは, 初期配置から, エージェント間でおこる相互作用を実行することにより動作を開始する. 状態が q_1 と q_2 を持つ 2 つのエージェントが出会い相互作用するとき, 遷移関係 δ が (q_1, q_2, q_1', q_2') ならば, 2 つのエージェントの状態は相互作用の結果としてそれぞれ状態 q_1', q_2' に変わる. 以下, δ を表記法 $(q_1, q_2) \to (q_1', q_2')$ を使って相互作用を表す. ただし, (q_1, q_2) について特に遷移が定義されていなければ, それは状態の変わらない遷移 $(q_1, q_2) \to (q_1, q_2)$ であるとする. もし任意の 2 エージェントの状態対 (q_1, q_2) に対して遷移が唯一 $(q_1, q_2) \to (q_1', q_2')$ のみ可能ならば, そのプロトコルは決定的であるという. ある 2 つのエージェントによる 1 回の相互作用により, 状態集合 C から C' が得られることを $C \to C'$ と表す. ここで, δ により (q_1, q_2, q_1', q_2') が定義されているとすると, C が 2 つの状態 q_1 と q_2 を含み C' は q_1 と q_2 を q_1' と q_2' に置き換えることによって C から得られるものとする. プロトコルの実行は, C_0 を初期配列としたとき, すべての $i \geq 0$ に対して $C_i \to C_{i+1}$ であるような有限で連続した配列 $C_0, C_1, C_2, \cdots$ である. このように, 実行は各相互作用が起こったときのシステムの連続したスナップショットとみなすことができる. 実際には分散して実行が行われるときに相互作用が同時に起こることがあるが, 実行を記録するときにその同時に起こった相互作用を任意に順序づけてよい.

2.4 公平性

相互作用の順番は, 敵対者が 2 つのエージェントによる相互作用のスケジュールをするため各エージェントには予測できない. プロトコルは敵対者がどんなスケジュールを選択しても正確な実行を行わなければならない. ここで, 意味のある計算が可能となるように敵対者によるスケジュールにいくつか条件を設ける. たとえば, あるエージェント集合をいくつかの独立した集合に分割して同じ集合に属するエージェントとのみ相互作用するようなスケジュールの下では, どのような意味のある計算もできそうにない. このようなスケジュールを排除するための条件が

公平性である.

スケジュールに課す公平性は, 本質的には敵対者が遷移できるにもかかわらずその遷移を永遠に避けるようなスケジュールを認めないことを言う. より正確にはもし C がある実行で無限に現れうる配列で, $C \to C'$ と定義されているとすると, C' もその実行で無限に現れなければならないということである. 別の言い方をすると, 遷移を繰り返すことで常に遷移可能な配列は有限時間内に遷移しなければならない.

3 ある属性が過半数以上であるかどうかを判定するプロトコル

Population Protocol Model を用いてある属性を持つエージェントが全体の過半数以上を占めるかどうかを判定するためのプロトコル [4] を紹介する. 方針として, 判定の対象となるエージェントを a, それ以外の全てのエージェントを b で表し, a と b が相互作用することによりお互いの状態を打ち消しあい, これを繰り返した結果, a が残るかどうかにより判定を行うプロトコルを考える.

過半数判定プロトコル

判定の対象となるエージェントを a, それ以外の全てのエージェントを b で表すとする.

入力アルファベット $\Sigma = \{a, b\}$.

出力幅 $Y = \{0, 1\}$.
ここで, 1 は a が b よりも多いことを表しているとする.

エージェントの取りうる状態 $Q = \{a, b, 0, 1\}$.

入力写像 $\iota(\sigma) = \sigma \quad (\sigma \in \Sigma)$.

出力写像 $\omega(a) = \omega(1) = 1$.
$\omega(b) = \omega(0) = 0$.

状態遷移の関係 δ:
$(a, b) \to (0, 0)$.
$(a, 0) \to (a, 1)$.
$(b, 1) \to (b, 0)$.
$(0, 1) \to (0, 0)$.

例えば, 状態 a のエージェントの個数を 10 個, 状態 b のエージェントの個数を 6 個とすると, いつかは状態 a のエージェント個数は 4 個, 状態 b のエージェントの個数は 0 個となり状態 a のエージェント以外のエージェントは状態 1 のエージェントとなる. このとき確かに全てのエージェントは収束し 1 を出力する. 正当性の証明は省略する.

4 ある属性が任意の割合より多いかどうかを判定するプロトコル

いま, 前述の [4] の方法を応用し, ある属性を持つエージェントが全体の q/p(p, q は, $p > q$ を満たす互いに素な自然数) より多いかどうかを判定するプロトコルを提案する. 方針としては, 各エージェントの状態に重みを持たせ a が全体のうち q/p より多いとき 1 を出力し, q/p 以下のとき 0 を出力するようにする.

4.1 割合超過判定プロトコル

判定の対象となるエージェントを a, それ以外の全てのエージェントを b で表すとする.

$\Sigma = \{a, b\}$.

$Y = \{0, 1\}$.

$Q = \{a(x), b(y), 0, 1\}$
$(x \in \{1, 2, \cdots, p - q\}, y \in \{1, 2, \cdots, q\})$.

$\iota(a) = a(p-q)$.
$\iota(b) = b(q)$.

$\omega(a(x)) = 1 = 1$
$\omega(b(y)) = 0 = 0$

δ :
$(0,1) \to (0,0)$.
$(a(x),0) \to (a(x),1)$.
$(b(y),1) \to (b(y),0)$.
$x \geq y$ のとき
$(a(x),b(y)) \to (a(x-y),0)$.
$x < y$ のとき
$(a(x),b(y)) \to (0,b(y-x))$.

4.2 プロトコルの正当性

まず, 第 1 段階として有限時間内で全てのエージェントは $a(x)$ または $b(y)$ のどちらかの状態のエージェントのみが残るかどちらの状態のエージェントも残らないかの状態に到達することを証明する.

次に, 第 1 段階の後, 有限時間内に全てのエージェントの出力が適当な値に収束し, 属性 a が全体の q/p より大きいときに限り全体が 1 を出力し, それ以外のときに全体が 0 を出力することを証明する.

補題 1. 割合超過判定プロトコルにおいて有限時間内で全てのエージェントは $a(x)$ または $b(y)$ のどちらかの状態のエージェントのみが残るかどちらの状態のエージェントも残らないかの状態に到達する.

証明) 初め, すべてのエージェントは $a(x)$ または $b(y)$ の状態のエージェントのみであり, 状態が $a(x)$, $b(y)$ のエージェントは状態が $b(y)$, $a(x)$ のエージェントと出会うまで重みは変わらない. また, 状態 $a(x)$ と $b(y)$ のどちらのエージェントも残っていて, 状態 $a(x)$ と $b(y)$ の重みが減るような遷移を永遠に避けるような遷移の仕方は公平性に反するので, 状態 $a(x)$ と $b(y)$ のどちらのエージェントも残っているときには有限時間内にその相互作用が起こる. よって, 状態 $a(x)$ と $b(y)$ のエージェントがどちらとも残り続けることはなく, このプロトコルにおいて有限時間内に全てのエージェントは $a(x)$ または $b(y)$ のどちらかの状態のエージェントのみが残るかどちらの状態のエージェントも残らないかの状態に到達する.

初めの状態 $a(x)$ のエージェントの個数を n, 状態 $b(y)$ のエージェントの個数を m とすると, すべての属性 a のエージェント重みの総計は $n(p-q)$ で, すべての属性 b のエージェント重みの総計は mq である. よって, 以下のようにいずれの場合も収束する.

1. 属性 a が全体の q/p より大きいとき
$n > (n+m)q/p \leftrightarrow n(p-q) > mq$ より, 状態 $a(x)$ と $b(y)$ による相互作用を繰り返した結果, 状態 $a(x)$ のエージェントが残る.

2. 属性 a が全体の q/p より小さいとき
$n < (n+m)q/p \leftrightarrow n(p-q) < mq$ より, 状態 $a(x)$ と $b(y)$ による相互作用を繰り返した結果, 状態 $b(y)$ のエージェントが残る.

3. 属性 a が全体の q/p のとき
$n = (n+m)q/p \leftrightarrow n(p-q) = mq$ より, 状態 $a(x)$ と $b(y)$ による相互作用を繰り返した結果, どちらの状態のエージェントも残らない.

(証明終)

補題 2. 割合超過判定プロトコルにおいて第 1 段階の後, 有限時間内に全てのエージェントの出力が適当な値に収束し, 属性 a が全体の q/p より大きいときに限り全体が 1 を出力し, それ以外のときに全体が 0 を出力する.

証明)

1. 状態 $a(x)$ のエージェントが残るとき

すべてのエージェントの状態は $a(x)$ または 0 または 1 である. このうち状態が $a(x)$ のエージェントは常に別の状態に変わらず, δ は $(0,1) \to (0,0)$ または $(a(x),0) \to (a(x),1)$ のみ考えればよい. また, $(a(x),0) \to (a(x),1)$ の遷移ばかりを繰り返すことにより 0 の状態のエージェントがなくなるような状態の配列は常に考えられる. よって, 公平性より有限時間内にはすべてのエージェントの状態は $a(x)$ または 1 になり, 各エージェントの状態が変化することもなくなる. そして, その時それらの出力はすべて 1 である.

2. $b(y)$ のエージェントが残るとき

すべてのエージェントの状態は $b(y)$ または 0 または 1 である. このうち状態が $b(y)$ のエージェントは常に別の状態に変わらず, δ は $(b(y),1) \to (b(y),0)$ と $(0,1) \to (0,0)$ のみ考えればよい. よって, 1 の状態のエージェントの数は減るだけなので有限時間内には, すべてのエージェントの状態が $b(y)$ または 0 になり, 各エージェントの状態が変化することもなくなる. そして, それらの出力はすべて 0 である.

3. $a(x)$ と $b(y)$ のエージェントがどちらも残らないとき

すべてのエージェントの状態は 0 または 1 である. δ は $(0,1) \to (0,0)$ のみ考えればよく, 1 の状態のエージェントの数は減るだけなので, 有限時間内にはすべてのエージェントの状態は 0 になり, 各エージェントの状態が変化することもなくなる. そして, それらの出力はすべて 0 である.

以上より, 割合超過判定プロトコルにおいて第 1 段階の後, 有限時間内に全てのエージェントの出力が適当な値に収束し, 属性 a が全体の q/p より大きいときに限り全体が 1 を出力し, それ以外のときに全体が 0 を出力する.

(証明終)

よって, 補題 1, 2 より以下が成立する.

定理 1. 割合超過判定プロトコルはある属性を持つエージェントが全体の q/p(p, q は, $p > q$ を満たす互いに素な自然数) より多いかどうかを判定する.

5 まとめと課題

本稿では，Population protocol を用いてある属性を持つエージェントが任意の割合より多いかどうかを判定するプロトコルを提案し, その正当性を示した.

今後の課題として既存手法との比較が挙げられる. また, 対象とするエージェントを複数設定し, そのなかで一定数より多く存在する対象の数を判定するためのプロトコルを考えていく予定である.

参考文献

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

[2] D. T. Gillespie : "Exact stochastic simulation of coupled chemical reaction", *Journal of Physical Chemistry*, 81(25):2340-2361, 1977.

[3] D. T. Gillespie : "A rigorous derivation of the chemical master equation", *Physica A*, 188:404-425, 1992.

[4] M. Mavroninicolas, J. Aspens and E. Ruppert: "An Introduction to Population Protocols", *The Distributed Computering Column*, pp.1–6.

マルチコアにおける並列ソートアルゴリズムの検証と提案

岡本 直樹　　藤原 暁宏

九州工業大学 大学院情報工学研究院

n-okamoto@zodiac30.cse.kyutech.ac.jp　　fujiwara@cse.kyutech.ac.jp

概 要

本研究では，基本演算であるソートについて，マルチコア CPU に対応した並列ソートアルゴリズムの検証と提案を行う．まず最初に，既存の並列ソートアルゴリズムであるマップソート，及び，並列マージソートの性能の検証を行う．この検証結果から，マップソートは偏った入力に対しては実行が遅くなることを示し，並列マージソートは，マージ操作に実行時間がかかり，全体の実行時間が遅くなることを示す．

次に，既存のアルゴリズムの検証結果を基に，既存のアルゴリズムである並列マージソートを改良し，より高速に実行可能な並列ソートアルゴリズムを提案する．また，提案アルゴリズムを実際のマルチコア環境に実装し，既存のアルゴリズムと比較を行う．この比較結果により，提案アルゴリズムでは，並列マージソートにおけるマージ操作の実行時間が短縮できることを示す．また，マップソートとの比較では，一般的な入力に対しては提案アルゴリズムとマップソートの実行時間は同等であり，マップソートが苦手とする偏った入力に対してはマップソートより高速に実行できることを示す．

1　はじめに

コンピュータが設計された当初から，処理速度の高速化は重要な研究対象であり，過去数十年間プロセッサの処理速度は，高クロック化によって改善されてきた．しかし，クロック周波数の増加に伴い発熱量や消費電力が増加し，シングルコアではこれ以上の性能向上が困難になって来ている．

そこで，プロセッサのアーキテクチャをシングルコアからマルチコアにすることで，プロセッサの性能改善を行う方法が主流となっている．例えば，Intel の Core 2 Duo や Core 2 Quad[2]，SUN microsystems の Ultra SPARC T1[5] 及び，AMD の Phenom[1]，といったマルチコア CPU がある．さらに身近な例としては，プレイステーション 3 に組み込まれた Cell プロセッサ [4] がある．

しかし，単にプロセッサコアの数を増やしても，ソフトウェア自体が並列化に対応していなければ，その性能の向上は期待できない．そこで，ソフトウェアの性能改善のためには，並列処理に対応したアルゴリズムで作成されたアルゴリズムが必要となる．加えて，マルチコアシステムのアーキテクチャは，既存の並列計算機とは異なるアーキテクチャなので，マルチコアシステムに適した並列アルゴリズムが必要となる．また，マルチコアシステムでは，クラスタ型並列処理環境と比べ，プログラムの並列化に必要な通信コストが大変小さいため，粒度の小さな並列化が可能である．

本研究では，基本演算であるソートについて，マルチコア CPU に対応した並列ソートアルゴリズムの検証と提案を行う．ソートは基本的な問題であり，多くのアプリケーションに基本操作として含まれている．ソートのデータ数を N とすると，逐次処理の場合は，$\Theta(N \log N)$ 時間で解くことができることがわかっている．また，P 個のプロセッサで並列処理で行った場合の理想的な実行時間は $O(\frac{N}{P} \log N)$ であり，既存のマルチコアに対応した並列ソートアルゴリズムとしては，マップソート [6] や並列マージソート [3] 等がある．

本研究では，まず最初に，既存のアルゴリズムであるマップソート，及び，並列マージソートの性能の検証を行う．この検証結果から，マップソートは偏った入力に対しては実行が遅くなることを示し，並列マージソートは，マージ操作に実行時間がかかり，全体の実行が遅くなることを示す．

次に，既存のアルゴリズムの検証結果を基に，並列マージソートを改良し，より高速に実行可能な並列ソートアルゴリズムを提案する．提案アルゴリズムでは，並列マージソートのマージ処理を簡略化することにより，偏った入力に対しても安定し，全体実行時間も高速なアルゴリズムを目指す．

更に，既存のアルゴリズムと比較を行う．この検比較結果により，提案アルゴリズムでは，並列マージソートにおけるマージ操作の実行時間が短縮できることを示す．また，マップソートとの比較では，一般的な入力に対しては提案アルゴリズムとマップソートの実行時間は同等であり，マップソートが苦手とする偏った入力に対してはマップソートより高速に実行できることを示す．

本稿は，以下のように構成されている．第 2 章では，マルチコア環境, 及び, 既存のソートアルゴリズム，について説明する．第 3 章では，既存のアルゴリズムのマルチコア環境における実際の性能を示し, 考察を行う．第 4 章では，本研究で提案するアルゴリズムについて説明する．第 5 章では，提案アルゴリズムのマルチコア環境での実験結果を示し, 考察を行う．最後に，第 6 章で本研究のまとめと今後の課題について述べる．

2　準備

本節では，マルチコア環境について概説するとともに，既知のマルチコア環境用のソートアルゴリズムであるマップソートと並列マージソートについて説明する．

2.1　マルチコアシステム

マルチコアシステムとは，1 つの CPU パッケージ内に複数の CPU コアを内蔵したシステムである．従来の

1 つのコアを持つ CPU はマルチコアに対してシングルコアと呼ばれ，10 個以上のコアを集積したプロセッサはメニーコアと呼ばれる．

マルチコアシステムのアーキテクチャを図 1 に示す．マルチコアシステムは，先に述べた共有メモリ型並列計算機と同様に，メインメモリを全てのコアで共有している．また，二次キャッシュ(L2) 及びメインメモリへの帯域幅も全てのコアで共有しているため，各コアが並列に処理を行う場合，1 つのコアが使用できる二次キャッシュのサイズが制限されると共に，二次キャッシュとメインメモリ間の転送がボトルネックとなる可能性がある．

また，SMT(Simultaneous multi-threading) と呼ばれる技術がある．これは，1 つの CPU 上で複数のスレッドを同時に実行させ，外部的に複数の CPU に見せる技術である．有名な実装例としては，Intel の HTT(Hyper Threading Technology) などがある．しかし，実際に存在しているコアは 1 つであるという点でマルチコア技術とは根本的に異なっている．現在のマルチコア CPU は SMT を使用したものが多く，各コアに SMT を搭載することで，全体のシステムでは，(コア数) × (スレッド数) 個のスレッドが同時処理可能である．例えば，SUN microsystems の Ultra SPARC T1 プロセッサは，8×4 の 32 スレッドが同時に実行可能である．

また，一概にマルチコア CPU といっても，様々な形式のものが存在し，大きく分けて，同種のコアを複数実装する「ホモジニアスマルチコア」と，異種のコアを複数実装する「ヘテロジニアスマルチコア」の 2 種類が存在する．Intel や AMD による計算機用マルチコア CPU はホモジニアスマルチコアであり，現在の計算機向けのマルチコア CPU の主流はこちらである．ヘテロジニアスマルチコアの例としては，ソニー，ソニーコンピュータエンタテインメント，IBM，東芝によって開発された Cell(Cell Broadband Engine) プロセッサがある．本研究では, ホモジニアスマルチコアを対象としたアルゴリズムの提案を行う．

2.2 マップソート

マップソート [6] は，マルチコア CPU において実用的なスピードアップを実現することを目的として提案された並列ソートアルゴリズムである．このマップソートでは, 入力データ配列を M 個の部分配列に分割するとともに，データが分布する範囲を L 個の区間に分割する．そして，複数部分配列から複数区間への写像 (マップ) を考える．マップの計算，複数部分配列から複数区間へのデータコピー，複数区間ごとのソート処理は各々並列に行うことができ，結果として $O(\frac{N}{P}\log N)$ の時間計算量を達成する．

ソートを P 個のコアを持つ SMP(Symmetric Parallel Proccessing) かつ共有メモリ型のマルチコア CPU で解くことを考える．このとき，コア数 P は入力データ数 N よりも遥かに小さいと仮定する．

マップソートでは，与えられた入力 S を M 個の部分集合 $S_1, S_2 \cdots, S_M$ に分割するとともに，全データ範囲を L 個の区間に分割する．この 2 種類の分割のために，$(M \times L)$ の 2 次元配列を用いる．入力部分配列 S_i に含まれ，j 番目の区間に属するデータの集合を D_{ij} とする．

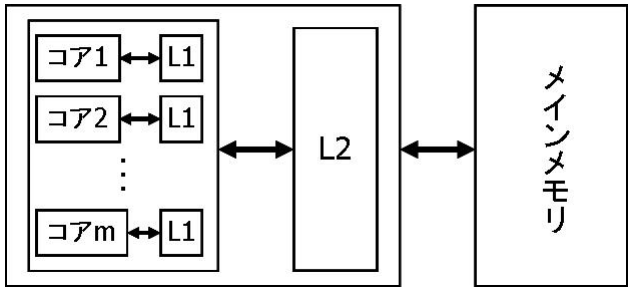


図 1: マルチコアシステムのアーキテクチャ

また，各 D_{ij} に対して，データの個数を数えるための 2 次元配列 C を $c_{ij} = |D_{ij}|$ と定義し，各区間の配列上の開始地点を示すための 2 次元配列 Q を以下のように定義する．

$$q_{ij} = \sum_{k_j=1}^{j-1}\sum_{k_i=1}^{M} c_{k_i k_j} + \sum_{k_i=1}^{i-1} c_{k_i j}$$

ただし，q_{ij} は次のように計算される．c_{ij} は部分集合 S_i に属し j 番目の区間に含まれるデータ数であるため，j 番目の区間に含まれるデータの総和は $\sum_{k_i=1}^{M} c_{k_i k_j}$ となる．これより，j 番目の区間に含まれるデータは，出力データ配列 T 上の $\sum_{k_j=1}^{j-1}\sum_{k_i=1}^{M} c_{k_i k_j}$ から開始されることになる．これにより，j 番目の区間に含まれ S_i に属するデータを $q_{ij} = \sum_{k_j=1}^{j-1}\sum_{k_i=1}^{M} c_{k_i k_j} + \sum_{k_i=1}^{i-1} c_{k_i j}$ から開始すると重なりなくデータを転送でき，これを q_{ij} とする．

以下にマップソートのアルゴリズムを示す．

入力 ： 全順序関係を持つ N 個のデータの列 $S = (s_1, s_2, \cdots s_N)$.

出力 ： 入力列の順列 $T = (t_1, t_2, \cdots t_N)$．ただし，任意の $i(1 \le i \le N-1)$ に対して，$t_i \le t_{i+1}$ が成り立つ

Step 1 : 入力データ S から $L-1$ 個の基準値を選び，それら基準値を境界とする L 個の区間を計算する．

Step 2 : 各データ $d \in S_i$ に対し，d を含む区間を求める．(求めた区間が先頭から j 番目と仮定とする．) 次に，2 次元配列 c_{ij} の値を $c_{ij} = c_{ij} + 1$ とする．(部分集合 S_i ごとに並列処理が可能である)

Step 3 : 全ての i，及び j の組み合わせに対して $q_{ij} = \sum_{k_j=1}^{j-1}\sum_{k_i=1}^{M} c_{k_i k_j} + \sum_{k_i=1}^{i-1} c_{k_i j}$ の計算を行い，2 次元配列 Q を求める．($\sum_{k_i=1}^{M} c_{k_i k_j}$ の演算は並列処理が可能である)

Step 4 : 各データ $d \in S_i$ に対し，d を含む区間を求める．(求めた区間を j 番目の区間と仮定とする．) 次に，出力データ配列 T の q_{ij} 番目に d を代入 ($T_{q_{ij}} = d$) する．最後に $q_{ij} = q_{ij} + 1$ とする．(部分集合 S_i ごとに並列処理が可能である)

Step 5 : 出力データ配列 T 上の各区間に対応する部分配列をソートする．(区間ごとに単体 CPU 向けに最適化されたソートアルゴリズムを用い，並列処理が可能である．)

マップソートの動作例を図2，図 3，図 4，図 5，及び，図 6 に示す．以下の動作例は，データ数 16 個の入力が与えられたとき，分割数 $M = 4$，区関数 $L = 4$ の条件でソートする場合の例である．

図 2 では Step 1 を実行し，3 個の基準値を選び 4 個の区間を作成する様子を示す．図 3 では Step 2 を実行し，カウント 2 次元配列を作成する様子を示す．図 3 は，各コアが部分配列の 1 つ目のデータについて Step 2 を実行した様子を示す．図 4 では Step 3 を実行し，2 次元配列 Q を作成する様子を示す．図 5 では Step 4 を実行し，出力配列にデータがコピーされる様子を示す．図 6 では Step 5 を実行し，各区間に含まれるデータ毎にソートする様子を示す．出力 T の色分けは，区間毎のデータ分割の様子である．

2.3 並列マージソート

並列マージソート [3] は，P 個のプロセッサを用いて N 個のデータに対して時間計算量 $O(\frac{N}{P}\log\frac{N}{P} + N \cdot \log P)$ で実行可能な並列ソートアルゴリズムである．並列マージソートでは，まずデータ数を均等に M 個のブロックに分け，各ブロックそれぞれを並列にソートする．その後，全てのブロックをヒープを用いてマージすることでソートを実現する．

ここで，入力データ数 N の入力配列 $S = (s_1, s_2, \cdots, s_N)$ を分割する数を M とし，分割後の部分配列を $D_1, D_2, \cdots, D_M$ とする．ただし，

$$D_i = \{s_{\frac{i\times N}{M}}, s_{\frac{i\times N}{M}+1}, \cdots, s_{\frac{(i+1)\times N}{M}-1}\} \quad (0 \le i \le M)$$

とする．

以下に並列マージソートのアルゴリズムを示す.

Step 1 : N 個の入力を M 個の部分配列に分け，それぞれに対して並列にクイックソートを行う.

Step 2 : ソート済みの M 個の部分配列について，それぞれの最小値からヒープを作成する.

Step 3 : ヒープから根に含まれるデータを取り出し T に格納する．更に, 最小値が含まれていた部分配列 D_i $(0 \le i \le M)$ から，次に小さい値をヒープに挿入する．本ステップを N 回繰り返し実行し, アルゴリズムを終了する.

並列マージソートの動作例を図 7，図 8，及び，図 9 に示す．以下の動作例は，データ数 16 個の入力が与えられたとき，分割数 $M = 4$ の条件でソートする場合の例である.

図 7，図 8 はそれぞれ Step 1，Step 2 の動作を示し，16 個の入力が 4 分割され，それぞれがソートされた後に，ヒープに挿入される様子を示す．図 9 は Step 3 の動作を示し，ヒープから根に含まれるデータを取りだし，出力配列にデータを格納する様子を示す.

3 既存のアルゴリズムの検証と考察

本章では，2 章で述べたソートアルゴリズムを C 言語で実装し，シミュレーションした結果を示す．またその結果に対して考察を行う.

3.1 実験環境

本研究における実験環境を 1 に示す.

表 1: 実験環境

CPU	Intel(R) Xeon E5320 1.86GHz
Memory	4GB
OS	Cent OS 4.6
コンパイラ	gcc 3.4.6

3.2 シミュレーションパラメタ

本研究では 2 種類の入力 (入力 1，入力 2) に対して実験を行った．表 2 に入力 1 のパラメタ，表 3 に入力 2 のパラメタを示す．入力 1 は, ランダムな文字集合であり, 各文字列の出現確率は当確率である. 一方, 入力 2 は, 文字の種類を 1 種類から 5 種類に固定することにより, 同じ文字列が非常に多い場合の入力である. この入力 2 は, 入力を複数の基準値により区間に区切ってからソートを行うマップソートが苦手とする入力であると予想される.

表 2: 入力 1

プロセッサ数	8
入力要素	8 文字の文字列 (a〜z)
入力文字種類数	26^8
入力要素数	100,000,000

表 3: 入力 2

プロセッサ数	8
入力要素	8 文字の文字列 (a〜z)
入力文字種類数	1 から 5
入力要素数	100,000,000

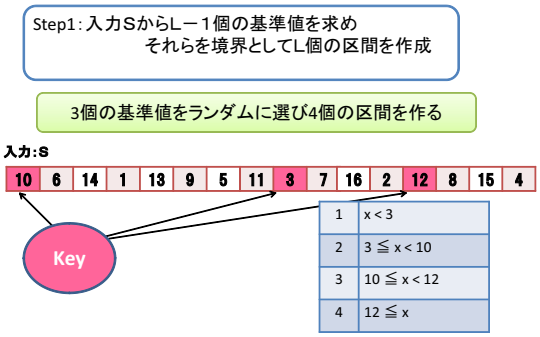


図 2: マップソートの動作例 (Step 1)

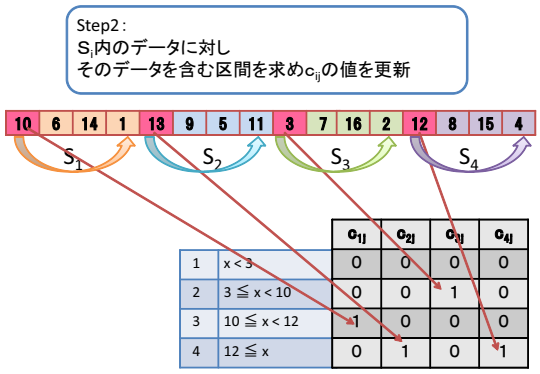


図 3: マップソートの動作例 (Step 2)

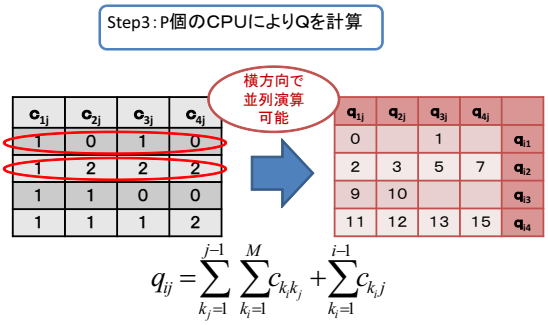


図 4: マップソートの動作例 (Step 3)

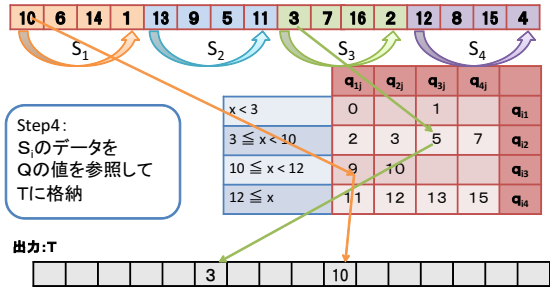


図 5: マップソートの動作例 (Step 4)

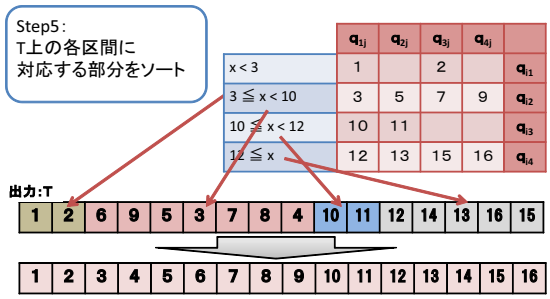


図 6: マップソートの動作例 (Step 5)

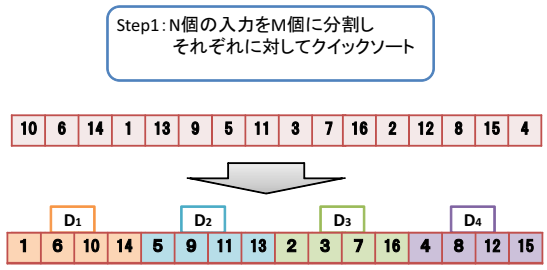


図 7: 並列マージソートの動作例 (Step1)

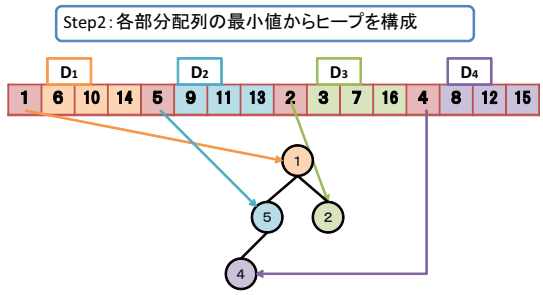


図 8: 並列マージソートの動作例 (Step 2)

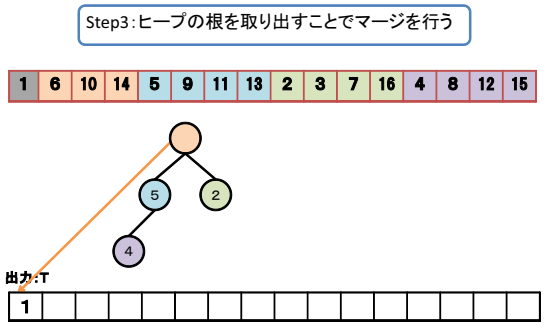


図 9: 並列マージソートの動作例 (Step 3)

3.3 実験結果

3.3.1 クイックソート

並列ソートソートアルゴリズムの実行時間の比較対象とするために，1CPU で入力 1 , 2 に対してクイックソートを実行した場合の実行時間を図 10 に示す．なお，本稿における実行時間とは，ランダムな 30 種類の入力に対する平均実行時間であり，図 10 のグラフの横軸は入力文字種類数，縦軸は実行時間を表している．この結果より，クイックソートは，入力文字の種類が少ない場合は実行が少し遅いことがわかる．

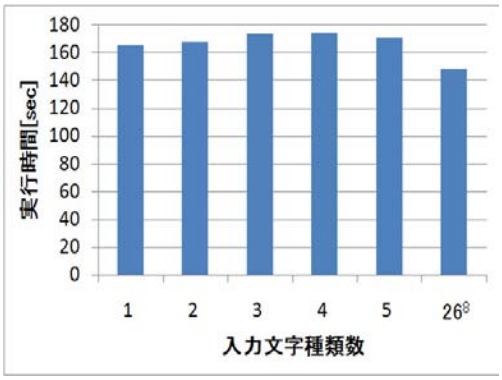


図 10: クイックソートの入力文字種類別実行時間

3.3.2 マップソート

まず最初に，入力 1 に対するマップソートの実験時間を図 11 に示す．図 11 のグラフでは，横軸が区間数 L を表しており，縦軸が実行時間を表している．また，グラフの各色は分割数 M の値を示している．図 11 から，実行時間はデータ分布区間数 L に大きく依存し，分割数 M にはほとんど依存しないことがわかる．この結果より，$M = 16$，$L = 16$ の場合が, マップソートが最も高速に動作するため，以降のマップソートでは，この値をデフォルトとして用いる．

次に，$M = 16$，$L = 16$ の場合における，マップソートの実行時間の最大値と最小値を表 4 に示す．表 4 より，マップソートの実行時間は入力により大きく変化することがわかる．これは，基準値の選択により，入力要素を均等に分割できていないという理由が考えられる．なぜならば，基準値の選び方により，Step 4 の結果生成される L 個の部分配列に偏りができ，Step 5 を並列実行する場合の負荷バランスが悪くなるためである．

最後に，入力 1 及び入力 2 に対するマップソートの実験時間を図 12 に示す．図 12 のグラフでは，横軸が入力文字種類数，縦軸が実行時間を表している．図 12 から，マップソートは偏った入力に対して実行が遅いことがわかる．

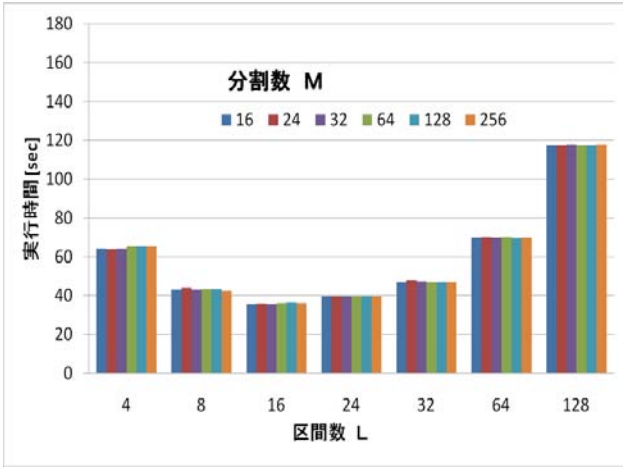


図 11: マップソートの M，L の変化における実行時間

3.3.3 並列マージソート

まず最初に，入力 1 に対する並列マージソートの実行時間を図 13 に示す．図 13 では，横軸が分割数 M を表しており，縦軸が実行時間を表している．更に，ソートフェーズ (Step1) とマージフェーズ (Step2，及び，Step3) にかかった時間を，それぞれ赤色と青色で表している．図 13 から，分割数 M が CPU 数 8 を超えてからはソートフェーズに大きな変化は見られないが，マージフェーズは分割数が増えるにつれて増加することがわかる．また，$M = 8$ の場合が, 並列マージソートが最も高速に動作するため，以降の並列マージソートでは，この値をデフォルトとして用いる．

次に，入力 1 及び入力 2 に対する並列マージソートの実行時間を図 14 に示す．図 14 のグラフでは，横軸が入力文字種類数，縦軸が実行時間を表している．図 14 から，並列マージソートは，偏った入力に対して安定した実行時間で動作することがわかる．

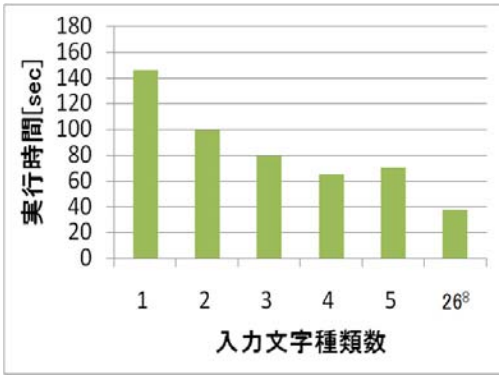


図 12: マップソートの入力文字種類数別実行時間

3.3.4 マップソートと並列マージソートの比較

入力 1 及び入力 2 に対するマップソートと並列マージソートの実験時間を，図 15 に示す．図 15 のグラフでは，横軸が入力文字種類数，縦軸が実行時間を表している．この結果から，入力 1 のようなランダムな入力に対しては，マップソートの方が高速であることがわかる．しかし, 入力 2 のような偏った入力に対しては, 並列マージソートの方が高速であることがわかる．

表 4: マップソートの実行時間の最小値及び最大値

	最小値	最大値	平均
マップソート	30.603	56.020	38.111

3.4 実験結果に対する考察

以上の実験結果から，マップソートの実行が遅くなる条件として，以下の 2 つが挙げられる．

条件 1： 入力の分布が特定の区間に偏っている．

条件 2： L 個の区間を作成する際に用いた基準値が不適当で，作成した区間の範囲に大きな差がある．

また，並列マージソートは，偏った入力に対して安定し，ランダムに決定する要素がアルゴリズム中に含まれていないため，同じ入力に対しては毎回同じ時間で実行できる利点がある．しかし，マージに大きく時間がかかるため，全体の実行が遅いという欠点がある．

4 提案アルゴリズム

本章では，2 つの提案アルゴリズムについて説明する．提案アルゴリズムの目標として、マップソートと同程度の実行時間で実行でき，更に，入力要素に左右されず実行時間が安定したソートを目標とする．これらの目標を実現するために、並列マージソートを基に，並列化に適するように改良することで高速化を図る．

4.1 並列クイックソートと逐次マージを用いたアルゴリズム

ヒープを用いたマージは，時間計算量 $O(N \log N)$ で実行可能であるが，3 章の実験より，実際には時間がかかることがわかった．そこで本章では，まず最初に，各部分配列の最小値を毎回比較することにより，ヒープ構成のステップを省略したマージ処理を用いるアルゴリズムを提案する．このマージ処理は，最も単純なマージ処理であるといえる．

Step 1： N 個の入力を M 個の部分配列 D_i $(0 \leq i \leq M)$ に分け，それぞれに対して並列にクイックソートを行う．

Step 2： ソート済みの M 個の部分配列について，それぞれの部分配列の最小値を比較し，更にその中の最小値を出力配列 T に格納する．格納した値は部分配列 D_i $(0 \leq i \leq M)$ から取り除く．本ステップを N 回繰り返し実行しアルゴリズムを終了する．

このアルゴリズムの動作例を図 16 に示す．以下の動作例は，データ数 16 個の入力が与えられたとき，分割数 $M = 4$ の条件でソートする場合の例である．この例では，16 個の入力 S が 4 つの部分配列 D に分割され，それぞれがクイックソートによりソートされる．その後，

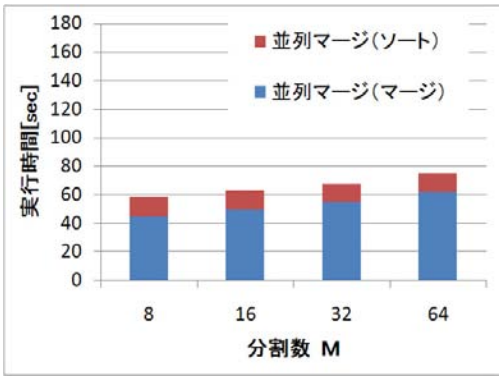


図 13: 並列マージソートの M の変化における実行時間

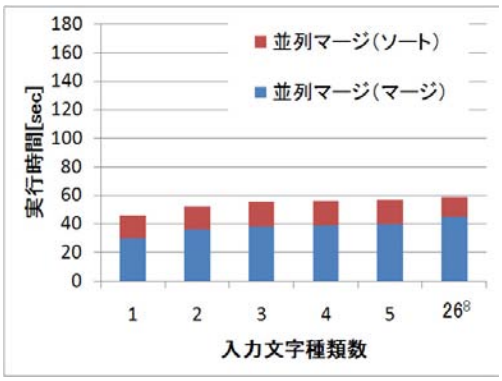


図 14: 並列マージソートの入力文字種類数別実行時間

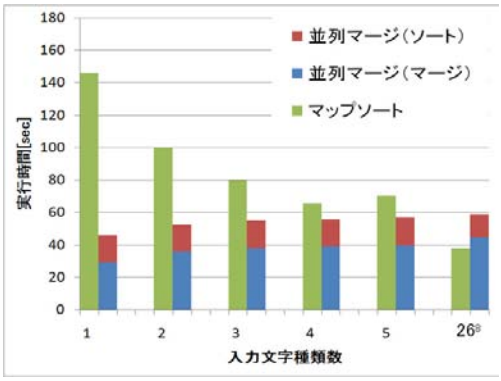


図 15: 入力 1,2 における実行時間の比較

各部分配列 D_i $(0 \leq i \leq 4)$ の最小値を比較し，出力配列 T に挿入する．挿入した値を取り除いた入力 S において再び Step 2 の処理を行う．この動作を 16 回繰り返すことで，ソートされた配列 T が作成される．

4.2 並列クイックソートと並列マージを用いたアルゴリズム

次に，2 つ目のソートアルゴリズムとして，並列にマージを行うアルゴリズムを提案する．この提案アルゴリズムでは，M 個の部分配列に対して，B 個ずつの部分配列を 1 つのブロックとして，各ブロックごとに並列にマー

ジを行う．(なお，B は $B \leq M$ を満たす自然数である．) その結果，$\frac{M}{B}$ 個のソート済みブロックが作成される．この $\frac{M}{B}$ 個の各ブロックを逐次にマージし，ソートを実現する．

以下に，提案アルゴリズムの詳細を示す．この詳細において，ソート済み出力配列の一時配列を T' とする．

Step 1 : N 個の入力を M 個の部分配列 D_i $(0 \leq i \leq M)$ に分け，それぞれに対して並列にクイックソートを行う．更に，ソート済みの M 個の部分配列を，$\frac{M}{B}$ 個のブロックに分ける．

Step 2 : $\frac{M}{B}$ 個の各ブロックに含まれる B 個の部分配列について，それぞれの最小値を比較し，その中での最小値を一時配列 T' に代入する．(ブロックごとに並列処理が可能である．) 更に，一時配列 T' に代入した値を部分配列から削除し，Step 2 を繰り返す．$\frac{M}{B}$ 個のブロック全てがソートされれば Step 3 に進む．

Step 3 : ソート済みの $\frac{M}{B}$ 個の部分配列について，それぞれの最小値を比較し，その中での最小値を出力配列 T に代入する．T' に代入した値を部分配列から削除し，Step 3 を繰り返す．本ステップを N 回繰り返しアルゴリズムを終了する．

このアルゴリズムの動作例を図 17 と図 18 に示す．以下の動作例は，データ数 16 個の入力が与えられたとき，分割数 $M = 4$，同時マージブロック数 $B = 2$，の条件でソートする場合の例である．

図 17 は Step 1，Step 2 の動作を示し，16 個の入力が 4 分割され，それぞれがソートされた後に，2 ブロックずつ並列にマージする様子を示す．図 18 は Step 3 の動作を示し，Step 2 のマージ処理で作成された 2 つのソート済部分配列を再びマージする様子を示す．

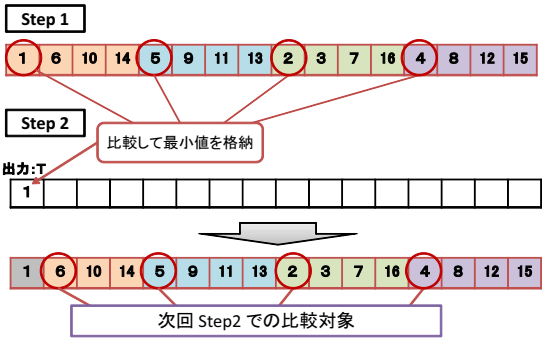


図 16: 並列クイックソートと逐次マージを用いたアルゴリズムの動作例

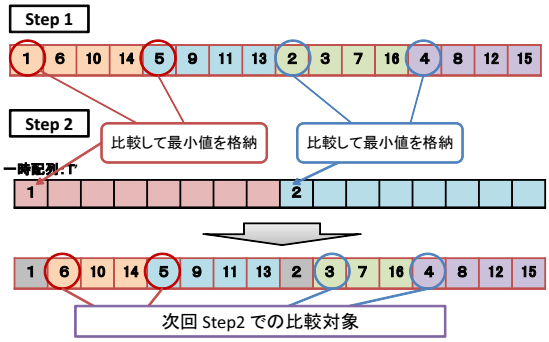


図 17: 並列クイックソートと並列マージを用いたアルゴリズムの動作例 (Step 1，及び，Step 2)

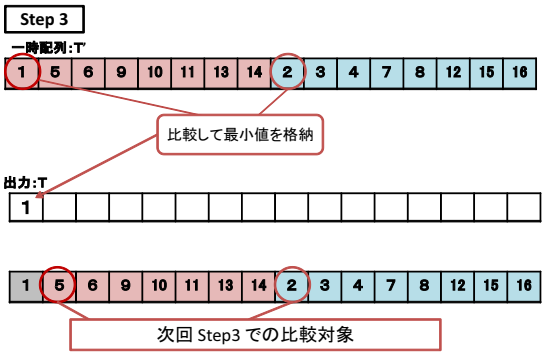


図 18: 並列クイックソートと並列マージを用いたアルゴリズムの動作例 (Step 3)

5　提案手法の検証と考察

本章では，4 章で提案したソートアルゴリズムを C 言語で実装し，シミュレーションした結果を示す．なお，シミュレーション環境は 3 章で説明した実験環境と同じである．また，各アルゴリズムの実験結果に対して考察を行う．

5.1　実験結果

以下の節では，4 章で提案した並列クイックソートと逐次マージを用いたアルゴリズムの結果と，並列クイックソートと並列マージを用いたアルゴリズムの結果を同時に示す．分割数 M ごとに，同時マージブロック数 B を変化させて測定し，最適な M と B の組み合わせを示す．なお，$M = B$ の場合が，並列クイックソートと逐次マージを用いたアルゴリズムの結果であり，それ以外の結果が，並列クイックソートと並列マージを用いた場合の結果である．

5.1.1　提案手法に対する実験結果

まず，入力 1 に対する提案アルゴリズムの実行時間を図 19，図 20，及び，図 21 に示す．これらの図では，横軸が同時マージブロック数 B を表しており，縦軸が実行時間を表している．更に，ソートフェーズとマージフェーズにかかった時間を，それぞれ橙色と紺色で表している．この結果から，$M = 8$，$B = 8$ の場合が最も高速に動作することがわかる．$M = 8$，$B = 8$ のとき，8 個のブロックを同時にマージするため，処理する CPU 数は 1

つである．つまり，マージを逐次で行うときが，提案手法が最も高速に動作することがわかる．よって，本実験環境においては，マージは並列に実行する必要がないことがわかる．

次に，入力 2 に対する提案アルゴリズムの実行時間を図 22，図 23，及び，図 24 に示す．図 22 は $M = 8$，$B = 8$，図 23 は $M = 16$，$B = 8$，及び，図 24 は $M = 32$，$B = 8$ の条件で実行したものである．これらは，入力 1 に対する実験結果において，各分割数 M における，最も高速に動作した M と B の組み合せを用いてアルゴリズムを実行しており，横軸が入力文字種類数，縦軸が実行時間を表している．どの図においても，入力文字種類数が少ないときでも安定して実行できることがわかる．特に，入力文字種類数が減少するにつれて，若干高速化する傾向がみられる．

5.1.2 実験結果に対する考察

まず最初に，入力 1 に対する，$M = 8$，$B = 8$ の場合での提案手法の実行時間の最大値と最小値を表 5 に示す．また，比較のためにマップソートの結果も同時に示す．表 5 より，提案手法は，アルゴリズム内にランダムで決定する要素がないため，実行時間は常に安定していることが確認できる．

次に，入力 1，2 に対する各アルゴリズムの実行時間を図 25 に示す．図 25 より，提案手法は既存の並列マージソートよりも，マージ処理が高速に行えていることがわかる．またマップソートと比較しても，同等の実行時間であるといえる．更に，偏った入力に対しては提案手法の方がマップソートより有効であることがわかる．

これらの実験結果より，提案手法は，マップソートよりもランダムな入力に対しては少し実行が遅くなるが，偏った入力に対しては大きく実行速度を改善できることがわかる．また，同じ入力に対する実行時間も安定している．よって，マップソートよりも提案手法の方が，より実用的な並列ソートアルゴリズムであるといえる．

表 5: 提案手法の実行時間の最小値, 及び, 最大値

	最小値	最大値	平均
マップソート	30.603	56.020	38.111
提案手法	40.858	41.430	41.198

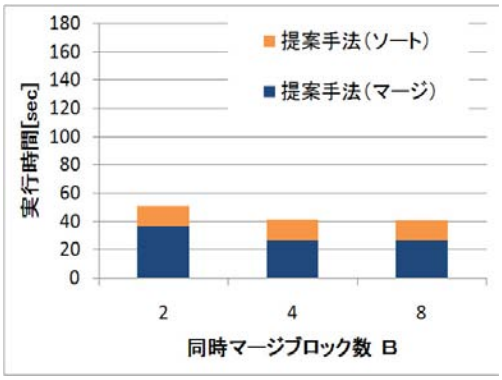


図 19: 提案手法の分割数別実行時間 (M=8)

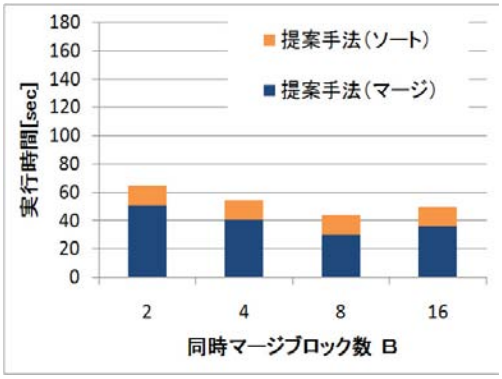


図 20: 提案手法の分割数別実行時間 (M=16)

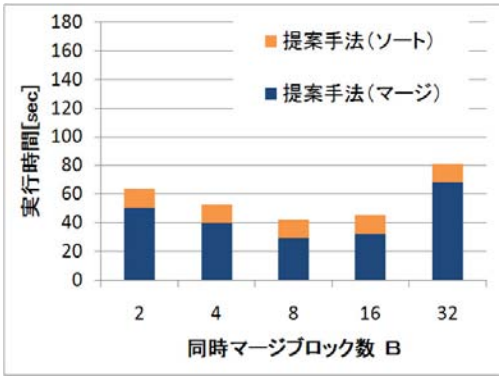


図 21: 提案手法の分割数別実行時間 (M=32)

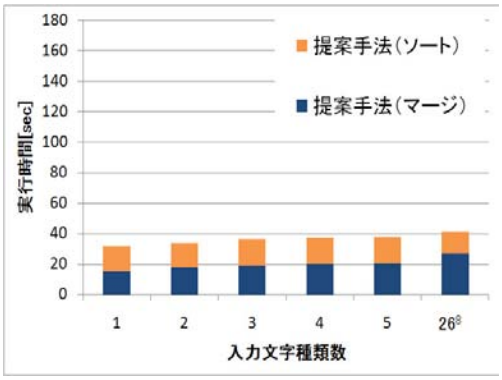


図 22: 提案手法の入力文字種類数別実行時間 (M=8, B=8)

6 まとめ

本研究では，まず最初に，既存の並列ソートアルゴリズムの性能の検証を行った．その結果，マップソートは，入力要素が偏った場合，実行が遅くなることがわかった．更に，マップソートは実行により，実行時間が大きく変化する問題点があることもわかった．また，並列マージソートは，どのような入力に対しても実行時間が大きく変化することはないが，マージ操作に時間がかかるため，全体の実行が遅くなることがわかった．

次に，並列マージソートを改良した並列ソートアルゴリズムを提案した．また，この提案アルゴリズムと既知のソートアルゴリズムの実験結果を比較し，検証を行った．検証の結果，提案アルゴリズムは，入力が偏っていない場合の実行時間がマップソートとほとんど変わらず，偏った入力に対してはマップソートよりも高速に実行可能であることがわかった．

今後の課題としては，より大きな入力に対する提案アルゴリズムの実用性の検証や，マージ操作の更なる高速化，さらには，全く新しいアイデアを用いた並列ソートアルゴリズムの提案等が挙げられる．

参考文献

[1] *AMD Multi-Core Products.* http://multicore.amd.com/en/Products/, 2006.

[2] *Multi-Core from Intel - Products and Platforms.* http://www.intel. com/multi-core/products.htm, 2006.

[3] D.Bader, V.Kanada, K.Madduri. SWARM:A Parallel Programming Framework for Multicore Processos. *Int'l Parallel and Distributed Processing Symp*, 2007.

[4] J.Kahle, M.Day, H.Hofstee, C.Johns, T.Maeurer, and D.Shippy. *Introduction to the Cell multiprocessor.* IBM J.Res.Dev.,49(4/5):589-604, 2005.

[5] P.Kongetira, K.Aingaran, and K.Olukotun. Niagara:a 32-way multithreaded Sparc processes. IEEE Micro, 25(2):21-29, 2005.

[6] 枝廣正人, 山下慶子. Map Sort:マルチコアプロセッサに向けたスケーラブルなソートアルゴリズム. 情報処理学会研究報告, 2007.

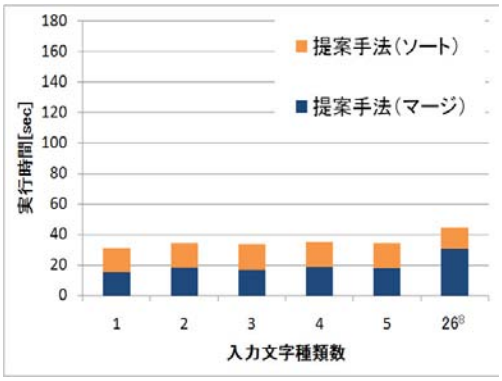


図 23: 提案手法の入力文字種類数別実行時間 (M=16, B=8)

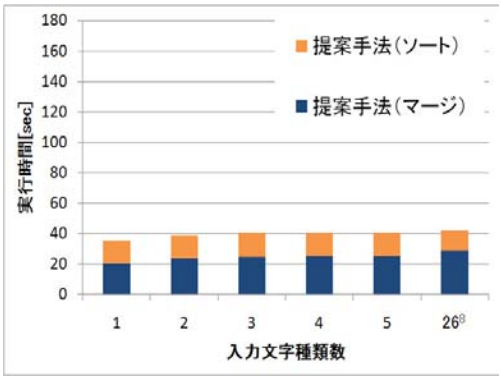


図 24: 提案手法の入力文字種類数別実行時間 (M=32, B=8)

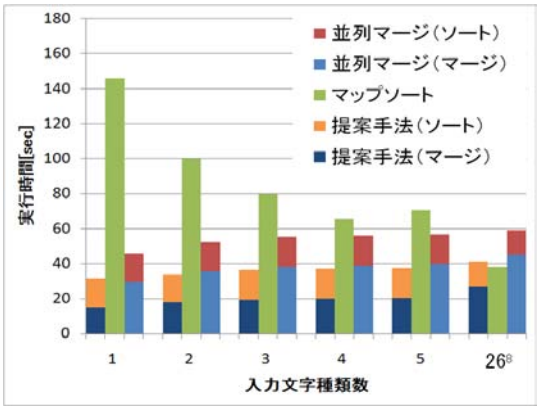


図 25: 入力文字種類数別の実行時間比較

膜計算における辞書演算及び最大値計算

田川 博文　　藤原 暁宏

九州工業大学 大学院情報工学研究院

h-tagawa@zodiac30.cse.kyutech.ac.jp　fujiwara@cse.kyutech.ac.jp

概 要

近年，新しい計算パラダイムの 1 つである膜計算が注目を集めている．本研究では，この膜計算において，辞書演算，及び，最大値計算を実行する P システムを提案する．まず，膜計算を用いて辞書演算である挿入，探索，削除の操作を実行す P システムを示す．これらの P システムはいずれも $O(mn)$ 種類のオブジェクトを用いて $O(1)$ ステップで実行可能である．さらに，この辞書演算を用いることにより，入力に対応する解を記憶するメモリ機能も実現可能であることを示す．次に，膜計算を用いて最大値を計算する P システムを示す．この P システムは，n 個の m ビットの 2 進数を表すオブジェクトに対してその最大値を計算するものであり，$O(mn)$ 種類のオブジェクトを用いて $O(m)$ ステップで実行可能である．

1　はじめに

現在のシリコンコンピュータは物理的な高速化の限界が存在するため，更なる計算能力の向上のためには，非シリコンコンピュータの開発が必要である．この非シリコンコンピュータの有望な候補として注目を集めているものに，自然界の生命活動を並列計算として扱うナチュラルコンピューティングがあり，このナチュラルコンピューティングの 1 つとして，膜計算 (P システム) が注目を集めている．

この膜計算は，Păun[3] によって提案された，生物の細胞活動を計算に用いる計算モデルである．生物の細胞内では，それぞれの膜で区切られた細胞内の要素 (核，細胞膜，液胞など) は独自に生命活動を行っており，膜計算では，この生命活動を並列計算と考えて計算モデルとしている．

膜計算では，細胞内の要素をオブジェクトと呼び，各オブジェクト，及び，膜に対して進化規則を適用し，オブジェクト，及び，膜を進化させることによって計算を行う．この計算モデルは，データ量が増加すると指数的に計算時間が増大するような問題に威力を発揮し，従来の計算機では指数的な時間を必要とする NP 完全問題を多項式ステップで解くことが可能となる．このモデルの提案以降，様々な NP 完全問題を解く P システム [1, 2, 4, 5] が提案されており，また，論理演算，加減算，及び，ソートを実行する P システム [7] も提案されている．

本研究では，この膜計算において，辞書演算，及び，最大値計算を実行する P システムを提案する．まず，膜計算を用いて辞書演算である挿入，探索，削除の操作を実行する P システムを示す．この P システムは，膜に含まれるオブジェクトとして保存されたデータに対して挿入，探索，削除の操作を実行するものであり，いずれの P システムも，$O(mn)$ 種類のオブジェクトを用いて $O(1)$ ステップで実行可能である．さらに，この辞書演算を用いることにより，入力に対応する解を記憶するメモリ機能も実現可能であることを示す．

次に，膜計算を用いて最大値を計算する P システムを示す．この P システムは，n 個の m ビットの 2 進数を表すオブジェクトに対してその最大値を計算するものであり，2 進数の最上位ビットから並列に比較を行うことにより，$O(mn)$ 種類のオブジェクトを用いて $O(m)$ ステップで実行可能である．

2　準備

2.1　膜構造とオブジェクト

膜計算の基本要素である膜構造とオブジェクトについて簡単に説明する．膜計算において用いられる膜構造とオブジェクトは，生物の細胞等をモデル化したものである．

膜計算で用いられる膜構造を図 1 の例を用いて説明する．

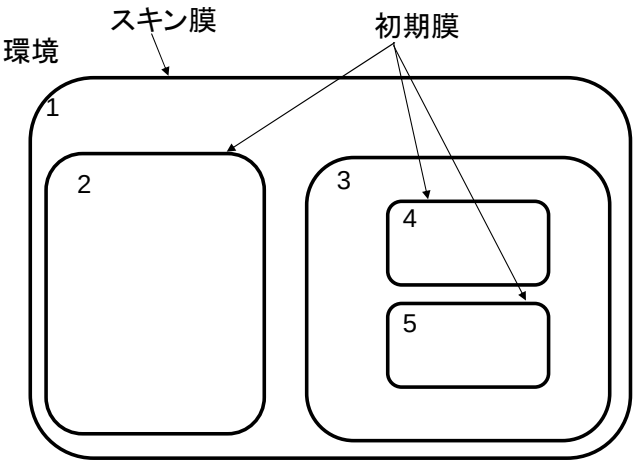


図 1: 膜構造

この膜構造は階層的にとらえることができ，最も外側の膜 (それ以上外側に膜がない膜) を**スキン膜**という．逆に，最も内側の膜 (それ以上内側に膜がない膜) を**初期**

膜という．また，それぞれの膜によって仕切られた空間を**膜空間**といい，スキン膜の外の空間を**環境**という．また，それぞれの膜は整数のラベルにより識別される．本研究では，整数 j によりラベル付けされた膜を，膜 j と表記する．

次に，膜構造の表記法について説明する．膜構造の表記法は膜に対応したラベルを添字とする左括弧と右括弧の組を用いて表記する．例えば，図 1 の膜構造は，以下のように表される．

$$[_1\ [_2\ ]_2\ [_3\ [_4\ ]_4\ [_5\ ]_5\ ]_3\ ]_1$$

なお，1 つの膜に含まれる膜の順序関係は存在しないので，以下のような表記でも，上記と同じ膜構造を表している．

$$[_1\ [_2\ ]_2\ [_3\ [_5\ ]_5\ [_4\ ]_4\ ]_3\ ]_1$$

また，各膜は，電気的極性 $e \in \{-, 0, +\}$ を持つ．例えば，図 1 の膜 5 が $+$ という極性を持つ場合以下のように記述する．

$$[_1\ [_2\ ]_2\ [_3\ [_4\ ]_4\ [_5\ ]_5^+\ ]_3\ ]_1$$

なお，一般的に，電気的極性 $e = 0$ である場合は，極性の表記を省略する．

次に，**オブジェクト**について説明する．各膜はオブジェクトと呼ばれる要素を含む．例えば，図 1 の膜 5 の中にオブジェクト a が含まれるとすると，膜構造は以下のように表記される．

$$[_1\ [_2\ ]_2\ [_3\ [_4\ ]_4\ [_5\ a\ ]_5\ ]_3\ ]_1$$

また，同じ膜内にオブジェクト a, b, c が存在すると，abc のように文字列として表記し，同じオブジェクトが複数存在する場合は，a^n と表記することにより，オブジェクト a が n 個存在することを表す．

最後に，膜計算の重要な要素である**進化規則**について説明する．進化規則とは，オブジェクトが進化するための規則であり，各膜において進化規則に従ってオブジェクトが進化していくことにより計算を行う．例えば，$a \to b$ という進化規則があった場合，これはオブジェクト a が次ステップでオブジェクト b に進化することを表している．この進化規則については次節で詳しく説明する．

2.2 膜計算モデル

本稿では，文献 [1] で提案されている P システムを用いる．1 つの P システム Π は以下のように定義される．

$$\Pi = (O, \mu, \omega_1, \omega_2, \cdots, \omega_m, R_1, R_2, \cdots, R_m, i_{in}, i_{out})$$

ここで，各要素の定義は以下の通りである．

O: 使用する全てのオブジェクトの集合

ω_j: 膜 $j \in H$ の中に初期状態で存在するオブジェクトの集合

R_j: 膜 $j \in H$ に対して適用される規則の集合

i_{in}: 入力膜のラベル

i_{out}: 出力膜のラベル

ここで，集合 H は使用するすべての膜のラベルの集合である．

本研究では，文献 [1] に基づき，オブジェクト進化規則，内部通信規則，外部通信規則，膜分解規則，及び，膜分割規則という 5 種類の進化規則を用いる．以下に，この 5 つの進化規則の定義を示す．

オブジェクト進化規則

オブジェクト進化規則は以下のように定義される．

$$[_h\ a\ ]_h^{e_1} \to [_h\ b\ ]_h^{e_2}$$

ただし，$h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$ である．

オブジェクト進化規則は，最も基本的な規則であり，各膜に存在するオブジェクトを別のオブジェクトに進化 (変化) させる規則である．また，オブジェクト進化規則は，オブジェクトだけが進化する場合は，膜のラベルと電気的極性を省略する．例えば，

$$[_h\ a\ ]_h^0 \to [_h\ b\ ]_h^0$$

という規則の場合は，

$$a \to b$$

と省略して記述する．

内部通信規則

内部通信規則は以下のように定義される．

$$a[_h\ ]_h^{e_1} \to [_h\ b\ ]_h^{e_2}$$

ただし，$h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$ である．

内部通信規則は，オブジェクトを進化させると同時に内側の膜に移動する規則である．

外部通信規則

外部通信規則は以下のように定義される．

$$[_h\ a\ ]_h^{e_1} \to b[_h\ ]_h^{e_2}$$

ただし，$h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$ である．

外部通信規則は内部通信規則と逆の働きをする進化規則であり，オブジェクトを進化させると同時に外側の膜に移動する規則である．

膜分解規則

膜分解規則は以下のように定義される．

$$[_h\ a\ ]_h^e \to b$$

ただし，$h \in H, e \in \{+, -, 0\}, a, b \in O$ である．

膜分解規則は，オブジェクトを進化させると同時に，そのオブジェクトが存在する膜を分解 (消去) する規則である．膜が消去されることにより，その膜の中のオブジェクトは，1 つ外側の膜の中に存在することになる．なお，スキン膜を分解することはできないので，そのような膜分解規則を定義することはできない．

膜分割規則

膜分割規則は以下のように定義される．

$$[_h\ a\ ]_h^{e_1} \rightarrow [_h\ b\ ]_h^{e_2}[_h\ c\ ]_h^{e_3}$$

ただし，$h \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$ である．

膜分割規則は，オブジェクトを進化させると同時に，そのオブジェクトが存在した膜を分割 (分裂) させる規則である．なお，スキン膜を分割することはできないので，そのような膜分割規則を定義することはできない．

次に，膜計算における進化規則の適用に関する基本的原理として，**最大並列則**と**非決定性**という 2 つの規則を説明する．最大並列則とは，ある時点において適用できるすべての規則が適用されるというものであり，これにより膜計算を並列計算とみなすことができる．また，非決定性とは，ある時点においてあるオブジェクトに適用できる規則は複数存在する場合は，適用される規則が非決定的に選択されるというものである．

本研究では，ある時点で，適用可能なすべての規則を適用し，次の状態に進化することを**遷移**と呼び，その遷移の計算量を $O(1)$ ステップと定義する．また，計算の終了は，適用できる規則がなく，それ以上進化が進まないときに計算が終了するものとする．

3 2進数を表すデータ表現

本研究では，文献 [7] で提案されたデータ構造を用い，2 進数のデータを表現する．以下に，文献 [7] で提案された 2 進数の 1 ビットを表すデータ構造を示す．

$$\langle A_i, B_j, V_{i,j} \rangle$$

ここで，$i(0 \leq i \leq n-1)$ は 2 進数の格納されるアドレスを表し，$j(0 \leq j \leq m-1)$ は 2 進数の中のビット番号を表す．さらに，$V_{i,j}(\in 0, 1)$ は，表すビットの論理値を表している．

上記のオブジェクトを用いて，アドレス i に格納された 2 進数の値 V_i は，

$$\langle A_i, B_{m-1}, V_{i,m-1} \rangle \langle A_i, B_{m-2}, V_{i,m-2} \rangle, \cdots, \langle A_i, B_0, V_{i,0} \rangle$$

という m 個のオブジェクトで表すことができる．ここで，

$$V_i = \sum_{j=0}^{m-1} V_{i,\,j} \times 2^j$$

である．例えば，アドレス 8 に格納された 2 進数 1011 は，以下の 4 つのオブジェクトにより表される．

$$\langle A_8, B_3, 1 \rangle \langle A_8, B_2, 0 \rangle \langle A_8, B_1, 1 \rangle \langle A_8, B_0, 1 \rangle$$

従って，n 個の m ビットの 2 進数は，$O(mn)$ 個のオブジェクトを用いて表すことができる．

4 辞書演算

本節では，辞書演算の基本操作である挿入，探索，削除の操作を実行する P システムを示す．まず，挿入，探索，削除のそれぞれの操作について，その P システムを定義し，P システムの動作の概要，実行例及び，P システムの計算量を示す．次に，挿入，探索，削除操作を連続して実行できる様に P システムの改良を行う．最後に，辞書演算を用いてメモリ機能を実行する P システムを示す．

4.1 挿入操作

4.1.1 膜計算における挿入操作の入出力

以下に，膜計算における挿入操作の入出力を表すオブジェクトを示す．

入力 アドレス I に格納された m ビットの 2 進数のデータを表すオブジェクトを入力とする．なお，これらのオブジェクトは，スキン膜に入力として与える．
$\langle A_I, B_0, V_{I,0} \rangle$，$\langle A_I, B_1, V_{I,1} \rangle$，$\cdots$，
$\langle A_I, B_{m-1}, V_{I,m-1} \rangle$
また，辞書演算のデータは，スキン膜内部のリスト膜と呼ばれる膜に格納されているものとする．

出力 入力の m ビットの 2 進数のデータが挿入された集合を出力とする．ただし，入力データと重複するデータが存在する場合は，入力のデータ集合をそのまま出力とする．

$$\begin{gathered}
\langle A_0, B_0, V_{0,0} \rangle,\ \langle A_0, B_1, V_{0,1} \rangle,\ \cdots, \\
\langle A_0, B_{m-1}, V_{0,m-1} \rangle \\
\langle A_1, B_0, V_{1,0} \rangle,\ \langle A_1, B_1, V_{1,1} \rangle,\ \cdots, \\
\langle A_1, B_{m-1}, V_{1,m-1} \rangle \\
\vdots \\
\langle A_i, B_0, V_{i,0} \rangle,\ \langle A_i, B_1, V_{i,1} \rangle,\ \cdots, \\
\langle A_i, B_{m-1}, V_{i,m-1} \rangle
\end{gathered}$$

4.1.2 P システムの概要

本節で示す挿入操作を行う P システムのアルゴリズムは，指定した m ビットの 2 進数のデータを集合に挿入する．

まず，入力の m ビットの 2 進数のデータをアドレス I に入れる．

$$\langle A_I, B_0, V_{I,0} \rangle, \langle A_I, B_1, V_{I,1} \rangle, \cdots, \langle A_I, B_{m-1}, V_{I,m-1} \rangle$$

このオブジェクトがスキン膜に入力として与えられる．

挿入操作を実行するアルゴリズムは，大きく分けて以下の 2 つのステップで構成される．まず，1 つ目のステップ (Step 1) では，入力オブジェクトが表すデータが保存されている集合のデータと重複しているか調べる．次に，2 つ目のステップ (Step 2) では，重複するデータが存在する場合は，このオブジェクトを挿入せず，重複す

るデータが存在しない場合は，このオブジェクトを集合に挿入する．

以下に挿入操作を行うアルゴリズムの概要を示す．

Step 1 以下のサブステップを実行することにより，重複するデータが存在するか否か調べる．

(1-1) アドレス I に格納された 2 進数のデータを表すオブジェクトを list というラベルで表された膜へ移動する．以下，この膜をリスト膜と呼ぶ．また，この際アドレス I' にアドレス I のデータを入れる．

(1-2) リスト膜において，入力の挿入データをアドレス I'' に保存する．それと同時に，ビット毎に用意された膜，I_0，I_1，$\cdots$，I_{m-1} 膜にそれぞれ，アドレス I' の 0，1，$\cdots$，$m-1$ ビットのデータを移動する．以後，ビット毎に用意された膜をビット膜と呼ぶ．
重複したデータが存在する場合は，アドレス I'' に保存しておいたデータを削除する．また重複したデータが存在しない場合は，アドレス I'' に保存したおいたデータを挿入する．

(1-3) 各ビット膜 I_0，I_1，$\cdots$，I_{m-1} 膜において，それぞれ，指定したデータの 0，1，$\cdots$，$m-1$ ビットのデータを集合内に存在するアドレスの個数分，CP(Copy) オブジェクトに複製する．
重複するデータの有無を調べるためには，集合のデータと指定したデータを 1 アドレス，1 ビット毎のオブジェクトと比較する必要がある．ゆえに，各ビット膜において，それぞれ，指定したデータを集合内に存在するアドレスの個数分だけ複製する．

(1-4) 各ビット膜において，指定したデータを複製したオブジェクトをリスト膜内の集合データと比較するために，リスト膜に移動する．

(1-5) リスト膜において，1 アドレス，1 ビット毎に集合内のデータと指定したデータが同じか否かを調べる．
同じデータの場合，judge というラベルで表された膜に $IMCH(i)$(Insert Match) オブジェクトを作成する．以下，この膜をジャッジ膜と呼ぶ．ここで，i には集合内のデータのアドレス番号が入る．この際，ジャッジ膜に $IC(0)$(Insert Count) オブジェクトを 1 つだけ作成する．この $IC(0)$ オブジェクトは，重複するデータがない場合，$IC(2)$ まで進化して，重複するデータが存在しないことを意味するオブジェクト $\langle A_{Iout}, B_0, 0\rangle$ をリスト膜へ作成する．また，指定したデータと集合のデータが異なる場合は，何もしない．

(1-6) ジャッジ膜において，$IMCH(i)$ オブジェクトが m ビット分存在するか否かを調べる．
m ビット分存在する場合は，アドレス i に指定したデータと重複するデータが存在する．ゆえに，重複するデータが存在することを意味するオブジェクト $\langle A_{Iout}, B_0, 1\rangle$ をリスト膜に作成する．m ビット分存在しない場合は，指定したデータと重複するデータは存在しないので，$IC(0)$ オブジェクトは $IC(2)$ まで進化し，重複するデータが存在しないことを意味するオブジェクト $\langle A_{Iout}, B_0, 0\rangle$ をリスト膜に作成する．

Step 2 重複するデータが存在しなければ，アドレス I'' に保存していた指定したデータを挿入する．重複するデータが存在するならば，アドレス I'' に保存していた指定したデータを削除する．
リスト膜において，オブジェクト $\langle A_{Iout}, B_0, 1\rangle$ が存在する場合，集合内に指定したデータと重複するデータが存在する．ゆえに，リスト膜を $+$ に帯電させてアドレス I'' に保存されているデータを削除する．また，オブジェクト $\langle A_{Iout}, B_0, 0\rangle$ が存在する場合は，集合内に指定したデータと重複したデータは存在しないので，リスト膜を $-$ に帯電させてアドレス I'' に保存されているデータを挿入する．

以下に辞書演算の挿入操作を実行する P システム Π_{insert} を示す．

$$\Pi_{insert} = (O, \mu, \omega_{skin}, \omega_{list}, \omega_{judge}, \omega_{I_j}, R_{skin}, R_{list}, R_{judge}, R_{I_j}, i_{in}, i_{out} | 0 \le j \le m-1)$$

ここで，Π_{insert} を構成するそれぞれの集合は以下ようになる．

- $O = \{\langle A_I, B_j, V_{I,j}\rangle, \langle A_{I'}, B_j, V_{I',j}\rangle, \langle A_{I''}, B_j, V_{I'',j}\rangle, \langle A_i, B_j, V_{i,j}\rangle, \langle A_{Iout}, B_0, V_{Iout,0}\rangle, CP, LTA(i), LTp, IMCH(i), IC(k) | 0 \le i \le n-1, 0 \le j \le m-1, 0 \le k \le 2, V_{I,j}, V_{I',j}, V_{I'',j} \in \{0,1\}\}$
- $\mu = [_{skin}\,[_{list}\,[_{judge}\,]_{judge}\,[_{I_0}\,]_{I_0}\,[_{I_1}\,]_{I_1} \cdots [_{I_{m-1}}\,]_{I_{m-1}}\,]_{list}\,]_{skin}$
- $\omega_{skin} = \phi$
- $\omega_{I_j} = \{CP^i | 0 \le i \le n-1\}$
- $\omega_{list} = \{LTA(-1)^m\}$
- $\omega_{judge} = \{LTA(-1)\}$
- $R_{skin} = \{[_{skin}\langle A_I, B_j, 0\rangle]_{skin} \to [_{list}\langle A_{I'}, B_j, 0\rangle]_{list}, [_{skin}\langle A_I, B_j, 1\rangle]_{skin} \to [_{list}\langle A_{I'}, B_j, 1\rangle]_{list}\}$
- $R_{list} = \{$
  $[_{list}\langle A_{I'}, B_j, 0\rangle LTA(i)]_{list} \to [_{list}\langle A_{I''}, B_j, 0\rangle LTA(i)]_{list}[_{I_j}\langle A_{I'}, B_j, 0\rangle]_{I_j}$,
  $[_{list}\langle A_{I'}, B_j, 1\rangle LTA(i)]_{list} \to [_{list}\langle A_{I''}, B_j, 1\rangle LTA(i)]_{list}[_{I_j}\langle A_{I'}, B_j, 1\rangle]_{I_j}$,
  $[_{list}\langle B_{I_0}, 0\rangle\langle A_0, B_0, 0\rangle]_{list} \to [_{list}\langle A_0, B_0, 0\rangle]_{list}[_{judge} IMCH(0) IC(0)]_{judge}$,

$[_{list}\langle B_{I_0},1\rangle\langle A_0,B_0,1\rangle]_{list}$
$\quad\rightarrow [_{list}\langle A_0,B_0,1\rangle]_{list}[_{judge}IMCH(0)IC(0)]_{judge}$,
$[_{list}\langle B_{I_j},0\rangle\langle A_i,B_j,0\rangle]_{list}$
$\quad\rightarrow [_{list}\langle A_i,B_j,0\rangle]_{list}[_{judge}IMCH(i)]_{judge}$,
$[_{list}\langle B_{I_j},1\rangle\langle A_i,B_j,1\rangle]_{list}$
$\quad\rightarrow [_{list}\langle A_i,B_j,1\rangle]_{list}[_{judge}IMCH(i)]_{judge}$,
$[_{list}\langle B_{I_0},0\rangle\langle A_0,B_0,1\rangle]_{list}$
$\quad\rightarrow [_{list}\langle A_0,B_0,1\rangle]_{list}[_{judge}IC(0)]_{judge}$,
$[_{list}\langle B_{I_0},1\rangle\langle A_0,B_0,0\rangle]_{list}$
$\quad\rightarrow [_{list}\langle A_0,B_0,0\rangle]_{list}[_{judge}IC(0)]_{judge}$,
$[_{list}\langle B_{I_j},0\rangle\langle A_i,B_j,1\rangle]_{list}$
$\quad\rightarrow [_{list}\langle A_i,B_j,1\rangle]_{list}$,
$[_{list}\langle B_{I_j},1\rangle\langle A_i,B_j,0\rangle]_{list}$
$\quad\rightarrow [_{list}\langle A_i,B_j,0\rangle]_{list}$,
$[_{list}\langle A_{Iout},B_0,0\rangle]_{list}\rightarrow [_{list}]^-_{list}$,
$[_{list}\langle A_{Iout},B_0,1\rangle]_{list}\rightarrow [_{list}]^+_{list}$,
$[_{list}\langle A_{I''},B_j,0\rangle]^+_{list}\rightarrow [_{list}]_{list}$,
$[_{list}\langle A_{I''},B_j,1\rangle]^+_{list}\rightarrow [_{list}]_{list}$,
$[_{list}\langle A_{I''},B_0,0\rangle]^-_{list}$
$\quad\rightarrow [_{list}\langle A_I,B_0,0\rangle]_{list}[_{I_0}CP]_{I_0}[_{judge}LTp]_{judge}$,
$[_{list}\langle A_{I''},B_0,1\rangle]^-_{list}$
$\quad\rightarrow [_{list}\langle A_I,B_0,1\rangle]_{list}[_{I_0}CP]_{I_0}[_{judge}LTp]_{judge}$,
$[_{list}\langle A_{I''},B_j,0\rangle]^-_{list}$
$\quad\rightarrow [_{list}\langle A_I,B_j,0\rangle]_{list}[_{I_j}CP]_{I_j}$,
$[_{list}\langle A_{I''},B_j,1\rangle]^-_{list}$
$\quad\rightarrow [_{list}\langle A_I,B_j,1\rangle]_{list}[_{I_j}CP]_{I_j}$,
$[_{list}LTA(-1)\langle A_{I'},B_0,0\rangle]_{list}$
$\quad\rightarrow [_{list}LTA(0)\langle A_0,B_0,0\rangle]_{list}[_{I_0}CP]_{I_0}$
$\quad [_{judge}LTp]_{judge}$,
$[_{list}LTA(-1)\langle A_{I'},B_0,1\rangle]_{list}$
$\quad\rightarrow [_{list}LTA(0)\langle A_0,B_0,1\rangle]_{list}[_{I_0}CP]_{I_0}$
$\quad [_{judge}LTp]_{judge}$,
$[_{list}LTA(-1)\langle A_{I'},B_j,0\rangle]_{list}$
$\quad\rightarrow [_{list}LTA(0)\langle A_0,B_j,0\rangle]_{list}[_{I_j}CP]_{I_j}$,
$[_{list}LTA(-1)\langle A_{I'},B_j,1\rangle]_{list}$
$\quad\rightarrow [_{list}LTA(0)\langle A_0,B_j,1\rangle]_{list}[_{I_j}CP]_{I_j}$,
$[_{list}LTA(i)\langle A_I,B_j,0\rangle]^-_{list}$
$\quad\rightarrow [_{list}LTA(i+1)\langle A_{i+1},B_j,0\rangle]_{list}$,
$[_{list}LTA(i)\langle A_I,B_j,1\rangle]^-_{list}$
$\quad\rightarrow [_{list}LTA(i+1)\langle A_{i+1},B_j,1\rangle]_{list}\}$

- $R_{I_j}=\{[_{I_j}\langle A_{I'},B_j,0\rangle]_{I_j}\rightarrow [_{I_j}]^-_{I_j}$,
$[_{I_j}\langle A_{I'},B_j,1\rangle]_{I_j}\rightarrow [_{I_j}]^+_{I_j}$,
$[_{I_j}CP]^-_{I_j}\rightarrow [_{I_j}\langle B_{I_j},0\rangle]_{I_j}$,
$[_{I_j}CP]^+_{I_j}\rightarrow [_{I_j}\langle B_{I_j},1\rangle]_{I_j}$,
$[_{I_j}\langle B_j,0\rangle]_{I_j}\rightarrow [_{I_j}CP]_{I_j}[_{list}\langle B_j,0\rangle]_{list}$,
$[_{I_j}\langle B_j,1\rangle]_{I_j}\rightarrow [_{I_j}CP]_{I_j}[_{list}\langle B_{I_j},1\rangle]_{list}\}$

- $R_{judge}=\{[_{judge}IMCH(i)^m]_{judge}$
$\quad\rightarrow [_{judge}]^+_{judge}[_{list}\langle A_{Iout},B_0,1\rangle]_{list}$,
$\quad [_{judge}IC(0)]_{judge}\rightarrow [_{judge}IC(1)]_{judge}$,
$\quad [_{judge}IC(1)]_{judge}\rightarrow [_{judge}IC(2)]_{judge}$,
$\quad [_{judge}IC(2)]_{judge}$
$\quad\rightarrow [_{list}\langle A_{Iout},B_0,0\rangle]_{list}[_{judge}]^+_{judge}$,
$\quad [_{judge}IC(1)]^+_{judge}\rightarrow [_{judge}]_{judge}$,
$\quad [_{judge}IMCH(i)]^+_{judge}\rightarrow [_{judge}]_{judge}$,
$\quad [_{judge}LTA(i)LTp]_{judge}$
$\quad\rightarrow [_{judge}LTA(i+1)]_{judge}\}$

- $i_{in}=skin$
- $i_{out}=list$

この計算モデルでは入力膜はスキン膜であり，出力膜はリスト膜である．

ここで，挿入操作を行う P システムにおける各オブジェクトの役割を説明する．

$LTA(i)$(Last Address): 集合の最終のアドレス i を保持するオブジェクト

CP(Copy): データの複製に必要なオブジェクト

$IMCH(i)$(Insert Match): 値が同じオブジェクトがある場合に，オブジェクトのアドレス i を保持するオブジェクト

$IC(0)$(Insert Count): 重複するデータを調べる際に，重複するデータの有無を調べるカウンタを意味するオブジェクト

LTp(Last Address Plus): $LTA(i)$ オブジェクトの i を 1 つ進めるためのオブジェクト

4.1.3 P システムの動作

膜計算における挿入操作の実行例を示すことにより，P システムの動作を説明する．ここでは，2 進数 10 のデータを，既に 2 進数 00，11 の入った集合に挿入される例を示す．よって，$\langle A_I,B_1,1\rangle$，$\langle A_I,B_0,0\rangle$ がスキン膜に入力として与えられるものとする．この膜計算モデルの初期状態は，図 2 である．

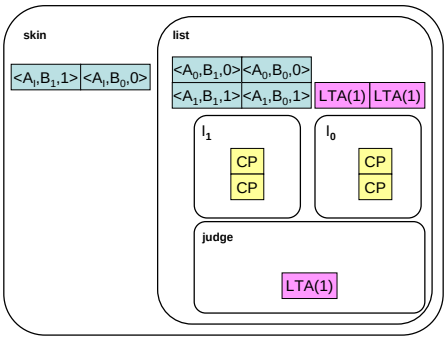


図 2: (1-1) 開始時における膜構造の状態

Step 1 重複するデータが存在するか調べる．

(1-1) アドレス I に格納された 2 進数のデータを表すオブジェクトをリスト膜へ移動する．

アドレス I に入っているオブジェクトをリスト膜へ移動する．この際，アドレス I' に値を入れる．

この例では，図 2 の状態に対して，進化規則

$$[_{skin}\langle A_I, B_1, 1\rangle]_{skin} \rightarrow [_{list}\langle A_{I'}, B_1, 1\rangle]_{list}$$

$$[_{skin}\langle A_I, B_0, 0\rangle]_{skin} \rightarrow [_{list}\langle A_{I'}, B_0, 0\rangle]_{list}$$

が適用され，図 3 の状態に遷移する．

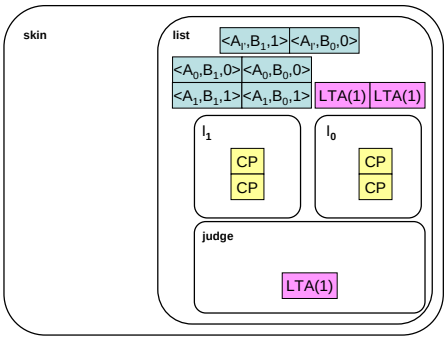


図 3: (1-2) 開始時における膜構造の状態

(1-2) リスト膜において，指定したデータをアドレス I'' に保存する．それと同時に，各ビット膜 I_0，I_1 膜にそれぞれ，アドレス I' の 0，1 ビット目のデータを移動する．

次に，重複するデータの有無を調べるために，アドレス I' のデータを 2 進数のデータにより，I_0，I_1 膜に指定したデータを移す．この際，アドレス I'' にデータを保存しておく．

この例では，前述の状態に対して，進化規則

$[_{list}\langle A_{I'}, B_1, 1\rangle LTA(1)]_{list}$
$\rightarrow [_{list}\langle A_{I''}, B_1, 1\rangle LTA(1)]_{list}[_{I_1}\langle A_{I'}, B_1, 1\rangle]_{I_1}$
$[_{list}\langle A_{I'}, B_0, 0\rangle LTA(1)]_{list}$
$\rightarrow [_{list}\langle A_{I''}, B_0, 0\rangle LTA(1)]_{list}[_{I_0}\langle A_{I'}, B_0, 0\rangle]_{I_0}$

が適用され，図 4 の状態へ進化する．

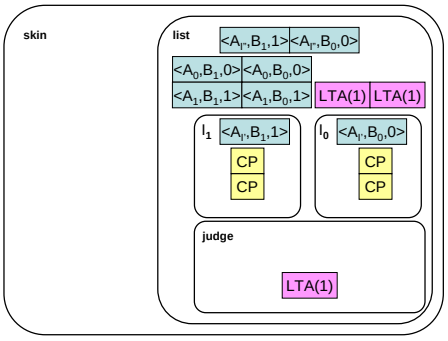


図 4: (1-3) 開始時における膜構造の状態

(1-3) 各ビット膜 I_0，I_1 膜において，それぞれ，指定したデータの 0，1 ビット目のデータを集合内に存在するアドレスの個数分，CP(Copy) オブジェクトに複製する．

ビット毎に用意された膜でアドレス I' のオブジェクトを複製する．各ビット膜において，アドレス I' の値が 0 の場合，そのビット膜を $-$ に帯電させ，値が 1 の場合はそのビット膜を $+$ に帯電させて，CP オブジェクトを全て同時に進化させる．

この例では，前述の状態に対して，進化規則

$$[_{I_1}\langle A_{I'}, B_1, 1\rangle]_{I_1} \rightarrow [_{I_1}]^{+}_{I_1}$$

$$[_{I_0}\langle A_{I'}, B_0, 0\rangle]_{I_0} \rightarrow [_{I_0}]^{-}_{I_0}$$

$$[_{I_1}CP]^{+}_{I_1} \rightarrow [_{I_1}\langle B_{I_1}, 1\rangle]_{I_1}$$

$$[_{I_0}CP]^{-}_{I_0} \rightarrow [_{I_0}\langle B_{I_0}, 0\rangle]_{I_0}$$

が適用され，図 5 の状態へ進化する．

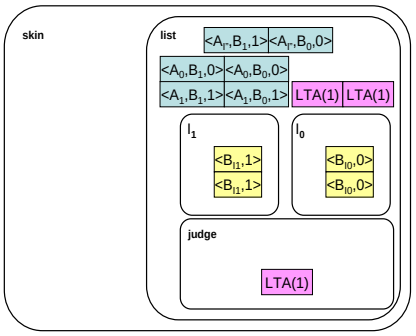


図 5: (1-4) 開始時における膜構造の状態

(1-4) 各ビット膜において，指定したデータを複製したオブジェクトをリスト膜へ移動する．

複製したオブジェクトを既にリスト膜内にあるデータと比較するために，リスト膜へ移す．この際，連続して挿入操作ができるように，アドレス I' のビット毎の値を複製したオブジェクトを CP オブジェクトにもどしておく．

この例では，前述の状態に対して，進化規則

$$[_{I_1}\langle B_{I_1}, 1\rangle]_{I_1} \rightarrow [_{I_1}CP]_{I_1}[_{list}\langle B_{I_1}, 1\rangle]_{list}$$

$$[_{I_0}\langle B_{I_0}, 0\rangle]_{I_0} \rightarrow [_{I_0}CP]_{I_0}[_{list}\langle B_{I_0}, 0\rangle]_{list}$$

が適用され，図 6 の状態へ遷移する．

(1-5) リスト膜において，1 アドレス，1 ビット毎に集合内のデータと指定したデータが同じか否かを調べる．

次に，$IMCH(i)$ オブジェクトをジャッジ膜に作成する．同じオブジェクトがある場合 $IMCH(i)$ オブジェクトができる．そうでない場合は $IMCH(i)$ オブジェクトは作成されない．また，このときアドレス 0 のビット 0 のオブジェクトに対してだけ，$IC(0)$ オブジェクトを作成する．

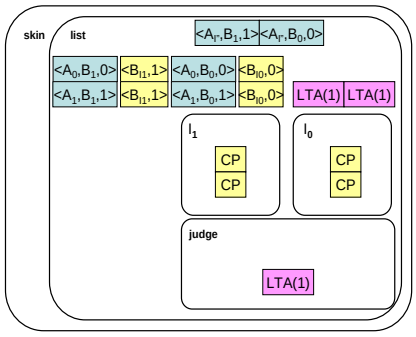


図 6: (1-5) 開始時における膜構造の状態

この例では，前述の状態に対して，進化規則

$[_{list}\langle B_{I_1},1\rangle\langle A_0,B_1,0\rangle]_{list} \rightarrow [_{list}\langle A_0,B_0,0\rangle]_{list}$

$[_{list}\langle B_{I_0},0\rangle\langle A_0,B_0,0\rangle]_{list}$
$\rightarrow [_{list}\langle A_0,B_0,0\rangle]_{list}[_{judge}IMCH(0)IC(0)]_{judge}$
$[_{list}\langle B_{I_1},1\rangle\langle A_1,B_1,1\rangle]_{list}$
$\rightarrow [_{list}\langle A_1,B_1,1\rangle]_{list}[_{judge}IMCH(1)]_{judge}$

$[_{list}\langle B_{I_0},0\rangle\langle A_1,B_0,1\rangle]_{list} \rightarrow [_{list}\langle A_1,B_0,1\rangle]_{list}$

が適用され，図 7 の状態へ遷移する．

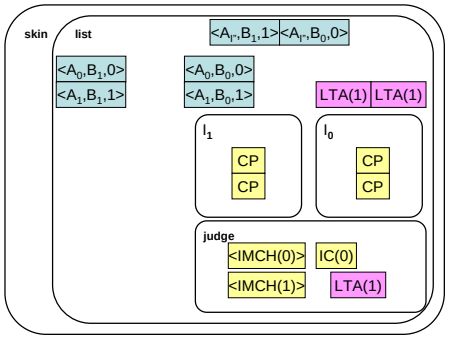


図 7: (1-6) 開始時における膜構造の状態

(1-6) ジャッジ膜において，$IMCH(i)$ オブジェクトが m ビット分存在するか否かを調べる．
この例においては，ジャッジ膜において，各 $IMCH(i)$ オブジェクトが 2 個，つまり，2 ビット分存在しないのでリスト膜内に指定したデータと同じデータはなかったということが判断できる．ゆえに $IC(0)$ オブジェクトが $IC(2)$ まで進化していき，$\langle A_{Iout},B_0,0\rangle$ がリスト膜へ作成される．このとき同時に，ジャッジ膜内に残ったオブジェクトを削除するために，ジャッジ膜を $+$ に帯電させる．ここで，$\langle A_{Iout},B_0,0\rangle$ が，リスト膜内に指定したデータと同じデータがないこと表す．
したがって，この例では，前述の状態に対して，進化規則

$$[_{judge}IC(0)]_{judge} \rightarrow [_{judge}IC(1)]_{judge}$$

$$[_{judge}IC(1)]_{judge} \rightarrow [_{judge}IC(2)]_{judge}$$

$$[_{judge}IC(2)]_{judge} \rightarrow [_{list}\langle A_{Iout},B_0,0\rangle]_{list}[_{judge}]^{+}_{judge}$$

$$[_{judge}IMCH(0)]^{+}_{judge} \rightarrow [_{judge}]_{judge}$$

$$[_{judge}IMCH(1)]^{+}_{judge} \rightarrow [_{judge}]_{judge}$$

が適用され，図 8 の状態になる．

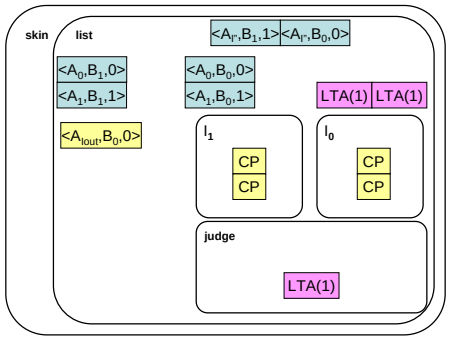


図 8: Step 2 開始時における膜構造の状態

また，もし重複したデータがあった場合は $\langle A_{Iout},B_0,1\rangle$ がリスト膜に作成され，リスト膜を $+$ に帯電させる．これにより，アドレス I'' に入っていたデータを削除する．この進化規則

$[_{judge}IMCH(i)^m]_{judge}$
$\rightarrow [_{judge}]^{+}_{judge}[_{list}\langle A_{Iout},B_0,1\rangle]_{list}$
$[_{list}\langle A_{Iout},B_0,1\rangle]_{list} \rightarrow [_{list}]^{+}_{list}$
$[_{list}\langle A_{I''},B_j,0\rangle]^{+}_{list} \rightarrow [_{list}]_{list}$
$[_{list}\langle A_{I''},B_j,1\rangle]^{+}_{list} \rightarrow [_{list}]_{list}$

が適用される．

Step 2 重複するデータが存在しなければ，アドレス I'' に保存していた指定したデータを挿入する．重複するデータが存在するならば，アドレス I'' に保存していた指定したデータを削除する．

この例では，リスト膜に，重複するデータが存在しなかったことを意味するオブジェクト $\langle A_{Iout},B_0,0\rangle$ が存在するので，重複するデータは存在しないことがわかる．ゆえに，挿入したいデータ，つまり，アドレス I'' に入っている値をリスト膜の集合に挿入する．
まず，重複するデータが存在しないことを意味するオブジェクト $\langle A_{Iout},B_0,0\rangle$ がリスト膜を $-$ に帯電させる．リスト膜が $-$ に帯電すると，アドレス I'' の値をアドレス I に入れる．同時に各ビット膜に CP オブジェクトを 1 つずつ作る．ジャッジ膜において，リスト膜に値が入るので，$LTA(i)$ オブジェクトを 1 つ進める必要がある．ゆえにジャッジ膜の $LTA(i)$ オブジェクトを 1 つ進めるために，オブジェクト LTp をジャッジ膜に作成する．

以上より，この例では，前述の状態に対して，進化規則

$$[_{list}\langle A_{Iout}, B_0, 0\rangle]_{list} \to [_{list}]^{-}_{list}$$

$$[_{list}\langle A_{I''}, B_1, 1\rangle]^{-}_{list} \to [_{list}\langle A_I, B_1, 1\rangle]_{list}[_{I_1}CP]_{I_1}$$

$$[_{list}\langle A_{I''}, B_0, 0\rangle]^{-}_{list} \to [_{list}\langle A_I, B_0, 0\rangle]_{list}[_{I_0}CP]_{I_0}[_{judge}LTp]_{judge}$$

が適用され，図 9 の状態に遷移する．

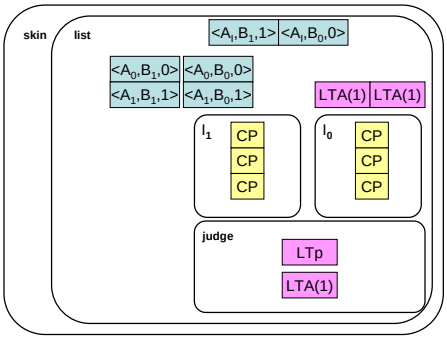


図 9: Step 2 における膜構造の状態

次に，リスト膜において，アドレス I の値を最終のアドレスに挿入する．また，ジャッジ膜において，$LTA(i)$ が 1 つ進む．この例では，前述の状態に対して，進化規則

$$[_{list}LTA(1)\langle A_I, B_1, 1\rangle]^{-}_{list} \to [_{list}LTA(2)\langle A_2, B_1, 1\rangle]_{list}$$

$$[_{list}LTA(1)\langle A_I, B_0, 0\rangle]^{-}_{list} \to [_{list}LTA(2)\langle A_2, B_0, 0\rangle]_{list}$$

$$[_{judge}LTA(1)LTp]_{judge} \to [_{judge}LTA(2)]_{judge}$$

が適用され，図 10 の状態に遷移する．

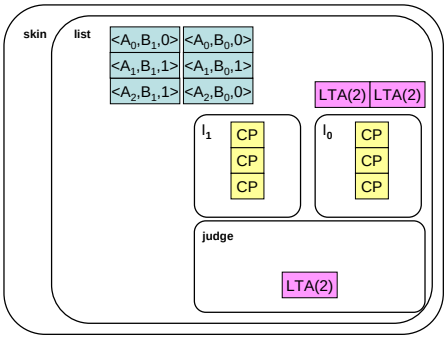


図 10: 挿入操作後の状態

これ以上，適用できる進化規則がないので計算は終了する．

4.1.4 P システムの計算量

挿入操作を実行する P システム Π_{insert} は，$O(mn)$ の進化規則の適用により実行が終了するので，以下の定理が得られる．

定理 1 n 個のアドレス，m ビットの 2 進数表記のオブジェクトを入力とし，辞書演算の挿入操作を計算する P システム Π_{insert} は，$O(mn)$ 種類のオブジェクト，$O(m)$ 個の膜，及び，$O(mn)$ 個の進化規則を用いて，$O(1)$ ステップで実行可能である． □

4.2 探索操作

探索操作では，以下の探索入出力オブジェクトで定義されるように，指定したデータが集合内つまりリスト膜内に存在すれば，スキン膜に真を出力し，存在しない場合はスキン膜に偽を出力する．

4.2.1 膜計算における探索操作の入出力

以下に，膜計算における探索操作の入出力を表すオブジェクトを示す．

入力 アドレス S に格納された m ビットの 2 進数のデータを表すオブジェクトを入力とする．なお，これらのオブジェクトは，スキン膜に入力として与える．
$\langle A_S, B_0, V_{S,0}\rangle$，$\langle A_S, B_1, V_{S,1}\rangle$，$\cdots$，
$\langle A_S, B_{m-1}, V_{S,m-1}\rangle$
また，辞書演算のデータは，スキン膜内部のリスト膜に格納されているものとする．

出力 スキン膜に，真を意味するオブジェクト $\langle A_{Sout}, B_0, 1\rangle$，もしくは，偽を意味するオブジェクト $\langle A_{Sout}, B_0, 0\rangle$ を出力する．

この探索操作は挿入操作と類似の操作なので，挿入操作を行う P システムと似たアイデアを用いた P システムにより実現可能である．(詳細は，[6] 参照のこと．) したがって，探索操作を行う P システム Π_{search} は以下の計算量で実現できる．

定理 2 n 個のアドレス，m ビットの 2 進数表記のオブジェクトを入力とし，辞書演算の探索操作を計算する P システム Π_{search} は，$O(mn)$ 種類のオブジェクト，$O(m)$ 個の膜，及び，$O(mn)$ 個の進化規則を用いて，$O(1)$ ステップで実行可能である． □

4.3 削除操作

削除操作では，指定した m ビットの 2 進数のデータが集合内，つまり，リスト膜内に存在すれば，その値を削除し，集合を出力する．また，存在しない場合はスキン膜に偽を出力する．

4.3.1 膜計算における削除操作の入出力

以下に，膜計算における削除操作の入出力を表すオブジェクトを示す．

入力 アドレス D に格納された m ビットの 2 進数を表すオブジェクトを入力とする．なお，これらのオブジェクトは，スキン膜に入力として与える．
$\langle A_D, B_0, V_{D,0}\rangle$，$\langle A_D, B_1, V_{D,1}\rangle$，$\cdots$，
$\langle A_D, B_{m-1}, V_{D,m-1}\rangle$
また，辞書演算のデータは，スキン膜内部のリスト膜に格納されているものとする．

出力 指定した m ビットの 2 進数のデータが削除された集合，

$$\begin{gathered}\langle A_0, B_0, V_{0,0}\rangle,\ \langle A_0, B_1, V_{0,1}\rangle,\ \cdots,\\ \langle A_0, B_{m-1}, V_{0,m-1}\rangle\\ \langle A_1, B_0, V_{1,0}\rangle,\ \langle A_1, B_1, V_{1,1}\rangle,\ \cdots,\\ \langle A_1, B_{m-1}, V_{1,m-1}\rangle\\ \vdots\\ \langle A_i, B_0, V_{i,0}\rangle,\ \langle A_i, B_1, V_{i,1}\rangle,\ \cdots,\\ \langle A_i, B_{m-1}, V_{i,m-1}\rangle\end{gathered}$$

もしくは偽を表すオブジェクト $\langle A_{Dout}, B_0, 0\rangle$ を出力する．

4.3.2 P システムの概要

本節で示す削除操作を行う P システムのアルゴリズムは，指定した m ビットの 2 進数のデータを集合から削除する操作である．

まず，指定した m ビットの 2 進数のデータをアドレス D に入れる．

$$\langle A_D, B_0, V_{D,0}\rangle, \langle A_D, B_1, V_{D,1}\rangle, \cdots, \langle A_D, B_{m-1}, V_{D,m-1}\rangle$$

このオブジェクトがスキン膜に入力として与えられる．

削除操作を実行するアルゴリズムは，大きくわけて以下の 2 つのステップで構成される．まず，1 つ目のステップ (Step 1) では，アドレス D に格納されたデータと同じデータがリスト膜内の集合内に存在するか否か調べる．次に，2 つ目のステップ (Step 2) では，同じデータがリスト膜内に存在する場合，そのデータを削除する．同じデータがリスト膜内に存在しない場合，何もしない．

以下に削除操作を行う P システムのアルゴリズムの概要を示す．

Step 1 以下のサブステップを実行することにより，指定したデータが存在するか調べる．

(1-1) アドレス D に格納された指定した m ビットの 2 進数のデータを表すオブジェクトを 1 ビット毎に，各ビット膜へ移動する．
リスト膜内の集合と同じデータがあるか比較するオブジェクトを作成するために，各ビット膜，ここでは，D_0，D_1，$\cdots$，D_{m-1} 膜に，それぞれ，アドレス D の 0，1，$\cdots$，$m-1$ ビットのデータを移動する．

(1-2) 各ビット膜，D_0，D_1，$\cdots$，D_{m-1} 膜において，それぞれ，指定したデータの 0，1，$\cdots$，$m-1$ ビットのデータを集合内に存在するアドレスの個数分，CP(Copy) オブジェクトに複製する．
指定したデータと同じデータの有無を調べるためには，集合のデータと指定したデータを 1 アドレス，1 ビット毎のオブジェクトと比較する必要がある．ここでは，各ビット膜において，それぞれ，指定したデータを集合内に存在するアドレスの個数分，CP オブジェクトに複製する．

(1-3) 指定したデータを複製したオブジェクトをリスト膜へ移動する．
(1-2) で複製したオブジェクトをリスト膜内の集合データと比較するために移動する．

(1-4) リスト膜において，1 アドレス，1 ビット毎に集合内のデータと指定したデータが同じか否かを調べる．
同じデータの場合，ジャッジ膜に $DEL(i)$(Delete) オブジェクトを作成する．ここで，i には集合内のデータのアドレス番号が入る．この際，ジャッジ膜に $DC(0)$(Delete Count) オブジェクトを 1 つだけ作成する．この $DC(0)$ オブジェクトは重複するデータが存在しない場合，$DC(2)$ まで進化し，重複するデータが存在しないことを意味するオブジェクト $\langle A_{Dout}, B_0, 0\rangle$ をリスト膜へ作成する．また，指定したデータと集合のデータが異なる場合は，何もしない．

(1-5) ジャッジ膜において，$DEL(i)$ オブジェクトが m ビット分存在するか調べる．
m ビット分存在する場合，アドレス i に指定したデータと同じデータが存在する．ゆえに，アドレス i に格納されているデータを削除するために，$DEL(i)$ オブジェクトをリスト膜へ移す．この際，LTm(Last Address Minus) オブジェクトを作成する．この LTm オブジェクトはジャッジ膜内の $LTA(i)$ オブジェクトの保持するアドレス i を 1 つ減らすためのオブジェクトである．また同時に，ジャッジ膜を $+$ に帯電させる．これは，必要のないオブジェクトを削除するためにジャッジ膜の状態を変化させるためである．m ビット分存在しない場合は，指定したデータと同じデータは存在しないので，$DC(0)$ オブジェクトが $DC(2)$ まで進化して，同じデータが存在しないことを意味するオブジェクト $\langle A_{Dout}, B_0, 0\rangle$ をリスト膜に作成する．

Step 2 以下のサブステップを実行することにより，指定したデータを削除して空白になったアドレスを埋める．

(2-1) 削除して空白になったアドレス i を保存する．
削除したいデータと同じデータが集合に存在

する場合，リスト膜に$DEL(i)$オブジェクトがm個存在する．ゆえに，このデータを削除し，空白となるアドレス番号iを保持する$EXA(i)$ (Exchange Address) オブジェクトを作成する．

(2-2) 最終アドレスのオブジェクトを，空白になったアドレスに入れる．
$EXA(i)$オブジェクトに格納された空白になったアドレスに，最終アドレスのオブジェクトを挿入する．もし，最終アドレスのオブジェクトが削除された場合は，$EXA(i)$オブジェクトを削除する．

以下に辞書演算の削除操作を実行するPシステムΠ_{delete}を示す．

$$\Pi_{delete} = (O, \mu, \omega_{skin}, \omega_{list}, \omega_{judge}, \omega_{D_j}, R_{skin}, R_{list}, R_{judge}, R_{D_j}, i_{in}, i_{out})$$

ここで，Π_{delete}を構成するそれぞれの集合は以下のようになる．

- $O = \{\langle A_D, B_j, V_{D,j}\rangle, \langle A_E, B_j, V_{E,j}\rangle, \langle A_i, B_j, V_{i,j}\rangle, \langle A_{Dout}, B_0, V_{Dout,0}\rangle, CP, CPm, LTA(i), LTm, DC(k), DEL(i), EXA(i) \mid 0 \le i \le n-1, 0 \le j \le m-1, 0 \le k \le 2, V_{D,j}, V_{E,j}, V_{i,j}, V_{Dout,0} \in \{0,1\}\}$
- $\mu = [_{skin}\ [_{list}\ [_{judge}\ ]_{judge}\ [_{D_0}\ ]_{D_0}\ [_{D_1}\ ]_{D_1} \cdots [_{D_{m-1}}\ ]_{D_{m-1}}\ ]_{list}\ ]_{skin}$
- $\omega_{skin} = \phi$
- $\omega_{D_j} = \{CP^i \mid 0 \le i \le n-1\}$
- $\omega_{list} = \{LTA(-1)^m\}$
- $\omega_{judge} = \{LTA(-1)\}$
- $R_{skin} = \{[_{skin}\langle A_D, B_j, 0\rangle]_{skin} \to [_{list}\langle A_D, B_j, 0\rangle]_{list}, [_{skin}\langle A_D, B_j, 1\rangle]_{skin} \to [_{list}\langle A_D, B_j, 1\rangle]_{list}\}$
- $R_{list} = \{$
  $[_{list}\langle A_D, B_0, 0\rangle LTA(-1)]_{list} \to [_{list}LTA(-1)]_{list}[_{skin}\langle A_{Dout}, B_0, 0\rangle]_{skin}$,
  $[_{list}\langle A_D, B_0, 1\rangle LTA(-1)]_{list} \to [_{list}LTA(-1)]_{list}[_{skin}\langle A_{Dout}, B_0, 0\rangle]_{skin}$,
  $[_{list}\langle A_D, B_j, 0\rangle LTA(-1)]_{list} \to [_{list}LTA(-1)]_{list}$,
  $[_{list}\langle A_D, B_j, 1\rangle LTA(-1)]_{list} \to [_{list}LTA(-1)]_{list}$,
  $[_{list}\langle A_D, B_j, 0\rangle LTA(i)]_{list} \to [_{list}LTA(i)]_{list}[_{D_j}\langle A_D, B_j, 0\rangle]_{D_j}$,
  $[_{list}\langle A_D, B_j, 1\rangle LTA(i)]_{list} \to [_{list}LTA(i)]_{list}[_{D_j}\langle A_D, B_j, 1\rangle]_{D_j}$,
  $[_{list}\langle B_{D_0}, 0\rangle\langle A_0, B_0, 0\rangle]_{list} \to [_{list}\langle A_0, B_0, 0\rangle]_{list}[_{judge}DEL(0)DC(0)]_{judge}$,
  $[_{list}\langle B_{D_0}, 1\rangle\langle A_0, B_0, 1\rangle]_{list} \to [_{list}\langle A_0, B_0, 1\rangle]_{list}[_{judge}DEL(0)DC(0)]_{judge}$,
  $[_{list}\langle B_{D_j}, 0\rangle\langle A_i, B_j, 0\rangle]_{list} \to [_{list}\langle A_i, B_j, 0\rangle]_{list}[_{judge}DEL(i)]_{judge}$,
  $[_{list}\langle B_{D_j}, 1\rangle\langle A_i, B_j, 1\rangle]_{list} \to [_{list}\langle A_i, B_j, 1\rangle]_{list}[_{judge}DEL(i)]_{judge}$,
  $[_{list}\langle B_{D_0}, 0\rangle\langle A_0, B_0, 1\rangle]_{list} \to [_{list}\langle A_0, B_0, 1\rangle]_{list}[_{judge}DC(0)]_{judge}$,
  $[_{list}\langle B_{D_0}, 1\rangle\langle A_0, B_0, 0\rangle]_{list} \to [_{list}\langle A_0, B_0, 0\rangle]_{list}[_{judge}DC(0)]_{judge}$,
  $[_{list}\langle B_{D_j}, 0\rangle\langle A_i, B_j, 1\rangle]_{list} \to [_{list}\langle A_i, B_j, 1\rangle]_{list}$,
  $[_{list}\langle B_{D_j}, 1\rangle\langle A_i, B_j, 0\rangle]_{list} \to [_{list}\langle A_i, B_j, 0\rangle]_{list}$,
  $[_{list}DEL(i)\langle A_i, B_j, 0\rangle]^+_{list} \to [_{list}LTm, EXA(i)]_{list}[_{D_j}CPm]_{D_j}$,
  $[_{list}DEL(i)\langle A_i, B_j, 1\rangle]^+_{list} \to [_{list}LTm, EXA(i)]_{list}[_{D_j}CPm]_{D_j}$,
  $[_{list}LTA(i)LTm]_{list} \to [_{list}LTA(i-1)]_{list}$,
  $[_{list}LTA(i)\langle A_{i+1}, B_j, 0\rangle]_{list} \to [_{list}\langle A_E, B_j, 0\rangle]_{list}$,
  $[_{list}LTA(i)\langle A_{i+1}, B_j, 1\rangle]_{list} \to [_{list}\langle A_E, B_j, 1\rangle]_{list}$,
  $[_{list}LTA(i-1)EXA(i)]_{list} \to [_{list}LTA(i-1)]_{list}$,
  $[_{list}EXA(i)\langle A_E, B_j, 0\rangle]_{list} \to [_{list}\langle A_i, B_j, 0\rangle]_{list}$,
  $[_{list}EXA(i)\langle A_E, B_j, 1\rangle]_{list} \to [_{list}\langle A_i, B_j, 1\rangle]_{list}\}$
- $R_{D_j} = \{[_{D_j}\langle A_D, B_j, 0\rangle]_{D_j} \to [_{D_j}]^-_{D_j}$,
  $[_{D_j}\langle A_D, B_j, 1\rangle]_{D_j} \to [_{D_j}]^+_{D_j}$,
  $[_{D_j}CP]^-_{D_j} \to [_{D_j}\langle B_{D_j}, 0\rangle]_{D_j}$,
  $[_{D_j}CP]^+_{D_j} \to [_{D_j}\langle B_{D_j}, 1\rangle]_{D_j}$,
  $[_{D_j}\langle B_{D_j}, 0\rangle]_{D_j} \to [_{D_j}CP]_{D_j}[_{list}\langle B_{D_j}, 0\rangle]_{list}$,
  $[_{D_j}\langle B_{D_j}, 1\rangle]_{D_j} \to [_{D_j}CP]_{D_j}[_{list}\langle B_{D_j}, 1\rangle]_{list}$,
  $[_{D_j}CPmCP]_{D_j} \to [_{D_j}]_{D_j}\}$
- $R_{judge} = \{[_{judge}DEL(i)^m]_{judge} \to [_{judge}LTm]^+_{judge}[_{list}DEL(i)^m]_{list}$,
  $[_{judge}DC(0)]_{judge} \to [_{judge}DC(1)]_{judge}$,
  $[_{judge}DC(1)]_{judge} \to [_{judge}DC(2)]_{judge}$,
  $[_{judge}DC(2)]_{judge} \to [_{list}\langle A_{Dout}, B_0, 0\rangle]_{list}[_{judge}]^+_{judge}$,
  $[_{judge}DC(1)]^+_{judge} \to [_{judge}]_{judge}$,
  $[_{judge}DEL(i)]^+_{judge} \to [_{judge}]_{judge}$,
  $[_{judge}LTA(i)LTm]_{judge} \to [_{judge}LTA(i-1)]_{judge}\}$
- $i_{in} = skin$
- $i_{out} = list$

この計算モデルでは入力膜はスキン膜である．また，出力膜はリスト膜である．

ここで，削除操作を行う P システムにおける各オブジェクトの役割を説明する．

$DEL(i)$(Delete): 値が同じオブジェクトがある場合にそのアドレス i を保持するオブジェクト

$DC(0)$(Delete Count): 指定したデータの有無調べる際の，カウンタを意味するオブジェクト

LTm(Last Address Minus): 指定したデータが削除された際に，$LTA(i)$ オブジェクトを 1 つ減らすためのオブジェクト

$EXA(i)$(Exchange Address): 指定したデータが削除されたときに空白になったアドレス i を保存するオブジェクト

CPm(Copy Minus): CP オブジェクトを 1 つ減らすためのオブジェクト

4.3.3 P システムの動作

膜計算モデルを用いて削除操作の実行例を示すことにより，P システムの動作を説明する．ここでは，2 進数 11 のデータを，既に 2 進数 00，11 の入った集合から削除する例を示す．$\langle A_D, B_1, 1\rangle$，$\langle A_D, B_0, 1\rangle$ がスキン膜に入力として与えられるものとする．

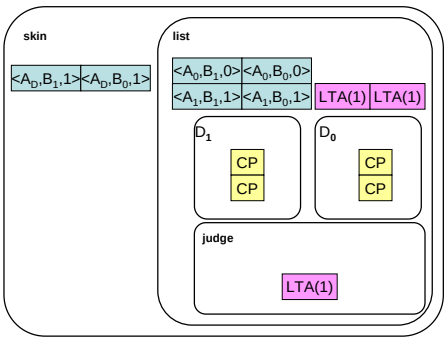


図 11: (1-1) 開始時における膜構造の状態

この膜計算モデルの初期状態は，図 11 である．ここで，$LTA(i)$ はリスト膜に最終のアドレスのビット個数分，ジャッジ膜に 1 つだけ存在する．また，CP オブジェクトはビット膜，ここでは，D_0 膜，D_1 膜にアドレスの個数分入っている．

Step 1 以下のサブステップを実行することにより，指定したデータが存在するか調べる．

(1-1) アドレス D に格納された指定した 2 ビットの 2 進数のデータを表すオブジェクトを 1 ビット毎に，各ビット膜へ移動する．この例では，前述の状態に対して，進化規則
$[_{skin}\langle A_D, B_1, 1\rangle]_{skin} \rightarrow [_{list}\langle A_D, B_1, 1\rangle]_{list}$
$[_{skin}\langle A_D, B_0, 1\rangle]_{skin} \rightarrow [_{list}\langle A_D, B_0, 1\rangle]_{list}$
が適用され，スキン膜からリスト膜へ移動する．

次に，この例では，前述の状態に対して，進化規則
$[_{list}\langle A_D, B_1, 1\rangle LTA(1)]_{list}$
$\rightarrow [_{list} LTA(1)]_{list}[_{D_1}\langle A_D, B_1, 1\rangle]_{D_1}$
$[_{list}\langle A_D, B_0, 1\rangle LTA(1)]_{list}$
$\rightarrow [_{list} LTA(1)]_{list}[_{D_0}\langle A_D, B_0, 1\rangle]_{D_0}$
が適用され，リスト膜から各ビット膜，ここでは，D_0，D_1 膜へ削除したい値を移動する．これにより，図 12 の状態に遷移する．

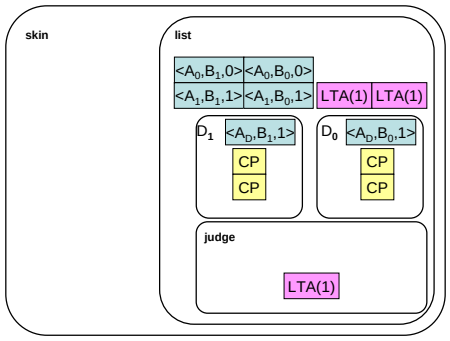


図 12: (1-2) 開始時における膜構造の状態

(1-2) 各ビット膜，D_0，D_1 膜において，それぞれ，指定したデータの 0，1 ビット目のデータを集合内に存在するアドレスの個数分，CP オブジェクトに複製する．
次に，削除したいデータの有無を調べるためにアドレス D のデータを各ビット膜おいて CP オブジェクトに複製する．この際 1 回の操作で複製するために，各ビット膜を値が 0 の場合は膜を $-$ に，値が 1 の場合は膜を $+$ に帯電させる．この例では，前述の状態に対して，進化規則

$$[_{D_1}\langle A_D, B_1, 1\rangle]_{D_1} \rightarrow [_{D_1}]^+_{D_1}$$

$$[_{D_0}\langle A_D, B_0, 1\rangle]_{D_0} \rightarrow [_{D_0}]^+_{D_0}$$

$$[_{D_1} CP]^+_{D_1} \rightarrow [_{D_1}\langle B_{D_1}, 1\rangle]_{D_1}$$

$$[_{D_0} CP]^+_{D_0} \rightarrow [_{D_0}\langle B_{D_0}, 1\rangle]_{D_0}$$

が適用され，図 13 の状態へ遷移する．

(1-3) 指定したデータを複製したオブジェクトをリスト膜へ移動する．
次に，削除したいデータの有無を調べるためにコピーしたオブジェクトをリスト膜へ移動する．この際，連続で削除操作ができるように，アドレス D のビット毎の値を複製したオブジェクトを CP オブジェクトを元に戻す．この例では，前述の状態に対して，進化規則

$$[_{D_1}\langle B_1, 1\rangle]_{D_1} \rightarrow [_{D_1} CP]_{D_1}[_{list}\langle B_{D_1}, 1\rangle]_{list}$$

$$[_{D_0}\langle B_{D_0}, 1\rangle]_{D_0} \rightarrow [_{D_0} CP]_{D_0}[_{list}\langle B_{D_0}, 1\rangle]_{list}$$

が適用され，図 14 の状態へ遷移する．

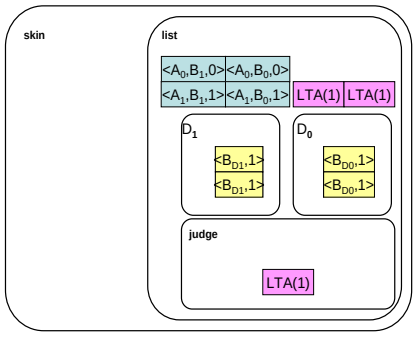


図 13: (1-3) 開始時における膜構造の状態

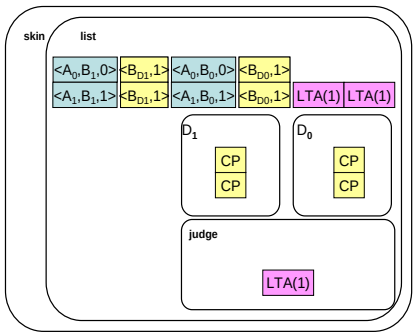


図 14: (1-4) 開始時における膜構造の状態

(1-4) リスト膜において，1 アドレス，1 ビット毎に集合内のデータと指定したデータが同じか否かを調べる．

各ビットごとで比較して，削除したい値と同じ値があれば，ジャッジ膜に $DEL(i)$ オブジェクトを作成する．このとき，i は削除された値のアドレス番号である．また，アドレス 0 のビット 0 のオブジェクトに対してだけ $DC(0)$ オブジェクトを作成する．違う値であれば，何も作成しない．

この例では前述の状態に対して，進化規則

$$[_{list}\langle B_{D_1},1\rangle\langle A_0,B_1,0\rangle]_{list} \rightarrow [_{list}\langle A_0,B_1,0\rangle]_{list}$$

$$[_{list}\langle B_{D_0},1\rangle\langle A_0,B_0,0\rangle]_{list} \rightarrow [_{list}\langle A_0,B_0,0\rangle]_{list}[_{judge}DC(0)]_{judge}$$

$$[_{list}\langle B_{D_1},1\rangle\langle A_1,B_1,1\rangle]_{list} \rightarrow [_{list}\langle A_1,B_1,1\rangle]_{list}[_{judge}DEL(1)]_{judge}$$

$$[_{list}\langle B_{D_0},1\rangle\langle A_1,B_0,1\rangle]_{list} \rightarrow [_{list}\langle A_1,B_0,1\rangle]_{list}[_{judge}DEL(1)]_{judge}$$

が適用され，図 15 の状態へ遷移する．

(1-5) $DEL(i)$ オブジェクトが 2 ビット分あるのか調べる．

この例では，ジャッジ膜において，各 $DEL(i)$ オブジェクトが 2 個つまり 2 ビット分存在するのでリスト膜内に削除したいデータと同じデータが存在する．ゆえにジャッジ膜に LTm オブジェクトを作成すると同時に，ジャッジ膜を $+$ に帯電させる．これは，ジャッジ膜内に残ったオブジェクトを削除するためである．またリスト膜内のデータを削除するために $DEL(i)$ をリスト膜へ移動させる．

この例では，前述の状態に対して，進化規則

$$[_{judge}DEL(1)^2]_{judge} \rightarrow [_{judge}LTm]^+_{judge}[_{list}DEL(1)^2]_{list}$$

$$[_{judge}DC(0)]_{judge} \rightarrow [_{judge}DC(1)]_{judge}$$

$$[_{judge}DC(1)]^+_{judge} \rightarrow [_{judge}]_{judge}$$

が適用され，図 16 の状態に遷移する．

また，もし削除したいデータがなかった場合は $DC(1)$ オブジェクトが $DC(2)$ まで進化して，スキン膜に偽を表すオブジェクト $\langle A_{out},B_0,0\rangle$ を出力する．この際ジャッジ膜を $+$ に帯電させてジャッジ膜内に残った必要ないオブジェクトを削除する．この進化規則

$$[_{judge}DC(0)]_{judge} \rightarrow [_{judge}DC(1)]_{judge}$$

$$[_{judge}DC(1)]_{judge} \rightarrow [_{judge}DC(2)]_{judge}$$

$$[_{judge}DC(2)]_{judge} \rightarrow [_{list}\langle A_{Dout},B_0,0\rangle]_{list}[_{judge}]^+_{judge}$$

$$[_{judge}DEL(i)]^+_{judge} \rightarrow [_{judge}]_{judge}$$

$$(0 \le i \le n-1)$$

が適用される．

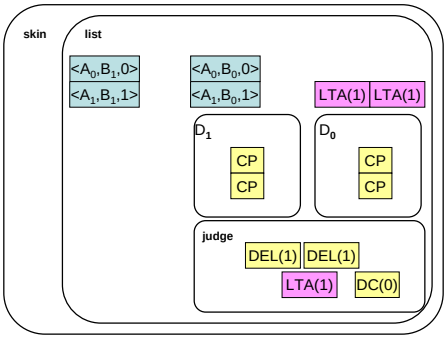


図 15: (1-5) 開始時における膜構造の状態

Step 2 指定したデータを削除して空白になったアドレスを埋める．

(2-1) 削除して空白になったアドレス i を保存する．

この例では，$DEL(1)$ が 2 ビット分あるので削除したいデータと同じデータがアドレス 1 に入っており，このデータを削除する．このとき同時に，リスト膜に $LTA(1)$ が保持しているアドレス番号を 1 つ減らすオブジェクト LTm，削除されて空白となるアドレス番号を保持するオブジェクト $EXA(i)$ を作成する．i には空白になったアドレス番号が入る．また，各ビット膜の CP オブジェクトを 1 つずつ削除する

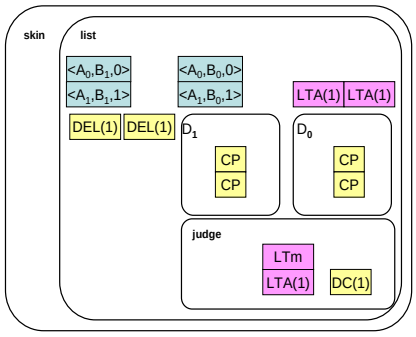


図 16: (2-1) 開始時における膜構造の状態

ために，各ビット膜に CPm オブジェクトを作成する．このとき同時に，ジャッジ膜において，LTm オブジェクトが $LTA(i)$ オブジェクトの値を 1 だけ減算する．

この例では，前述の状態に対して，進化規則

$[_{list} DEL(1)\langle A_1, B_1, 1\rangle]_{list}$
$\rightarrow [_{list} LTmEXA(1)]_{list}[_{D_1} CPm]_{D_1}$
$[_{list} DEL(1)\langle A_1, B_0, 1\rangle]_{list}$
$\rightarrow [_{list} LTmEXA(1)]_{list}[_{D_0} CPm]_{D_0}$
$[_{judge} LTA(1)LTm]_{judge} \rightarrow [_{judge} LTA(0)]_{judge}$

が適用され，図 17 の状態へ遷移する．

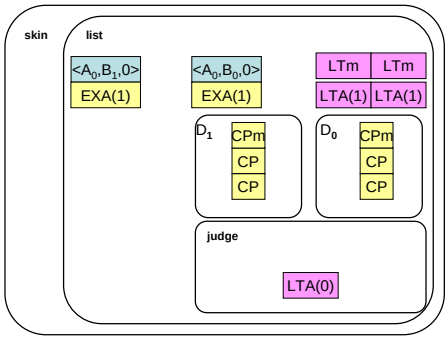


図 17: (2-1) における膜構造の状態

次に，リスト膜において LTm オブジェクトが $LTA(i)$ の値を 1 だけ減算し，各ビット膜において，CPm が CP オブジェクトを削除する．この例では，前述の状態に対して，進化規則

$$[_{list} LTA(1)LTm]_{list} \rightarrow [_{list} LTA(0)]_{list}$$

$$[_{D_0} CPmCP]_{D_0} \rightarrow [_{D_0}]_{D_0}$$

$$[_{D_1} CPmCP]_{D_1} \rightarrow [_{D_1}]_{D_1}$$

が適用され，図 18 の状態へ遷移する．

(2-2) 最終のアドレスのオブジェクトを空白になったアドレスに入れる．

次にリスト膜において，最後のアドレスのデータが削除されたので，アドレス $E(Exchange)$

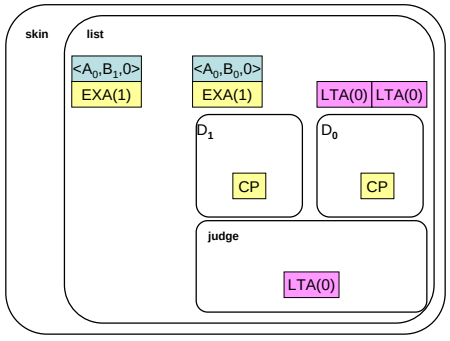


図 18: (2-2) における膜構造の状態

にはデータが入っていない．ゆえに，$EXA(i)$ オブジェクトだけが残るのでこれを削除する．

この例では，前述の状態に対して，進化規則

$$[_{list} LTA(0)EXA(1)]_{list} \rightarrow [_{list} LTA(0)]_{list}$$

が適用され，図 19 の状態へ遷移する．

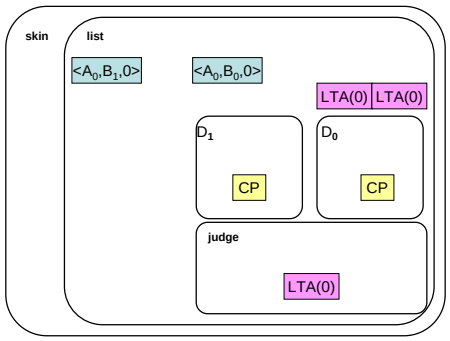


図 19: 削除操作後の状態

この例では，これ以上，適用できる進化規則がないので計算を終了する．

4.3.4 P システムの計算量

削除操作を実行する P システム Π_{delete} は，$O(mn)$ 回の進化規則の適用により実行が終了するので，以下の定理が得られる．

定理 3 n 個のアドレス，m ビットの 2 進数表記のオブジェクトを入力とし，辞書演算の削除操作を計算する P システム Π_{delete} は，$O(mn)$ 種類のオブジェクト，$O(m)$ 個の膜，及び，$O(mn)$ 個の進化規則を用いて，$O(1)$ ステップで実行可能である． □

4.4 辞書演算の改良

挿入，探索，削除を実現するそれぞれの P システムでは，異なる膜のラベルを用いているので，挿入，探索，

削除の操作をまとめて行うことはできない．そこで，膜の名前の変更といくつかの進化規則の修正を行い，これら 3 つの P システムを統合し 1 つの P システムとすることにより，3 つの辞書演算を実行する P システム Π_{dic} を作成することができる．(本稿では説明を割愛し，文献 [6] にて詳細を記述する．) なお，挿入，探索，削除のそれぞれの操作に必要な計算量は，各操作を実現する P システムの計算量と同じである．

4.5 メモリ機能

メモリ機能とは，入力に対応する解を記憶して，計算時間を短縮するものである．膜計算の辞書演算を用いることでメモリ機能は実行可能なことを示す．

膜計算におけるメモリ機能には挿入，探索操作がある．挿入操作の入力は，ある演算の入力とその入力の解の 2 つとし，出力は入力と解が挿入された集合とする．また，探索操作の入力は，ある演算の入力で，出力はその入力に対応する解である．入力を *input list* 膜，入力に対応する解を *solution list* 膜に記憶する．以下，それぞれ入力リスト膜，解リスト膜と呼ぶ．入力と解はアドレス番号で対応させる．

例えば，AND 演算，$0 \wedge 0 = 0$，$1 \wedge 1 = 1$ を記憶した状態は図 20 のようになる．

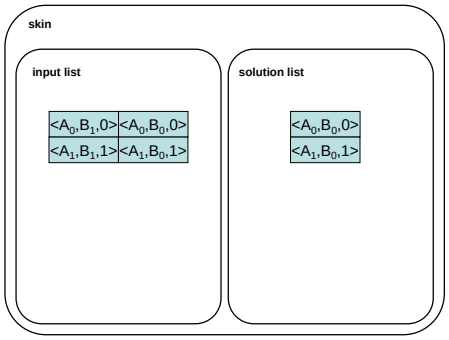


図 20: メモリ機能の記憶状態

4.5.1 メモリ機能 P システム

以下にメモリ機能の挿入操作，探索操作を実行する P システム Π_{memory} を示す．

$$\Pi_{memory} = (O, \mu, \omega_{skin}, \omega_{input_{list}}, \omega_{solution_{list}}\omega_{judge}, \omega_{I_j}, \omega_{S_j}, R_{skin}, R_{input_{list}}, R_{solution_{list}}, R_{judge}, R_{I_j}, R_{S_j}, i_{in}, i_{out})$$

ここで，Π_{memory} を構成するそれぞれの集合は以下ようになる．

- $O = \{\langle A_I, B_j, V_{I,j}\rangle, \langle A_{I'}, B_j, V_{I',j}\rangle, \langle A_{I''}, B_j, V_{I'',j}\rangle, \langle A_i, B_j, V_{i,j}\rangle, \langle A_i, B_{j'}, V_{i,j'}\rangle, \langle A_{Isol}, B_{j'}, V_{Isol,j'}\rangle, \langle A_{Isol'}, B_{j'}, V_{Isol',j'}\rangle, \langle A_S, B_j, V_{S,j}\rangle, \langle A_{Iout}, B_0, V_{Iout,0}\rangle, \langle A_{Sout}, B_0, V_{Sout,0}\rangle, CP, LTA(i), LTp, SC(k), IC(k), IMCH(i), SMCH(i) | 0 \le i \le n-1, 0 \le j \le m-1, 0 \le j' \le m'-1, 0 \le k \le 2, V_{I,j}, V_{I',j}, V_{I'',j}, V_{Isol,j'}, V_{Isol',j'}, V_{i,j}, V_{i,j'} V_{Iout,0}, V_{Sout,0} \in \{0,1\}\}$
- $\mu = [_{skin}\ [_{input_{list}}\ [_{judge}\ ]_{judge} [_{I_0}\ ]_{I_0}\ [_{I_1}\ ]_{I_1} \cdots [_{I_{m-1}}\ ]_{I_{m-1}} [_{S_0}\ ]_{S_0}\ [_{S_1}\ ]_{S_1} \cdots [_{S_{m-1}}\ ]_{S_{m-1}}]_{input_{list}} [_{solution_{list}}]_{solution_{list}}\ ]_{skin}$
- $\omega_{skin} = \phi$
- $\omega_{I_j} = \{CP^i\}$
- $\omega_{S_j} = \{CP^i\}$
- $\omega_{input_{list}} = \{LTA(-1)^m\}$
- $\omega_{solution_{list}} = \{LTA(-1)^{m'}\}$
- $\omega_{judge} = \{LTA(-1)\}$

  進化規則については，辞書演算の挿入操作を行う P システムの進化規則に対して以下の変更をおこなう．
  まず，R_{skin} に以下の進化規則を追加する．
  $[_{skin}\langle A_{Isol}, B_{j'}, 0\rangle]_{skin} \rightarrow [_{solution_{list}}\langle A_{Isol'}, B_{j'}, 0\rangle]_{solution_{list}}$
  $[_{skin}\langle A_{Isol}, B_{j'}, 1\rangle]_{skin} \rightarrow [_{solution_{list}}\langle A_{Isol'}, B_{j'}, 1\rangle]_{solution_{list}}$
  $[_{skin}\langle A_{Iout}, B_0, 0\rangle]_{skin} \rightarrow [_{solution_{list}}\langle A_{Iout}, B_0, 0\rangle]_{solution_{list}}$
  $[_{skin}\langle A_{Iout}, B_0, 1\rangle]_{skin} \rightarrow [_{solution_{list}}\langle A_{Iout}, B_0, 1\rangle]_{solution_{list}}$
  次に，$R_{input_{list}}$ は R_{list} の以下の進化規則を変更する．他の R_{list} の進化規則は $R_{input_{list}}$ として，そのまま使用する．
  (変更前)
  $[_{list}\langle A_{Iout}, B_0, 0\rangle]_{list} \rightarrow [_{list}]^-_{list}$
  $[_{list}\langle A_{Iout}, B_0, 1\rangle]_{list} \rightarrow [_{list}]^+_{list}$
  (変更後)
  $[_{input_{list}}\langle A_{Iout}, B_0, 0\rangle]_{input_{list}} \rightarrow [_{skin}\langle A_{Iout}, B_0, 0\rangle]_{skin}[_{input_{list}}]^-_{input_{list}}$
  $[_{input_{list}}\langle A_{Iout}, B_0, 1\rangle]_{input_{list}} \rightarrow [_{skin}\langle A_{Iout}, B_0, 1\rangle]_{skin}[_{input_{list}}]^+_{input_{list}}$
  $R_{solution_{list}}$ について，以下のように定義する．
- $R_{solution_{list}} = \{$
  $[_{solution_{list}}\langle A_{Iout}, B_0, 0\rangle]_{solution_{list}} \rightarrow [_{solution_{list}}]^-_{solution_{list}}$,
  $[_{solution_{list}}\langle A_{Iout}, B_0, 1\rangle]_{solution_{list}} \rightarrow [_{solution_{list}}]^+_{solution_{list}}$,
  $[_{solution_{list}}\langle A_{Isol'}, B_{j'}, 0\rangle]^+_{solution_{list}} \rightarrow [_{solution_{list}}]_{solution_{list}}$,
  $[_{solution_{list}}\langle A_{Isol'}, B_{j'}, 1\rangle]^+_{solution_{list}} \rightarrow [_{solution_{list}}]_{solution_{list}}$,
  $[_{solution_{list}}\langle A_{Isol'}, B_0, 0\rangle]^-_{solution_{list}}$

$\rightarrow [_{solution_{list}}\langle A_{Isol}, B_0, 0\rangle]_{solution_{list}}$
$[_{solution_{list}}\langle A_{Isol'}, B_0, 1\rangle]^{-}_{solution_{list}}$
$\rightarrow [_{solution_{list}}\langle A_{Isol}, B_0, 1\rangle]_{solution_{list}}$
$[_{solution_{list}} LTA(-1)\langle A_{Isol}, B_0, 0\rangle]_{solution_{list}}$
$\rightarrow [_{solution_{list}} LTA(0)\langle A_0, B_0, 0\rangle]_{solution_{list}}$
$[_{solution_{list}} LTA(-1)\langle A_{Isol}, B_0, 1\rangle]_{solution_{list}}$
$\rightarrow [_{solution_{list}} LTA(0)\langle A_0, B_0, 1\rangle]_{solution_{list}}$
$[_{solution_{list}} LTA(-1)\langle A_{Isol}, B_{j'}, 0\rangle]_{solution_{list}}$
$\rightarrow [_{solution_{list}} LTA(0)\langle A_0, B_{j'}, 0\rangle]_{solution_{list}}$
$[_{solution_{list}} LTA(-1)\langle A_{Isol}, B_{j'}, 1\rangle]_{solution_{list}}$
$\rightarrow [_{solution_{list}} LTA(0)\langle A_0, B_{j'}, 1\rangle]_{solution_{list}}$
$[_{solution_{list}} LTA(i)\langle A_{Isol}, B_{j'}, 0\rangle]_{solution_{list}} \rightarrow$
$[_{solution_{list}} LTA(i+1)\langle A_{i+1}, B_{j'}, 0\rangle]_{solution_{list}}$,
$[_{solution_{list}} LTA(i)\langle A_{Isol}, B_{j'}, 1\rangle]_{solution_{list}} \rightarrow$
$[_{solution_{list}} LTA(i+1)\langle A_{i+1}, B_{j'}, 1\rangle]_{solution_{list}}\}$

4.5.2 メモリ機能の挿入操作

辞書演算の挿入操作を用いることにより，入力に対応する解を記憶する．まず，指定した入力を入力リスト膜，解リスト膜に保存する．ここで，入力リスト膜は辞書演算におけるリスト膜と同じである．

メモリ機能の挿入操作の入出力

入力 入力をアドレス I に格納された m ビットの 2 進数のデータ，解をアドレス $Isol$ に格納された m' ビットの 2 進数のデータを表すオブジェクトとして入力とする．つまり，アドレス I のオブジェクトに対応する解がアドレス $Isol$ のオブジェクトとなる．なお，これらのオブジェクトは，スキン膜に入力として与える．また，すでに入力済みの入力と解は，スキン膜内部の入力リスト膜，解リスト膜に格納されているものとする．

$\langle A_I, B_0, V_{I,0}\rangle$，$\langle A_I, B_1, V_{I,1}\rangle$，$\cdots$，
$\langle A_I, B_{m-1}, V_{I,m-1}\rangle$

$\langle A_{Isol}, B_0, V_{Isol,0}\rangle$，$\langle A_{Isol}, B_1, V_{Isol,1}\rangle$，$\cdots$，
$\langle A_{Isol}, B_{m'-1}, V_{Isol,m'-1}\rangle$

出力 入力の m ビットの 2 進数のデータと解の m' ビットの 2 進数のデータが挿入された集合を出力とする．

$\langle A_0, B_0, V_{0,0}\rangle$，$\langle A_0, B_1, V_{0,1}\rangle$，$\cdots$，
$\langle A_0, B_{m-1}, V_{0,m-1}\rangle$
$\langle A_1, B_0, V_{1,0}\rangle$，$\langle A_1, B_1, V_{1,1}\rangle$，$\cdots$，
$\langle A_1, B_{m-1}, V_{1,m-1}\rangle$
$\vdots$
$\langle A_i, B_0, V_{i,0}\rangle$，$\langle A_i, B_1, V_{i,1}\rangle$，$\cdots$，
$\langle A_i, B_{m-1}, V_{i,m-1}\rangle$

$\langle A_0, B_0, V_{0,0}\rangle$，$\langle A_0, B_1, V_{0,1}\rangle$，$\cdots$，
$\langle A_0, B_{m'-1}, V_{0,m'-1}\rangle$
$\langle A_1, B_0, V_{1,0}\rangle$，$\langle A_1, B_1, V_{1,1}\rangle$，$\cdots$，
$\langle A_1, B_{m'-1}, V_{1,m'-1}\rangle$
$\vdots$
$\langle A_i, B_0, V_{i,0}\rangle$，$\langle A_i, B_1, V_{i,1}\rangle$，$\cdots$，
$\langle A_i, B_{m'-1}, V_{i,m'-1}\rangle$

4.5.3 挿入操作を行う P システムの概要

メモリ機能を持つ挿入操作を行う P システムは，辞書演算の挿入操作の P システムを改良することで実行可能である．

まず，入力の m ビットの 2 進数のデータをアドレス I に，解の m' ビットの 2 進数のデータをアドレス $Isol$(Insert solution) に入れる．
$\langle A_I, B_0, V_{I,0}\rangle$，$\langle A_I, B_1, V_{I,1}\rangle$，$\cdots$，$\langle A_I, B_{m-1}, V_{I,m-1}\rangle$

$\langle A_{Isol}, B_0, V_{Isol,0}\rangle$，$\langle A_{Isol}, B_1, V_{Isol,1}\rangle$，$\cdots$，
$\langle A_{Isol}, B_{m'-1}, V_{Isol,m'-1}\rangle$
このらのオブジェクトがスキン膜に入力として与えられる．入力に対応するアドレス I のオブジェクトは入力リスト膜に，解に対応するアドレス $Isol$ のオブジェクトは解リスト膜に移動する．入力リスト膜においてアドレス I のオブジェクトが入力リスト膜のデータと重複していないか調べる．このアルゴリズムは辞書演算の挿入操作を行う P システムのアルゴリズムと同じなので説明は省略する．重複データが存在する場合は，指定した入力に対応するオブオブジェクト，解に対応するオブジェクトを挿入せず，重複データが存在しない場合のみ，指定した入力に対応するオブジェクト，解に対応するオブジェクトを挿入する．

4.5.4 挿入操作を行う P システムの動作

膜計算においてメモリ機能の挿入操作を実行する実行例を示すことにより，アルゴリズムの動作を説明する．ここでは，論理積の AND 演算に対応する入力と解の対を例として説明する．まず，入力に対応するデータを 2 進数 10，解に対応するデータを 2 進数 0 を入力として与える．よって，
$\langle A_I, B_1, 1\rangle$，$\langle A_I, B_0, 0\rangle$，$\langle A_{Isol}, B_0, 0\rangle$
がスキン膜に入力として与えられるものとする．

また，既知の解として，入力リスト膜には 2 進数 00，11 が，解リスト膜にはそれらに対応する解 0，1 が集合として入っているものとする．

初めに，スキン膜から入力リスト膜，解リスト膜にアドレス I，$Isol$ のオブジェクトをそれぞれ移動する．入力リスト膜において，アドレス I のオブジェクトが重複するか調べる．重複する場合，$\langle A_{Iout}, B_0, 1\rangle$ が，そうでない場合，$\langle A_{Iout}, B_0, 0\rangle$ がスキン膜へ作成される．ここまでのアルゴリズム動作は辞書演算の挿入操作のアルゴリズム動作と同じなので省略する．ここではスキン膜に，重複するデータが存在しないことを示すオブジェクト $\langle A_{Iout}, B_0, 0\rangle$ が作成される．この状態を図 21 に示す．

この状態に適用可能な進化規則は，
$[_{skin}\langle A_{Iout}, B_0, 0\rangle]_{skin}$

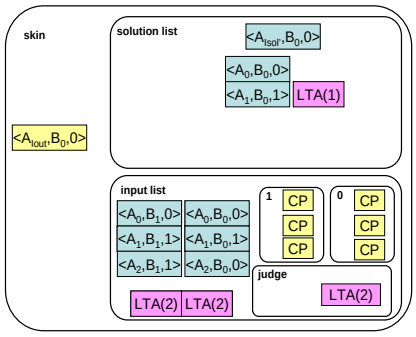


図 21: メモリ機能の挿入操作の状態 1

$\rightarrow [_{solution_{list}}\langle A_{Iout}, B_0, 0\rangle]_{solution_{list}}$

となり，図 22 の状態となる．

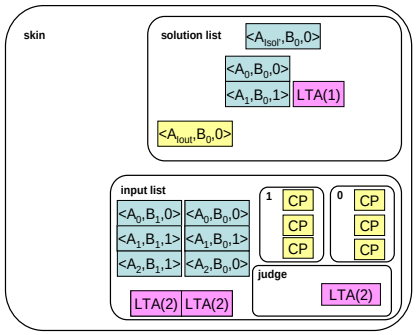


図 22: メモリ機能の挿入操作の状態 2

解リスト膜に保存しておいた，アドレス $Isol'$ のオブジェクトを解リスト膜の集合に挿入するために，進化規則，

$[_{solution_{list}}\langle A_{Iout}, B_0, 0\rangle]_{solution_{list}}$
$\rightarrow [_{solution_{list}}]^{-}_{solution_{list}}$
$[_{solution_{list}}\langle A_{Isol'}, B_0, 0\rangle]^{-}_{solution_{list}}$
$\rightarrow [_{solution_{list}}\langle A_{Isol}, B_0, 0\rangle]_{solution_{list}}$

が適用され，図 23 の状態となる．

最後に，アドレス $Isol$ のデータを一番末のアドレスに挿入する．このとき，以下の進化規則，

$[_{solution_{list}} LTA(1)\langle A_{Isol}, B_0, 0\rangle]_{solution_{list}} \rightarrow$
$[_{solution_{list}} LTA(2)\langle A_2, B_0, 0\rangle]_{solution_{list}}$

が適用され，図 24 となる．

ここで，入力リスト膜にはアドレス 0，1，2 にそれぞれ 2 進数 00，11，10 が記憶されている．また，解リスト膜にはアドレス 0，1，2 にそれぞれ 2 進数 0，1，0 が記憶されている．論理積演算の入力に対応する解が記憶されているのがわかる．以上で，適用できる進化規則はないので計算終了となる．

4.5.5 メモリ機能の探索操作

辞書演算の探索操作を用いることにより，入力に対応する解を出力する．

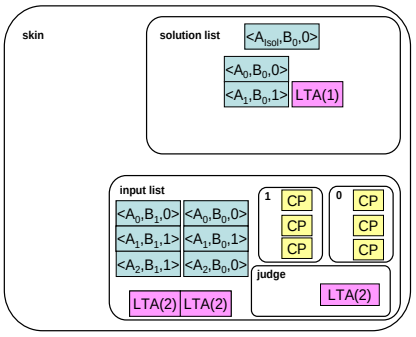


図 23: メモリ機能の挿入操作の状態 3

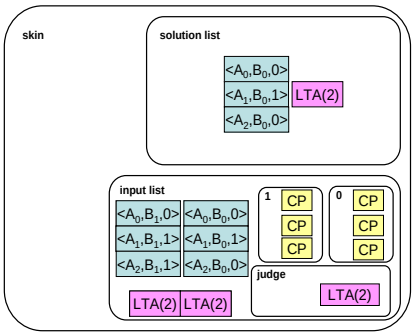


図 24: メモリ機能の挿入操作の状態 4

メモリ機能の探索操作の入出力

入力 入力をアドレス S に格納された m ビットの 2 進数のデータ表すオブジェクトとして入力とする．これらのオブジェクトを，スキン膜に入力として与える．また，すでに入力済みの入力と解は，スキン膜内部の入力リスト膜，解リスト膜に格納されているものとする．
$\langle A_S, B_0, V_{S,0}\rangle$，$\langle A_S, B_1, V_{S,1}\rangle$，$\cdots$，
$\langle A_S, B_{m-1}, V_{S,m-1}\rangle$

出力 入力の m ビットの 2 進数のデータが入力リスト膜に存在する場合，それに対応する解 m' ビットの 2 進数のデータを出力する．そうでない場合は偽を表すオブジェクト $\langle A_{Sout}, B_0, 0\rangle$ を出力する．

4.5.6 探索操作を行う P システムの概要

メモリ機能を持つ探索操作を行う P システムのアルゴリズムは，辞書演算の探索操作の P システムを改良することで実行可能である．

まず，入力の m ビットの 2 進数のデータを保存した以下のオブジェクトをスキン膜に入力として与える．
$\langle A_S, B_0, V_{S,0}\rangle$，$\langle A_S, B_1, V_{S,1}\rangle$，$\cdots$，
$\langle A_S, B_{m-1}, V_{S,m-1}\rangle$
入力に対応するアドレス S のオブジェクトは入力リスト膜に移動する．入力リスト膜において，アドレス S

のオブジェクトが入力リスト膜のデータと同じデータが存在するか調べる．このアルゴリズムは辞書演算の探索操作と同じなので説明は省略する．同じデータが存在する場合は，指定した入力に対応する解を出力する．

以上のように，メモリ機能の探索操作は，辞書演算の探索操作の改良により実現できる．以下に，メモリ機能を実現する P システムの進化規則として，辞書演算の探索操作を行う P システムの進化規則からの変更点を示す．

まず，R_{skin} に以下の進化規則を追加する．

$[_{skin} SMCH(i)]_{skin}$
$\rightarrow [_{solutionlist} SMACH(i)]_{solution_{list}}$

次に，R_{judge} を以下のように変更する．

(変更前)
$[_{judge} SMCH(i)^m]_{judge}$
$\rightarrow [_{judge}]^+_{judge} [_{list} \langle A_{Sout}, B_0, 1\rangle]_{list}$

(変更後)
$[_{judge} SMCH(i)^m]_{judge}$
$\rightarrow [_{judge}]^+_{judge} [_{input_{list}} SMCH(i)]_{input_{list}}$

さらに，$R_{input_{list}}$ の進化規則を以下のように変更する．

(変更前)
$[_{list} \langle A_{Sout}, B_0, 1\rangle]_{list} \rightarrow [_{skin} \langle A_{Sout}, B_0, 1\rangle]_{skin}$

(変更後)
$[_{input_{list}} SMCH(i)]_{input_{list}} \rightarrow [_{skin} SMCH(i)]_{skin}$

最後に，$R_{solution_{list}}$ に以下の進化規則を追加する．

$[_{solution_{list}} SMCH(i)]_{solution_{list}}$
$\rightarrow [_{solution_{list}} MCH(i)^{m'}]_{solution_{list}}$
$[_{solution_{list}} \langle A_i, B_{j'}, 0\rangle MCH(i)]_{solution_{list}}$
$\rightarrow [_{skin} \langle A_{out}, B_{j'}, 0\rangle]_{skin} [_{solution_{list}} \langle A_i, B_{j'}, 0\rangle]_{solution_{list}}$
$[_{solution_{list}} \langle A_i, B_{j'}, 1\rangle MCH(i)]_{solution_{list}}$
$\rightarrow [_{skin} \langle A_{out}, B_{j'}, 1\rangle]_{skin} [_{solution_{list}} \langle A_i, B_{j'}, 1\rangle]_{solution_{list}}$

4.5.7 探索操作を行う P システムの動作

膜計算においてメモリ機能の探索操作を実行する実行例を示すことにより，P システムの動作を説明する．ここでは，論理積の AND 演算に対応する入力と解を例として説明する．まず，入力に対応するデータを 2 進数 11 を入力として与える．よって，

$\langle A_S, B_1, 1\rangle$，$\langle A_S, B_0, 1\rangle$

がスキン膜に入力として与えられるものとする．

また，既に，入力リスト膜にはアドレス 0，1 に 2 進数 00，11 が，解リスト膜にはそれらに対応する解 0，1 が集合として入っているものとする．

初めに，スキン膜から入力リスト膜にアドレス S のオブジェクトを移動し，入力リスト膜において，アドレス S のオブジェクトと同じデータがあるか調べる．同じデータが存在する場合，スキン膜に $SMCH(1)$ が作成され，そうでない場合，$\langle A_{Sout}, B_0, 0\rangle$ がスキン膜へ作成される．ここまでの動作は辞書演算の探索操作の動作と同じなので省略する．この場合は，スキン膜に入力リスト膜にアドレス 1 に同じデータが存在することを表すオブジェクト $SMCH(1)$ が作成される．この状態を図 25 に示す．

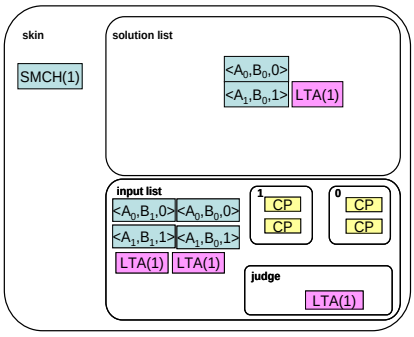


図 25: メモリ機能の探索操作の状態 1

この状態に適用可能な進化規則は，

$[_{skin} SMCH(1)]_{skin}$
$\rightarrow [_{solutionlist} SMACH(1)]_{solution_{list}}$
$[_{solution_{list}} SMCH(1)]_{solution_{list}}$
$\rightarrow [_{solution_{list}} MCH(1)^{m'}]_{solution_{list}}$

となり，図 26 の状態となる．これは，アドレス 1 の解を出力するために $MCH(1)$ オブジェクトを m' ビット分複製している．ここでは，m'=1 となる．

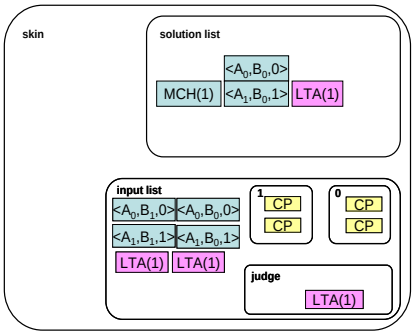


図 26: メモリ機能の探索操作の状態 2

次に，$MCH(1)$ オブジェクトを m' ビット分作成したので，アドレス 1 に記憶しておいた入力 11 に対応する解 1 を出力する．このとき，以下の進化規則，

$[_{solution_{list}} \langle A_1, B_0, 1\rangle MCH(1)]_{solution_{list}}$
$\rightarrow [_{skin} \langle A_{out}, B_0, 1\rangle]_{skin} [_{solution_{list}} \langle A_1, B_0, 1\rangle]_{solution_{list}}$

が適用され，図 27 の状態となる．

ここで，確かに入力 11 に対応する解 1 が出力されたのがわかる．以上で，適用できる進化規則はないので計算終了となる．

4.5.8 P システムの計算量

メモリ機能の挿入操作，探索操作を実行する P システム Π_{memory} は，$O(mn)$ の進化規則の適用により実行が終了するので，以下の定理が得られる．

定理 4 n 個のアドレス，m ビットの 2 進数表記のオブジェクトを入力とし，メモリ機能の挿入操作を計算す

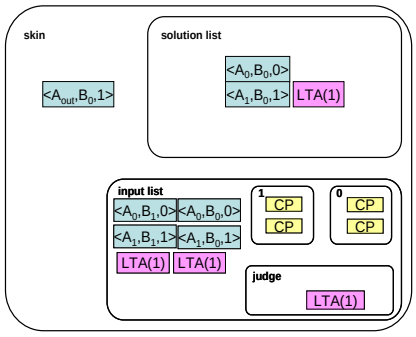


図 27: メモリ機能の探索操作の状態 3

る P システム Π_{memory} は，$O(mn)$ 種類のオブジェクト，$O(m)$ 個の膜，及び，$O(mn)$ 個の進化規則を用いて，$O(1)$ ステップで実行可能である．□

5 最大値計算

本節では，最大値を計算する P システムを示す．最大値計算とは，入力として全順序関係が定義されている n 個の m ビットの 2 進数のデータ $d_0, d_1, \cdots, d_{n-1}$ が与えられたときに，そのデータの最大値を出力する操作である．まず最初に，最大値計算の P システムを定義し，P システムのアルゴリズムの概要を示す．次に，P システムの実行を具体例を用いて説明し，最後に，P システムの計算量を検証する．

5.1 膜計算における最大値計算の入出力

入力はアドレス max_S の 0 ビット目に値 1 を入れて，スキン膜に入力として与える．集合にデータがない場合は偽が出力され，集合にデータが存在する場合は，最大値をアドレス A' に入れて，出力する．

入力 計算開始を知らせるためのオブジェクト $\langle A_{max_S}, B_0, 1\rangle$

なお，これらのオブジェクトは，スキン膜に入力として与える．また，最大値計算のデータは，スキン膜内部のリスト膜と呼ばれる膜に格納されているものとする．

出力 データの集合内の最大の 2 進数を表すオブジェクトの集合
$\langle A'_i, B_0, V_{i,0}\rangle$，$\langle A'_i, B_1, V_{i,1}\rangle$，$\cdots$，
$\langle A'_i, B_{m-1}, V_{i,m-1}\rangle$

5.2 概要

以下に最大値計算を行う P システムのアルゴリズムの概要について示す．本アルゴリズムでは，入力の n 個の 2 進数に対して，最上位ビットから 1 ビットずつ検証を行う．各ビットの検証において 1 の値を持つ 2 進数が存在する場合は，その 2 進数を最大値の候補として残し，その他の値は削除する．各ビットの検証において 0 の値をもつ 2 進数しか存在しない場合は，その 2 進数全てを最大値の候補として残す．この操作を 0 ビット目まで行うと，残った 2 進数が最大値を表すことになる．

最大値計算のアルゴリズムを以下にまとめる．

Step 1 (1-1) から (1-3) を最上位ビット $j = m-1$ から最下位ビット $j = 0$ まで，m 回繰り返す．

(1-1) j ビット目のデータを表すオブジェクトを judge というラベルで表された膜へ移動する．以下，この膜をジャッジ膜と呼ぶ．

(1-2) j ビット目のデータにおける最大値候補を調べる．
j ビット目のデータの検証において 1 の値を持つ 2 進数が存在する場合は，その 2 進数を最大値の候補として残し，最大値候補のオブジェクトをアドレス A'_i に入れ直す．その他の値は，アドレス A''_i に入れ直す．j ビット目のデータの検証において 0 の値をもつ 2 進数しか存在しない場合は，その 2 進数全てを最大値の候補として残し，最大値候補のオブジェクトをアドレス A'_i に入れ直す．

(1-3) j ビットのデータにおける最大値候補のアドレスを count というラベルのついた膜へ保存する．このとき同時に，最大値候補の検証が終わったオブジェクトを list というラベルのついた膜へ移動する．以下，これらの膜をそれぞれ，カウント膜，リスト膜と呼ぶ．
アドレス A''_i に格納されたオブジェクトは最大値候補から外れるので，次の検証を行わない．また，アドレス A'_i に格納されたオブジェクトは最大値候補となるオブジェクトなので，次の $j = j-1$ ビット目のデータの検証で検証される．

以下に最大値計算を実行する P システム Π_{max} を示す．

$$\Pi_{max} = (O, \mu, \omega_{skin}, \omega_{list}, \omega_{judge}, \omega_{count}, R_{skin}, R_{list}, R_{judge}, R_{count}, i_{in}, i_{out})$$

ここで，Π_{max} を構成するそれぞれの集合は以下のようになる．

- $O = \{\langle A_i, B_j, V_{i,j}\rangle, \langle A'_i, B_j, V_{i,j}\rangle, \langle A''_i, B_j, V_{i,j}\rangle, MXC(k) \mid 0 \le i \le n-1, 0 \le j \le m-1, V_{i,j}, k \in \{0,1\}\}$
- $\mu = [_{skin}\ [_{list}\ [_{judge}\ [_{count}\ ]_{count}\ ]_{judge}\ ]_{list}\ ]_{skin}$
- $\omega_{skin}, \omega_{judge}, \omega_{count} = \phi$
- $\omega_{list} = \{\langle A_i, B_j, V_{i,j}\rangle\}$
- $R_{skin} = \{[_{skin}\langle A_{max_S}, B_0, 1\rangle]_{skin} \rightarrow [_{list}\langle A_{max_S}, B_0, 1\rangle]_{list}\}$

- $R_{list} = \{$
  $[_{list}\langle A_{max_S}, B_0, 1\rangle]_{list} \to [_{list}]^-_{list}$,
  $[_{list}\langle A_i, B_{m-1}, 0\rangle]^-_{list}$
  $\to [_{list}]_{list}[_{judge} MXC(1)\langle A_i, B_{m-1}, 0\rangle]_{judge}$,
  $[_{list}\langle A_i, B_{m-1}, 1\rangle]^-_{list}$
  $\to [_{list}]_{list}[_{judge} MXC(1)\langle A_i, B_{m-1}, 1\rangle]_{judge}$,
  $[_{list}\langle A'_i, B_j, 0\rangle\langle A_i, B_{j-1}, 0\rangle]_{list}$
  $\to [_{judge} MXC(1)\langle A_i, B_{j-1}, 0\rangle]_{judge}$
  $[_{list}\langle A'_i, B_j, 0\rangle]_{list}$,
  $[_{list}\langle A'_i, B_j, 0\rangle\langle A_i, B_{j-1}, 1\rangle]_{list}$
  $\to [_{judge} MXC(1)\langle A_i, B_{j-1}, 1\rangle]_{judge}$
  $[_{list}\langle A'_i, B_j, 0\rangle]_{list}$,
  $[_{list}\langle A'_i, B_j, 1\rangle\langle A_i, B_{j-1}, 0\rangle]_{list}$
  $\to [_{judge} MXC(1)\langle A_i, B_{j-1}, 0\rangle]_{judge}$
  $[_{list}\langle A'_i, B_j, 1\rangle]_{list}$,
  $[_{list}\langle A'_i, B_j, 1\rangle\langle A_i, B_{j-1}, 1\rangle]_{list}$
  $\to [_{judge} MXC(1)\langle A_i, B_{j-1}, 1\rangle]_{judge}$
  $[_{list}\langle A'_i, B_j, 1\rangle]_{list}$,
  $[_{list} A'_i\langle A'_i, B_j, 1\rangle]_{list}$
  $\to [_{skin}\langle A'_i, B_j, 1\rangle]_{skin}[_{list}\langle A'_i, B_j, 1\rangle]^+_{list}$,
  $[_{list} A'_i\langle A'_i, B_j, 0\rangle]_{list}$
  $\to [_{skin}\langle A'_i, B_j, 0\rangle]_{skin}[_{list}\langle A'_i, B_j, 0\rangle]^+_{list}$,
  $[_{list}\langle A'_i, B_j, 0\rangle]^+_{list} \to [_{list}\langle A_i, B_j, 0\rangle]_{list}$,
  $[_{list}\langle A'_i, B_j, 1\rangle]^+_{list} \to [_{list}\langle A_i, B_j, 1\rangle]_{list}$,
  $[_{list}\langle A''_i, B_j, 0\rangle]^+_{list} \to [_{list}\langle A_i, B_j, 0\rangle]_{list}$,
  $[_{list}\langle A''_i, B_j, 1\rangle]^+_{list} \to [_{list}\langle A_i, B_j, 1\rangle]_{list}\}$
- $R_{judge} = \{$
  $[_{judge} MXC(1)\langle A_i, B_j, 1\rangle]_{judge}$
  $\to [_{judge} MXC(0)\langle A'_i, B_j, 1\rangle]^+_{judge}$,
  $[_{judge} MXC(0)\langle A_i, B_j, 0\rangle]^+_{judge}$
  $\to [_{judge}\langle A''_i, B_j, 0\rangle]^-_{judge}$,
  $[_{judge} MXC(0)\langle A'_i, B_j, 1\rangle]^+_{judge}$
  $\to [_{judge}\langle A'_i, B_j, 1\rangle]^-_{judge}$,
  $[_{judge} MXC(1)\langle A_i, B_j, 0\rangle]_{judge}$
  $\to [_{judge} MXC(0)\langle A_i, B_j, 0\rangle]_{judge}$,
  $[_{judge} MXC(0)\langle A_i, B_j, 0\rangle]_{judge}$
  $\to [_{judge} MXC(0)\langle A'_i, B_j, 0\rangle]^-_{judge}$,
  $[_{judge}\langle A'_i, B_j, 1\rangle]^-_{judge}$
  $\to [_{judge}]_{judge}[_{count} A'_i]_{count}[_{list}\langle A'_i, B_j, 1\rangle]_{list}$,
  $[_{judge}\langle A'_i, B_j, 0\rangle]^-_{judge}$
  $\to [_{judge}]_{judge}[_{count} A'_i]_{count}[_{list}\langle A'_i, B_j, 0\rangle]_{list}$,
  $[_{judge}\langle A''_i, B_j, 0\rangle]^-_{judge}$
  $\to [_{judge}]_{judge}[_{list}\langle A''_i, B_j, 0\rangle]_{list}$,
  $[_{judge} A'^m_i]_{judge} \to [_{list} A'^m_i]_{list}\}$
- $R_{count} = \{[_{count} A'^m_i]_{count}$
  $\to [_{count}]^+_{count}[_{judge} A'^m_i]_{judge}$,
  $[_{count} A'_i]^+_{count} \to [_{count}]_{count}\}$
- $i_{in} = skin$
- $i_{out} = skin$

ここで，最大値計算を行う P システムおける各オブジェクトの役割を説明する．

$MXC(k)$(Max Count): 1 ビット毎に最大値を調べるオブジェクト

5.3 P システムの動作

膜計算モデルを用いて最大値計算の実行例を示すことにより，P システムの動作を説明する．この例では，2 進数 00，11，10 の入った集合が最大値を計算する入力として与えられ，計算の開始を知らせるためのオブジェクト $\langle A_{max_S}, B_0, 1\rangle$ がスキン膜に与えられることにより，動作が開始されるものとする．この膜計算モデルの初期状態は，図 28 である．

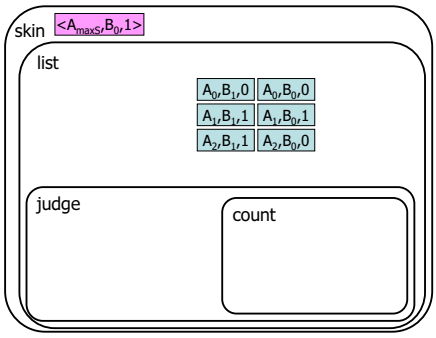


図 28: (1-1) 開始時の $j = 1$ における膜構造の状態

(1-1) 最上位ビットのデータをジャッジ膜へ移動する．
この例では，前述の状態に対して，進化規則

$$[_{skin}\langle A_{max_S}, B_0, 1\rangle]_{skin} \to [_{list}\langle A_{max_S}, B_0, 1\rangle]_{list}$$

が適用され，スキン膜の $\langle A_{max_S}, B_0, 1\rangle$ がリスト膜へ移動する．
次に，最大値計算を最上位ビットから始めるためにリスト膜を $-$ に帯電させる．この例では，前述の状態に対して，以下の進化規則

$$[_{list}\langle A_{max_S}, B_0, 1\rangle]_{list} \to [_{list}]^-_{list}$$

が適用される．リスト膜が $-$ に帯電すると，最大値計算が開始される．この例では，前述の状態に対して，進化規則
$[_{list}\langle A_0, B_1, 0\rangle]^-_{list}$
$\to [_{list}]_{list}[_{judge} MXC(1)\langle A_0, B_1, 0\rangle]_{judge}$
$[_{list}\langle A_1, B_1, 1\rangle]^-_{list}$
$\to [_{list}]_{list}[_{judge} MXC(1)\langle A_1, B_1, 1\rangle]_{judge}$
$[_{list}\langle A_2, B_1, 1\rangle]^-_{list}$
$\to [_{list}]_{list}[_{judge} MXC(1)\langle A_2, B_1, 1\rangle]_{judge}$
が適用され，図 29 の状態へ遷移する．

(1-2) 最上位ビットのデータにおける最大値候補を調べる．

ジャッジ膜において，そのビットにおける最大値候補を検証する．ここでは，$MXC(k)$ オブジェクト

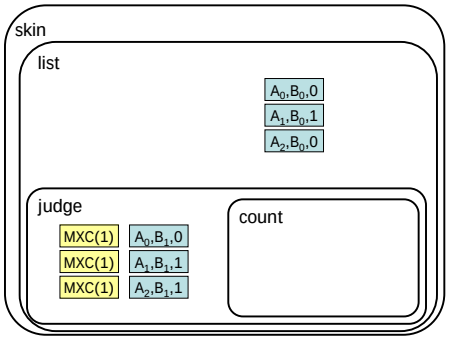


図 29: (1-2) 開始時の $j = 1$ における膜構造の状態

を用いてこのビットにおける最大値候補を検証する．ここで，k には値 1，0 が入り，$MXC(1)$ から $MXC(0)$ へと進化する．この際，2 進数に 1 がある場合はそのビットにおける最大値候補は 1 に決まるので，その値をアドレス A からアドレス A' に入れ直す．この例では，前述の状態に対して，進化規則

$[_{judge}MXC(1)\langle A_0, B_1, 0\rangle]_{judge}$
$\quad \rightarrow [_{judge}MXC(0)\langle A_0, B_1, 0\rangle]_{judge}$
$[_{judge}MXC(1)\langle A_1, B_1, 1\rangle]_{judge}$
$\quad \rightarrow [_{judge}MXC(0)\langle A'_1, B_1, 1\rangle]^{+}_{judge}$
$[_{judge}MXC(1)\langle A_2, B_1, 1\rangle]_{judge}$
$\quad \rightarrow [_{judge}MXC(0)\langle A'_2, B_1, 1\rangle]^{+}_{judge}$

が適用され，図 30 の状態へ遷移する．

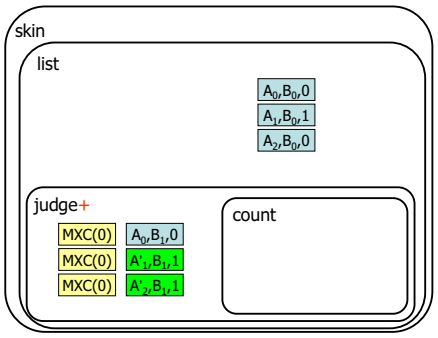


図 30: (1-2) の $j = 1$ における膜構造の状態

次に，このビットにおける，最大値候補が 1 と決まったので，このビットの値が 0 の値は最大値とはならない．そこで，それらの値をアドレス A からアドレス A'' に入れ直す．また同時にジャッジ膜を出力状態の $-$ に帯電させる．この例では，前述の状態に対して，進化規則

$[_{judge}MXC(0)\langle A_0, B_1, 0\rangle]^{+}_{judge}$
$\quad \rightarrow [_{judge}\langle A''_0, B_1, 0\rangle]^{-}_{judge}$
$[_{judge}MXC(0)\langle A'_1, B_1, 1\rangle]^{+}_{judge}$
$\quad \rightarrow [_{judge}\langle A'_1, B_1, 1\rangle]^{-}_{judge}$
$[_{judge}MXC(0)\langle A'_2, B_1, 1\rangle]^{+}_{judge}$
$\quad \rightarrow [_{judge}\langle A'_2, B_1, 1\rangle]^{-}_{judge}$

が適用され，図 31 の状態へ遷移する．

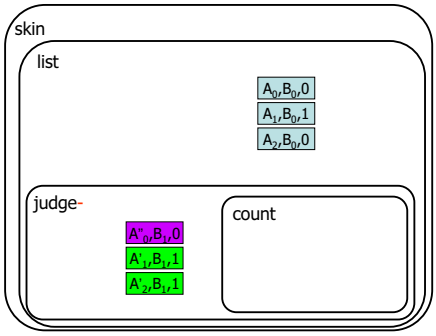


図 31: (1-3) 開始時の $j = 1$ における膜構造の状態

(1-3) 最上位ビットのデータにおける最大値のアドレスを保存する．

カウント膜へどのアドレスが最大値候補かの情報を渡す．また同時に，リスト膜へ最大値候補のアドレスの情報，もしくは最大値候補ではないアドレスの情報を渡す．この例では，前述の状態に対して，進化規則

$[_{judge}\langle A''_0, B_1, 0\rangle]^{-}_{judge}$
$\quad \rightarrow [_{judge}]_{judge}[_{list}\langle A''_0, B_1, 0\rangle]_{list}$
$[_{judge}\langle A'_1, B_1, 1\rangle]^{-}_{judge}$
$\quad \rightarrow [_{judge}]_{judge}[_{count}A'_1]_{count}[_{list}\langle A'_1, B_1, 1\rangle]_{list}$
$[_{judge}\langle A'_2, B_1, 1\rangle]^{-}_{judge}$
$\quad \rightarrow [_{judge}]_{judge}[_{count}A'_2]_{count}[_{list}\langle A'_2, B_1, 1\rangle]_{list}$

が適用され，図 32 の状態へ遷移する．

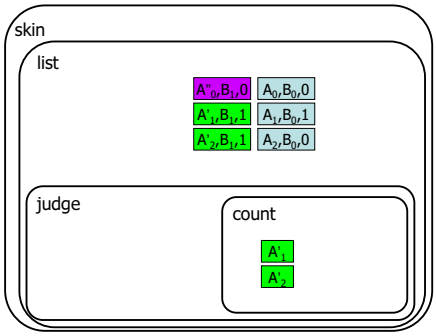


図 32: (1-4) 開始時の $j = 0$ における膜構造の状態

次に，0 ビット目のデータに対して，(1-1) (1-3) の操作を行い，図 33 の状態に遷移する．

0 ビット目まで計算が終了したので，最後に最大値を出力する．カウント膜にアドレスが m ビット分あるデータを出力すればよい．この例では，前述の状態に対して，進化規則

$$[_{count}A'^{\,2}_1]_{count} \rightarrow [_{count}]^{+}_{count}[_{judge}A'^{\,2}_1]_{judge}$$

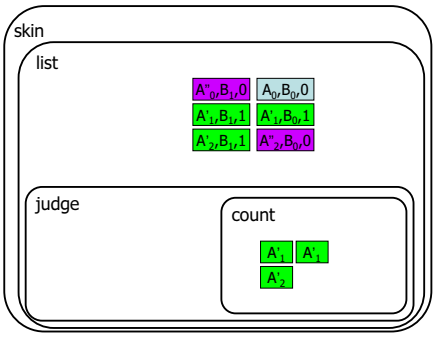


図 33: 0 ビット目の最大値計算終了時の膜構造の状態

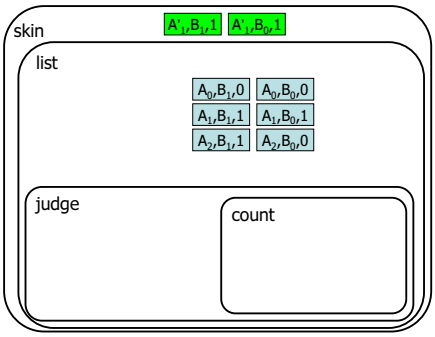


図 34: 最大値出力

$$[_{judge}A_1'^{\,2}]_{judge} \rightarrow [_{list}A_1'^{\,2}]_{list}$$

が適用される．

ここで，カウント膜において，残ったオブジェクトを削除しておく．

$$[_{count}A_2']^{+}_{count} \rightarrow [_{count}]_{count}$$

最後に，リスト膜からスキン膜へ最大値を出力する．この例では，前述の状態に対して，進化規則

$[_{list}A_1'\langle A_1', B_1, 1\rangle]_{list}$
$\rightarrow [_{skin}\langle A_1', B_1, 1\rangle]_{skin}[_{list}\langle A_1', B_1, 1\rangle]^{+}_{list}$
$[_{list}A_1'\langle A_1', B_0, 1\rangle]_{list}$
$\rightarrow [_{skin}\langle A_1', B_0, 1\rangle]_{skin}[_{list}\langle A_1', B_0, 1\rangle]^{+}_{list}$

が適用される．この後リスト膜を $+$ に帯電させているのは，アドレス A'，A'' をアドレス A に戻すためである．この例では，前述の状態に対して，進化規則

$$[_{list}\langle A_1', B_1, 1\rangle]_{list} \rightarrow [_{list}\langle A_1, B_1, 1\rangle]_{list}$$
$$[_{list}\langle A_1', B_0, 1\rangle]_{list} \rightarrow [_{list}\langle A_1, B_0, 1\rangle]_{list}$$
$$[_{list}\langle A_2', B_1, 1\rangle]_{list} \rightarrow [_{list}\langle A_2, B_1, 1\rangle]_{list}$$
$$[_{list}\langle A_0'', B_1, 0\rangle]_{list} \rightarrow [_{list}\langle A_0, B_1, 0\rangle]_{list}$$
$$[_{list}\langle A_2'', B_0, 0\rangle]_{list} \rightarrow [_{list}\langle A_2, B_0, 0\rangle]_{list}$$

が適用され，図 34 の状態へ遷移する．

これ以上，適用できる進化規則がないので計算を終了する．

5.4 P システムの計算量

最大値計算を実行する P システム Π_{max} は，$O(mn)$ の進化規則の適用により実行が終了するので，以下の定理が得られる．

定理 5 n 個のアドレス，m ビットの 2 進数表記のオブジェクトを入力とし，最大値計算を実行する P システム Π_{max} は，$O(mn)$ 種類のオブジェクト，$O(1)$ 個の膜，及び，$O(mn)$ 個の進化規則を用いて，$O(m)$ ステップで実行可能である． □

6 まとめ

本研究では，膜計算を用いて辞書演算，及び，最大値計算を実行する P システムを示した．これらの P システムにより，辞書演算の 3 つの挿入操作，探索操作，及び，削除操作を，n 個の m ビットの 2 進数を表すオブジェクトを用いて，それぞれ $O(mn)$ 種類のオブジェクト，$O(m)$ 個の膜，$O(mn)$ 個の進化規則を用いて $O(1)$ ステップで実行できることを示した．さらに，辞書演算を用いることにより，膜計算を用いてメモリ機能も $O(1)$ ステップで実現可能なことを示した．

また，最大値計算については，提案 P システムにより，n 個の m ビットの 2 進数を表すオブジェクトを用いて，$O(mn)$ 種類のオブジェクト，$O(1)$ 個の膜，及び，$O(mn)$ 個の進化規則を用いて，$O(m)$ ステップで実行できることを示した．

今後の課題として，それぞれの P システムで使用されるオブジェクト，膜，及び，進化規則の数の削減などが挙げられる．

参考文献

[1] A. Leporati and C. Zandron. P systems with input in binary form. *International Journal of Foundations of Computer Science*, Vol. 17, pp. 127–146, 2006.

[2] L. Pan and A. Alhazov. Solving HPP and SAT by p systems with active membranes and separationrules. *Acta Informatica*, Vol. 43, No. 2, pp. 131–145, 2006.

[3] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, Vol. 61, No. 1, pp. 108–143, 2000.

[4] G. Păun. P system with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, Vol. 6, No. 1, pp. 75–95, 2001.

[5] C. Zandron, G. Rozenberg, and G. Mauri. Solving NP-complete problems using P systems with active membranes. *Proceedings of the Second International Conference on Uncoventional Models of Computation*, pp. 289–301, 2000.

[6] 田川. 膜計算における辞書演算及び最大値計算. 九州工業大学卒業論文, 2007.

[7] 立石, 藤原. 膜計算における基本演算アルゴリズム. 第 3 回情報科学ワークショップ, 2007.

PC演習室環境と共生するPCクラスタの構成法とその評価について

大柚　智[†]，桝田　秀夫[‡]
s-ohyu07@dsm.cis.kit.ac.jp， h-masuda@kit.ac.jp
† 京都工芸繊維大学大学院　工芸科学研究科　情報工学専攻
‡ 京都工芸繊維大学　情報科学センター

概要　近年，パソコンの普及率が高まり，PC演習室を設置する教育機関が増えている．通常，PC演習室には多数のパソコンが備え付けられているが，夜間・休日は警備の問題等でPC演習室を閉室しており，備え付けのパソコンが遊休状態にある場合が多い．また，平日昼間でも演習室に備え付けられているパソコンの全てが利用されているわけではないため，遊休状態となる時間 (遊休時間) が存在する．PC演習室にある夜間・休日の遊休時間を有効利用するため，パソコンを夜間・休日にPCクラスタとして運用するシステムがあるが，平日昼間の遊休時間を考慮していないため，PC演習室の稼働率を上げる余地が残されている．そこで，本稿ではPC演習室にあるすべての遊休時間を有効利用することを目指したシステムとして，仮想計算機技術を用いて，PC演習室のパソコンを常時PCクラスタとして運用することを提案した．そして，実験的に研究室内の常用PCの4台を用いてPCクラスタを構成してその性能を評価した．

キーワード　PCクラスタ，仮想計算機

1　はじめに

通常，演習室には多数のパソコンが備え付けられているが，夜間・休日は警備の問題等で演習室を閉室している場合が多い．そのため，夜間・休日はシステムの稼働率が低くなり，計算機資源を有効活用できているとは言い難い．そこでシステムの稼働率を上げるために，演習室に備え付けられているパソコンをクラスタノードとするPCクラスタを構築し，そのPCクラスタでHigh Performance Computingを行うことによる計算資源の活用が考えられている．

演習室に備え付けられているパソコンをクラスタノードとして運用している教育機関の例としては，広島大学 [1] や大阪工業大学 [2] などが挙げられる．これらの教育機関が採用しているのは，講義や演習が行われない夜間・休日に，パソコンをクラスタノードとして運用するシステムである．しかし，講義や演習が行われている平日昼間でも，全てのパソコンが使用されているわけではないため，平日昼間の遊休時間にもパソコンをクラスタノードとして運用できれば，計算資源がより有効に活用できるようになると考えられる．

そこで本稿では，仮想計算機技術を用い，ホストOSを演習用OSとして，ゲストOSをクラスタノード用OSとするクラスタノードを考え，演習室にあるPCを全ての遊休時間にクラスタノードとして利用することを提案する．

2　要件

PC演習室と共生するPCクラスタの構成法を考えるにあたり，以下の点を満たすシステムを目指す.[3]

クラスタ利用が演習利用の妨げとならないこと．

セキュリティやメンテナンス上の問題が無いこと．

3　実験環境

本稿では，演習用 OS 及びクラスタノード用 OS として，CentOS5.2 を用い，仮想計算機技術 [4] として User Mode Linux(UML)[5] を，クラスタリングのためのミドルウェアとして Score[6] を用いることを考えた．そして，これらのソフトウェアを研究室内の常用 PC4 台にインストールしてクラスタノードとした．また，同様に CentOS5.2 と Score をスケジューラ用 PC にインストールした．実験を行ったハードウェア環境を表 1 に示す．

表 1: ハードウェア環境

	クラスタノード	スケジューラ
	HP Proliant ML115	
CPU	Athlon 3500+	
HDD	320GB 7200rpm	
メモリ	512MB	1GB

4　評価

クラスタノード用 OS に仮想計算機技術を用いた際の PC クラスタ (仮想計算機による PC クラスタ) としての性能について評価を行う．

4.1　評価方法

クラスタノード用 OS に Native な CentOS が用いられている PC クラスタ (通常の PC クラスタ) と仮想計算機による PC クラスタのそれぞれに関して，ノードの数を 1，2，3，4 台と増やしていき，N-Queen 問題の N ＝ 15 における計算時間の変化を測定した．N-Queen 問題とは，N × N の盤面にチェスのクイーンを N 個並べて，どのクイーンも互いに張り合わないような局面の数を求める問題である．図 1 に N=4 における解の例を示す．

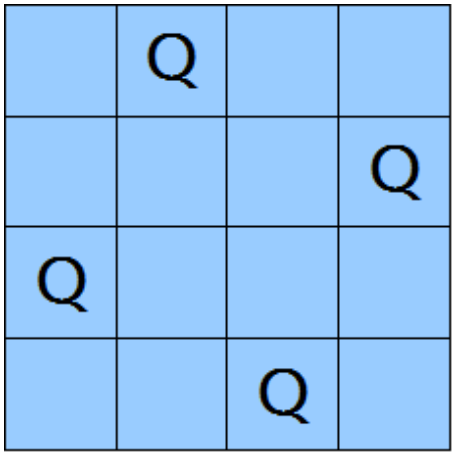


図 1: 4-Queen 問題の解の例

4.2　評価結果

計算にかかった時間のグラフを図 2，通常の PC クラスタにおいて，ノード 1 台の時の相対性能を 1 としたグラフを図 3 に示す．図 3 のように，双方でノード台数に比例した性能向上が見られたが，仮想計算機による PC クラスタは通常の PC クラスタと比較すると，およそ 15%の性能低下があった．

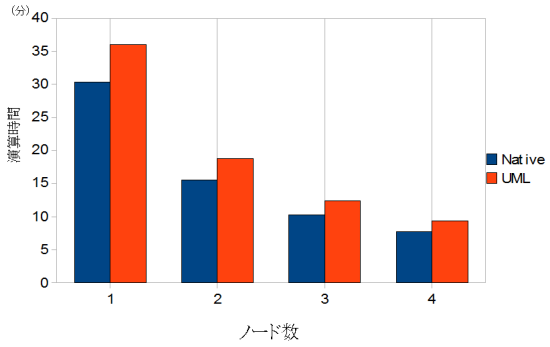


図 2: 計算時間

4.3　ノード機能停止時の性能

4.2 の結果を受けて，演習室環境を想定した実験として，クラスタノードが 4 台の場合の N-Queen 問題の計算において，計算開始から 5 分の時点で演習目的のユーザが 1 名ログインしてきたと仮定し，その時点でノード 1 台の機能を停止させて同様の測定を行った．この際，チェックポイントを 3 分ごとに生成するように設定した。この測定の結果を図 4 に示す．

5　考察

ノードの台数が 1，2，3，4 台の測定では，台数に比例して計算性能の向上が見られた．このことから，N-Queen 問題のような CPU バウンドなア

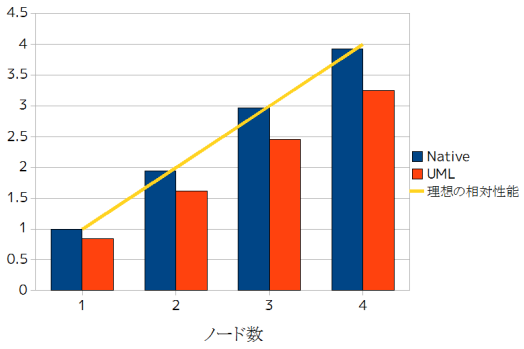


図 3: 相対性能

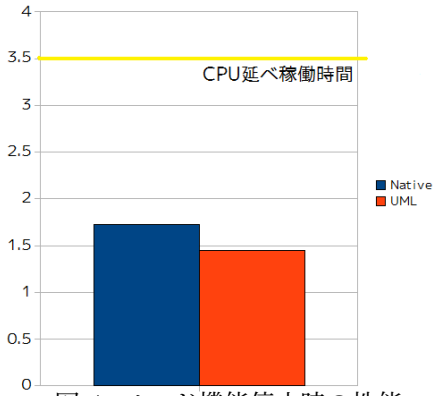


図 4: ノード機能停止時の性能

プリケーションでは，仮想化の有無に関わらずノードの台数の増加に従って PC クラスタの計算性能が向上すると考えられる．

5.1 ノード機能停止時の動作

実環境を想定してノード機能を停止させた測定では，CPU の延べ稼働時間に対しての性能がともに 50%を下回った．これは，チェックポイントの生成が 3 分ごとであったために，機能停止を行ったノードに関して，計算開始 3 分から 5 分の間の 2 分間の計算結果が破棄されてしまったことと，スケジューラとして，途中まで計算の終わったタスクが再分割できなかったため，1 台のノードにタスクを割付けてしまったことが原因であると考えられる．

これらの解決策としては，静的にチェックポイントの生成を行うだけではなく，ノードが機能を停止する際にもチェックポイントの生成を行うことや，最初にタスクを分割する際，ノードの台数個にタスクを分割するのではなく，ノードの台数×k 個にタスクを分割することなどが挙げられる．

6 まとめ

本稿では，PC 演習室と共生するために，UML を用いてホスト OS を演習用 OS，ゲスト OS をクラスタノード用 OS として並行稼働させる方式の PC クラスタについて述べ，実験的に研究室内の常用 PC を用いた性能評価を行った．

今後の課題としては，よりノード数を増やした場合の挙動や，I/O バウンドなアプリケーションでの性能を評価すること，演習利用とクラスタノード利用の切替えに伴うクラスタ性能の低下を防ぐための，パラメータチューニング方法の検討などが挙げられる．

参考文献

[1] 庄司 文由，隅谷 孝洋，石井 光雄:
“教育端末の遊休時間を利用した HPC 環境”，DSM シンポジウム，(2005)

[2] 大阪工業大学キャンパスグリッド:
http://www.oit.ac.jp/dim/facilities.html

[3] 大柚 智, 桝田 秀夫:
“PC 演習室環境と共生する PC クラスタの実装にむけたクラスタノードの構成”, 情報科学ワークショップ,(2007)

[4] 桝田 秀夫，齊藤 明紀:
“Xen と unionfs を用いたサーバファーム運用の可能性の検討”，情報処理学会研究報告，(2006)

[5] User Mode Linux:
http://user-mode-linux.sourceforge.net/

[6] Score
http://www.pccluster.org/

Shape Recognition in Sensor Networks

Yang YANG Sayaka KAMEI Satoshi FUJITA

Department of Information Engineering
Graduate School of Information Engineering, Hiroshima University

Abstract

A large number of applications for wireless sensor networks are tightly associated with answering such query that "in which regions in the environment have the event happened?" In this paper we study the problem of recognizing the *shape* of such event region. Suppose that a large number of sensor nodes are scattered in a geometric region, with nearby nodes directly communicating with each other. Our goal is to approximate the shape of the event region only with the information of direct neighbors. We propose a simple distributed algorithm which constructs a distance field thus identifies several critical points which are capable of capturing the feature of the event region. By collecting information of the nodes in the event region, especially the critical points, the shape of the event region can be effectively approximated.

Keywords: Shape Recognition, Distance Field, Gradient, Sensor Networks.

1 Introduction

The study on wireless sensor networks have recently emerged as an active research area. A typical wireless sensor network consists of a large number of sensor nodes which "sense" the external environment and collaboratively work to process the sensed data. The goal of many wireless sensor networks is to detect and track changes in the monitored environment. Such scenarios arise in target tracking[1] and in the task of robot navigation[2]. In these situations, the problems such as event detection and the event location report must be taken into consideration. For example, consider a network of sensor nodes that are capable of sensing the gas leakage, an important query in this situation could be: "Which regions in the environment have a gas concentration greater than a certain threshold?" We refer to the process of getting answers to such kind of query as event region detection.

Wireless sensor networks are often autonomous systems with severe energy constraints, and individual sensor nodes lack global topology knowledge. In such conditions, self-organizing, energy efficient and distributed algorithms are required for network operation. In this paper, we propose a scheme that tackles the problem of event region detection by recognizing the shape of the region covered by an event. We focus on the case of detecting a single static event. The proposed scheme is capable of presenting the event region as a whole image by collecting messages

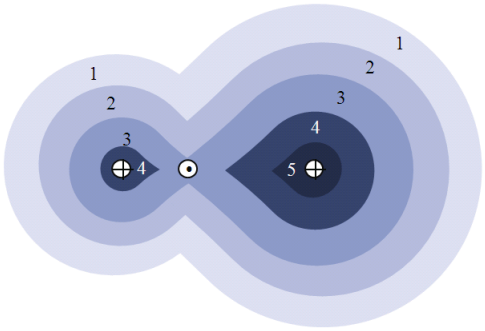


Figure 1: Distance field and critical points.

of quite small size, and it runs in a completely distributed manner. Since lacking for global coordination, we also propose a method to approximate the location. Our approach recognizes the shape of the continuous event region by constructing a distance field within the region, thus identifies several critical points capturing the feature of the shape. The distance field is constructed by assigning each node a value, representing its distance to the boundary of the event region. Since the values present the local thickness, by collecting the information of the nodes with different values respectively, and the location of critical points, we can approximate the shape of the event region at the base station.

2 Related Works

Most of the existing work on event region detection can be classified into two categories; i.e., boundary detection approaches and fault-tolerant approaches.

Boundary detection schemes try to capture the event region by providing a description of its boundary. Chintalapudi *et al.* [3] employs a classifier-based edge detection mechanism to generate a linear polynomial denoting the boundary by sampling the neighboring area. Another class of boundary detection schemes base on constructing a quadtree. Nowak *et. al.* [4] proposed a cluster head based approach for edge detection where a quadtree is constructed, with each leaf corresponding to rectangular subregions. However, our approach is capable of catching the shape of the entire area spanned by the event rather than only the boundary.

Fault-tolerant schemes implement the real sensor readings rather than the binary presence. By assuming that

event measurements are spatially correlated, fault-tolerant approaches focus on distinguishing fault sensor measurements thus disambiguating events by exchanging readings among neighboring sensors. Krishnamachari *et al.* [5] proposed an approach based on a distributed Bayesian algorithm for detecting and correcting such faults. However, it assumes that each sensor knows its own geographical location, which is impossible in most of the cases for wireless sensor networks. Moreover, the fault-tolerant approaches focus on enabling the nodes to determine which sensor readings are interesting, lacking for collaboration among sensors to come to a final decision about the shape of event region.

Unlike the approaches listed above, our scheme is capable of capturing the feature of the entire event region instead of detecting only its occurrence [5] or just the boundary of the event region [3],[4].

3 System Model

Consider a wireless sensor network consisting of a large number of sensor nodes distributed over a given square region. Each node has a capability of detecting an occurrence of event, and a capability of communicating with nearby nodes via wireless communication. For simplicity, we assume that all nodes have identical sensing radius and identical communication radius. We also assume that the density of nodes is sufficiently high such that: 1) the whole square region is covered by the sensing range of sensor nodes, 2) each event is detected by at least two sensor nodes, and 3) the underlying wireless communication network is connected. Each sensor node receives a signal from the environment, and makes a binary decision by the strength of the received signal; i.e., if it exceeds a predetermined threshold, then it recognizes that an event actually happens.

We prepare two base stations s_1 ands_2. The first sink s_1 serves as a data aggregation point, while, since we do not assume a mechanism to explicitly acquire the location information such as GPS, another sink s_2 is deployed for approximating the location collaborating to s_1. Two base stations have more resources than the individual sensor nodes in terms of power, computation, and the memory. The location of base stations is fixed at two corners facing to an edge of the square region.

4 Proposed Scheme

4.1 Initialization

Initially, two sinks flood a gradient setting message over the network. Each node receiving the messages keeps record of the minimum hop counts to two sinks. After calculating the minimum hop counts to each sink, each node will have a tuple (d_1, d_2), where d_i represents the minimum hop count to sink s_i from the current node. In what follows, we refer to the tuple as the "address" of the node.

An event is considered as a 0/1, where presence of a 1 indicates the occurrence of the event and a 0 denotes its absence. As soon as a node detects an event, it forwards a message informing the detection towards the gradient descent direction to s_1. There may be several nodes detecting the same event thus forwarding a message simultaneously. To cut down the cost arisen by the communication, the informing messages should be: 1) merged if they encounter each other, or 2) suppressed if a node has already forwarded the same message from any sensor node, while 3) in the case that the message does not share any node with other messages along its path to s_1, such message will be forwarded to s_1 without suppression.

4.2 Construct the Distance Field

Upon receiving the informing message, s_1 disseminates a query asking what happened and in which area, along the reversed path. Each node receiving the query disseminates it along the reversed path until the query encounters a node with presence of 1. Thus the construction of the distance field is triggered.

We define the *distance field* as the continuous region consisting of nodes with sensor readings equal to 1. The boundary nodes are the nodes who have at least one 1-hop neighbor has reading 0. We construct a distance field such that each node is assigned a value h representing the minimum distance to the boundary. Since we do not assume global coordination, our only measure of distance in the network is the number of hops to the boundary nodes. In the following, we would like to illustrate the details of the distance field construction phase by explaining the action of node with presence of 1 and 0 respectively.

- The action of nodes with reading 1 (the nodes inside the event region):

  On receiving the query: The node u broadcasts the query by radius r and at the same time request(REQ) for the reading of its 1-hop neighbors by appending the tuple $(u, 1)$, which means the node ID and the reading of u.

  On receiving the query with the REQ: The node v replies to u with the tuple $(v, 1)$, then changes the appended REQ tuple to $(v, 1)$ and broadcast the query.

  After broadcasting the REQ: The node u waits a certain time that long enough for its furthest 1-hop neighbor's reply reaching. Then node u maintains a neighbor table recording the neighbors' ID and their readings. *If* u has one or more neighbors with reading 0, as Table 1 indicates, u assigns itself the h-value as 1, and does not change any more. Then u broadcasts $(u, h(1))$, which means the h-value of node u is equal to 1. *Otherwise*, in the case that all the neighbors' reading is 1, u has to wait for the broadcasting of its neighbors' h-value.

  On receiving the broadcasting of h-value: *If* node v has already had its neighbor table with all the neighbors' readings are 1, v records the value into

ID	reading	h
self	1	1
29	0	null
77	0	null
52	1	null
⋮	⋮	⋮

Table 1: Neighbor table of boundary nodes.

its neighbor table, and assigns itself an h-value according to: h_v :=Min $h_{neighbors} + 1$, the h-value of v will always be the minimum h-value of its neighbors plus 1, as Table 2 shows. *If* node v has not yet had its neighbor table, v keeps record of the tuple and waits until the table has been constructed to make a decision.

ID	reading	h
self	1	2
29	1	1
77	1	null
52	1	null
⋮	⋮	⋮

Table 2: Neighbor table of inner nodes.

- The action of nodes with reading 0 (the nodes outside the event region):

  On receiving the query with the REQ: The node w replies to u with the tuple $(w, 0)$.

In this way, the h-value of the boundary nodes will always be determined first, then the h-value of the inner nodes will be calculated by plus 1 to the minimum h-value among its 1-hop neighbors.

4.3 Local Identification of Critical Points

As the distance field construction phase finished, the continuous event region has been separated into several components with different h-values, similar to the $iso-contours$. The component of an iso-value x is the collection of points p with h-value $h_p = x$.

As we decrease the h-values from local maxima to local minima, the components on the iso-contours may merge together, thus the merge saddles will be identified.

4.3.1 Identify local maxima

A node p is said to be **local maximum** if all its 1-hop neighbors have h-value no greater than h_p.Since each node maintains a neighbor table recording its neighbors' ID and h-values, a node can easily identify itself as a local maximum by referring to its neighbor table. A local maximum has all the neighboring h-values no greater than itself.

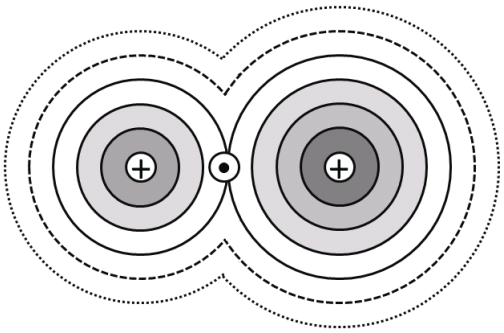


Figure 2: $\bigoplus$ indicates a local maximum. $\bigodot$ indicates the merge saddle point. Dark color means bigger h-values. When we start from local maxima and decrease the h-value, at a saddle point, two iso-contour components start to merge into one.

4.3.2 Identify saddle points

Unlike the local maxima, the saddle points are not easy to be identified, for they have larger and smaller h-values in its neighborhood in an alternating way. We would like to implement a sweep algorithm to spread the information about the peaks of the event region, thus identify the saddle points.

The sweep algorithm we implement here is similar to the one in [6]. We explain the detail with the sweep top down. Each sweep is initiated and labeled by a critical point (a local maximum or a saddle node). A node identifies itself as a local maximum if it discovers that all its 1-hop neighbors have h-value no greater than itself. Notice that, there may be several local maxima of a same peak, but only the information of one local maximum can be spread representing the peak. In order to resolve the ambiguity, we give a priority to the local maximum with smallest node ID. It then initiates a sweep top down. The sweep algorithm runs in a distributed fashion on all the nodes. A node has two possible status, *swept* and *not swept*. Each local maximum node initializes itself as a swept node. When a node has all of its higher neighbors in the swept state, it changes itself to be swept.

ID	h	*status*
self	3	not swept
85	3	not swept
36	5	swept
41	4	swept
⋮	⋮	⋮

Table 3: Neighbor table in terms of status.

In the sweep initiated by a local maximum p, the sweep message carries the tuple (p, h_p, d), i.e., the node ID of p, the h-value of p and the minimum distance to p. Each node being swept will keep this information, as well as from which node it received this information. We define a *descending path* as a path in which each node has an

h-value no greater than its precedent. During the sweep, the information about a local maximum p is propagated along descending paths from p. In addition, each node swept learns the shortest ascending path which eventually lead to the local maximum.

If a node gets two sweep messages from different local maxima, this indicates that two contour components start to merge, thus a saddle should be identified.

The **contour component** $C(u)$ for a node u is defined as the set of nodes which have h-values equal to h_u. Since the nodes advance the sweep frontier in a distributed fashion, it may happen that more than one nodes receive the sweep messages from two peaks. Thus we need to define a saddle point rigorously and resolve the ambiguity.A node s is said to be **saddle point** if it is the one has highest h-value with two descending path from different local maxima. A saddle point is the node where two contour components merge. Now we show how to identify the saddle point. A node that first receives two sweep messages from different peaks p_1, p_2 will promote itself to be a *potential saddle* $s(p_1, p_2)$. In a distributed setting, we need to worry about: two nodes s_1, s_2 (or more) may become potential saddles for the same two peaks. In this case only one saddle node (the one with highest h-value) should survive for two same peaks.

The problem will be resolved by spreading the saddle nodes messages, described below. Once a node s_1 becomes a potential saddle for two peaks p_1, p_2, it stops forwarding the sweep messages from p_1, p_2, and starts a new sweep by propagating the tuple $(s_1, M(p_1, p_2), h_{s_1}, d_{p_1}, d_{p_2})$, indicating that s_1 is the merge saddle of two critical points p_1, p_2. As a node receives messages from saddle nodes, it records the message with largest h-value. If another potential saddle s_2 with $h_{s_2} < h_{s_1}$ received or has recorded such message, it loses its competition to be the real saddle. A potential saddle identifies itself to be a real saddle, if it has never received such messages.

4.4 Approximate the Shape

The h-value recorded at critical points provides the local thickness, which can be used to approximate the shape. We can approximate the shape of the event region at the base station, by collecting the following information:

- For critical points: the tuple $((d_{\text{sink1}}, d_{\text{sink2}}), h)$, indicating the location and the local thickness, should be collected.
- For individual nodes(including critical points): the tuple $(h, 1)$, denoting the h-value of a node within the event region, should be counted with different h-values respectively.

Our scheme enables the sensor nodes within the event region to provide extremely small size information, by collecting which we can easily approximate the shape of the event region at the base station.

5 Simulations

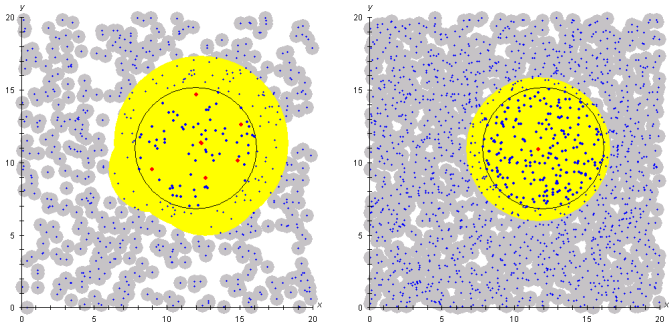
In this section, we report on simulation results for our algorithm. We implemented the algorithm to construct the distance field and identify the critical points, thus approximate the event region. Our simulation do not take into consideration of the networking details, e.g., packet loss, delay and channel contention. Additionally, we do not take into account the errors arisen by inaccuracy location information. This set of simulations aims to verify the correctness of the algorithm and evaluate the feasibility of our approach on the algorithmic level.

5.1 Effect of Network Density

As Figure 3 illustrates, we simulated our scheme in a 20 units by 20 units square region under different network densities. Nodes are deployed in a perturbed distribution, where each node is assigned a random position within the square region. We implemented a round event region centered at location $(12, 11)$ with radius 4.2. The big black circle indicates the real event region boundary. The nodes inside it detected the event. We assume the communicating radius is 1 unit, and two nodes can directly communicate to each other if their distance is no bigger than 1. We set the radius of gray circles 0.5, thus nodes covered by continuous gray regions can communicate directly or indirectly. The network density is varied by adjusting the number of sensor nodes. As expected, as the network density increases, the performance of the algorithm improves.

Figure 3 illustrates the comparison of the performance of our scheme with 600 nodes and 1600 nodes sampling. The red dots indicate the local maxima calculated by our scheme. The yellow region is the approximated event region mapped by the information of critical points.

The unsatisfactory result in Figure 3(i) is due to insufficient connectivity. The random distribution tends to have clustered nodes and communication holes. Thus, in sparse regions, several nodes are incorrectly judged to be local maxima, and the event region is then approximated by the mistaken critical points. However, since the h-values of the critical points are varied, the size of the approximated event region is limited. When the average node



(i) (ii)

Figure 3: The performance of our algorithm with (i)600 nodes sampling (the average node degree is 4) and (ii)1600 nodes sampling (the average node degree is 12).

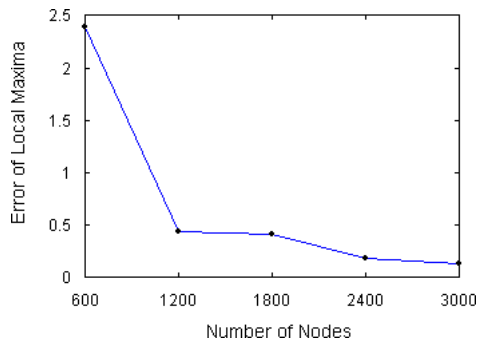
degree reaches 12, as Figure 3(ii) indicates, the communication graph is better connected, and the result proved.

5.2 Approximation Quality

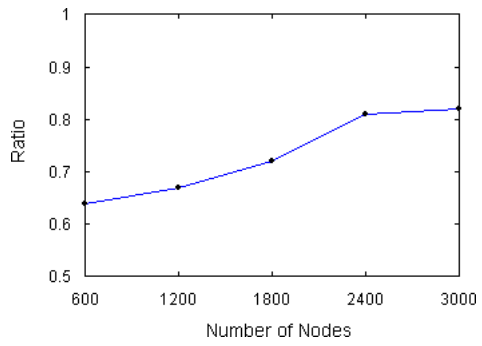
We evaluate the quality of approximations produced by our algorithm by performing extensive simulations in various network densities. We implement the same scenario as Subsection 5.1, e.g., the 20×20 square area, the round event region and the communication model. Figure 4 indicates the approximation quality.

Figure 4(i) shows the average absolute distance from local maxima to the real event region center $(12, 11)$. In the case of 600 nodes, several mistaken local maxima were identified due to clustered nodes and communication holes. Thus, the average distance is then calculated by implementing the mistaken local maxima as well. As the number of nodes increases, the average distance from local maxima to $(12, 11)$ converges. Under high density of 3000 nodes, absolute error 0.14 still exists due to the absence of node at exact $(12, 11)$.

Figure 4(ii) illustrates the ratio of that the event region radius 4.2 versus the largest distance from $(12, 11)$ to the approximated event region boundary. For the best case, the h-value of local maximum is 5, hence the upper bound of the ratio is limited to $4.2/5 = 0.84$. According to our simulation result, with 3000 nodes, the ratio can reach to 0.82. The error to the best case was arisen by the distance from local maximum to $(12, 11)$.



(i)



(ii)

Figure 4: (i)The average error of local maxima. (ii)the ratio of that the event region radius 4.2 versus the largest distance from $(12, 11)$ to the approximated event region boundary.

Through simulations, we found that the connectivity is important to our proposed scheme. Sparse network with insufficient connectivity is the major troubling issue for distance field construction, and effect the calculation of the critical points, thus degrade the quality of approximations.

6 Concluding Remarks

In this paper we proposed a distributed algorithm that recognizes the shape of the continuous event region by constructing a distance field within the region, thus identifies several critical points catching the feature of the shape. Our approach is capable of giving an whole image of the event region.

As future work, we will evaluate the performance of the proposed scheme in more realistic manner, and enhance the approximation quality by focusing on effective approximating strategies.

References

[1] J. Shin, L. Guibas and F. Zhao, "A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks," *in Proc. 2nd Intertional Conference on Information Processing in Sensor Networks (IPSN), Vol. 2634, pp. 223-238*, 2003.

[2] M. Rosencrantz, G. Gordon and S. Thrun, "Decentralized sensor fusion with distributed particle filters," *in Proc. Conference on Uncertainty in Artificial Intelligence (UAI), Acapulco, Mexico*, 2003.

[3] K. K. Chintalapudi and R. Govindan, "Localized Edge Detection in Sensor Fields," *in Proc. IEEE ICC Conference on Sensor Network Protocols and Applications (SNPA), pp. 1-11*, 2003.

[4] R. Nowak and U. Mitra, "Boundary Estimation in Sensor Networks: Theory and Methods," *in Proc. 2nd Intertional Conference on Information Processing in Sensor Networks (IPSN), Vol. 2634, pp. 80-95*, 2003.

[5] B. Krishnamachari and S. Iyengar, "Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks," *in Proc. Journal of IEEE Transactions on Computers, Vol. 53, No. 3, pp. 241-250*, March, 2004.

[6] P. Skraba, Q. Fang, A. Nguyen and L. Guibas, "Sweeps over wireless sensor networks," *in Proc. 5th International Conference on Information Processing in Sensor Networks (IPSN), pp. 143-151*, 2006.

センサネットワークのための自律分散経路集約手法

何　杏平　　亀井　清華　　藤田　聡
広島大学大学院工学研究科

概 要

センサは通常バッテリ駆動であるため, センサネットワークではその電力消費を抑えることが必要となる. 中でもデータ送信にかかる電力がもっとも多い. 各センサはイベントを検知すると, そのイベントのデータを中継ノードを通じて全てのシンクまで送信する. そこで, そのデータをシンクに送信する際の経路に対して, 総送信電力が軽減されるような経路を設定する手法を提案する.

キーワード　センサネットワーク, 複数ソースと複数シンク, 経路集約, 自律分散

1　はじめに

近年, ナノテクノロジー技術と無線通信技術の発展により, 小型のセンサノードで形成されるセンサネットワークが注目を浴びている. センサネットワークでは通信は最もエネルギー消費量の高い操作であるため, 中継ノードが各ソースから送られてきたすべてのデータをそのままシンクに向けて転送するとエネルギー効率の低下を招くことになる. このため, データを伝播する際にデータ集約を行うことにより, できるだけメッセージの数を削減することが重要である. ここで, ソースとはセンサーネットワーク上で情報を検知したノード, シンクは外部から情報を取り出すための集積ポイントに対応する.

本論文では複数ソースと複数シンクが存在するセンサネットワークを対象とし, 各センサが収集したデータが複数シンクに届けられるまでの経路に着目する. 中継ノードが複数のセンサから送られてくるデータを集約することで, ネットワーク全体でのデータ送信の回数を抑えることが可能となるような経路を設定する手法を提案する.

2　経路集約問題

センサネットワークのルーティング方式はデータ集約を考慮するかしないかによって Address-centric routing(ACR) と Data-centric routing(DCR) の二種類に分けられる. ACR ではデータ集約を考慮しないで最短パスで直接シンクにデータを送信する. DCR では各ソースからシンクまでの経路を集約し, データの送信回数を減らすことで電力消費を抑える. 本論文では, DCR に着目し, 送信経路の経路集約を目標とする.

例えば, 図 1(a) においてソース A は中継ノード C を介して, ソース B は中継ノード D を介して別々に送信している. この時, 合計 6 回のデータ送信が行われる. 一方, 図 1(b) においては, ソース A, B ともに中継ノード D を介してシンクに送信している. この時, ノード D でデータ集約が行われると, 合計 5 回しかデータ送信を行わない. このように適当な経路を選ぶことで, データを集約して送信することが可能となり, ネットワーク全体のデータの総送信回数が削減できる.

ネットワーク全体のデータの総送信回数が最小となるような経路木を構築する問題を経路集約問題と呼ぶ. 実際にデータを集約するためには, 各ノード上でデータパケットの同期をとる必要があるが, 本研究ではデータ集約については他の研究に譲り, 経路集約問題についてのみ議論する.

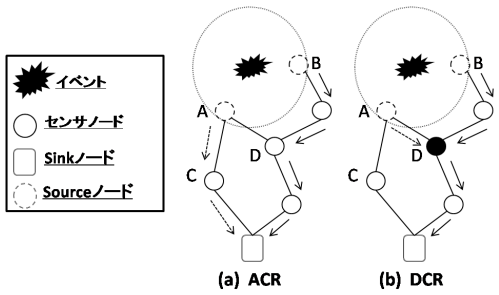


図 1: 経路集約

3　関連研究

文献 [3] は, センサが任意に配置されたセンサネットワークにおいて経路集約問題は NP 困難であると証明した. そして, 下記のような近似解を求める三つのアプローチを提示した.

- Center at nearest source(CNS): シンクに一番近いソースを代理シンクとし, 代理シンクは他のソースからのデータを集約してシンクに送信する.

- Shortest path tree(SPT)：各ソースは最短パスでシンクに送信し, データが重なるノードでデータ集約する.
- Greedy incremental tree(GIT)：シンクに一番近いソースが最短パスで基幹を作り, 残りのソースは基幹に一番近いノードに便乗する.

これらのアプローチを元に様々なアルゴリズムが提案されている. 文献 [1] は単数ソース複数シンクのモデルに対して SPT を応用した最小遅延のルーティングアルゴリズムを提案した. 一方, 文献 [2] は複数ソース単数シンクのモデルに対して GIT を応用したアルゴリズムを提案した. これは伝送経路はある程度集約できるが, 遅延が高くなる場合があることが分かっている. しかし, 複数ソース複数シンクのモデルに対するアルゴリズムは著者の知るかぎりでは未だ提案されていない.

4 モデル

ネットワークは大量のセンサノードで構成されており, センサーノードの集合を V とする. 各ノードが区別できるように識別 ID が付けられているものとする.

各ノード $u \in V$ にはそれぞれ隣接ノードの集合 $N(u)$ が割り当てられており, ノード u は $N(u)$ 中のすべてのノードに向けて 1 ステップでメッセージを放送することができる. メッセージの競合 (collision) はここでは考えない.

ノード間の隣接関係は対称であり[1], ノード集合 V と隣接関係 $\{N(u) : u \in V\}$ とによってつくられるグラフは連結であると仮定する.

V 中の任意のノード u, v が与えられたとき, u から v へのメッセージ送信は, u と v を結ぶパス上で, u から v に向けてメッセージ転送を繰り返すことによって実現できる[2]. このとき u をメッセージの送信元 (originator) と呼び, v をメッセージの宛先 (destination) と呼ぶ[3].

送信元のもっているメッセージが宛先に受け取られるまでに実行された放送回数をそのメッセージ送信の**ホップ数**と呼ぶ. ノード u からノード v までの最短ホップ数を $hop(u, v)$ と記す. ホップ数はメッセージ転送に使われたパスのリンク数と一致し, メッセージ転送で使われたエネルギー量をあらわしている.

V の部分集合として, ソースの集合 S とシンクの集合 D が存在するものとする. ここで, $|S| \geq 1, |D| = 2$ とする. 以下では簡単のため, S 中の各ノードは, 自分が情報を検知したことを, D 中のすべてのノードに対して知らせなくてはならないものとする. ただし, 各ソースが情報を検知するタイミングについては, 何も仮定をおかない.

提案方式は自律分散方式であるので集中管理を行うノードを想定しない. メッセージ交換で得られた情報を元に経路木を計算し, 近似解を求める.

[1]すなわち $u \in N(v)$ ならば $v \in N(u)$ であると仮定する.

[2]各ノードは基本的に周囲に向けてメッセージを放送する機能しかもっていないが, メッセージ中にそのメッセージを受信すべき隣接ノードを書き込むことができるため, 一対一のメッセージ通信をシミュレートすることができる.

[3]以下では 1 回のメッセージ送信 (放送) に関しては, 送信者 (sender), 受信者 (receiver) という言葉を用いて, 送信元, 宛先とは明確に区別する.

5 提案手法

5.1 概要

提案手法の基本アイデアは, 各シンク $d_i \in D$ に対してトークン t_i を対応付け, それらのトークンの移動によって経路の集約過程を制御することである.

トークン t_i が置かれているノードを変数 x_i であらわし, 変数 $x_1, x_2, \ldots, x_{|D|}$ によってあらわされるノードたちの集合を X とする (変数 x_i の値は d_i に初期化される). X は多重集合であり, アルゴリズムの進行に伴って変化することに注意しよう (X の初期値はもちろん D である). ここで, X に対して, ソースを根とした中継ノード数が最小になるような木を MSPT と定義する. S 中の各ソース s_j から X に向けてそれぞれ独立に MSPT を張り, そのようにして張られた MSPT のコストの総和を $\Psi(X)$ と記す.

アルゴリズムは, 以下のステップを繰り返す:

Step 0: x を X 中の任意のノードとする.

Step 1: X 中の x を v で置き換えることで得られるノード集合を X_v とする. $\Psi(X_{x'}) = \min_{v \in N(x)}\{\Psi(X_v)\}$ を満たす x の隣接ノード x' に注目し, もし $\Psi(X_{x'})+1 < \Psi(X)$ であれば, ノード x は自身の後継者として x' を選択し, Step 2 に進む. そのような x' が存在しなければ, 次にチェックすべきトークンをもっているノードを新しく x とした上で, Step 1 に戻る. ただし, 実行が一巡して再び x に戻ってくるまでの間に X 中のどの要素についても後継者が選択されなければ, Step 3 に進む.

Step 2: x と選択された後継者 x' とを結ぶリンクを解の一部として確定した上で $X := X_{x'}$ とし (すなわち x のもっているトークンを x' に移動し), 次にチェックすべきトークンをもっているノードを新しい x として, Step 1 に戻る.

Step 3: それまでに確定されたリンクの集合と, S 中の各ソースから X に向けてそれぞれ独立に張られた MSPT の和集合を解として出力し, アルゴリズムを終了する.

5.2 詳細

まず, $X = \{x_1, x_2\}$ とし, x_1 が後継者を探す状況を考える. シンク数が 2 のとき, 各 $s_j \in S$ から X に向けて張られた MSPT は唯一の分岐点 r_j をもつから, $\Psi(X)$ は次のように与えられる：

$$\Psi(X) \ = \ \sum_{s_j \in S} hop(s_j, x_2) + \sum_{r_j \in S} hop(r_j, x_1) \qquad (5.1)$$

S と x_2 は固定されているから, x_1 を隣接ノード v で置き換えることに伴う $\Psi(X)$ の変化は, 各分岐点から x_1 までのホップ数の和, すなわち上式の第二項の変化分に一致する (ただし x_1 と v とでは, 分岐点の位置が異なる可能性がある). 以下では, どの隣接ノードがもっとも大きな減少分をもつのかを分散的に調べる方法について述べる.

アルゴリズムでは以下の 2 種類のメッセージが用いられる. 各メッセージには基本情報として, 発信元ノード (originator), メッセージ送信者 (sender), 発信元から受信ノードまでのホップ数がそれぞれ付加されている.

- REQ: そのノードまでのホップ数を他ノードに周知するためにフラッディングされるメッセージ.
- Signal(next,Dest,count): イベントの発生を検知したソースからシンクに転送されるメッセージ. ここで Dest はメッセージの目的地の集合 ($\subseteq X$) であり, next はこのメッセージの指定された受信者である. また count は分岐点からのホップ数をカウントするためのフィールドである.

ノード u のホップカウントベクトル hcv は, 任意の $v \in V$ に対して, hcv$[v]$ の値が $hop(u, v)$ と一致するか未定義であるような局所配列である. hcv$[v]$ の値は v から REQ メッセージをフラッディングすることで分散的に計算される. アルゴリズムの前処理では, 各ノードは $S \cup D$ 中のノード v たちに対して hcv$[v]$ を計算し, 記録しておく.

イベントを検知したソースは, そのことを通知するメッセージ Signal を, Dest と next フィールドを適切に設定した上で周囲に向けて放送する. 隣接ノードから Signal を受け取ったノード u は, 自分がそのメッセージの next フィールドで指定された受信者であるかどうかを確認し, そうである場合のみ, 以下の処理を実行する (そうでなければ, 単にそのメッセージを無視する).

Step 1: 局所変数 α, β を $\alpha := N(u)$, $\beta :=$ Signal.Dest のように初期化する.

Step 2: α 中の各 v に対して $\beta_v \stackrel{\text{def}}{=} \{y \in \beta : hop(v, y) < hop(u, y)\}$ を求め, そのサイズが最大となるような w ($\in \alpha$) に着目する. 受け取った Signal メッセージの next フィールドを w, Dest フィールドを β_w, count フィールドを以下のようにそれぞれ更新したものを周囲に向けて放送する：$|\beta| > 1$ のとき count $:= 0$, $|\beta| = 1$ のとき count := count+1.

Step 3: $\alpha := \alpha \setminus \{v\}$, $\beta := \beta \setminus \beta_w$ とする. もし $\beta = \emptyset$ であれば終了し, そうでなければ Step 2 へ戻る.

後継者の探索: 現在の X に向けてのホップ数をすべてのノードが正しくもっていれば, 各ソースから X までの MSPT を各ノードが分散的に計算することができる. また $x_i \in X$ が MSPT を通して各ソースから受け取った Signal メッセージ中の count フィールドの値の総和は, 分岐点から x_i までのホップ数の総和, すなわち (5.1) 式の第二項と一致する. したがって $x_i \in X$ の後継者を探索するためには, x_i のすべての隣接ノードに REQ をフラッディングさせてそれらに対する hcv を構成し, 隣接ノードごとに別々の MSPT を構成してそれらに Signal を流した後で, count フィールドの総和を計算すればよい. フラッディングは, $N(x_i)$ 内部でのゴシップ処理の後で, $N(x_i)$ 外部に向けて発信するという手順で効率よく実現できる.

ここで, 後継者候補からのフラッディングコストは, 以下の性質を利用することで削減できると考えている.

1. 最も遠いソースよりも遠くへ伝える必要はない (メッセージ発信時に TTL を設定して制御できる). 後継者候補の情報は, S から X へ至る最短経路上に存在しえないノードが知っていても役に立たない情報だからである.
2. u が v から REQ を受け取ったとき, もし S に関する u の hcv と v の hcv の間にベクトルの意味での大小関係があり, u の hcv のほうが等しいか大きければ, u は v から受け取ったメッセージを次のノードに転送しなくてもよい. そのようなノードも, 上と同様, 最短経路上に存在しないことがはっきりしているノードだからである.
3. x_i までのホップ数と u までのホップ数が一致すれば, u からの REQ をそれ以上転送しなくてもよい. それよりも後継者候補から遠い場所では, x_i までのホップ数をそのまま流用できる.

6 おわりに

本稿では, 複数のソースと 2 つのシンクが存在する場合の経路集約問題を考察し, 近似解法を提案した. さらにこれをシンクが 3 つ以上の場合に適応することを考えると, MSPT をどのように構築するかの考察が必要となる.

また, $|S| > 2$ の場合は, $|D| = 1$ とした場合でも, 各ソースからシンクに向かうパスの併合 (merge) が消費電力の減

少につながる保証はない. たとえば併合によって転送経路の総リンク数が最小化されたとしても, あるソース $s_1 \in S$ からシンクまでの距離が 2 倍になれば, s_1 からの発信頻度だけが突出して高い状況では, 併合しない場合に比べて 2 倍近くの電力を消費することになる. したがって現実的な観点からは, 各パスの伸長 (stretch) に何らかの制限を加えた上で, その制限の範囲内で併合を試みるのが妥当なアプローチであると考えられる. 現段階では, 各パスの伸長に以下のような制限を加えることを考えている.

> 各ソース s_i から最も遠いシンクまでのホップ数を h_i とするとき, パスの併合によって得られるネットワークの各シンクまでのホップ数は, h_i を超えてはならない.

この制限によって, 消費される電力量は, 制限を付加しない場合の最適値よりも大きくなるが, ひとつのソース s_i だけが検知情報を発信している状況でも, その総リンク数は $h_i \times |D|$ で抑えられることになる (最低でも h_i なので, 下界値の $|D|$ 倍以下であることが保証される). 加えて上記の制限方法は, ソースからシンクに情報を転送する際の遅延時間を保証するという点でも有効である. すなわち, 各ソース s_i が検知情報を D 中のシンクたちに向かって転送する状況では, 上記の制限によって, "h_i ホップ後にすべてのシンクが情報を受け取る" という条件が満足されることになる.

今後, 上記の制限つき集約問題手法のについてアルゴリズムを検討するとともに, シンクが三つ以上のモデルが主な課題として残されている. また, シミュレーションによる経路木の集約度および探索コストを評価をする予定である.

参考文献

[1] S. Chuang, C. Chen, and C. Jiang. Minimum-delay energy-efficient source to multisink routing in wireless sensor networks. *SIGNAL PROCESSING*, 87(12):2934–2948, 2007.

[2] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the 22th IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

[3] B. Krishnamachari, D. Estrin, and S. Wicker. Modeling data-centric routing in wireless sensor networks. In *Proceedings of IEEE INFCOM*, 2002.

二連結性を保証する連結センサカバーアルゴリズム

白石 忠士　　藤原 暁宏

九州工業大学 大学院情報工学研究院

t-shiraishi@zodiac30.cse.kyutech.ac.jp　　fujiwara@cse.kyutech.ac.jp

概 要

センサネットワークにおいて，センサの集合が得た情報を利用するためには，情報を特定の場所に集約する．この操作は一般に照会と呼ばれているが，この照会を行うには，任意のセンサ間が連結である必要がある．一方，センサネットワークにおける重要なテーマとして，通信コストの削減がある．この通信コストの削減手法の一つとして，照会に必要なセンサの部分集合を求める手法が研究されており，その一つとして，連結かつ要求領域を検知領域で被覆する連結センサカバーを求める手法がある．

本研究では，二連結性を保証する連結センサカバーを求めるアルゴリズムを提案する．まず，集中型アルゴリズムと分散型アルゴリズムを示し，次に，それらのアルゴリズムをシミュレーション環境に実装し，既知のアルゴリズムとの比較を行う．この実験結果より，提案アルゴリズムにより，従来手法と同等のセンサ数で，高い確率で二連結性を持つ連結センサカバーが得られることを示す．

1　はじめに

センサネットワークとは，小型自律センサで構成される無線通信ネットワークである．各センサは検知領域内に存在する対象物の観測を行うことができ，通信領域内に存在するセンサと無線通信によるメッセージ交換が可能である．このセンサネットワークにおいて，ある特定の領域の監視を行いたい場合，通常，監視を行いたい領域 (要求領域) に多数のセンサが配布され，各センサが自身の検知領域内の監視を行い，検知して得た情報を一つの計算機に収集し処理を行う．この操作をセンサネットワークにおいて一般に照会と呼ぶが，この照会を行うためには，任意のセンサ間は連結である必要がある．

一方，センサは利用環境上，小型なバッテリーを内蔵することが求められているため，バッテリー容量が小さい．また，充電が困難な場所でも使用することが一般的であるため，電力の消費を抑える必要がある．この電力の消費を抑える手段の一つとして，通信コストの削減がある．通信コストの削減はセンサネットワークにおいて，重要なテーマであり，様々な研究がなされている [1, 4, 5]．特に，照会に最低限必要なセンサだけを使って照会を行う場合，有効な省電力化を図ることができる．一般に，この様な連結かつ要求領域をセンサ自身の検知領域で被覆するセンサの集合を連結センサカバーと呼ぶ．なお，与えられたセンサ集合と要求領域に対して，センサ数が最小の連結センサカバーを求める問題は NP 困難であることが知られている [1]．

この連結センサカバーに対して，Gupta ら [1] は，連結センサカバーを求める逐次アルゴリズムを提案し，さらにアルゴリズムを拡張して分散環境へ適用している．また森川ら [5] は，分散環境において要求領域の被覆を複数センサからのアルゴリズム開始により並行に行う手法を提案している．しかし，Gupta らや森川らのアルゴリズムは，求めた連結センサカバーが連結であることは保証しているが，二連結であることを保証していない．そのため，故障などにより，連結センサカバーを構成するセンサの一つが使用不可能になった場合，ネットワーク全体の連結性を保つことができず，照会が実行できなくなる恐れがある．

本研究では，二連結性を保証する連結センサカバーを求めるアルゴリズムを提案する．提案アルゴリズムでは，まず最初に要求領域に沿って環状の連結センサカバーを構築し，次に，環状の連結センサカバーに含まれる任意の 2 センサと通信可能なセンサを連結センサカバーに追加することにより，二連結性を保証する．このアイデアに基づき，本研究では集中型アルゴリズムと分散型アルゴリズムを示す．次に，それらのアルゴリズムをシミュレーション環境に実装し，既知のアルゴリズムとの比較を行う．この実験結果より，提案アルゴリズムにより，従来手法と同等のセンサ数で，高い確率で二連結性を持つ連結センサカバーが得られることを示す．

本稿の構成は，以下のようになっている．2 章でシステムモデルの説明と問題定義，及び，既存手法の紹介を行う．3 章で提案手法の説明を行い，4 章ではシミュレーションによる評価を行う．最後に 5 章でまとめと今後の課題について述べる．

2　準備

2.1　システムモデル

本研究で対象とするセンサネットワークは，センサネットワークをグラフを用いてモデル化したものである．

センサネットワーク $G = (V, E)$ は，n 個のセンサ集合 $V = \{I_1, I_2, ..., I_n\}$ とリンク集合 E で定義される．各センサ I_i にはそれぞれ固有の識別番号 ID_i が割り振られている．また，各センサ I_i は二次元平面 R 上に配置されており，自身の地理的位置を知っているものとする．

各センサ I_i は，自身を内部に含む通信半径 t の円状の通信領域 T_i を持ち，その中に存在する他のセンサとのみ無線通信が可能である．そのセンサ I_i，I_j が互いに通

信可能であるとき，グラフ G 上のセンサ I_i，I_j 間に双方向通信リンク $e_{ij} \in E$ が存在し，センサ I_i，I_j は隣接するという．本研究では隣接するセンサ I_i，I_j 間のみでメッセージの送受信が可能であるとする．また，ネットワークの中のセンサは全て起動しており，センサ I_i がセンサ I_j に送信したメッセージはセンサ I_j が必ず受信する．このとき，通信の衝突などによるメッセージの消失，及び，メッセージの改変は起こらないとする．また，センサ I_i，I_j 間はメッセージの交換のみによってのみ情報交換を行う．

センサ I_i の検知半径 s による検知領域 S_i とは，センサ I_i が平面 R 上の監視を行う円状の領域であり，検知領域 S_i はセンサ I_i を内部に含む．

また，センサネットワーク G において，互いの検知領域に共通領域が存在する任意のセンサ I_i，I_j の最大通信距離をリンク半径 r_G とよぶ．この任意のセンサ間の距離は，検知領域が重なり合うことから $2s$ 以内であると考えられる．このとき，I_i と I_j を結ぶ直線上に十分なセンサが存在するならば，通信経路は $\frac{2s}{t}+1$ ホップで I_i と I_j は通信可能である．よって，リンク半径 r_G は，$\frac{2s}{t}+1$ である．また，センサに対して，メッセージを送信するために経由した双方向通信リンクの数をホップ数という．

図 1 にセンサネットワークの概念図を示す．白と黒の点で表される 30 個のセンサで構成されており，黒いセンサ I_1 の検知領域が S_1 で表され，通信領域が T_1 で表されている．黒いセンサ I_1 と通信領域内の白いセンサの間にはリンクが存在する．

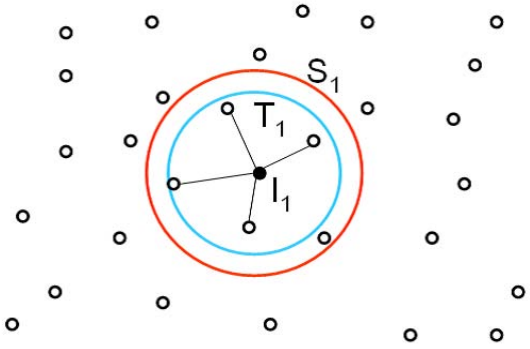


図 1: センサネットワークの概念図

2.2 連結センサカバー問題

連結センサカバー問題とは，センサネットワーク G，及び，被覆すべき領域が与えられたとき，少数の連結なセンサを用いて被覆すべき領域をセンサの検知領域で被覆する問題である．被覆すべき領域を要求領域 R_Q と呼ぶが，要求領域 R_Q はセンサネットワーク G が存在する二次元平面 R 上の連続する凸な平面領域であるとする．また，連結センサカバーにおいては，センサ数が少なければ少ない程，使用センサを減らすことができ，よい連結センサカバーであると考えられる．以下に連結センサカバーについて，更に詳しく定義する．

定義 1 (連結センサカバー問題)

センサネットワーク G=(V, E) に存在する各センサ I_i の検知領域を S_i とする．センサネットワーク G に対して要求領域 R_Q が与えられたとき，以下の条件を満たすセンサの部分集合 M=$\{I_{i_0}, I_{i_1}, ..., I_{i_m}\}(\subseteq V)$ を連結センサカバーとよぶ．

1. $R_Q \subseteq (S_{i_1} \cup S_{i_2} ... \cup S_{i_m})$
2. G の M による部分グラフは連結である　□

ここで，線分を用いた点被覆問題は NP 困難であることが知られており [1]，与えられたセンサ集合と要求領域に対して，センサ数が最小の連結センサカバーを求める問題もまた NP 困難として知られている．

本稿では，センサ数 n は連結センサカバーを構築するのに十分に大きいとする．すなわち，要求領域 R_Q を被覆可能な連結センサカバー M は必ず存在する．また，各センサ I_i には要求領域 R_Q の情報が与えられているとする．

図 2 に連結センサカバーの例を示す．センサ集合 $\{I_1, I_2, ..., I_9\}$ は連結センサカバーである．センサ $I_1, I_3, I_5, I_6, I_8, I_9$ の検知領域で要求領域 R_Q を被覆し，センサ I_2, I_4, I_7 によって集合に含まれるセンサ間の連結性保っている．

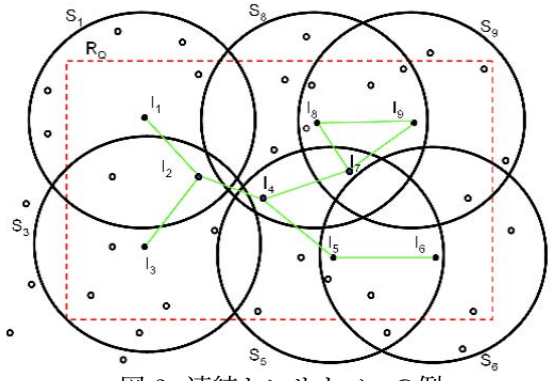


図 2: 連結センサカバーの例

2.3 既存の連結センサカバーアルゴリズム

本節では，既存の連結センサカバーアルゴリズムである Gupta らのアルゴリズム [1] について説明する．Gupta らのアルゴリズムでは，要求領域内の任意の 1 センサからグリーディー法を元にしたアルゴリズムを開始する．また，連結センサカバーを構築し終えるまで，センサネットワークでは以下の値を保持する．

連結センサカバーの集合 ： ラウンド k 終了時に，アルゴリズムによってすでに連結センサカバーに含まれているセンサの集合 M^k.

候補パス集合 ： 各ラウンドにおける候補パスの集合 SP.

候補パス ：最も新しく連結センサカバーに加えられた候補パス P .

コーディネータ ：最も新しく連結センサカバーに加えられたセンサ C^k.

Gupta らのアルゴリズムでは，ラウンド k において以下の操作を実行する．最初に，M^k に含まれていないセンサの中から，M^k に含まれるセンサと定数個のセンサを介して通信可能なセンサを選出する．どの周辺センサを連結センサカバーに加えるかは，M^k に含まれる 1 センサが決定する．このセンサを M^k の コーディネータ C^k とよぶ．アルゴリズムの開始直後では，M^0 に含まれるただ 1 つのセンサがコーディネータ C^0 となる．コーディネータとなるセンサは，カバーにセンサが追加される度に変わり，$C^{k_1} \neq C^{k_2} (k_1 \neq k_2)$ である．

次に，コーディネータはある条件により一つのセンサを選び，コーディネータとそのセンサまでの通信経路上のセンサを連結センサカバーに加えて M^{k+1} とする．このとき，コーディネータによって選ばれたセンサがラウンド $k+1$ のコーディネータ C^{k+1} となり，アルゴリズムはセンサの検知領域によって要求領域 R_Q を完全に被覆し終えたとき終了する．

Gupta らのアルゴリズムは以下の 4 つのフェーズからなる．

Gupta らの分散型アルゴリズム

1. **探索フェーズ** ：探索フェーズでは，コーディネータ C^k が周辺センサへ探索メッセージを送信する．探索メッセージにはカバー M^k に含まれるセンサの座標，及び，コーディネータ C^k が送信者である情報を付加する．探索メッセージはブロードキャストによってコーディネータ C^k から定められた $2r_G$ ホップ内に存在するすべてのセンサに転送される．すなわち，コーディネータ C^k の検知半径 S_i と共通の検知領域を持つセンサ I_j，及び，センサ I_j の検知領域 S_j と共通の検知領域をもつセンサまで探索メッセージが届くようになっている．探索メッセージを受信した各センサは，最初に受信した探索メッセージのみを受信し，その探索メッセージが経由した経路をセンサ I_i の候補パス P_i とする．

2. **応答フェーズ** ：探索メッセージを受信したセンサ I_i は，コーディネータ C^k へ応答メッセージを返す．ただし，応答メッセージを返すのは自身の検知領域 S_i とカバー M^k の検知領域 $S(M^k)$ が共通領域を持つ場合のみである．この応答メッセージは，探索メッセージが経由した経路，すなわち，候補パスを逆に辿って，コーディネータ C^k まで伝えられる．応答メッセージには候補パス (パス上に存在するセンサ ID の列) と，パス上の各センサの検知領域の情報が付加される．

3. **決定フェーズ** ：決定フェーズでは，コーディネータが応答メッセージによって得られた各候補パス P_i を候補パス集合 SP^k に追加，もしくは，更新し，さらに各候補パス P_i の検知領域 $S(P_i)$ を保持しておく．候補パス P_i の検知領域 $S(P_i)$ は以下の式 (1) で求められる．

$$S(P_i) = \cup_{I_j \in P_i} (S_j \cap R_Q) \tag{1}$$

応答フェーズで得られた各候補パスの情報から，コーディネータ C^k はセンサ集合 M^k に追加すべき候補パス P_{new} を選出し，選出された候補パス P_{new} 上の全てのセンサをセンサ集合 M^k に追加する．

候補パス P_{new} の選出方法は以下の通りである．コーディネータ C^k は，すべての応答メッセージを受信すると，候補パス集合 SP^k に含まれる候補パスの中から以下の式 (2) で表される候補パスの評価値が最大となるパスを選出する．

$$\text{評価値} = \frac{\text{候補パス上のセンサで被覆する未被覆のサブエレメント}^*\text{数}}{\text{候補パス上の } M \text{ に含まれていないノード数}} \tag{2}$$

評価値が最大となったパスを P_i とする．コーディネータ C^k は決定メッセージを P_i 上のセンサに送信し，P_i 上のセンサは新たに連結センサカバーを構成するセンサになる．すなわち，センサ集合 M^{k+1} はカバー M^k に含まれるセンサと，新たに追加した P_i 上のセンサで構成される．また，センサ I_i がセンサ集合 M_{k+1} のコーディネータ C^{k+1} になる．決定メッセージには候補パス集合 SP^k，SP^k に含まれる各候補パスの検知領域の情報，及び，カバー M^k の情報を付加し，コーディネータ C^{k+1} は候補パス集合 SP^k を自身の候補パス集合 SP^{k+1} の初期値とする．

4. **繰り返し** ：以上の動作をカバー M^{k+1} の検知領域が要求領域 R_Q を完全に被覆するまで繰り返す．

アルゴリズムの動作例を図 3 に示す．図 3 では，センサ I_1 がコーディネータ C^k があるとき，候補パス P_2^k，P_3^k，P_4^k，P_6^{k-1}，P_9^{k-2} が存在する．候補パス P_6^{k-1}，P_9^{k-2} はラウンド k 以前に得られた候補パスである．この場合は，評価値が最大となる候補パスは P_4^k であり，このパス上にあるすべてのセンサを M^k に追加され，連結センサカバー M^{k+1} が構成される．

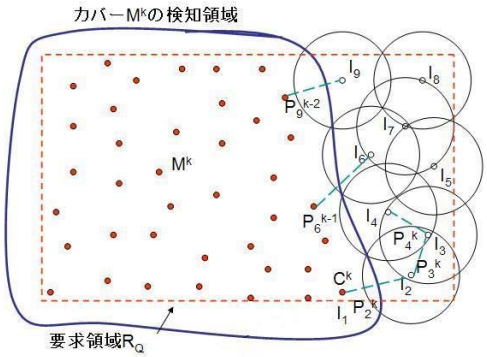


図 3: Gupta らの手法の動作例

*複数のセンサの検知領域が交差することで区切られる最小の領域

2.4 二連結性判定アルゴリズム

本節では，グラフの二連結性の定義と二連結性を確認するためのアルゴリズムについて説明する．

アルゴリズムが構築する連結センサカバーを一つの無向グラフ $G = (V, E)$ と捉えると，センサを節点 $v, u \subseteq V$，任意のセンサ間 u, v のリンクを辺 (u, v) と考えることができる．グラフ G から k 個の節点を取り除いて G を非連結にする操作を，k-点切断という．G を k-点切断する k に対し，最小の k の値を連結度といい，グラフがどの程度固く結び付いているかの尺度になる．また，取り除くことによってグラフが非連結になる節点を関節点という．最小でも 2 点切断を行わなければ非連結にならないグラフを，二連結グラフという [3].

このように，グラフの二連結性は，そのグラフに関節点が存在するか否かにより決定することができる．以下では，関節点の計算方法 [2] を説明し，グラフの二連結性を調べる方法を説明する．

まず，グラフに対して深さ優先探索を行い，深さ優先極大木を作りつつ，行きがけ順に番号 $num[v]$ を振る．次に，「各節点から深さ優先極大木に沿って 0 段以上下がった後に後退辺を 1 度だけ使ってどこまで上へ戻れるか」を表す値 $low[v]$ を計算する．この値の計算には，各節点 v に対して帰りがけに，

- $num[v]$
- $num[v$ から出ている後退辺の行き先の節点 $]$
- $low[v$ の子 $]$

の 3 つの値のうちの最小値を割り当てていけばよい．ここで，後退辺とは深さ優先極大木において節点 v から節点 v の祖先に向かう辺のことである．

これら 2 つの値 num と low を用いて，各節点が関節点か否かは以下の条件で判断できる．まず，深さ優先極大木における根に 2 個以上の子があった場合には，その根が関節点である．それ以外の節点 v については，v の子である節点 w で $low[w] \geq num[v]$ を満たすものが存在すれば，w からは v の祖先である節点に戻る経路が存在しないことになるので，v が関節点であることがわかる．

上記のアルゴリズムを使用して，本研究のアルゴリズムで得られる連結センサかバーの二連結性を判定している．

3 二連結を保証する連結センサカバーアルゴリズム

本章では，二連結性を保証する連結センサカバーを求めるアルゴリズムについて説明する．ます，アルゴリズムの基本的なアイデアを説明し，そのアイデアを集中型と分散型で実装したアルゴリズムを示す．

3.1 提案手法のアイデア

本節では，提案手法のアイデアについて説明する．提案手法では，まず最初に，図 4 のようにあらかじめ要求領域に沿って環状の連結センサカバーを構築する．このような環状の連結センサに関しては，明らかに二連結性が保証されている．したがって，環状の連結センサカバーの任意の 2 つのセンサと連結になるようにカバーにセンサを加えていくと，二連結性を保証したまま，連結センサカバーを構築することができる．

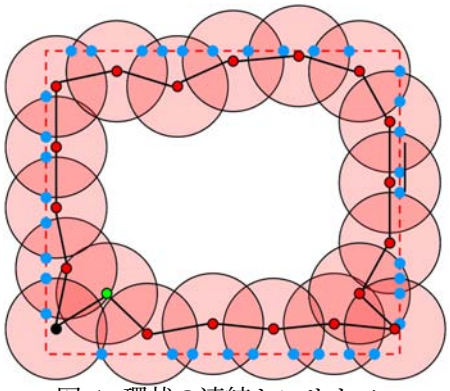


図 4: 環状の連結センサカバー

次に，連結センサカバーにセンサを加える場合のアイデアについて説明する．Gupta らの手法では，追加するセンサとして隣り合うセンサの検知領域との共通領域が少ない検知領域を持つセンサが選出される傾向にある．このため，図 5 のように，連結センサカバーに含まれているセンサ集合の検知領域の間に，小さな隙間が生じる．その結果，その隙間を被覆するためのセンサが必要となり，最終的な連結センサカバーのセンサ数が大きくなってしまう．

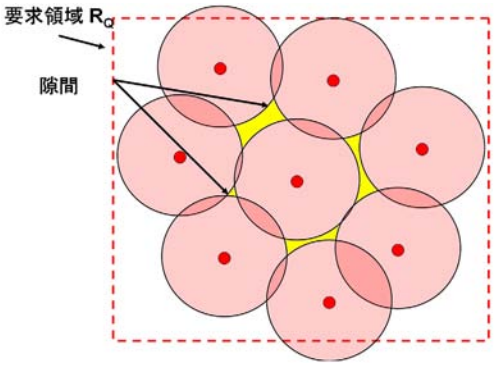


図 5: Gupta らの手法によるカバー M^k の傾向

そこで提案手法では，二連結性を保証し，かつ，センサ数の少ない連結センサカバーを構築できるように，追加するセンサの条件として，以下の条件を用いる．

条件 1 ： 座標 $(0, 0)$ に最も近い未被覆の要求領域 (以下，最近未被覆点とよぶ) を，自身の検知領域に含む．

ただし，コーディネータと通信可能なセンサが，上記の条件を満たせなかったとき，二連結性を保証できず，コーディネータが連結センサカバーの切点となってしまう．

そこで，応答メッセージを返す条件として，以下の条件を加えることにより，二連結性を保証する．

条件２： コーディネータと通信可能であり，かつ，コーディネータ以外の連結センサカバーに含まれるセンサと通信可能である．

上記の条件を用いることにより，二連結性を保証しつつ，少ないセンサ数で連結センサカバーを構築することができると考えられる．

3.2 提案集中型アルゴリズム

提案集中型アルゴリズムの概略を以下に述べる．提案アルゴリズムでは，要求領域と自身の検知領域とで２つの検知領域交点をつくるセンサから開始する．また，アルゴリズムは，要求領域に沿って，連結センサカバーを構築するフェーズ１と，要求領域の内部で連結センサカバーを構築するフェーズ２により構成される．

また，フェーズ１では，各ラウンド k で以下の値を計算する．

候補センサ１： 以下の条件を満たすセンサ I_i を候補センサ $C1_i$ とする．

1. センサ I_i は M^k に含まれていない．
2. M^k に含まれるセンサの検知領域と要求領域の検知領域交点を被覆することができる．

候補パス： $C1$ から M^k に含まれるセンサまでの最短通信経路を候補パス P_i とする．

検知領域交差点： M^k に含まれるセンサ I_i の検知領域 S_i と要求領域 R_Q が交差することによってできる交点を CP_i とする．

フェーズ２では，各ラウンド k で以下の値を計算する．

候補センサ２： 以下の条件を満たすセンサ I_i を候補センサ $C2_i$ とする．

1. センサ I_i は M^k に含まれていない．
2. 最近未被覆点を被覆する．
3. M^k に含まれる２センサと通信することができる．

最近未被覆点： ラウンド k に置いて，最近未被覆点を A^k とする．

具体的な提案アルゴリズムについて以下に示す．

入力： 要求領域 R_Q，及び，センサの集合 $I_0, I_1, ..., I_{n-1}$．ただし，各センサ I_i について，検知領域 S_i と通信領域 T_i が定義されている．

出力： 連結センサカバー M．

集中型アルゴリズム

アルゴリズムの開始時では $k = 0$ とし，センサ集合 M^0 は空集合である．

フェーズ１： 以下のステップ１からステップ４を，カバー M^k が環状になるまで繰り返す．

ステップ１： 要求領域 R_Q と自身の検知領域とで２つの検知領域交点があるセンサ I_i をカバー M^0 に加える．

ステップ２： 検知領域交点 CP_i を，自身の検知領域で被覆することができるセンサの集合を求め，候補センサ $C1_i$ とする．

ステップ３： 探索した各候補センサ $C1_i$ に対し，候補パス P_i を求める．

ステップ４： すべての候補パスに対し，以下の式 (3) で表される評価値を計算する．

$$\text{評価値} = \frac{\text{候補パス上のセンサで被覆する未被覆の要求領域}}{\text{候補パス上のカバーに含まれていないノード数}} \quad (3)$$

次に，候補パスの中から計算した評価値が最大となるものを選出する．最後に，候補パス上のセンサを M^k に加え，$k = k + 1$ とする．

フェーズ２： 以下のステップ１からステップ３を最近未被覆点がなくなるまで繰り返す．

ステップ１： 最近未被覆点 A^k を，自身の検知領域で被覆することができ，かつ，M^k に含まれる２センサと通信可能である候補センサ $C2_i$ を求める．

ステップ２： 探索した各候補センサ $C2_i$ に対し，候補パス P_i を求める．

ステップ３： すべての候補パスに対し，式 (3) で示される評価値を計算し，評価値が最大となる候補パスを選出する．この選出した候補パス上のセンサを M^k に加え，$k = k + 1$ とする．

3.3 提案分散型アルゴリズム

提案分散型アルゴリズムの概略を以下に述べる．提案アルゴリズムでは，任意の１センサからアルゴリズムを開始する．また開始センサは，コーディネータと呼ばれるセンサネットワーク内で唯一のセンサとなる．(アルゴリズムの実行とともに，コーディネータは別のセンサになる．) アルゴリズム終了まで，コーディネータは，以下の値を保持する．

連結センサカバーの集合： 各ラウンド k 終了時に，連結センサカバーに含まれているセンサの集合を M^k とする．

コーディネータ： 各ラウンド k で探索メッセージや決定メッセージを送信するセンサを C^k とする．ただしアルゴリズム開始時は，アルゴリズムを開始するセンサが C^0 である．

分散型アルゴリズム

分散型アルゴリズムは以下のステップにより構成される．

ステップ１： 座標 (0, 0) を検知領域に含むセンサを求める．

ステップ２： 要求領域に沿って環状のセンサカバーを求める．

ステップ 3 : 要求領域内の連結センサカバーを求める.

以下に各ステップのアルゴリズムを説明する.

ステップ 1

ステップ 1 では，探索フェーズ 1，応答フェーズ 1，決定フェーズ 1 で構成 され，コーディネータが座標 (0, 0) を検知領域に含むまで繰り返される．コーディネータが座標 (0, 0) を検知領域に含むと，ステップ 2 に移行する.

探索フェーズ 1 探索フェーズ 1 では，コーディネータが周囲の通信可能なセンサのみに探索メッセージ 1 をブロードキャストする．探索メッセージ 1 に付加されている情報は，コーディネータのセンサ ID である.

応答フェーズ 1 探索メッセージ 1 を受け取ったセンサは，コーディネータに応答メッセージ 1 を返信する．応答メッセージ 1 に付加されている情報は，自身のセンサ ID と座標である.

決定フェーズ 1 コーディネータは応答メッセージ 1 より，最も座標 (0, 0) に近いセンサを選出する．選出したセンサに，決定メッセージ 1 を送信する．決定メッセージ 1 を受け取ったセンサは，コーディネータとなり，探索フェーズ 1 を開始する.

ステップ 2

ステップ 2 は，探索フェーズ 2，応答フェーズ 2，決定フェーズ 2，及び，接続フェーズで構成される．カバー M^k と要求領域の交点がすべて被覆されるまでは，探索フェーズ 2，応答フェーズ 2，決定フェーズ 2 が順番に繰り返され，M^k と要求領域の交点がすべて被覆されると，接続フェーズを実行後に，ステップ 3 に移行する.

探索フェーズ 2 探索フェーズ 2 では，各ラウンド k のコーディネータ C^k が周辺センサに探索メッセージ 2 をブロードキャストする．探索メッセージに付加されている情報は，コーディネータ C^k が送信元であるという情報，設定されたホップ数，及び，M^k に含まれているセンサの集合である．メッセージはコーディネータ C^k から，設定されるホップ数内に存在するすべてのセンサに転送される．なお，ホップ数の初期値は $r_G = \frac{2s}{t}$(s は検知半径，t は通信半径の値) である．探索メッセージ 2 を受信したセンサは，最初に受信したメッセージのみを受信し，探索メッセージ 2 が経由した経路をセンサ I_i の候補パスとする.

探索フェーズ 2 の例を図 6 に示す．赤い点は連結センサカバーに含まれるセンサを表しており，緑色の点はコーディネータを表している．また，黒い点は連結センサカバーに加えられていないセンサである．青い点は，連結センサカバーに加えられたセンサの検知領域と要求領域の未被覆の交点である．さらに，

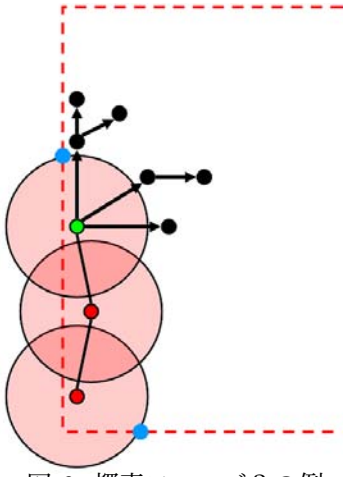


図 6: 探索フェーズ 2 の例

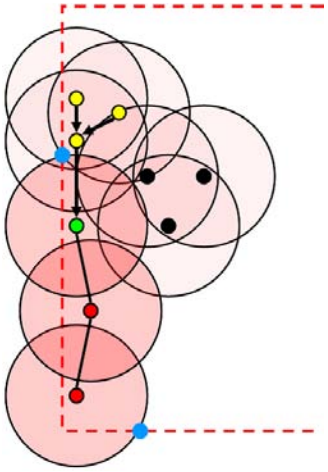


図 7: 応答フェーズ 2 の例

赤い円は検知領域を表しており，黒い線は通信可能であることを，黒い矢印はメッセージが矢印方向に送信されていることを示している．赤い破線は要求領域であることを示している．なお，通信ホップ数は 2 である.

応答フェーズ 2 コーディネータ C^k からの探索メッセージ 2 を受信したセンサ I_i の検知領域が，コーディネータ C^k の検知領域と要求領域の交点を被覆することができる場合，応答メッセージ 2 をコーディネータ C^k に向けて送信する．伝達経路は応答メッセージ 2 を送信するセンサ I_i の候補パスを逆に辿ったものである．応答メッセージ 2 には，候補パス (パス上にあるセンサ ID と座標) と，パス上の各センサの検知領域の情報が付加される.

図 7 に例を示す．赤い点は連結センサカバーに含まれるセンサを表しており，緑色の点はコーディネータを表している．また，黒い点は連結センサカバーに加えられていないセンサを表しており，黄色の点応答メッセージを返すセンサを表している．また，黒い矢印は応答メッセージ 2 が送信されることを表している.

決定フェーズ２ 決定フェーズ２では，応答メッセージ２によって得られた候補パスの中から，評価値が最大のものを選出し，連結センサカバーに加える．評価値の計算方法は (3) 式と同様である．

コーディネータ C^k は決定メッセージ２を P_i^k 上のセンサに送信し，P_i^k 上のセンサは新たに連結センサカバーを構成するセンサになる．なお，ラウンド $k+1$ でコーディネータ C^{k+1} となるセンサは，選出された候補パス上の先端にあるセンサ I_i である．決定メッセージ２には連結センサカバーに含まれるセンサの情報が付加される．

また決定フェーズ２では，応答メッセージ２が得られなかった場合，探索メッセージ２に付加する情報であるホップ数を１増やす．ホップ数を増やしたことによって M^k が更新された場合，再びホップ数を以下の式 (4) に示すホップ数に戻す．

$$\text{ホップ数} = \frac{2s}{t} \tag{4}$$

接続フェーズ 接続フェーズでは，環状の連結センサカバーを完成させる．

コーディネータ C^k は，通信可能なセンサに接続メッセージ１を送信する．メッセージに付加する情報は，M^k に含まれるセンサの情報，及び，コーディネータ C^k が送信元であるという情報である．接続メッセージ１を受け取ったセンサのうち，コーディネータ C^0 と通信可能なセンサは接続メッセージ２を返信する．接続メッセージ２に付加する情報は，自身のセンサ ID と座標である．

コーディネータ C^k は，接続メッセージ２を返信したセンサの中から，評価値が最大になるセンサを選出し，連結センサカバーに加える．評価値の計算式は以下に示す．

$$\text{評価値} = \text{センサが被覆する未被覆の要求領域} \tag{5}$$

なお，評価値が等しいときは，コーディネータ C^0 から最も遠いセンサを選出する．

コーディネータ C^k は接続メッセージ３を，新しく連結センサカバーに加えられたセンサに送信する．接続メッセージ３に付加される情報は，M^k に含まれるセンサの情報である．接続メッセージ３を受け取ったセンサは，次のラウンドのコーディネータとなり，探索フェーズ２に移行する．

接続フェーズの例を図8，及び，図9に示す．図8は接続フェーズを行う前の連結センサカバー，図9は接続フェーズを行った後の連結センサカバーである．

ステップ３

ステップ３では，探索フェーズ３，応答フェーズ３，決定フェーズ３で構成され，未被覆の領域がなくなるまでこれらのフェーズが，この順番で繰り返される．

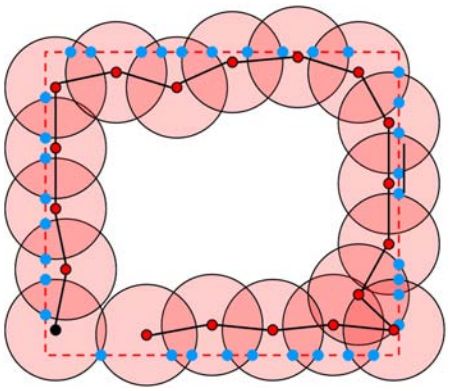


図 8: 接続フェーズに移行する前の状態

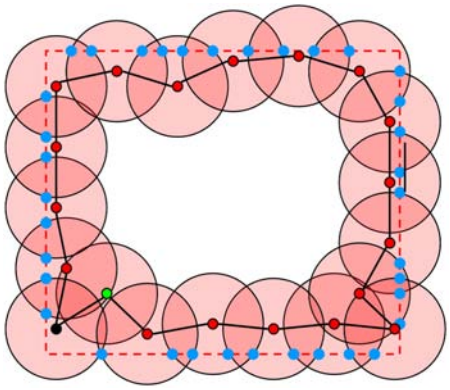


図 9: 接続フェーズ実行後の状態

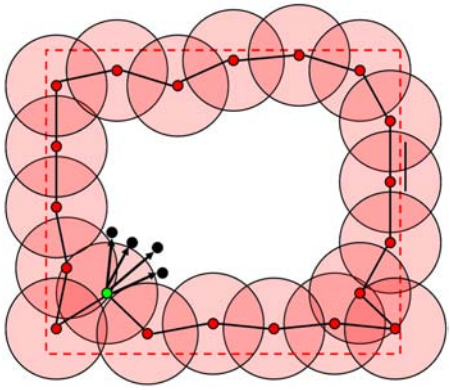


図 10: 探索フェーズ３の例

探索フェーズ３ 探索フェーズ３では，コーディネータ C^k が通信できるセンサに探索メッセージ３をブロードキャストする．探索メッセージ３に付加されている情報は，コーディネータ C^k が送信元であるという情報，M^k に含まれているセンサの座標である．

図10に例を示す．赤い点は連結センサカバーに含まれるセンサを表しており，緑色の点はコーディネータを表している．また，黒い点は連結センサカバーに加えられていないセンサを表している．黒い矢印はメッセージが矢印方向に送信されていることを示している．

応答フェーズ３ コーディネータ C^k からの探索メッセー

ジ３を受信したセンサは，以下の２つの条件を満たすとき，応答メッセージ３を返信する． 応答メッセージ３に付加される情報は，自身のセンサ ID と座標である．

条件１： 最近未被覆点を自身の検知領域に含む．

条件２： コーディネータと通信可能であり，かつ，コーディネータ以外の連結センサカバーに含まれるセンサと通信可能である．

図 11 に例を示す．黄色の点は応答メッセージ３を返信するセンサを表しており，青い点は，最近未被覆点ある．

決定フェーズ３ 決定フェーズ３では，応答メッセージ３によって得られたセンサの情報の中から，評価値が最大なものを選出し，連結センサカバーに加える．評価値の計算方法は (5) 式と同様である．

コーディネータ C^k は決定メッセージ３を選出されたセンサに送信し，決定メッセージ３を受け取ったセンサが次のラウンドのコーディネータとなる．決定メッセージ３には連結センサカバーに含まれるセンサの情報が付加される．

4 実験結果

本章では，既存手法である Gupta らの手法 [1] と森川らの手法 [5] による分散型アルゴリズムと，提案手法の集中型アルゴリズムと分散型アルゴリズムのシミュレーションを行い，その結果を比較する．

4.1 シミュレーションモデル

本研究のシミュレーションでは，一辺の長さが 100 の正方形を要求領域 R_Q とする．また，入力のセンサ集合は，センサ数が 4000, 5000, 6000, 7000，及び，8000 とし，各センサ集合を要求領域 R_Q 内にランダムに配置する．各センサ I_i の持つ検知領域 S_i は検知半径 $s = 4$，通信領域 T_i は通信半径 t の単一円とし，通信半径 t は 4, 5, 6, 7, 8 に対して計測を行う．各センサはラウンド方式で動作し，１回のラウンドで各センサはメッセージを受信し，メッセージに対する適切な計算を行い，計算結果としてメッセージの送信を行うものとする．なお，本研究における実験環境を表 1 に示す．

4.2 シミュレーション結果

まず，Gupta らの手法，森川らの手法，及び，提案手法の分散型アルゴリズムにより得られた連結センサカバーの通信コストに対して評価を行う．

図 12～ 図 16 はセンサ数が 4000～8000 の場合の各手法の通信コストを比較したものである．これらの図より，提案手法は，二連結性を保証しない Gupta らの手法よりも非常に通信コストが小さく，同じく二連結性を保証しない森川ら手法の通信コストとほとんど変わらないことがわかる．

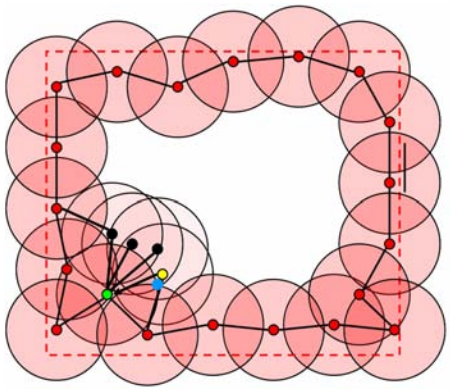


図 11: 応答フェーズ３の例

表 1: 実験環境

CPU	Core 2 Duo E4600 2.4GHz
Memory	2GB
OS	Cent OS 4.6
言語	LEDA6.1 & g++4.1.2

次に，Gupta らの手法，森川らの手法，提案手法の分散型アルゴリズム，及び，提案手法の集中型アルゴリズムにより得られた連結センサカバーのセンサ数に対して評価を行う．

図 17～21，はセンサ数が 4000～8000 の場合で，各手法を比較したものである．これらの図より，通信半径が 6 以上のとき，森川らの手法と，提案手法の集中型のアルゴリズムによって得られる連結センサカバーが，最も使用するセンサが少なくて済むことがわかる．また，提案手法の分散型と森川らの手法を比べても，連結センサカバーに使用したセンサの数は，同程度であることがわかる．

最後に，提案手法の集中型，及び，分散型のアルゴリズムによる二連結性の保証率を表 2，及び，表 3 に示す．これらの表からわかるように，通信半径が小さく，センサ数が少ないときは，二連結性を持つ連結センサカバーを得られていない場合が多い．これは，提案手法では連結センサカバーに加えるセンサの探索を直接通信できるセンサのみに行っているからである．コーディネータと直接通信できるセンサだけでは，最近未被覆点を検知領域に含み，二連結性を保証するセンサを発見することができない場合が多いため，通信半径が小さいほど二連結性を保証する割合が低くなっている．この保証率を 100 % にすることが今後の研究課題である．

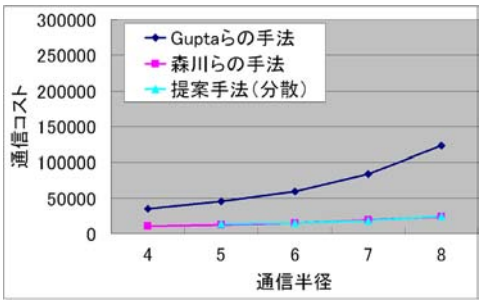


図 12: 通信コスト (センサ数 4000)

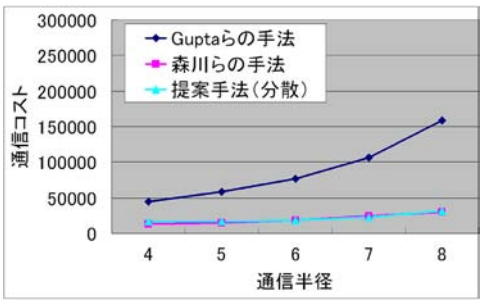


図 13: 通信コスト (センサ数 5000)

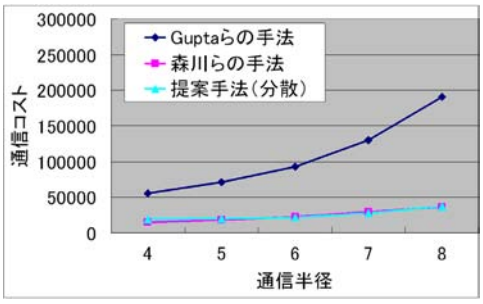


図 14: 通信コスト (センサ数 6000)

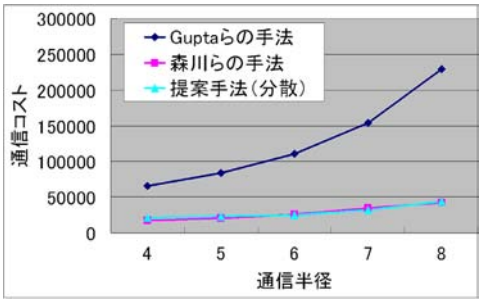


図 15: 通信コスト (センサ数 7000)

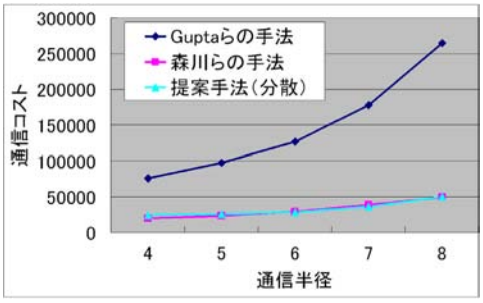


図 16: 通信コスト (センサ数 8000)

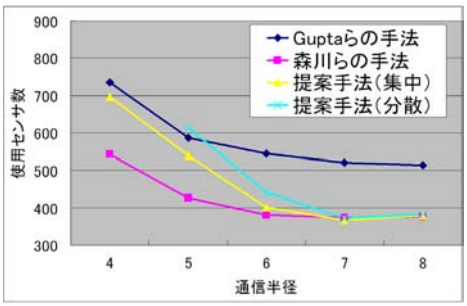


図 17: センサカバーに使用したセンサ数 (センサ数 4000)

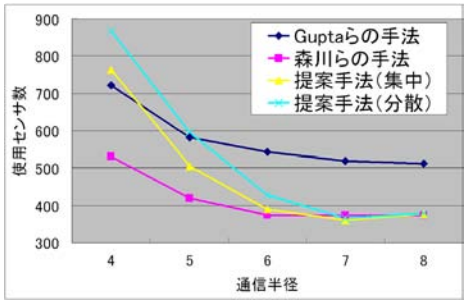


図 18: センサカバーに使用したセンサ数 (センサ数 5000)

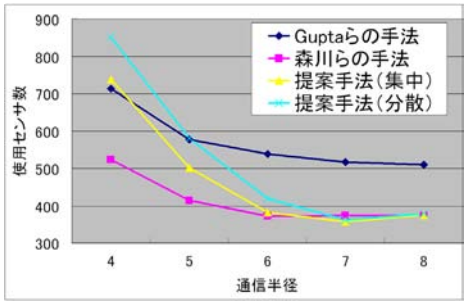


図 19: センサカバーに使用したセンサ数 (センサ数 6000)

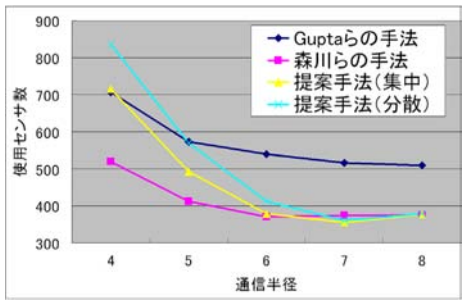


図 20: センサカバーに使用したセンサ数 (センサ数 7000)

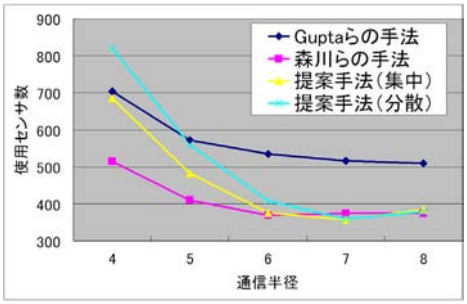


図 21: センサカバーに使用したセンサ数 (センサ数 8000)

5 まとめ

本研究では，二連結性を保証する連結センサカバーを求めるアルゴリズムを提案した．また，提案手法を，集中型と分散型のシミュレーション環境に実装し，既存手法の分散型アルゴリズムと比較した．

提案分散アルゴリズムは，集中型アルゴリズムや既存の分散型アルゴリズムと同程度のセンサ数と通信コストで，高い確率で二連結性を保証する連結センサカバーを構築できることを示した．

今後の課題としては，二連結性の保証率を高めることや，検知領域の異なるセンサを用いたセンサネットワークへの対応などが挙げられる．

参考文献

[1] H. Gupta, S. R. Das, and Q. Gu. Connected sensor cover:self-organization of sensor networks for efficient query execution. *Proc of MobiHoc 03*, 2003.

[2] 五十嵐健夫．データ構造とアルゴリズム．数理工学社, 2007.

[3] 茨木俊秀. C によるアルゴリズムとデータ構造. 株式会社昭晃堂, 1999.

[4] 高田亮．検知領域交点を考慮した連結センサカバーアルゴリズムに関する研究. 九州工業大学, 修士論文, 2008.

[5] 森川雅和, 鈴木朋子, 大下福仁, 角川裕次, 増澤利光. 領域被覆のためのセンサネットワークアルゴリズム. 情報処理学会研究報告，pp.357-362, 2007.

表 2: 提案集中アルゴリズムにより求められた連結センサカバーの二連結性充足率 (縦軸：通信半径，横軸：センサ数)

	4000	5000	6000	7000	8000
4	2/100	80/100	90/100	93/100	92/100
5	55/100	98/100	99/100	99/100	99/100
6	92/100	100/100	100/100	100/100	100/100
7	99/100	100/100	100/100	100/100	100/100
8	100/100	100/100	100/100	100/100	100/100

表 3: 提案分散アルゴリズムにより求められた連結センサカバーの二連結性充足率 (縦軸：通信半径，横軸：センサ数)

	4000	5000	6000	7000	8000
4	0/100	73/100	83/100	76/100	83/100
5	5/100	92/100	92/100	91/100	95/100
6	13/100	87/100	95/100	91/100	90/100
7	27/100	95/100	90/100	88/100	93/100
8	38/100	94/100	90/100	93/100	93/100

1万円ゲームについて

広島大学　中野浩嗣

10000円ゲーム(1)

- 2プレイヤーゲーム(プレイヤーA, Bとする)
- 最初にA, Bはそれぞれ10000円を所持.
- 1ゲームは10ラウンドで, 所持金の10000円全てを10ラウンドで賭ける.
  - AとBのiラウンド目の掛け金を, それぞれ, aiとbiとすると,
    a1+a2+・・・+a10=b1+b2+・・・+b10=10000
- 各ラウンドにおいて, 小さい金額を賭けた方が, 両方の掛け金をもらえる. 引き分けは, 自分の掛け金をそれぞれもらう.
  ai<biのとき, Aがai+biを得る.
  ai>biのとき, Bがai+biを得る.
  ai=biのとき, AとBは, それぞれai(=bi)を得る.

10000円ゲーム(2)

- 相手がこれまでに賭けた金額を知ることができる. (iラウンドにはi-1ラウンドまでの相手の掛金を知っている)
- 10ラウンド終了時に多く(1円でも)お金を持っているプレーヤーの勝ち.
- 1対1対決で1000回連続で勝負を行い, 勝ち数が多い(1勝でも)多い方が勝ち.
- 1000回勝負の際に, 過去の勝負を記憶して用いても良い. (対戦相手の過去の傾向を利用できる.)
- 研究室全員参加の総当りで順位を決める.

対戦例

- 戦略always1000:全ラウンド1000円を出す.
- 戦略minus1:最初は0円を出す.
  2回目以降は, 相手が1つ前のラウンドで出した金額ー1を出す.
  最終ラウンドは残り金額を出す.

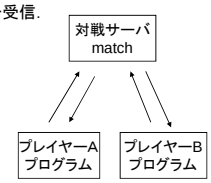
対戦結果

	always1000の掛金	minus1の掛金	always1000の累積	minus1の累積
1:	1000	[0] ->	0	1000
2:	1000	[999] ->	0	2999
3:	1000	[999] ->	0	4998
4:	1000	[999] ->	0	6997
5:	1000	[999] ->	0	8996
6:	1000	[999] ->	0	10995
7:	1000	[999] ->	0	12994
8:	1000	[999] ->	0	14993
9:	1000	[999] ->	0	16992
10:	[1000]	2008 ->	3008	16992
	3008	[16992]		

↑ minus1の勝ち

対戦サーバ

- 対戦サーバ:C言語で書かれた比較的簡単なプログラム
  - 各プレイヤーのプログラムをfolk+execで子プロセスとして実行
  - このとき, 対戦回数(通常1000回)を引数として渡す.
  - 子プロセスと双方向パイプ(pipeとdup2を使用)で通信.
  - ラウンド毎に, 各プレイヤーの賭金を受信.
  - 各プレイヤーに相手の賭金を送信.
  - 勝敗を判定し結果を表示.
- プレイヤーのプログラム
  - 対戦回数(通常1000)を第1引数で受け取る.
  - 各ラウンドの賭金を標準出力に書く.
  - 相手の賭金を標準入力から読む.



プレイヤープログラムの例1

- 戦略always1000
  - 全ラウンド1000円を賭ける.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc,char **argv){
  int n,j,k,num_game;

  num_game=atoi(argv[1]);

  for(k=1;k<=num_game;++k)
    for(j=1;j<=10;++j){
      printf("1000¥n");
      fflush(stdout);
      scanf("%d",&n);
    }

  exit(EXIT_SUCCESS);
}
```

プレイヤープログラムの例2

- 戦略minus1
  - 最初は0円を賭ける.
  - 2回目以降は，相手が1つ前のラウンドで出した金額ー1を賭ける.
  - 最終ラウンドは残り金額を賭ける.

```
int main(int argc,char **argv){
 int n,i,j;
 int num_game;

 num_game=atoi(argv[1]);

 for(i=1;i<=num_game;++i){
  int bet;
  int sum=0;
  printf("0¥n"); //1
  fflush(stdout);
  scanf("%d",&n);

  for(i=2;i<=9;++i){
   bet=(n>0?n-1:0);
   printf("%d¥n",bet); //2-9
   fflush(stdout);
   sum+=bet;
   scanf("%d",&n);
  }

  bet=10000-sum;
  printf("%d¥n",bet); //10
  fflush(stdout);
  sum+=bet;
  scanf("%d",&n);
 }

 exit(EXIT_SUCCESS);
}
```

プレイヤープログラムの例3

- 戦略random
  - 各ラウンドは，[0,残り所持金]の一様乱数の値を賭ける.
  - 最終ラウンドは，残り金額を全て賭ける.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc,char **argv){
 int n,i;
 int num_game;

 num_game=atoi(argv[1]);

 srand(time(NULL));

 for(j=1;j<=num_game;++j){
  int bet;
  int remain=10000;

  for(i=1;i<=9;++i){
   bet=random()%remain;
   printf("%d¥n",bet); //1-9
   fflush(stdout);
   remain-=bet;
   scanf("%d",&n);
  }

  bet=remain;
  printf("%d¥n",bet); //10
  fflush(stdout);
  scanf("%d",&n);
 }

 exit(EXIT_SUCCESS);
}
```

対戦例1：always1000 vs minus1

```
------------------------------------
game 1
 1:  1000 [   0] ->     0  1000
 2:  1000 [ 999] ->     0  2999
 3:  1000 [ 999] ->     0  4998
 4:  1000 [ 999] ->     0  6997
 5:  1000 [ 999] ->     0  8996
 6:  1000 [ 999] ->     0 10995
 7:  1000 [ 999] ->     0 12994
 8:  1000 [ 999] ->     0 14993
 9:  1000 [ 999] ->     0 16992
10: [1000]  2008  ->  3008 16992
    3008  [16992]
minus1 won.
------------------------------------
```

```
------------------------------------
game 1000
 1:  1000 [   0] ->     0  1000
 2:  1000 [ 999] ->     0  2999
 3:  1000 [ 999] ->     0  4998
 4:  1000 [ 999] ->     0  6997
 5:  1000 [ 999] ->     0  8996
 6:  1000 [ 999] ->     0 10995
 7:  1000 [ 999] ->     0 12994
 8:  1000 [ 999] ->     0 14993
 9:  1000 [ 999] ->     0 16992
10: [1000]  2008  ->  3008 16992
    3008  [16992]
minus1 won.
------------------------------------
always1000 won : 0, minus1 won : 1000, Draw : 0
```

minus1の全勝

対戦例2:always1000 vs random

```
------------------------------------
game 1
 1: [1000]  4910  ->  5910     0
 2: [1000]  2601  ->  9511     0
 3: [1000]  2211  -> 12722     0
 4:  1000  [ 115] -> 12722  1115
 5:  1000  [  88] -> 12722  2203
 6:  1000  [  14] -> 12722  3217
 7:  1000  [   1] -> 12722  4218
 8:  1000  [  12] -> 12722  5230
 9:  1000  [  11] -> 12722  6241
10:  1000  [  37] -> 12722  7278
  [12722]   7278
always1000 won.
------------------------------------
```

```
------------------------------------
game 1000
 1: [1000]  5593  ->  6593     0
 2: [1000]  2541  -> 10134     0
 3: [1000]  1743  -> 12877     0
 4:  1000  [  19] -> 12877  1019
 5:  1000  [  77] -> 12877  2096
 6:  1000  [  17] -> 12877  3113
 7:  1000  [   3] -> 12877  4116
 8:  1000  [   4] -> 12877  5120
 9:  1000  [   2] -> 12877  6122
10:  1000  [   1] -> 12877  7123
  [12877]   7123
always1000 won.
------------------------------------
always1000 won : 863, random won : 137, Draw : 0
```

always1000の863勝137敗

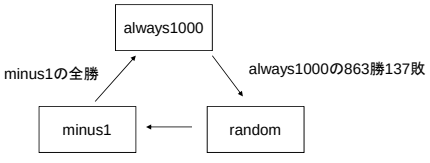
対戦例3:random vs minus1

```
------------------------------------
game 1
 1:  8350 [   0] ->     0  8350
 2: [1335]  8349  ->  9684  8350
 3: [  80]  1334  -> 11098  8350
 4: [  29]    79  -> 11206  8350
 5:    86  [  28] -> 11206  8464
 6: [  52]    85  -> 11343  8464
 7: [  13]    51  -> 11407  8464
 8:    31  [  12] -> 11407  8507
 9: [  19]    30  -> 11456  8507
10: [   5]    32  -> 11493  8507
  [11493]   8507
random won.
------------------------------------
```

```
------------------------------------
game 1000
 1:  4744 [   0] ->     0  4744
 2: [3916]  4743  ->  8659  4744
 3: [ 600]  3915  -> 13174  4744
 4: [ 515]   599  -> 14288  4744
 5: [  41]   514  -> 14843  4744
 6:   146  [  40] -> 14843  4930
 7: [  14]   145  -> 15002  4930
 8:    20  [  13] -> 15002  4963
 9: [   1]    19  -> 15022  4963
10: [   3]    12  -> 15037  4963
  [15037]   4963
random won.
------------------------------------
random won : 714, minus1 won : 286, Draw : 0
```

randomの714勝286敗

3つの対戦プログラムの関係

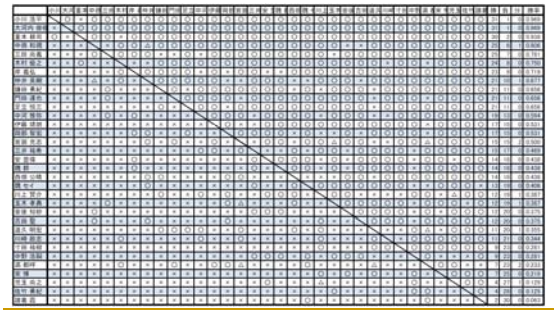


randomの714勝286敗

研究室でコンテスト

- 研究室全員参加の総当り(リーグ戦)
- 1対1の対戦(1000回勝負)で勝ち数の多い方が勝ち！

コンテスト結果



10000円ゲームコンテストの特徴

- プログラミングの初心者や苦手な人も参加することができるので教育向き
- かなり熱中した学生がいた.
- 10000円ゲーム単純だが奥が深い.
  - いろいろな戦略:なるべく相手より少しだけ少ない金額を出して勝ちを稼ぐ.
  - (予想)最強プログラムは存在しない. あるプログラムAが最強だとしたら, そのソースコードを修正して, プログラムAに勝つプログラムBを必ず作ることができる.

1万円ゲーム優勝プログラム

広島大学
小川浩平

概要

- 様々なプログラムの紹介
- 優勝プログラムの紹介

様々なプログラムの紹介

1万円ゲームプログラミング

- プログラムの紹介
  - 定数を出すプログラム
  - 乱数を利用するプログラム
  - 相手の出した金額を利用するプログラム
  - パターン解析を行うプログラム
  - 複数の戦略からひとつを選んで実行するプログラム

定数を出すプログラム

- 決められた金額を常に出すプログラム
  - 常に1000を出すプログラム

1試合
1: 1000
2: 1000
3: 1000
4: 1000
5: 1000
6: 1000
7: 1000
8: 1000
9: 1000
10: 1000

⇨

2試合
1: 1000
2: 1000
3: 1000
4: 1000
5: 1000
6: 1000
7: 1000
8: 1000
9: 1000
10: 1000

⇨

3試合
1: 1000
2: 1000
3: 1000
4: 1000
5: 1000
6: 1000
7: 1000
8: 1000
9: 1000
10: 1000

……

毎ラウンド1000を出すプログラム

- 毎ラウンド1000を出すプログラムに対して
  - 1000に対して少ない金額をだせば勝てる
  - 999を出すのが最善の手

相手の金額	自分の金額	獲得金額
1000	0	1000
	500	1500
	999	1999

1: 1000 [999] → 0 1999
2: 1000 [999] → 0 3998
3: 1000 [999] → 0 5997
4: 1000 [999] → 0 7996
5: 1000 [999] → 0 9995
6: 1000 [999] → 0 11994
7: 1000 [999] → 0 13993
8: 1000 [999] → 0 15992
9: 1000 [999] → 0 17991
10: [1000] 1009 → 2009 17991
2009 [17991]

- 毎試合同じ金額を出すのではなく乱数を使うプログラム

乱数を利用するプログラム

- ランダムに金額を出す
  - それぞれのラウンドでランダムな2000以下の金額を出すプログラム
    （10ラウンド目は残りのすべての金額を出す）

1試合	2試合	3試合
1: 139	1: 918	1: 666
2: 632	2: 1197	2: 88
3: 1389	3: 1127	3: 1074
4: 1128 ⇒	4: 718 ⇒	4: 914 .....
5: 1903	5: 584	5: 245
6: 938	6: 1500	6: 322
7: 607	7: 1590	7: 123
8: 621	8: 1263	8: 903
9: 1522	9: 0	9: 795
10: 1121	10: 1103	10: 4870

乱数を利用するプログラム

- 試合毎に出す金額が変わるので、同じ金額を出し続けるプログラムより対処が難しい
- このプログラムに勝つにはどんなプログラムがいいか？
  - 1ラウンド目に大きな金額を出すプログラム

1ラウンド目に大きな金額を出すプログラム

- 1ラウンド目に7000以上の大きな金額を出すプログラム
  - 1ラウンド目は負けてしまうが自分が出す残り金額が少なくなるので以降のラウンドで勝つことができる
  - 1ラウンド目で、試合の負けが確定してしまう可能性がある

```
1:  7000  [ 931] ->     0  7931
2: [ 259]   947  ->  1206  7931
3: [ 244]   945  ->  2395  7931
4: [ 337]   933  ->  3665  7931
5: [ 489]  1557  ->  5711  7931
6: [ 273]   940  ->  6924  7931
7: [ 113]   940  ->  7977  7931
8: [ 315]   933  ->  9225  7931
9: [ 498]   936  -> 10659  7931
10: [ 472]   938  -> 12069  7931
 [12069]   7931
```

```
1:  8456  [1897] ->     0 10353
2: [ 100]  1043  ->  1143 10353
3: [  15]   247  ->  1405 10353
4: [ 204]  1826  ->  3435 10353
5: [  21]  1884  ->  5340 10353
6: [ 123]   610  ->  6073 10353
7: [ 320]  1530  ->  7923 10353
8: [ 311]   340  ->  8573 10353
9: [ 341]   417  ->  9332 10353
10: [ 109]   206  ->  9647 10353
    9647  [10353]
```

1ラウンド目に大きな金額を出すプログラム

- 1ラウンド目で試合での負けが即確定する条件

自分が出す金額	相手が出した時 試合での自分の負けが確定する金額
7000	3001～6999
8000	2001～7999
9000	1001～8999

もし1ラウンド目に2000以下の金額しか出さないことが分かっていれば7000～7999までの金額をだせば試合での負けは確定しない

1ラウンド目に大きな金額を出すプログラム

- 最初のラウンドで7000以上8000未満の大きい金額を出し、他のラウンドは1000以下の小さな金額を出すプログラム

1試合	2試合	3試合
1: *7258*	1: *7900*	1: *7546*
2: 589	2: 289	2: 435
3: 80	3: 261	3: 163
4: 416 ⇒	4: 740 ⇒	4: 74 .....
5: 16	5: 24	5: 239
6: 601	6: 131	6: 172
7: 144	7: 193	7: 558
8: 221	8: 122	8: 574
9: 398	9: 147	9: 69
10: 277	10: 193	10: 170

大きい金額を出すプログラム

- 実行例
  2000以下の金額を出すプログラムに対して
  - 1ラウンド目では負けてしまうが、試合での負けは確定しない
  - それ以降のラウンドでほぼ勝つことができる
    - 最終的に勝利できる

```
1: 7870  [ 660]  ->     0   8530
2: [  240] 1396  ->  1686 8530
3: [ 121]  1896  ->  3653 8530
4: [ 234]   796  ->  4683 8530
5: [ 427]  1130  ->  6240 8530
6: [ 200]   541  ->  6981 8530
7: [ 228]  1336  ->  8545 8530
8:  344  [ 164] ->  8545  9038
9: [  11]   499  ->  9055  9038
10: [ 325]  1582  -> 10962  9038
 [10962]  9038
```

大きい金額を出すプログラム

- 最初に大きな金額を出すようなプログラムに勝つにはどのようなプログラムがよいか？
  - 相手の大きい金額に対応して、それよりも少し少ない金額を出すプログラム

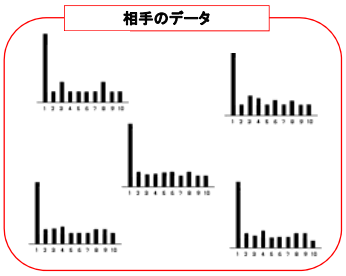
相手の出した金額を利用するプログラム

- 前の試合で出した相手の金額を利用するプログラム
  - 前の試合の相手の出した金額を基にし
    1～9ラウンドまでは少し少ない金額
    10ラウンド目に残りの金額を出すプログラム

前の試合の相手		自分				
1: 7260	-90	1: [7170]	**7260**	->	14430	0
2: 590	-60	2: [530]	**590**	->	15550	0
3: 90	-45	3: [45]	**90**	->	15685	0
4: 400	-10	4: [390]	**400**	->	16475	0
5: 20	-10	5: [10]	**20**	->	16505	0
6: 700	-30	6: [670]	**700**	->	17875	0
7: 60	-15	7: [45]	**60**	->	17980	0
8: 200	-30	8: [170]	**200**	->	18350	0
9: 400	-50	9: [350]	**400**	->	19100	0
10: 280	+340	10: 620	**[280]**	->	19100	900
		[19100] 900				

パターン解析を行うプログラム

- 過去の相手の出したデータを蓄積しそれを基にして自分の出す金額を決定するプログラム



自分が出す金額

1: 6000
2: 600
3: 300
4: 1100
5: 500
6: 100
7: 100
8: 1000
9: 600
10: 100

パターン解析を行うプログラム

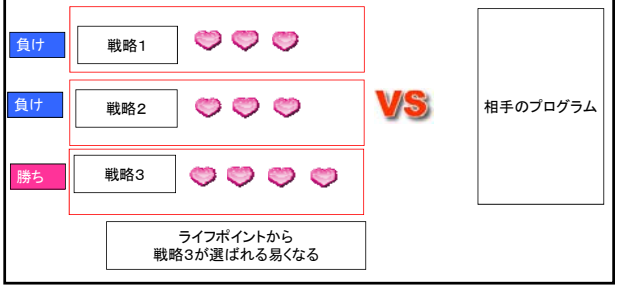
- 相手の金額を利用、パターン解析を行う
  - 同じような傾向で金額を出すプログラムには勝つことができる
- 同じような傾向を持たないようにするにはどうすればよいか？

複数の戦略からひとつを選んで実行するプログラム

- 複数の戦略からひとつを選んで実行する
  - プログラムに傾向の違う複数の戦略を実装する
  - 各々の戦略にその戦略を使う優先度を表す、ライフポイントを設定する
  - 負けるとライフポイントを減らし、勝ったときに増やすようにする
  - そのライフポイントが大きい戦略から優先的に選んでいく
    - 勝ち易いプログラムが選ばれやすくする
    - 相手に傾向を読まれにくくなる

複数の戦略からひとつを選んで実行するプログラム

- 例



優勝プログラム

優勝プログラム

- 相手の勝ちまでの残り・自分の勝ちまでの残りを考慮したプログラム
- 対戦中の状況を逐一蓄える
  - 試合の状況に合わせて出す金額を変えていく

優勝プログラム

- 出し方を状況に合わせて変えていく

1ラウンド　100～1000
2ラウンド　相手の出す期待値を基にする出し方
⋮
iラウンド　相手の出す期待値を基にする出し方
(自分の獲得金額が8500以上)
i+1ラウンド 負けない出し方
i+2ラウンド 勝つための出し方
⋮
9ラウンド　勝つための出し方
10ラウンド　残りの金額すべて

優勝プログラム

- 相手の出す期待値を基にする出し方
  - 相手の残り金額を残りラウンド数で割った金額（相手が次にベットすると予想される期待値）を基にして、出す金額を決める

出す金額をbetとし以下のようにbetを定める

A/10<bet<A
(A=(相手の残りの金額)÷[残りのラウンド数])

1: [700] 1000
⇒ 2: [198] [30]
⇒ 3: [1076] 4000
⇒ 4: [175] 1000

A= (相手の残りの金額)÷残りのラウンド数

2でのbet→　100<bet<1000

優勝プログラム

- 出し方を状況に合わせて変えていく

1ラウンド　100～1000
2ラウンド　相手の出す期待値を基にする出し方
⋮
iラウンド　相手の出す期待値を基にする出し方
(自分の獲得金額が8500以上)
i+1ラウンド 負けない出し方
i+2ラウンド 勝つための出し方
⋮
9ラウンド　勝つための出し方
10ラウンド　残りの金額すべて

優勝プログラム

- 負けない出し方
  自分の金額が8500以上になった次のラウンド
  - 自分が有利になったと考え以下のような金額を出す

[相手が9999までに必要な金額]- [相手が出すべき残りの金額]

優勝プログラム

[相手が9999までに必要な金額]- [相手が出すべき残りの金額]

- 相手が9999までに必要な金額から、相手が出すべき残りの金額を引く
  - もし相手が残りの金額を全部出してきても そのラウンドでは必ず相手は9999以下になる
    - そのラウンドで自分の試合の負けは確定しない

優勝プログラム

1: 958 [25]-> 0 983
2: 193 [36]-> 0 1212
3:[1111] 7684-> 8795 1212
4: 6532 [2255]-> 8795 9999

bet= 8788 – 2256 = 6532

- 相手が残りをすべて出してもそのラウンドで試合の負けは確定しない

優勝プログラム

1: 958 [25]-> 0 983
2: 193 [36]-> 0 1212
3:[1111] 7684-> 8795 1212
4: 6532 [185]-> 8795 7929

3ラウンド目

自分が出すべき残り	7738	相手が出すべき残り	2255
自分の獲得金額	8795	相手の獲得金額	1212

4ラウンド目
（負けない出し方を出した後）

自分が出すべき残り	**1206**	相手が出すべき残り	2070
自分の獲得金額	**8795**	相手の獲得金額	7929

自分の獲得金額＞相手の獲得金額
自分の出すべき残り＜相手の出すべき残り

出した後以降の展開が非常に有利になる

優勝プログラム

- 出し方を状況に合わせて変えていく

1ラウンド 100～1000
2ラウンド 相手の出す期待値を基にする出し方
⋮
iラウンド 相手の出す期待値を基にする出し方
(自分の獲得金額が8500以上)
i+1ラウンド 負けない出し方
i+2ラウンド 勝つための出し方
⋮
9ラウンド 勝つための出し方
10ラウンド 残りの金額すべて

優勝プログラム

- 勝つための出し方
  自分の勝利条件を考えだす金額を決定する

bet < [自分の勝ちまでの金額] / 2

1: 322 [242] -> 0 564
2: 929 [286] -> 0 1779
3: 299 [193] -> 0 2271
4: [1111] 7350 -> 8461 2
5: [66] 79 -> 8606 22
6: 5878 [97] -> 8606 8
7: [400] 500 -> 9506 8246
8: [205] 342 -> 10053 8246

700以下の金額を出す
250以下の金額を出す
勝ちまで後約1400
勝ちまで後約500

優勝プログラム

- 相手に自分の出す傾向が読まれにくいようにするため
  - 試合が確定した次のラウンドは残りの金額をすべて出すようにした

1: 322 [242] -> 0 564
2: 929 [286] -> 0 1779
3: 299 [193] -> 0 2271
4: [1111] 7350 -> 8461 2271
5: [66] 79 -> 8606 2271
6: 5878 [97] -> 8606 8246
7: [400] 500 -> 9506 8246
8: [205] 342 -> 10053 8246
9: 790 [171] -> 10053 9207

優勝プログラム

- 定数を出すプログラム、乱数を利用するプログラム
  - その試合の金額を基にして自分が出す金額を変えているので勝つことができる
- 解析を行うプログラム
  - 一試合ごとそれぞれの戦況に合わせて金額を出しているので過去の出し方を利用されても特に影響はなく勝つことができる

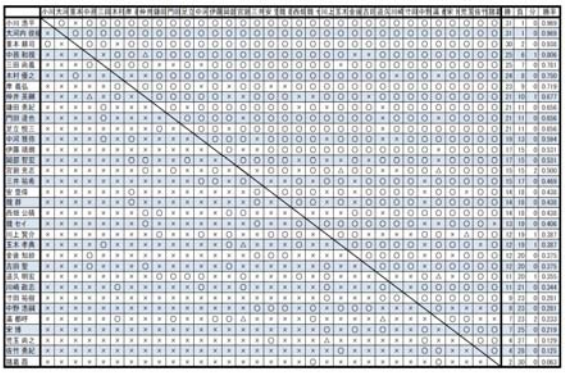
実行例

A: 相手の出す期待値を基にする出し方
B: 負けない出し方
C: 勝つための出し方

```
1:  958 [  25] ->    0  983
2:  193 [  36] ->    0 1212   } A
3: [1111] 7684 -> 8795 1212
4: 6532 [ 185] -> 8795 7929   } B
5: [ 152]  195 -> 9142 7929
6: [ 150]  168 -> 9460 7929
7:  113 [   4] -> 9460 8046
8: [  87]  160 -> 9707 8046   } C
9: [  70]   72 -> 9849 8046
10: [ 634] 1471 -> 11954 8046
 [11954]  8046
```

```
1:  974 [ 134] ->    0 1108
2:  785 [ 168] ->    0 2061
3: [ 608] 7617 -> 8225 2061   } A
4: [  58]  186 -> 8469 2061
5: [ 120]  221 -> 8810 2061   } B
6: 6264 [ 122] -> 8810 8447
7: [  87]  218 -> 9115 8447   } C
8: [ 122]  224 -> 9461 8447
9:   98 [  44] -> 9461 8589
10: [ 884] 1066 -> 11411 8589
 [11411]  8589
```

大きい金額を出した後は自分の出すべき金額が少ないので有利
無駄に大きい金額を出していないので負ける確率が少ない



終わり

学習者が多角的な視点から議論するための助言を計算機が自動的に提供する手法の提案

青木志乃 †，長瀧寛之，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 コンピュータサイエンス専攻
† E-mail：s-aoki@ist.osaka-u.ac.jp

概要 本稿では，協調学習において議論活動を行うメンバに対し，多角的な視点から議論をさせるための助言を与える議論支援システムの提案を行う．本システムは複数グループがチャットを用いた同一テーマの議論を行うことを想定しており，各グループのメンバは議論前に調査資料として WWW から収集したページを提示しつつ議論を行う．本システムの目標は，議論中の発言と提示された資料，グループ間の議論進捗の差異などを基に，複数グループの議論の状況をリアルタイムで把握し，状況に応じた助言を各グループメンバあるいはグループ全体に対して行う機能の実現である．本システムによって，多数の議論グループに対しても多角的な視点から議論をさせることが可能になる．

1 はじめに

協調学習では，教師から与えられたテーマについてグループ間で議論を行う作業が頻繁に発生する．その際に教師は，学習者の議論が停滞したり，学習者が見落としている情報が存在する場合に**助言**を行う [1]．例えば，議論が盛り上がるような話題や，学習者が見落としている視点からの意見を与えることが挙げられる．教師の助言によって，グループに新たな情報がもたらされ，議論が活性化されることが期待できる．しかし，複数グループが同時に議論を行う場合，教師は複数グループの議論内容を同時に把握しなければならず，各グループに対し適切な助言を行うのは困難になる．そこで本研究では，計算機が複数グループの議論内容を把握し適切な助言を与えるシステム (以下**本システム**) を提案する．計算機によって教師の助言を代替させることで，教師の数が少なくても，多数のグループが多角的な視点から議論を行うことが期待できる．

2 計算機による助言方法

2.1 既存の議論支援システム:HAKASE

本システムは既存の議論支援システム HAKASE[2] を拡張することで構築する．HAKASE は,「あるテーマに対する 2 つの異なる主張 A と B のどちらが適切か」という議題でのチャット議論 (**議論フェーズ**) とそれを行うための事前調査 (**調査フェーズ**) を支援する Web アプリケーションである．HAKASE は調査フェーズにおいて，資料収集用のインターフェースを提供し，資料と主張 A,B との関連性を学習者に登録させる．登録された資料は，議論フェーズにおいて議論用のインターフェース上に一覧表示 (図 1) し，かつ議論中の単語を解析することで，現在の議論の流れに近い資料ほど上位に表示するように随時更新を行う．更に，資料へのリンクを発言に付与するボタンをインターフェース上に用意するこ

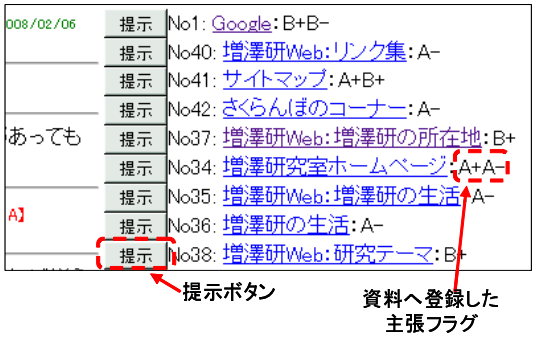


図 1: HAKASE の議論インターフェースにおける資料一覧

とで，学習者は資料の引用を伴った発言が容易にできる．これにより，資料収集の手間の削減と，資料を基にした客観的発言の意識付けを図っている．本研究は，適切なタイミングで適切な資料などを積極的にアドバイスとして送ろうと試みるものである．

2.2 本研究のアプローチ

現状の HAKASE では，議論の流れに適した資料を一覧表示しているが，能動的に資料提示をアドバイスに利用しているわけではない．本研究は，議論の停滞や偏った視点での議論を把握し，適切なタイミングで適切な資料提示を行うなど，より積極的にアドバイザとしての役割を HAKASE に担わせようとするものである．

議論への助言を計算機によって行うためには，議論中の任意の時点で何の話題が取り上げられているかを，計算機が自動で判断する必要がある．しかし発言の文字列や引用資料のみから話題のキーワードを取り出すのは容易ではない．そこで筆者は，[2] において HAKASE の評価実験で行われた議論の発言履歴を解析し，その結果，議論の開始時にはまず主張 A，B を比較検討する基準 (以下**比較基準**) を何にするか学習者間で議論するパターンが多いという事実に着目した．そして，比較基準は，

議論中の小テーマとして機能するため，比較基準を話題のキーワードとして扱うことができると考えた．つまり，議論中に現れる比較基準のキーワードを計算機があらかじめ把握しておけば，議論中の話題の把握が可能となり，把握した話題に基づいた助言のタイミング検出や助言内容の選択も行えるのではないかという着想に至った．

2.3 比較基準の把握方法

議論中の発言履歴だけで比較基準のキーワードを判断するのは容易ではない．そこで，計算機が比較基準のキーワードを議論前に把握できるよう，比較基準を決める学習者のやり取りを独立したフェーズ (以下**基準決定フェーズ**) とし，調査フェーズの前に基準決定フェーズを置く事とした．基準決定フェーズでは，学習者に，グループ内で発言のやりとりを経て，議論の際に取り扱う比較基準のキーワードを明示的に登録させる．また，調査フェーズにおいて，学習者が資料を登録する際に，資料と関連のある比較基準のキーワードを選択させる．登録した比較基準のキーワードと資料により，本システムは議論中の発言と予め把握した比較基準キーワードとの関連性を判定する事で，現在何の比較基準をもとに議論が行われているかを判断する．

2.4 話題の把握方法

助言を行うために，本システムは議論フェーズにおいてどんな話題が話されているかを把握する必要がある．本研究では，システムは基準決定フェーズにおいて登録された比較基準のキーワードを話題の候補とし，議論フェーズにおける発言等から現在どの比較基準が議論されているかを判定する事で話題の把握を行う．比較基準のキーワードを HAKASE が判定する方法として以下の 3 つの手法を提案する．

- 比較基準のキーワードを含む発言から判断
- 発言に引用された資料から判断
- 特定の資料内のテキストに多く存在する単語を含む発言から判断

発言に比較基準のキーワードが含まれている場合，本システムは含まれた比較基準のキーワードについて議論していると判断する．また，発言に資料が引用された場合，システムは，学習者が資料を登録した際に選択した関連する比較基準のキーワードについて議論されていると判断する．同様に，特定の資料内のテキストに多く存在する単語を含む発言が行われた場合，本システムは該当する単語を含む資料と関連する比較基準のキーワードについて議論されていると判断する．

2.5 助言のタイミング

2.4 章で把握した話題に基づき，本システムは新たな話題や新たな視点に関する情報を助言として与える．本研究では，助言に適切なタイミングとして,「話題が変化した時」と「議論が停滞した時」を候補として考えている．

話題が変化した時は，グループ内で直前まで議論されていた話題の情報が出尽くして収束したか，話題に区切りを付けて次の話題に移ることが予想されるため，このタイミングで直前まで議論されていた話題に関する新たな情報を与えることで，対象となる話題に対していっそう深い議論を展開していく事が期待される．タイミングの検出方法として本システムは，2.4 章の手法によって判断した比較基準が，直前に判断した比較基準と異なっていた場合に話題が変化したと判定し，変化する前の話題について助言を行う．

議論が停滞した時はグループ内で議論が行き詰っていると予想されるため，このタイミングで議論中の話題に関する新たな情報を与える事で議論の行き詰まりを解消する事が期待される．タイミングの検出方法として，本システムはグループ内で一定時間発言が行われなかった場合に議論が停滞していると判定し，議論中の話題について助言を行う．

2.6 提示する助言内容の選択

本研究は，助言対象となるグループ (以下**助言グループ**) 以外の他グループで引用された「資料」と「発言」を助言として与えることを考える．なぜなら，他グループにおいて引用された資料や発言は，助言グループのメンバが知らない情報が含まれている事があるからである．よって，本システムは全グループが引用した資料を記録し，2.5 章のタイミングにあわせて助言グループに必要な資料を助言として与える．ただし，他グループで引用された資料であっても，助言グループのメンバが引用した資料であれば助言グループにとって既知であるとみなし助言候補から外す．また，資料が引用された周辺の発言は，資料に関する情報が記載してある確率が高いので，本システムは資料が引用された周辺の発言も資料と共に助言として与える．

他グループで引用された資料のうち，助言グループが助言のタイミングの直前で議論していた比較基準に関連した資料がある場合，本システムは話題に適した資料であると判断し，その資料と引用時の

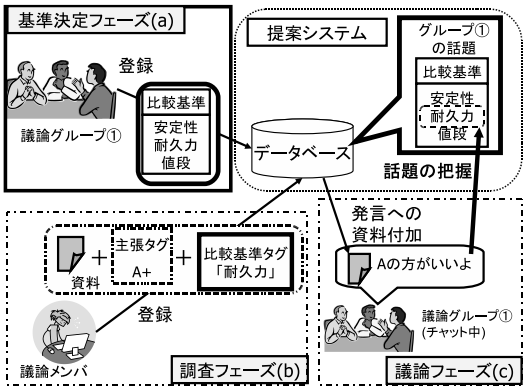


図 2: 提案システム (太枠内が本研究での拡張部分)

周辺発言を優先して助言に用いる．また，助言のタイミングにおいて助言グループのメンバが支持する主張が偏っていた場合，本システムは主張の偏りを無くすために少数派の主張を支持する内容の資料と引用時の周辺発言を優先して助言に用いる．

助言グループが助言タイミングの直前で議論していた比較基準に関する資料が存在しない場合，本システムは助言グループが助言のタイミング以前に議論した比較基準に関する資料と引用時の周辺発言を与える．これは，一度議論した話題に関して他グループが遅れて議論を行った際に，助言グループにとって新しい視点となる資料が引用される可能性があるためである．

助言グループが議論してきた比較基準に関する資料が存在しない場合，本システムは他グループの話題ごとの発言数を比較し，発言が多い話題を議論が盛り上がるような話題と判断し，その話題で引用された資料と周辺発言を助言として与える．

助言として資料と周辺発言を与える際，話題の参考のために資料に関連する比較基準のキーワードも補足情報として与える．

3 提案システムの概要

本システムの概要を図 2 に示す．基準決定フェーズ (a) おいて本システムは，学習者グループの定めた比較基準を登録させ，調査フェーズ (b) にて資料登録の際に内容に関連ある比較基準を学習者に選択させる．議論フェーズ (c) において，学習者グループがチャットで議論をしている際にシステムは 2.4 章の方法を用いて発言から話題を取得し，2.5 章のタイミングを検出した際に 2.6 章で述べた助言をグループに与える．

4 まとめ

本稿では，協調学習における学習者同士の議論を多角的に行うための助言を計算機で行う手法について紹介した．今後は，本システムの構築作業として，HAKASE に本稿で紹介した手法を実現するための機能を追加する必要がある．また，本システムを用いた議論学習を実施することにより，提案手法による助言の効果についての評価を行う予定である．

参考文献

[1] 稲葉 昌子，岡本 敏雄. “協調学習における議論支援戦略”. 電子情報通信学会技術研究報告, Vol.98, No.35, pp77–84, 1998.

[2] 林 昌弘，長瀧 寛之，大下 福仁，角川 裕次，増澤 利光. “議論活動における調査資料の活用を支援するシステム HAKASE の構築”. 情報処理学会研究報告, Vol.2008, No.13, pp.119–126, 2008.

A Tiny Processing System for Education and Small Embedded Systems on the FPGAs

Koji Nakano, Kensuke Kawakami, Koji Shigemoto, Yuki Kamada, Yasuaki Ito
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, JAPAN

Abstract

The main contribution of this paper is to present a simple, scalable, and portable tiny processing system which can be implemented in various FPGAs. Our processing system includes a 16-bit processor, a cross assembler, and a cross compiler. The 16-bit processor runs in 89MHz on the Xilinx Spartan-3A family FPGA XC3S700A using 336 out of 5888 slices (5.7%) and in 76MHz on the Altera Cyclon III family EP3C25F324 using 569 out of 24624 logic elements (2.3%). Every instruction can be executed in only one clock cycle, that is, CPI=1. Using a cross assembler and a cross compiler that we have developed, a C-based language program can be translated into a machine language object code, which can be executed on the 16-bit processor. The source codes of our processing system are very simple and compact. The 16-bit processor is designed by Verilog HDL using 268 lines, and the cross assembler is written in 38 lines using Perl language. The cross compiler has 23 lines of Flex grammar file for lexical analysis, and 90 lines of Bison grammar file for context analysis and code generation. Hence, our tiny processing system is portable and easy to understand and the function expansion is not difficult. Actually, the tiny processing system has been used for the embedded system course of graduate students as a course material. As real-life applications, we have developed a PONG-like mini game and an RSA encryption/decryption system based on the tiny processing system. Therefore, our tiny processing system benefits computer system education and small embedded system development.

1 Introduction

An FPGA (Field Programmable Gate Array) is a programmable VLSI in which a hardware designed by users can be embedded instantly. Typical FPGAs consist of an array of programmable logic blocks (slices or logic elements), memory blocks, and programmable interconnect between them. The logic block usually contains four-input logic functions and/or several registers. The memory block is usually a dual-port RAM which can be read/written a word of data for distinct addresses in the same time. Using design tools provided by FPGA vendors or third party companies, a hardware logic designed by users using hardware description languages can be embedded in FPGAs. Thus, the FPGAs are widely used for platform of embedded system as well as computer system education. Also, it has been shown that a lot of computation can be accelerated using a circuit implemented in FPGAs [4, 5, 11, 12, 13, 14].

The main contribution of this paper is to present a simple, scalable, and portable tiny processing system which can be implemented in various FPGAs. The processing system includes an N-bit processor TINYCPU, a cross assembler TINYASM, and a cross compiler TINYC. Using TINYC and TINYASM, C-based language programs can be translated into a machine language object code, which can be executed in TINYCPU. Table 1 summarizes our tiny processing system. In this table, the code sizes in lines include blank lines.

Our tiny processing system has a lot of advantages as follows:

Simple and compact The source codes of our processing system are very simple and compact. TINYCPU has 268 lines with Verilog HDL, TINYASM is described in 38 lines using Perl language. TINYC has 23 lines of Flex grammar file for lexical analysis, and 90 lines of Bison grammar file for context analysis. Thus, our tiny processing system is easy to understand and the function expansion is very easy.

Low CPI Every instruction of TINYCPU can be executed in only one clock cycle, that is, CPI (Clock Per Instruction)=1. Thus, the performance our 16-bit version of TINYCPU is 89MIPS(Million

Table 1. Our tiny processing system and its code size

		language	module or function	code size (lines)
TINYCPU	N-bit Processor	Verilog HDL	definitions	36
		Verilog HDL	ALU	39
		Verilog HDL	state machine	20
		Verilog HDL	stack	34
		Verilog HDL	memory	29
		Verilog HDL	top module	110
		total		268
TINYASM	Cross Assembler	Perl		38
TINYC	Cross Compiler	Flex	lexical analysis	23
		Bison	context analysis code generation	90
		total		113
total				419

Instructions Per Second) on XC3S700A, and 76MIPS on EP3C25F324.

Minimum usage of block RAMs Altera and Xilinx FPGAs have dual-port block RAMs as building blocks. For example, Altera Cyclon III family FPGA EP3C25F324 and Xilinx Spartan III Family FPGA XC3S700A have 66 9k-bit block RAMs and 20 18k-bit block RAMs, respectively. Read-/Write operations for two distinct addresses can be performed in the same time for a dual-port block RAM. We use this function of dual-port to fetch an instruction code and execute a read/write operation in the same time. Thus, instruction codes and data can be in the same block RAM, and TINYCPU uses only one block RAM as a memory to store both instruction codes and data for the minimum configuration. Also, the size of memory can be extended easily.

Scalable The word size of TINYCPU can be changed. In other words, for arbitrary integer $N \geq 8$, TINYCPU can take an N-bit architecture. Any N-bit architecture can be generated by just setting parameter N in the top Verilog HDL module of our source code.

Portable Our Verilog HDL code for TINYCPU follows Verilog-95 standard, and thus it can be synthesized by tools supporting Verilog-95 as well as Verilog 2001. Therefore, most of logic design tools can synthesize our Verilog HDL code for the processor and generate its net list. Actually, Xilinx ISE Foundation, Altera Quartus II, and third party logic synthesis tool Synplify Pro can synthesize it, and Icarus Verilog [9] can perform the simulation correctly. Also, Perl, Flex, and Bison tools that we have used are standard UNIX tools. They can also be executed on Microsoft Windows-based PCs using Cygwin [7], which provides a Linux-like environment on them.

Benchmark TINYCPU has a lot of fundamental logic components including, registers, counters, memories, combinational circuits for arithmetic and logic operations, state machines, multiplexers, tri-state buses, etc. The arithmetic and logic operations include addition, subtraction, multiplication, negation, shift, comparison, and so on. Further, since the Verilog HDL code for TINYCPU is portable, it can be a standard benchmark to measure the performance of FPGAs and logic design tools. Therefore, TINYCPU can be a standard benchmark circuit design to measure the goodness of logic design and synthesis tools as well as the performance of FPGAs.

Education Since our processing system including a processor, a cross assembler, and a cross compiler is simple, they are easy to understand for students. Thus it can be used as course materials for learning basics of computer and embedded systems from various aspects.

Since our main target is to design a processor, a cross assembler, and a cross compiler as simple as possible, The efficiency including used hardware resources and the clock frequency is of secondary importance. Nevertheless, the performance of our TINYCPU is 89MIPS for XC3S700A and 76MIPS for EP3C25F324.

Several processors designed for embedding in FPGAs have been presented. Xilinx Inc. offers 8-bit CPU, Picoblaze [23] and 32-bit CPU Microblaze [25]. Altera Corp. provides 32-bit Nios II processor [3]. They are optimized for their FPGAs and porting to the other FPGAs is not possible. Embedded processors for executing Java Virtual Machine codes on the FPGA have been presented [16, 18], but they are too complicated.

We have also developed a PONG-like mini game [17] and an RSA encryption/decryption system using our tiny processing system. The RSA encryption/decryption system was submitted to the Design Wave Magazine Design Contest 2008 and won vice-champion [6].

2 The Architecture of TINYCPU

TINYCPU is a scalable processor that we have developed. This processor is a pure stack architecture [10] and does not have an accumulator or a register set. Instead, it has an operation stack, which is used for all operations including store/load operations and arithmetic and logic operations. In the standard configuration, the word size of TINYCPU is 16 bits. In other words, the width of the data bus and the stack is 16 bits. As we will explain later, the word size can be changed to any integer $N \geq 8$.

TINYCPU is designed using Verilog HDL. The Verilog HDL code is written as simple as possible. Figure 1 illustrate the block diagram of TINYCPU. It has six components including state machine `state`, 12-bit program counter `pc`, 16-bit output buffer `obuf`, 16-bit ALU `alu`, 16-bit stack `stack`, 16-bit data and 12-bit address memory `ram`. It also uses 16-bit data bus `dbus` and 12-bit address bus `abus`.

Every instruction of TINYCPU is a 16-bit word. Table 2 shows the list of all instructions of TINYCPU. It has 11 control instructions and 19 instructions for arithmetic and logic operations. In the table, operands `I` and `A` show immediate and address values, respectively, and `f` is a 5-bit code to specify an operation. Also, `top` and `next` denote the top and the second elements of the stack. Hence, the binary operations are performed for `next` and `top` and the resulting value is stored in `top`. The unary operations are performed for `top`. The readers may think that TINYCPU has too few instructions. However, these instructions are sufficient to execute machine codes generated by C-based language programs that we will explain later.

Quite surprisingly, every instruction in Table 2 can be performed in one clock cycle. The function of a dual-port block RAM is used to fetch an instruction and read/write data in the same time. Thus, during the execution of an instruction, TINYCPU fetches next instruction.

We will show how TINYCPU is designed by the Verilog HDL using an example. List 1 shows the Verilog HDL source code `alu.v` for ALU (Arithmetic and Logic Unit). In the first line, Verilog HDL source code `defs.v`, which has miscellaneous mappings of binary constant numbers to instruction codes and states, is included. The word size N can be changed by specifying the parameter value. This module has N-bit integers `a` and `b` as inputs and N-bit integer `s` as output. It also has 5-bit input `f` which is used to select a function out of 19 functions. We assume that `a`, `b` and `s` are signed and 2's complements if they are treated as integers.

List 1. The Verilog HDL source code alu.v for ALU (Arithmetic and Logic Unit)

```
1  `include "defs.v"
2
3  module alu(a, b, f, s);
4  parameter N = 16;
5
6    input [N-1:0] a, b;
7    input [4:0]   f;
8    output [N-1:0] s;
9    reg [N-1:0] s;
10   wire [N-1:0] x,y;
11
12   assign x = {~a[N-1],a[N-2:0]};
13   assign y = {~b[N-1],b[N-2:0]};
14
15   always @(a or b or x or y or f)
16     case(f)
17       `ADD : s = b + a;
18       `SUB : s = b - a;
19       `MUL : s = b * a;
20       `SHL : s = b << a;
21       `SHR : s = b >> a;
22       `BAND: s = b & a;
23       `BOR : s = b | a;
24       `BXOR: s = b ^ a;
25       `AND : s = b && a;
26       `OR : s = b || a;
27       `EQ : s = b == a;
28       `NE : s = b != a;
29       `GE : s = y >= x;
30       `LE : s = y <= x;
31       `GT : s = y > x;
32       `LT : s = y < x;
33       `NEG : s = -a;
34       `BNOT : s = ~a;
35       `NOT : s = !a;
36       default : s = {N{1'bx}};
37     endcase
38
39 endmodule
```

We use two nets `x` and `y` for the technical reasons to follow Verilog-95 standard. In Verilog-95, a bit vector are treated as an unsigned integer. Thus, we use two nets `x` and `y` to obtain correct results of comparisons

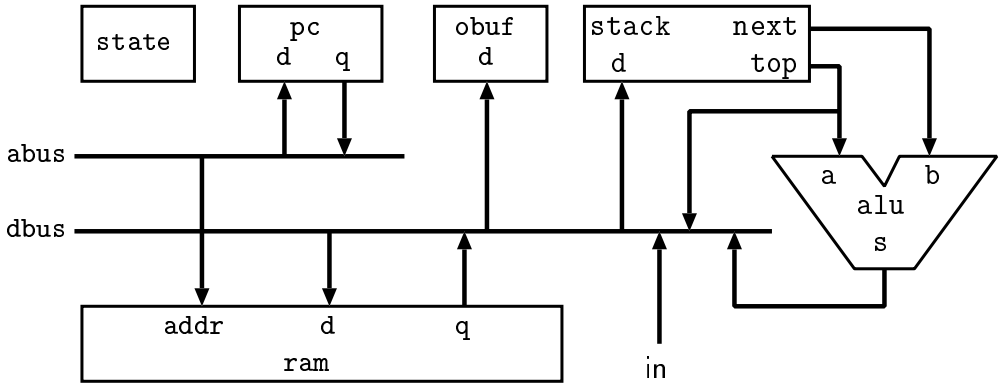


Figure 1. TINYCPU architecture

Table 2. Instruction set of TINYCPU: Mnemonic names and instruction codes

	Mnemonic	Machine Code (HEX)	Operation
1	HALT	0000	Stop
2	PUSHI I	1000+I	I→ top
3	PUSH A	2000+A	mem[A]→ top
4	POP A	3000+A	top → mem[A]
5	JMP A	4000+A	A→pc
6	JZ A	5000+A	A→pc if top=0
7	JNZ A	6000+A	A→pc if top!=0
8	LD	7000	mem[top]→top
9	ST	8000	top→ mem[next]
10	IN	D000	in→top
11	OUT	E000	top→out
12	OP f	F000+f	Perform operation f
	ADD	F000	next + top → top
	SUB	F001	next - top → top
	MUL	F002	next * top → top
	SHL	F003	next >> top → top
	SHR	F004	next << top → top
	BAND	F005	next & top → top
	BOR	F006	next | top → top
	BXOR	F007	next ^ top → top
	AND	F008	next && top → top
	OR	F009	next || top → top
	EQ	F00A	next == top → top
	NE	F00B	next != top → top
	GE	F00C	next >= top → top
	LE	F00D	next <= top → top
	GT	F00E	next > top → top
	LT	F00F	next < top → top
	NEG	F010	- top → top
	BNOT	F011	~ top → top
	NOT	F012	! top → top

>=, <=, >, and < for signed integers a and b. It should be clear that x = a +2^{15} and y = b +2^{15} hold for 16-bit signed integers a and b and 16-bit unsigned integers x and y. Also, the results of comparisons for unsigned integers x and y is equal to those for signed integers a and b. If we used Verilog 2001 [20] or later, which supports signed integers, we did not have to use two nets x and y. However, since we want to design TINYCPU by Verilog-95 [19] for portability, we use such technical conversion technique. Actually, some logic design, synthesis, and simulation tools give wrong results for comparison of signed integers. For example, Icarus Verilog [9] does not return correct results for comparisons of "signed" nets.

3 Cross Assembler and Cross Compiler

We have designed a cross assembler and a cross compiler for TINYCPU. The Assembler, TINYASM, translates an assembly language program into a machine code, which is a list of pairs of a 12-bit address and a 16-bit instruction. The compiler, TINYC, translates a C-based language program into an assembly language program for TINYASM. TINYC language supports 16-bit signed integers and its 1-dimensional array, and if, if-else, while, do, and goto statements. Also, it has basic arithmetic and logic operations including addition (+), subtraction (−), multiplication (∗), negation (−), bit shifts (<<, >>), bitwise logic operations (&, |, ^, ~), logic operations (&&, ||, !), and comparisons (==, ! =, >, >=, <, <=).

List 2 shows an example of TINYC language program collatz.c. Using TINYC compiler and TINYASM assembler, collatz.c is translated into a TINYASM assembly language program and a TINYCPU machine code in List 3. The C-language program in List 2 computes the formula in Collatz conjecture [1, 22] as follows. Consider the following operation on an arbitrary positive integer n:

- If the number is odd, triple it and add one, that is, $n \leftarrow 3n + 1$.
- If the number is even, divide it by two, that is, $n \leftarrow n/2$.

The Collatz conjecture asks if iterating this operation returns 1 for any initial value n. For example, if $n = 3$ then, we have the following sequence by iterating the operation.

$$3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

It remains open if the Collatz conjecture is true. The readers should have no difficulty to confirm that TINYC program collatz.c correctly repeats the operation of Collatz conjecture until n becomes 1.

List 3 shows the TINYASM assembly language program and TINYC machine code obtained using our TINYC compiler and TINYASM assembler. It contains a list of labels with their address values, and a machine code, which is a list of 16-bit instructions and initial values of variables.

List 2. C-based language program collatz.c for Collatz conjecture

```
n=in;
while(n>1){
  out(n);
  if(n&1){
    n= n*3+1;
  } else {
    n = n>>1;
  }
 }
out(n);
halt;
int n;
```

Surprisingly, the source programs for TINYASM and TINYC are very simple and compact. TINYASM that we have developed has 38 lines using Perl language [21]. We have developed compiler TINYC using the lexical scanner generating tool Flex [15] and the parser generating tool Bison [8]. The source code of TINYC consists of two files: a Flex grammar file that defines how an input C-based language program is converted into a sequence of tokens, and a Bison grammar file that defines how the token sequence are parsed and machine code are generated. Our Flex grammar file has 23 lines, and Bison grammar file has 90 lines (Figure 1).

List 3. The translated assembly language program and machine program of collatz.c for Collatz conjecture

```
*** LABEL LIST ***
_001F 018
_001T 002
_002F 013
_002T 017
n 01B

*** MACHINE PROGRAM ***
000:D000              IN
001:301B              POP n
          _001T:
002:201B              PUSH n
003:1001              PUSHI 1
004:F00E              GT
005:5018              JZ _001F
006:201B              PUSH n
```

```
007:E000                OUT
008:201B                PUSH n
009:1001                PUSHI 1
00A:F005                BAND
00B:5013                JZ _002F
00C:201B                PUSH n
00D:1003                PUSHI 3
00E:F002                MUL
00F:1001                PUSHI 1
010:F000                ADD
011:301B                POP n
012:4017                JMP _002T
            _002F:
013:201B                PUSH n
014:1001                PUSHI 1
015:F004                SHR
016:301B                POP n
            _002T:
017:4002                JMP _001T
            _001F:
018:201B                PUSH n
019:E000                OUT
01A:0000                HALT
01B:0000        n: 0
```

4 Scalability of TINYCPU and the Performance Evaluation

The word size of TINYCPU is 16 bits width in its standard configuration, and the data bus and the stack has 16-bit width. The word size of our TINYCPU can be N bits for any integer $N \geq 8$. Users can change the word size by specifying the value of parameter N in the top module of Verilog HDL for TINYCPU.

Table 3 shows the performance of TINYCPU for each word size N. It includes clock frequency, used slices, and 18-bit multipliers in the Xilinx Sparta-3A family FPGA XC3S700A-5, which has 5888 slices and 20 18-bit multipliers. It also shows clock frequency, used logic elements, and 9-bit multipliers in the Altera Cyclon III family FPGA EP3C25F324C6, which has 24624 logic cells and 132 9-bit multipliers. We have used ISE Foundation 9.2i for XC3S700A, and Quartus II 7.2 for EP3C25F324 for logic synthesis and timing analysis. XC3S700A is used in Spartan-3A starter kit [28], which is sold for 189USD, and EP3C25F324 is used in Cyclon III FPGA starter kit [2], which is sold for 199USD. So these FPGAs are in the same price range. We can see that the clock frequencies of these FPGA devises are almost the same. Further, we can compare the hardware resources for these two FPGAs. *A slice* of Sparta-3A family FPGA has two four-input LUTs (Look Up Table), two storage elements, and two multiplexers. *A logic element* of Cyclon III family FPGA contains a four-input LUT, a programmable register, a carry chain connection, and a register chain connection. Thus, a slice of Spartan-3A family FPGA has larger capacity than a logic element of Cyclon III family FPGA. From Table 3, we can confirm that the number of used slices is 60-90% of the number of used logic elements. TINYCPU is portable, and has a lot of fundamental logic components including, registers, counters, memories, combinational circuits for arithmetic and logic operations, state machines, multiplexers, tri-state buses, etc, it can be used as a benchmark circuit design to measure the goodness of logic design and synthesis tools as well as the performance of FPGAs.

5 Implementation in the FPGA board

We have implemented our TINYCPU system in the FPGA boards, Spartan-3E starter kit [24] (Figure 2) and Spartan-3A starter kit [26]. The Spartan-3E and Spartan-3A starter kits FPGA boards are equipped with Spartan-3E family FPGA XC3S500E [27, 29] and Spartan-3A family FPGA XC3S700A [27, 28], respectively. Both FPGA boards have various switches (slide switches, button switches and rotary switches), LEDs, and LCD. They also has ports for VGA, PS/2, and Ethernet.

Figure 2. Spartan-3E Starter Kit

We have implemented our TINYCPU system in these FPGA board to confirm that it works properly. In our implementation, various values including program counter, data bus, address bus, output buffer, etc, are displayed in the LCD display.

We have also developed VGA display, keyboard, and mouse controllers for the FPGA boards of Spartan-3E and Spartan-3A Starter kits. These controllers are connected to TINYCPU and can be operated by TINY-

Table 3. Performance of scalable TINYCPU for various word sizes.

	Xilinx XC3S700A-5			Altera EP3C25F324C6		
bits	clock (MHz)	slices (out of 5888)	18-bit multiplier (out of 20)	clock (MHz)	logic elements (out of 24624)	9-bit multiplier (out of 132)
8	111	181 (3.0%)	1 (5%)	85	191 (0.8%)	1(0.8%)
16	89	336 (5.7%)	1 (5%)	76	569 (2.3%)	2(1.5%)
24	63	480 (8.2%)	3 (15%)	62	849 (3.4%)	6 (4.5%)
32	60	650 (11%)	3 (15%)	60	1096 (4.5%)	6 (4.5%)
40	52	809 (14%)	6 (30%)	48	1404 (5.7%)	12 (9.1%)
48	50	1002 (17%)	6 (30%)	51	1716 (7.0%)	12 (9.1%)
56	44	1202 (20%)	10 (50%)	47	1995 (8.1%)	20 (15%)
64	43	1394 (24%)	10 (50%)	45	2313(9.4%)	20 (15%)

CPU using input/output instructions. We have developed two applications using our tiny processing system including TINYCPU and controllers. The first application is a PONG-like [17] mini game (Figure 3), which is a two-player computer game based on the sport ping pong. This game uses two mice connected to a PS/2 port through a PS/2 splitter. Each player control a racket in the display using a mouse. The second application is an RSA encryption/decryption system (Figure 3), which is a given task for Design Wave Magazine Design Contest [6]. In our RSA encryption/decryption system, plain text is given using a keyboard connected to the FPGA board. The results of the encryption/decryption are exhibited as plain text and hexadecimal in the display. Our system won vice-champion of the contest [6]. These two actual implementation results prove that our tiny processing system can be used for developing a small embedded system.

6 Course Material for Education

We have used degenerated version of our tiny processing system as course materials for embedded system design class of graduate students. This class is organized in 8 weeks of 5 hours each. We have used Spartan-3E Starter Kit (Figure 2) and Spartan-3AN Starter Kit and students implement the tiny processing system in the FPGA board and confirm it works correctly by operating it. The contents of each of the eight weeks as follows: (1) Full Adders, N-bit Adders and ALU, (2) Flip-Flops, Counters, State Machines, and Stacks, (3) Buses and Memories, (4) TINYCPU design, (5) Assembler TINYASM design using Perl language, (6) Flex and Bison, (7) Compiler TINYC design, and (8) TINYC programming. Also, students are required to extend the function of TINYC, TINYASM, and TINYC.

Using this course material, students can learn digital circuit design using HDL, processor architectures, assembler design, assembly language programming, and compiler design. Thus, our processing system can be used as a good course material to learn basics of computer and embedded system by experiment.

7 Concluding Remarks

We have presented a tiny processing system which can be implemented in various FPGAs. This tiny processing system has a lot of advantages and merits including simplicity, scalability, portability, low CPI, and minimum usage of block RAMs. We have implemented processor of the system in Altera and Xilinx FPGAs and evaluate the performance. Also, two applications, PONG-like mini game and RSA encryption/decryption system implemented in the Xilinx FPGA board. We have used the tiny processing system as a course material of embedded system class for graduate students.

We have a lot of future works that we plan to do. First, we extend instructions of TINYCPU to support signal and image processing. We also plan to embed two or more TINYCPU in a FPGA for parallel computation. Since the 16-bit version of TINYCPU uses only 2.3% logic elements of EP3C24F324, it may be possible to embed 40 TINYCPUs into a single FPGA. Further, developing a tiny operating system for TINYCPU is also an interesting research theme.

References

[1] E. Akin. Why is the 3x+1 problem hard ? *Contemporary Mathematics*, 356:1–20, 2002.

[2] Altera Corp. *Cyclon III FPGA Starter kit User Guide*, 2007.

[3] Altera Corp. *Nios II Processor Reference Handbook*, 2008.

Figure 3. PONG-like mini game and RSA encryption/decryption system

[4] J. L. Bordim, Y. Ito, and K. Nakano. Accelerating the CKY parsing using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):803–810, May 2003.

[5] J. L. Bordim, Y. Ito, and K. Nakano. Instance-specific solutions to accelerate the CKY parsing for large context-free grammars. *International Journal on Foundations of Computer Science*, pages 403–416, 2004.

[6] The results of design wave design contest 2008. *Desgin Wave*, 5:124–125, May 2008.

[7] Cygwin. http://www.cygwin.com/.

[8] C. Donnelly and R. Stallman. *Bison: The YACC-compatible Parser Generator*. Free Software Foundation, 1995.

[9] Icarus verilog. http://icarus.com/eda/verilog/.

[10] P. Koopman. *Stack Computers: the new wave*. Ellis Horwood, 1989.

[11] R. Lin, K. Nakano, S. Olariu, M. C. Pinotti, J. L. Schwing, and A. Y. Zomaya. Scalable hardware-algorithms for binary prefix sums. *IEEE Trans. on Parallel and Distributed Systems*, 11(8):838–850, August 2000.

[12] K. Nakano and E. Takamichi. An image retrieval system using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):811–818, May 2003.

[13] K. Nakano and K. Wada. Integer summing algorithms on reconfigurable meshes. *Theoretical Computer Science*, 197:57–77, 1998.

[14] K. Nakano and Y. Yamagishi. Hardware n choose k counters with applications to the partial exhaustive search. *IEICE Trans. on Information & Systems*, 2005.

[15] G. T. Nicol. *Flex: The Lexical Scanner Generator*. Free Software Foundation, 1993.

[16] C. Pitter and M. Schoeberl. Towards a Java multiprocessor. In *Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems (JTRES 2007)*, pages 144–151, Vienna, Austria, September 2007. ACM Press.

[17] Pong story. http://www.pong-story.com/.

[18] W. Puffitsch and M. Schoeberl. picoJava-II in an fpga. In *Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems (JTRES 2007)*, pages 213–221, Vienna, Austria, September 2007. ACM Press.

[19] D. E. Thomas and P. R. Moorby. *The Verilog Hardware Description Language, Fourth Edition*. Kluwer Academic, 1998.

[20] D. E. Thomas and P. R. Moorby. *The Verilog Hardware Description Language, Fifth Edition*. Kluwer Academic, 2002.

[21] L. Wall, T. Christiansen, and J. Orwant. *Programming Perl*. O'Reilly, 2000.

[22] E. W. Weisstein. Collatz problem. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/CollatzProblem.html.

[23] Xilix Inc. *Picoblaze: Embedded Micro Controller User Guide*, 2005.

[24] Xilix Inc. *Spartan-3E Starter Kit Board Users Guide*, 2006.

[25] Xilix Inc. *Microblaze Processor Reference Guide*, 2007.

[26] Xilix Inc. *Spartan-3A/3AN Starter Kit Board Users Guide*, 2007.

[27] Xilix Inc. *Spartan-3 Generation FPGA User Guide*, 2008.

[28] Xilix Inc. *Spartan-3A FPGA Family: Data Sheet*, 2008.

[29] Xilix Inc. *Spartan-3E FPGA Family: Complete Data Sheet*, 2008.

Accelerating Montgomery Modulo Multiplication for Redundant Radix-64k Number System on the FPGA using Dual-Port Block RAMs

Koji Shigemoto, Kensuke Kawakami, Koji Nakano
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, JAPAN

Abstract

The main contribution of this paper is to present hardware algorithms for redundant radix-2^r number system in the FPGA to accelerate Montgomery modulo multiplication with many bits, which have applications in security systems such as RSA encryption and decryption. Quite surprisingly, our hardware algorithm for Montgomery modulo multiplication of two dr-bit numbers can be completed in only $d+1$ clock cycles. Since most FPGAs have 18-bit multipliers and 18k-bit block RAMs, it makes sense to let $r = 16$. Our hardware algorithm for Montgomery modulo multiplication for 256-bit numbers runs only 17 clock cycles using redundant radix-$64k$ (i.e. radix-2^{16}) number system. The experimental results for Xilinx Virtex-II Pro Family FPGA XC2VP100-6 show that the clock frequency of our circuit is independent of d. Further, the hardware algorithm for 1024-bit Montgomery modulo multiplication using the redundant number system is 3 times faster than that using the conventional number system. Also, for 256-bit Montgomery modulo multiplication, our hardware algorithm runs in 0.322μs, while a previously known implementation runs in 1.22μs although our implementation uses less than a half slices.

1 Introduction

An FPGA (Field Programmable Gate Array) is a programmable VLSI in which a hardware designed by users can be embedded instantly. Typical FPGAs consist of an array of programmable logic blocks (slices), memory blocks, and programmable interconnect between them. The logic block contains four-input logic functions implemented by a look up table and/or several registers. Using four-input logic functions, registers, and their interconnects, any combinational circuits and sequential logic can be implemented. The memory block is a dual-port RAM which can perform read and/or written operations for a word of data to two distinct or same addresses in the same time. Usually, the dual-port RAM supports synchronous read and synchronous write. The read and/or write operations are performed at the rising edge of clock if read and/or write enable inputs are high. The dual-port RAM outputs data in a specified address after the rising edge. Similarly, data is written to a specified address at the rising edge of clock if write enable is high. Thus, a clock cycle necessary to perform read/data operations. Using design tools provided by FPGA vendors or third party companies, a hardware logic designed by users using hardware description languages can be embedded in FPGAs. It has been shown that a lot of computation can be accelerated using a circuit implemented in FPGAs [3, 4, 7, 11, 12].

It is well known that the addition of two n-bit numbers can be done using a ripple carry adder with the cascade of n full adders [5]. The ripple carry adder has a carry chain through all the n full adders. Thus, the delay time to complete the addition is proportional to n. The carry look-ahead adder [5, 13] which computes the carry bits using the prefix computation can reduce the depth of the circuit. Although the delay time is $O(\log n)$, its constant factor is large and the circuit is much more complicated than the ripple carry adder. Hence, it is not often to use the carry look-ahead adder for actual implementations. On the other hand, redundant number systems can be used to accelerate addition. Using redundant number systems, we can remove long carry chains in the addition. The readers should refer to [13] (Chapter 3) for comprehensive survey of redundant number systems.

The main contribution of this paper is to present hardware algorithms for redundant radix-2^r number system in the FPGA to speed Montgomery modulo multiplication [10], which have applications in security systems such as RSA encryption and decryption system [14]. Montgomery modulo multiplication is used to speed the modulo multiplication $X \cdot Y \cdot 2^{-R} \bmod M$ for R-bit numbers X, Y, and M. The idea of Montgomery modulo multiplication is not to use direct modulo computation, which is very costly in terms of the computing time and hardware resources. By iterative computation of Montgomery modulo

multiplication, the modulo exponentiation $P^E \bmod M$ can be computed, which is a key operation for RSA encryption and decryption [2].

This paper shows implementations of redundant radix-2^r number system in the FPGA for arithmetic operations and then presents hardware algorithms for Montgomery modulo multiplication. The key feature of our implementation for Montgomery modulo multiplication is to use

- redundant radix-2^r number system for interim results,
- dual-port block RAMs to compute $k \cdot M$ such that $(X \cdot Y + k \cdot M) \bmod 2^r = 0$, and
- 18-bit multipliers in an effective manner.

Since each digit of redundant radix-2^r number system has $r+2$ bits and most FPGAs has 18-bit multiplies as building blocks, it makes sense to let $r = 16$.

Quite surprisingly, our hardware algorithms for Montgomery modulo multiplication runs in $d+1$ clock cycles for d-digit redundant radix-2^r number system which takes integers up to $2^{dr}-1$. For example, our hardware algorithm for 1024-bit Montgomery modulo multiplication runs for $1024/16+1 = 65$ clock cycles. From the experimental results for the Xilinx Virtex II Pro Family FPGA XC2VP100-6. our implementation for 1024-bit Montgomery modulo multiplication runs in 1.23μs. Further, the speed up factors of our hardware algorithm using the redundant number system over those using the conventional number system are 3 for 1024-bit Montgomery modulo multiplication.

Several hardware implementations have been presented for Montgomery modulo multiplication [2, 6, 8, 9]. For example, it was shown in [8] that 256-bit Montgomery modulo multiplication can be done in 93 clock cycles of 76.17 MHz (=1.22μs) using 64 18-bit multipliers and 4663 slices on the Xilinx Virtex II Pro Family FPGA XC2VP125-7. In our implementation, it can be done in 17 clock cycles of 52.86MHz (=0.322μs) using 2054 slices and 16 18-bit multipliers and 8 18k-bit block RAMs. Table 1 summarizes the comparison of the performance for 256-bit Montgomery modulo multiplication. Our implementation is more than five times faster although it uses lesser hardware resources on a lesser grade FPGA. Also, the implementation presented in [1] uses 16 times more slices than ours.

2 Non-Redundant and Redundant Radix-2^r Numbers

In this paper, we use the following notation to represent the consecutive bits in a number. For a number X, let $X[i,j]$ ($i \geq j$) be consecutive bits from i-th to j-th bits, where the least significant bits is 0-th bit. For example, $X[6,2] = 11100$ for $X = 11110000$.

Before defining redundant radix-2^r number, we start with a non-redundant radix-2^r number. *A d-digit non-redundant radix-2^r number* is a sequence X of d r-bit numbers $(X_{d-1}, X_{d-2}, \ldots, X_0)$. The value of X is $\sum_{i=0}^{d-1} X_i \cdot 2^{ir}$ and, it takes an integer up to $\sum_{i=0}^{d-1}(2^r-1)\cdot 2^{ir} = 2^{dr}-1$. Hence, it is also a conventional dr-bit binary number. Also, for any integer X, its d-digit non-redundant radix-2^r number $(X_{d-1}, X_{d-2}, \ldots, X_0)$ is unique.

A d-digit redundant radix-2^r number is a sequence X of d $(r+2)$-bit numbers $(X_{d-1}, X_{d-2}, \ldots, X_0)$. The value of X is $\sum_{i=0}^{d-1} X_i \cdot 2^{ir}$. We call, for each X_i with $r+2$ bits, $X_i[r-1,0]$ and $X_i[r+1,r]$, *principal bits* and *redundant bits*, respectively. For example, $X = (\underline{00}0101, \underline{01}0011, \underline{11}1111, \underline{10}1111)$ is a 4-digit redundant radix-2^4 number, where underlined binary numbers are redundant bits. The value of X can computed as follows:

	<u>00</u>	<u>01</u>	<u>11</u>	<u>10</u>	
+		0101	0011	1111	1111
	00	0110	0111	0001	1111

Clearly, for any integer X, its d-digit redundant number may not be unique. For example, the value of $Y = (\underline{00}0110, \underline{00}0111, \underline{00}0001, \underline{00}1111)$ is equal to that of X although they have different numbers in each corresponding digit. Since the all the redundant bits of this redundant radix-2^4 number are zero, it can be converted to the non-redundant radix-2^4 number by just removing the redundant bits. Also, the non-redundant numbers can be converted to the equivalent redundant numbers by attaching redundant bits <u>00</u> to each digit.

From the definition, the value of a d-digit redundant radix-2^r number X is up to $\sum_{i=0}^{d-1}(2^{r+2}-1)\cdot 2^{ir} = \frac{(2^{dr}-1)(2^{r+2}-1)}{2^r-1} > 2^{dr}$. However, we assume that *the valid value* of X is up to $2^{dr}-1$. If the value of X is greater than $2^{dr}-1$, it is regarded as *overflow*. For example, 4-digit redundant radix-2^4 numbers ($\underline{01}0000$, $\underline{00}0000$, $\underline{00}0000$, $\underline{00}0000$) and ($\underline{00}1101$, $\underline{11}0000$, $\underline{00}0000$, $\underline{00}0000$) are overflows, because their values are greater than $2^{16}-1$. We assume that, if the resulting value of an operation is a d-digit redundant radix-2^r number and it is greater than $2^{dr}-1$, it is not necessary for a circuit or a program performing the operation to guarantee the correct result due to *the overflow error*. Clearly, the redundant bits $X_{d-1}[r+1,r]$ of the most significant digit X_{d-1} of a d-digit redundant radix-2^r number X are not zero, then the value of X is overflow. Note that X can be overflow even if $X_{d-1}[r+1,r]$ is zero.

In this paper, we present hardware algorithms for various operations for redundant radix-2^r numbers. We assume that input numbers and the resulting numbers are not overflows, and the redundant bits of the most significant digit are always zero.

Table 1. The performance evaluations of our implementation and known implementation for 256-bit Montgomery modulo multiplication

	speed			hardware resources		
	freq(MHz)	cycles	time(μs)	slices	multipliers	block RAMs
Our implementation	52.86	17	0.322	2054	16	8
McIvor *et al.* [8]	76.17	93	1.22	4663	64	-
Khaleel *et al.* [1]	2.50	-	0.40	34345	-	-

3 Arithmetic Operations of Redundant/Non-redundant Numbers

3.1 Addition of Non-redundant Numbers

Let us observe the addition over non-redundant numbers. Let $X = (0101, 1010, 0101, 1001)$ and $Y = (0100, 0101, 1010, 1001)$ be 4-digit non-redundant radix-2^4 numbers. The sum $X + Y$ can be computed as follows:

$$\begin{array}{cccccc} & 0101 & 1010 & 0101 & 1001 \\ + & 0100 & 0101 & 1010 & 1001 \\ \hline & 1010 & 0000 & 0000 & 0010 \end{array}$$

Clearly, the carry from the least significant digit is propagated to the most significant digit. We call such carry *block carry*. In other words, for two d-digit non-redundant radix-2^r numbers X and Y, if $X_0 + Y_0 \geq 2^r$, then the block carry $c_0 = 1$. Also, if $X_i + Y_i + c_{i-1} \geq 2^r$ $(1 \leq i \leq d-1)$, then the block carry $c_i = 1$. Hence, the addition has a carry chain from the least significant and the most significant digit, and it increases the delay if the addition is implemented by a combinational circuit.

3.2 Block-Carry-Free Addition for Redundant Numbers

Let us see the computation of the sum of two redundant numbers. For two 4-digit redundant radix-2^4 numbers $X = (\underline{00}0101, \underline{11}0011, \underline{11}1111, \underline{10}1111)$ and $Y = (\underline{00}0011, \underline{10}1111, \underline{01}1111, \underline{01}0001)$, their sum $Z = X + Y$ can be computed by the position sum as follows:

$$\begin{array}{cccccc} & \underline{11} & & \underline{11} & \underline{10} & \\ & 0101 & & 0011 & 1111 & 1111 \\ & \underline{10} & & \underline{01} & \underline{01} & \\ + & 0011 & & 1111 & 1111 & 0001 \\ \hline & \underline{00}1101 & & \underline{01}0110 & \underline{10}0001 & \underline{01}0000 \end{array}$$

Clearly, the addition has no block carry. Let us see the addition of two d-digit redundant radix-2^r numbers X and Y. The sum $Z = X + Y$ can be computed as follows:

$$\begin{aligned} Z_0 &= X_0[r-1,0] + Y_0[r-1,0] \\ Z_i &= X_{i-1}[r+1,r] + X_i[r-1,0] + Y_{i-1}[r+1,r] \\ &\quad + Y_i[r-1,0] \quad (1 \leq i < d) \end{aligned}$$

Hence, $Z_0 < 2^r + 2^r = 2^{r+1}$ and $Z_i < 4 + 2^r + 4 + 2^r < 2^{r+2}$ holds if $r \geq 2$. Thus, Z is a correct redundant radix-2^r number.

Let us design a combinational circuit to compute the sum $Z = X + Y$. Let $\mathrm{ADD}(2, 2, r, r)$ denote an adder circuit that computes the sum of two 2-bit and two r-bit integers. Also, let $\mathrm{ADD}(A, B, C, D)$ denote the resulting value of the sum of 2-bit numbers A and B, and r-bit numbers C and D. Clearly, $Z_0 = \mathrm{ADD}(0, 0, X_0[r-1,0], Y_0[r-1,0])$ and $Z_i = \mathrm{ADD}(X_{i-1}[r+1,r], Y_{i-1}[r+1,r], X_i[r-1,0], Y_i[r-1,0])$. Thus we have,

Lemma 1 *The addition of two d-digit redundant radix-2^r numbers can be computed using d adders* $\mathrm{ADD}(2, 2, r, r)$ *without block carries, whenever $r \geq 2$.*

Since the computation is performed independently in each $\mathrm{ADD}(2, 2, r, r)$, the circuit for Lemma 1 has no block carry chain. Thus, the delay time of the circuit is small and independent of d.

3.3 Block-Carry-Free Multiplication of Redundant Numbers

We show that the multiplication of 3-digit and 1-digit redundant radix-2^4 numbers can be computed without block carry. Let $X = (\underline{01}0011, \underline{10}0011, \underline{10}1111)$ and $Y = (\underline{10}0101)$. The product $X \cdot Y$ can be computed using 6-bit×6-bit=12-bit multiplications as follows.

$$\begin{array}{cccccc} & & & \underline{01}0011 & \underline{10}1001 & \underline{01}0001 \\ \times & & & & & \underline{10}0101 \\ \hline & & & 0010 & 0111 & 0101 \\ & & 0101 & 1110 & 1101 & \\ + & 0010 & 1011 & 1111 & & \\ \hline & \underline{00}0010 & \underline{01}0000 & \underline{01}1111 & \underline{01}0100 & \underline{00}0101 \end{array}$$

Clearly, we do not have the block carries. Let us formally confirm that the multiplication of d-digit and 1-digit redundant radix-2^r numbers can be computed without block carries. Let X and Y be d-digit and 1-digit redundant radix-2^r numbers. Also, let $P_i = X_i \cdot Y$ $(0 \leq i \leq d-1)$ be the partial multiplication. Since both X_i and Y has $r+2$ bits, P_i has $2r+4$ bits. We can compute the product $S = X \cdot Y$ as follows.

$$\begin{aligned} S_0 &= P_0[r-1,0] \\ S_1 &= P_0[2r-1,r] + P_1[r-1,0] \\ S_i &= P_{i-2}[2r+3,2r] + P_{i-1}[2r-1,r] \\ &\quad + P_i[r-1,0] \quad (2 \leq i \leq d-1) \\ S_d &= P_{d-2}[2r+3,2r] + P_{d-1}[2r-1,r] \\ S_{d+1} &= P_{d-1}[2r+3,2r] \end{aligned}$$

Hence, $S_0 < 2^r$, $S_1 < 2^r + 2^r = 2^{r+1}$, $S_d < 2^r$, and $S_{d+1} < 2^4$ hold. Also, if $r \geq 3$ then $S_i < 2^4 + 2^r + 2^r \leq 2^{r+2}$ holds. Thus, $S = (S_{d+1}, S_d, \ldots, S_0)$ is a redundant radix-2^r number.

Let $\mathrm{MUL}(r+2, r+2)$ and $\mathrm{ADD}(4, r, r)$ denote combinational circuits to compute the $(2r+4)$-bit product of two $(r+2)$-bit numbers and the $(r+2)$-bit sum of one 4-bit and two r-bit numbers. Each of the partial products $P_{d-1} = X_{d-1} \cdot Y$, $P_{d-2} = X_{d-2} \cdot Y$, …, $P_0 = X_0 \cdot Y$ can be computed using $\mathrm{MUL}(r+2, r+2)$. After that, each S_i can be computed using $\mathrm{ADD}(4, r, r)$. Thus, we have

Lemma 2 *The product of d-digit and 1-digit redundant radix-2^r numbers can be computed using d $\mathrm{MUL}(r+2, r+2)$s, and d $\mathrm{ADD}(4, r, r)$s, whenever $r \geq 3$.*

Next, to show a circuit to compute two d-digit redundant numbers, we will show how to add a $(d+1)$-digit radix-2^r number C to the product $X \cdot Y$. More specifically, we will show how to compute $S = X \cdot Y + C$. Later, C is used to store interim results of the product sum. We can compute each digit of the sum T can be computed as follows.

$$\begin{aligned} T_0 &= P_0[r-1,0] + C_0[r-1,0] \\ T_1 &= P_0[2r-1,r] + P_1[r-1,0] \\ &\quad + C_0[r+1,r] + C_1[r-1,0] \\ T_i &= P_{i-2}[2r+3,2r] + P_{i-1}[2r-1,r] \\ &\quad + P_i[r-1,0] + C_{i-1}[r+1,r] \\ &\quad + C_i[r-1,0] \quad (2 \leq i \leq d-1) \\ T_d &= P_{d-2}[2r+3,2r] + P_{d-1}[2r-1,r] \\ &\quad + C_{d-1}[r+1,r] + C_d[r-1,0] \\ T_{d+1} &= P_{d-1}[2r+3,2r] + C_d[r+1,r] \end{aligned}$$

Clearly, each T_i can be computed using $\mathrm{ADD}(2, 4, r, r, r)$, and the resulting value has no more than $r+2$ bits if $r \geq 5$. Thus, T is a $(d+2)$-digit redundant radix-2^r number and we have,

Lemma 3 *For a d-digit redundant radix-2^r number X, a 1-digit redundant radix-2^r number Y, and a $(d+1)$-digit redundant radix-2^r number C, the product sum $X \cdot Y + C$ can be computed using d $\mathrm{MUL}(r+2, r+2)$s, $d+2$ $\mathrm{ADD}(2, 4, r, r, r)$s, and a $(d+1)(r+2)$-bit registers, whenever $r \geq 5$.*

Let $T = \mathrm{PS}(X, Y, C)$ denote the circuit (or function) for Lemma 3. Using $\mathrm{PS}(X, Y, C)$ we can compute the sum C of two d-digit redundant radix radix-2^r numbers X and Y. Let $X = (X_{d-1}, X_{d-2}, \ldots, X_0)$ and $Y = (Y_{d-1}, Y_{d-2}, \ldots, Y_0)$ be two d-digit redundant radix radix-2^r numbers. We will show how to compute the product $P = (P_{2d-1}, P_{2d-2}, \ldots, P_0) = X \cdot Y$ using $\mathrm{PS}(X, Y_i, C)$. We compute partial products $X \cdot Y_0$, $X \cdot Y_1$, …, $X \cdot Y_{d-1}$ in turn. We use $C = (C_d, C_{d-1}, \ldots, C_0)$ to denote registers storing a interim $(d+1)$-digit redundant radix radix-2^r number. We first compute $\mathrm{PS}(X, Y_0, 0)$. Then, P_0 is the least significant digit $\mathrm{PS}(X, Y_0, 0)[r+1, 0]$. We store the remaining $d+1$ digits $\mathrm{PS}(X, Y_0, 0)[(d+1)(r+2)-1, r+2]$ in C. After that, we compute $\mathrm{PS}(X, Y_1, C)$. Clearly, P_1 is the least significant digit $\mathrm{PS}(X, Y_1, C)[r+1, 0]$ holds, and then we store the remaining $d+1$ digits $\mathrm{PS}(X, Y_1, C)[(d+1)(r+2)-1, r+2]$ in C. Continuing similarly, we can obtain the product $M = X \cdot Y$. The details are spelled out as follows:

$C \leftarrow 0$;
for $i = 0$ to $d-1$ do
  begin
    Compute $\mathrm{PS}(X, Y_i, C)$;
    $P_i \leftarrow \mathrm{PS}(X, Y_i, C)[r+1, 0]$;
    $C \leftarrow \mathrm{PS}(X, Y_i, C)[(d+2)(r+2)-1, r+2]$;
  end
$(P_{2d-1}, P_{2d-2}, \ldots, P_d) \leftarrow (C_{d-1}, C_{d-2}, \ldots, C_0)$;

We have the following theorem:

Theorem 4 *For two d-digit redundant radix-2^r numbers X and Y, the product $X \cdot Y$ in the redundant radix-2^r representation can be computed in d clock cycles using d $\mathrm{MUL}(r+2, r+2)$s, $d+2$ $\mathrm{ADD}(2, 4, r, r, r)$s, and a $(d+1)(r+2)$-bit register, whenever $r \geq 5$.*

4 Montgomery Modulo Multiplication

In the RSA encryption/decryption, the modulo exponentiation $C = P^E \bmod M$ or $P = C^D \bmod M$ are computed, where P and C are plain and cypher text, and (E, M) and (D, M) are encryption and decryption keys. Usually, the number of bits in P, E, D, and M is 256 or larger. Also, the modulo exponentiation is repeatedly computed for fixed E, D, and M, and various P and C. Since modulo operation is very costly in terms of

the computing time and hardware resources, we use *Montgomery modulo multiplication* [10], which does not use direct modulo operations. In Montgomery modulo multiplication, three R-bit numbers X, Y, and M are given, and $(X \cdot Y + k \cdot M) \cdot 2^{-R} \bmod M$ is computed, where an integer k is selected such that the least significant R bits of $X \cdot Y + k \cdot M$ are zero. The value of k can be computed as follows. Let $(-M^{-1})$ denote the minimum non-negative number such that $(-M^{-1}) \cdot M \equiv -1(\text{ or } 2^R - 1) \pmod{2^R}$. If M is odd, then $(-M^{-1}) < 2^R$ always holds. We can select k such that $k = ((X \cdot Y) \cdot (-M^{-1}))[r-1, 0]$. For such k, $(X \cdot Y + k \cdot M)[r-1, 0]$ are zero. For the reader's benefit, we will confirm this fact using an example as follows. Let $X = 10010011(147)$, $Y = 01011100(92)$, $M = 11111011(251)$, and $R = 8$. We have the product $X \cdot Y = 011010011010100(13524)$. Next, we need select an integer k such that the least significant R bits of $X \cdot Y + k \cdot M$ are zero. We have $(-M^{-1}) = 11001101(205)$, because $(-M^{-1}) \cdot M \equiv 1100100011111111(51455) \equiv -1 \pmod{2^R}$. We select $k = (X \cdot Y)[R-1, 0] \cdot (-M^{-1}) = 11000100(196)$. Then, we have the product $k \cdot M = 1100000000101100(49196)$ and the product sum $X \cdot Y + k \cdot M = 1111010100000000(62720)$. Thus, we have $(X \cdot Y + k \cdot M)[r-1, 0] = 00000000$ and $(X \cdot Y + k \cdot M) \cdot 2^{-R} = (X \cdot Y + k \cdot M)[2R-1, R] = 11110101(245)$.

Since $0 \leq X, Y < M < 2^R$ and $0 \leq k < 2^R$, we can guarantee that $(X \cdot Y + k \cdot M) \cdot 2^{-R} < 2M$. Thus, by subtracting M from $(X \cdot Y + k \cdot M) \cdot 2^{-R}$, we can obtain $(X \cdot Y + k \cdot M) \cdot 2^{-R} \bmod M$ if it is not less than M.

Since $X \cdot Y + k \cdot M \equiv X \cdot Y \pmod{M}$, we write $(X \cdot Y + k \cdot M) \cdot 2^{-R} \bmod M = X \cdot Y \cdot 2^{-R} \bmod M$. Let us see how Montgomery modulo multiplication is used to compute $C = P^E \bmod M$ using an example. Suppose we need to compute $C = P^E \bmod M$. For simplicity, we assume that E is a power of two. Since R and M are fixed, we can assume that $2^{2R} \bmod M$ is computed beforehand. We first compute $P \cdot (2^{2R} \bmod M) \cdot 2^R \bmod M = P \cdot 2^R \bmod M$ using the Montgomery modulo multiplication. We then compute the square $(P \cdot 2^R \bmod M) \cdot (P \cdot 2^R \bmod M) \cdot 2^{-R} \bmod M = P^2 \cdot 2^R \bmod M$. It should be clear that, by repeating the square computation using the Montgomery modulo multiplication, we have $P^E \cdot 2^R \bmod M$. After that, we multiply 1, that is $(P^E \cdot 2^R \bmod M) \cdot 1 \cdot 2^{-R} \bmod M = P^E \bmod M$ is computed. In this way, cypher text C is obtained.

4.1 Block-Carry-Free Implementation of Montgomery Modulo Multiplication

Recall that in the Montgomery modulo multiplication, R bit numbers X, Y, and M are given. In this subsection, we assume X and Y are a d-digit redundant radix-2^r number and a 1-digit redundant radix-2^r number, respectively. We will show a circuit to compute the Montgomery modulo multiplication $(X \cdot Y + k \cdot M) \cdot 2^{-r}$ for such X, Y, and M. We assume that the value of X and Y are given to the circuit as inputs, M is fixed and $(-M^{-1})$ is computed beforehand. This assumption makes sense if Montgomery modulo multiplication is used to compute the modulo exponentiation for RSA encryption and decryption.

Recall that, using the circuit for Lemma 2, $X \cdot Y$ can be computed using d $\mathrm{MUL}(r+2, r+2)$s and d $\mathrm{ADD}(4, r, r)$s. After computing $X \cdot Y$, we need to compute k such that the least significant r bits of $(X \cdot Y + k \cdot M)$ are zero. We can compute $k = ((X \cdot Y)[r-1, 0] \cdot (-M^{-1}))[r-1, 0]$ using a $\mathrm{MUL}(r, r)$. Once k is obtained, the product $k \cdot M$ is computed using the circuit for Lemma 2. Finally, the sum $(X \cdot Y + k \cdot M)$ is computed by the circuit for Lemma 1. Note that both $X \cdot Y$ and $k \cdot M$ are $(d+1)$-digit redundant radix-2^r numbers. However, since the least significant digit of $X \cdot Y$ and $k \cdot M$ are zero, we can omit the addition of the least significant digit. The readers should refer to Figure 1 for illustrating the circuit for Lemma 5.

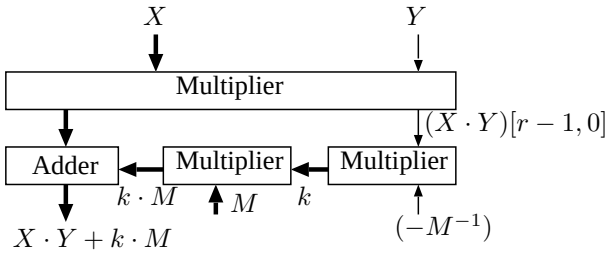


Figure 1. Circuit to compute $(X \cdot Y + k \cdot M)$ using multipliers

To compute the multiplication $X \cdot Y$, we can use a circuit for Lemma 2 which uses d $\mathrm{MUL}(r+2, r+2)$s and d $\mathrm{ADD}(4, r, r)$. To compute k, we use a $\mathrm{MUL}(r, r)$. After that to compute the multiplication $k \cdot M$, we also use a circuit for Lemma 2 and the addition $X \cdot Y + k \cdot M$ can be computed using d $\mathrm{ADD}(2, 2, r, r)$ by Lemma 1.

Therefore, we have,

Lemma 5 *Montgomery modulo multiplication $(X \cdot Y + k \cdot M) \cdot 2^{-r}$ for d-digit X and 1-digit Y of redundant radix-2^r representation can be computed using $2d+1$ $\mathrm{MUL}(r+2, r+2)$s, $2d$ $\mathrm{ADD}(4, r, r)$s, and d $\mathrm{ADD}(2, 2, r, r)$, without block carries, whenever $r \geq 4$.*

4.2 Montgomery Modulo Multiplication Using a Memory

The circuit for Lemma 5 has a cascade of three multipliers, which can be a long critical path. Also, it needs too many multipliers. We remove multipliers for computing k

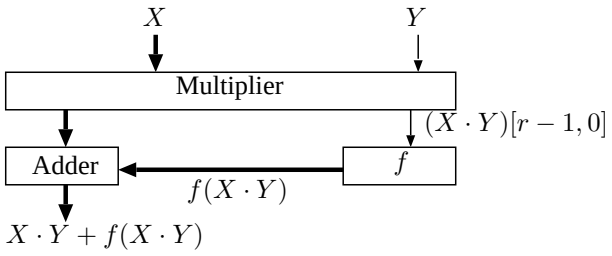


Figure 2. Circuit to compute $X \cdot Y \cdot + f(X \cdot Y)$ using a memory

to improve the circuit for Lemma 5. The key idea is to use a memory to look up the value of $k \cdot M$.

Let f be a function such that $f(Z) = (Z[r-1,0] \cdot (-M^{-1}))[r-1,0] \cdot M$. The function f can be computed using a 2^r word $(d+1)r$-bit memory as follows. The value of $f(i)$ $(0 \leq i \leq 2^r - 1)$ is stored in address i of the memory in advance. Then, by reading address $Z[r-1,0]$ of the memory, we can obtain the value of $f(Z)$ in one clock cycle. Using this memory, $f(X \cdot Y)$ can be computed in one clock cycle. After that, the addition $X \cdot Y + f(X \cdot Y)$ can be computed using $d+1$ $\mathrm{ADD}(2,2,r,r)$s from Lemma 1. Figure 2 illustrates the circuit to compute $X \cdot Y + f(X \cdot Y)$.

Note that the least significant digit of $X \cdot Y + f(X \cdot Y)$ is always zero. Hence, we can omit the computation of the least significant digit of f and the following addition. Thus, we use a 2^r word dr-bit memory for computing $f(X \cdot Y)$ and d $\mathrm{ADD}(2,2,r,r)$s to compute the sum $X \cdot Y + f(X \cdot Y)$. Therefore, we have,

Lemma 6 *Montgomery modulo multiplication $(X \cdot Y + k \cdot M) \cdot 2^{-r}$ for d-digit X and M, and 1-digit Y of redundant radix-2^r representation can be computed using d $\mathrm{MUL}(r+2, r+2)$s, $d+2$ $\mathrm{ADD}(2,4,r,r,r)$s, d $\mathrm{ADD}(2,2,r,r)$, and a 2^r-word dr-bit memory, without block carries, whenever $r \geq 5$.*

If $r = 16$ and $dr = 1024$, then the circuit for Lemma 6 needs $64k$-word 1024-bit memory of size $64M$ bits. Since the size of block memory of current FPGAs is up to few mega bits, this circuit cannot be implemented in FPGAs.

4.3 Montgomery Modulo Multiplication Using a Fewer Memory

We will reduce the size of memory to compute the function f. Recall that, k is a r-bit number such that the least significant r bits of $X \cdot Y + k \cdot M$ are zero. Let r-bit k number partition into two $r/2$ bits such that $\overline{k} = k[r-1, r/2]$ and $\underline{k} = k[r/2-1, 0]$. We can compute the values of $\overline{k}$ and $\underline{k}$ separately as follows. Let $(-M^{-1})$ be the minimum non-negative integer such that $(-M^{-1}) \cdot M \equiv -1 \pmod{2^{r/2}}$. Also, let $\overline{XY} = (X \cdot Y)[r-1, r/2]$ and $\underline{XY} = (X \cdot Y)[r/2-1, 0]$. We set $\underline{k} = \underline{XY} \cdot (-M^{-1})$. Then, the least significant $r/2$ bits of $X \cdot Y + \underline{k} \cdot M$ are zero. Let g be a function such that $g(Z) = ((Z[r/2-1,0] \cdot M)[r-1, r/2] \cdot (-M^{-1}) + c$ and $c = 0$ if $(Z \cdot M)[r/2-1, 0] = 0$ and $c = 1$ otherwise. Function g can be computed using a combinational circuit with $r/2$ input bits and $r/2$ out put bits. We set $\overline{k} = (\overline{XY} + g(\underline{XY}))[r/2-1, 0]$. Then, the least significant digit of $X \cdot Y + \underline{k} \cdot M + \overline{k} \cdot M \cdot 2^{r/2}$ is zero.

We will implement this idea in the same way as Lemma 6. Instead of computing $\underline{k}$ and $\overline{k}$, we compute $\underline{k} \cdot M$ and $\overline{k} \cdot M$ using a memory. Let h be a function such that $h(Z) = (Z[r/2-1,0] \cdot (-M^{-1}))[r/2-1,0] \cdot M$. Similarly to f, function g can be computed using $2^{r/2}$-word $(d(r+2) + r/2)$-bit memory. Then, $\underline{k} \cdot M = h(\underline{X \cdot Y})$ and $\overline{k} \cdot M = h(\overline{XY} + g(\underline{XY}))$ holds. Thus, $k \cdot M = \underline{k} + \overline{k} \cdot 2^{r/2} = h(\underline{X \cdot Y}) + h(\overline{XY} + g(\underline{XY})) \cdot 2^{r/2}$. The readers should refer to Figure 3 for illustrating the circuit to compute $X \cdot Y + k \cdot M = X \cdot Y + h(\underline{XY}) + h(\overline{XY} + g(\underline{XY})) \cdot 2^{r/2}$. Since a FPGAs has dual port memories, two modules to compute h in Figure 3 can be computed by a single $2^{r/2}$-word $(dr + r/2)$-bit dual port memory in the same time. The readers may think that a combinational circuit to compute g is not necessary. However, block RAMs in most FPGAs to implement a memory support only synchronous read. Thus, one clock cycle is necessary to read a memory. It follows that, if we use a memory to implement the computation of g, two clock cycles are necessary to compute $h(\overline{XY} + g(\underline{XY}))$.

The circuit to compute g is small. If $r = 16$, then a combinational circuits with 8 input bits and 8 output bits to compute g are used, and it is feasible. For example, a two-digit 7-segment decoder has 8 input bits and 14 output bits. So, necessary hardware resource to compute g is comparable to that for the two-digit 7-segment decoder. Also, we can omit the computation of the least significant $r/2$-bit of h. Thus, a single $2^{r/2}$-word dr-bit dual-port memory can compute two functions h's in the same time.

Let us evaluate the hard aware resources necessary to compute $X \cdot Y + h(\underline{X \cdot Y}) + h(\overline{X \cdot Y} + g(\underline{X \cdot Y})) \cdot 2^{r/2}$. The multiplication $X \cdot Y$ can be computed using d $\mathrm{MUL}(r+2, r+2)$s and d $\mathrm{ADD}(4,r,r)$s from Lemma 2. Function $g(\underline{X \cdot Y})$ can be computed using a combinational circuits with 8 input bits and 8 output bits and addition $\overline{X \cdot Y} + g(\underline{X \cdot Y})$ can be computed $\mathrm{ADD}(r/2, r/2)$. After that the value of function h for two arguments can be computed using a $2^{r/2}$-word dr-bit dual-port memory. Finally, the sum $(X \cdot Y) + h(\underline{X \cdot Y}) + h(\overline{X \cdot Y} + g(\underline{X \cdot Y})) \cdot 2^{r/2}$ can be computed using d $\mathrm{ADD}(2,2,2,r,r,r)$ by straightforward generalization of Lemma 1. Consequently, we have,

Lemma 7 *Montgomery modulo multiplication $(X \cdot Y + k \cdot M) \cdot 2^{-r}$ for d-digit X and M, and Y and 1-digit Y of redundant radix-2^r representation can be computed using d*

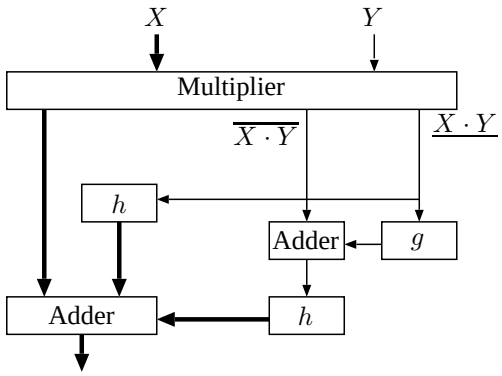
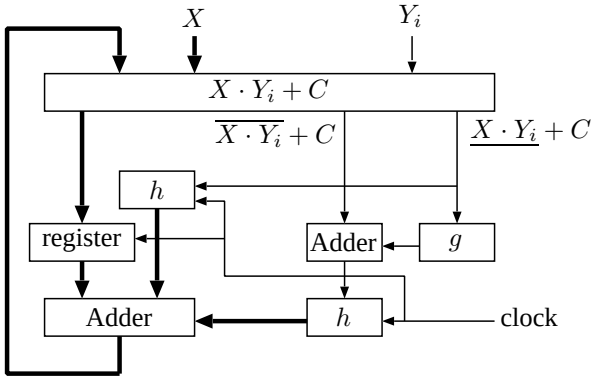


Figure 3. Circuit to compute $X \cdot Y + h(\underline{X \cdot Y}) + h(\overline{X \cdot Y} + g(\underline{X \cdot Y})) \cdot 2^{r/2}$



$X \cdot Y_i + C + k \cdot M$

Figure 4. Circuit to compute $X \cdot Y_i + C + k \cdot M$

$\mathrm{MUL}(r+2, r+2)$*s,* d $\mathrm{ADD}(4, r, r)$*s, one* $\mathrm{ADD}(r/2, r/2)$*,* d $\mathrm{ADD}(2, 2, 2, r, r, r)$*, a* $2^{r/2}$*-word* dr*-bit dual-port memory, and a combinational circuit with* $r/2$*-bit input and* $r/2$*-bit output, without block carries, whenever* $r \geq 4$*.*

4.4 Montgomery Modulo Multiplication for Two d-digit Numbers

Recall that the multiplication of d-digit and 1-digit numbers are shown in Lemma 2. we have extended Lemma 2 to Theorem 4 which shows the computation of multiplication of two d-digit numbers. The same technique can be used to extend Lemma 7 to compute the Montgomery modulo multiplication of d-digit redundant radix-2^r numbers. In other words, we compute partial products $X \cdot Y_0$, $X \cdot Y_1$, ..., $X \cdot Y_{d-1}$ in turn, and compute their sum, which is equal to $X \cdot Y$.

Suppose that, as shown in Theorem 4, we use $(d+1)(r+2)$ bit register C to store $(d+1)$-bit redundant radix-2^r numbers as an interim result. To compute $X \cdot Y_i + C$ we use a circuit for Lemma 3. After that, the value of $k \cdot M$ such that the least significant digit of $X \cdot Y_i + C + k \cdot M$ is zero is obtained using the circuit for Lemma 7. Note that it uses a memory to compute function h. Thus, one clock cycle is necessary to compute $k \cdot M$. Additional one clock is necessary to store the resulting value of $X \cdot Y_i + C + k \cdot M$ in C. This implementation requires two clock cycles to compute $(X \cdot Y_i + C + k \cdot M) \cdot 2^{-r}$ and store it in C.

We can reduce these two clock cycles into one as follows. As illustrated in Figure 4, the output $X \cdot Y_i + C$ is stored in register instead of storing C. Then, the following adder computes $X \cdot Y_i + C + k \cdot M$. In this way, the value of $X \cdot Y_i + C + k \cdot M$ can be done in one clock cycle. The readers should refer to Figure 4 for illustrating the circuit.

Let us evaluate necessary hardware resources. To compute $X \cdot Y_i + C$ we use a circuit for Lemma 3, which uses d $\mathrm{MUL}(r+2, r+2)$s, $d+2$ $\mathrm{ADD}(2, 4, r, r, r)$s. As before, function g can be computed using a combinational circuit with 8 input bits and 8 output bits and function h can be computed using a dr-bit $2^{r/2}$-word dual-port memory. Also, we need one $\mathrm{ADD}(r/2, r/2)$ to compute $\overline{X \cdot Y} + g(\underline{X \cdot Y})$, and $(d+1)(r+2)$-bit register to store the value of $X \cdot Y_i + C$. After that, the sum $(X \cdot Y) + h(\underline{X \cdot Y}) + h(\overline{X \cdot Y} + g(\underline{X \cdot Y})) \cdot 2^{r/2}$ can be computed using d $\mathrm{ADD}(2, 2, 2, r, r, r)$s. Thus, we have

Theorem 8 *Montgomery modulo multiplication* $(X \cdot Y + k \cdot M) \cdot 2^{-dr}$ *for three* d*-digit redundant radix-*2^r *numbers* X*,* Y *and* M *can be computed in* d *clock cycles using* d $\mathrm{MUL}(r+2, r+2)$*s,* $d+2$ $\mathrm{ADD}(2, 4, r, r, r)$*s,* d $\mathrm{ADD}(2, 2, 2, r, r, r)$*, a* $2^{r/2}$*-word* dr*-bit dual port memory, and a combinational circuit with* $r/2$*-bit input and* $r/2$*-bit output, and a* $(d+1)(r+2)$*-bit register, without block carries, whenever* $r \geq 5$*.*

5 Experimental Results

We have evaluated the performance of redundant radix-2^r circuits using Virtex II Pro Family FPGA XC2VP100-6, which has 99,216 slices, 444 18-bit multipliers, and 444 18k-bit dual-port block RAMs. We have used XST in ISE Foundation 9.2i for logic synthesis and analysis. Since this FPGA has 18-bit multipliers as building blocks, it makes sense to let $r = 16$. Thus, we use redundant radix-64k (i.e. radix-2^{16}) number system.

Table 2 shows the experimental results of Montgomery modulo multiplication for d-digit redundant radix-64k numbers shown in Theorem 8. In addition to the circuit for Theorem 8, the experimental results include the circuit to subtract M from the resulting value $(X \cdot Y + k \cdot M) \cdot 2^{-dr}$ to guarantee that it is smaller than M. Thus, the circuit runs in $d+1$ clock cycles. For example, for $dr = 1024$-bit in-

Table 2. Montgomery modulo multiplication of two d-digit redundant/non-redundant radix-64k numbers

	bits	64	128	256	512	1024
	clock cycles	5	9	17	33	65
redundant	clock(MHz)	53.10	52.74	52.86	52.48	52.95
	time (μs)	0.094	0.171	0.322	0.629	1.23
	slices	560	1067	2054	3990	7883
	multipliers	4	8	16	32	64
	block RAMs	2	4	8	15	29
non-redundant	clock(MHz)	54.36	47.31	37.66	27.05	17.16
	time (μs)	0.092	0.190	0.451	1.22	3.79
	slices	453	740	1363	2574	4958
	multipliers	4	8	15	31	61
	block RAMs	2	4	8	15	29

put, it takes 65 clock cycles to complete the computation. The table also shows the experimental results of the circuits obtained by changing those for Theorem 8 to use the non-redundant number system. The experimental results show that the clock frequency of the circuits for the redundant number system is constant for every number of bits. On the other hand, the clock frequency for non-redundant number system decreases as the number of bits increases. The speed up factor of our hardware algorithm using the redundant number system over those using the conventional number system is 3 for 1024-bit Montgomery modulo multiplication.

6 Conclusions

We have introduced redundant radix-2^r number system for Montgomery modulo multiplication and its implementation on the FPGA. Our implementation for 256-bit Montgomery modulo multiplication is four times faster using less hardware resources in the FPGA than the previously know implementation.

References

[1] O. Al-Khaleel, C. Papachristou, F. Wolff, and K. Pekmestzi. FPGA-based design of a large moduli multiplier for public-key cryptographic systems. In *Proc. of International Conference on Computer Design*, pages 314 – 319, 2007.

[2] T. Blum and C. Paar. High-radix montgomery modular exponentiation on reconfigurable hardware. *IEEE Trans. on Computers*, 50(7):759–764, 2001.

[3] J. L. Bordim, Y. Ito, and K. Nakano. Accelerating the CKY parsing using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):803–810, May 2003.

[4] J. L. Bordim, Y. Ito, and K. Nakano. Instance-specific solutions to accelerate the CKY parsing for large context-free grammars. *International Journal on Foundations of Computer Science*, pages 403–416, 2004.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[6] C. K. Koc, T. Acar, and B. S. Kaliski, Jr. Analyzing and comparing Montgomery multiplication algorithms — assessing five algorithms that speed up modular exponentiation, the most popular method of encrypting and signing digital data. *IEEE Micro*, 16(3):26–33, 1996.

[7] R. Lin, K. Nakano, S. Olariu, M. C. Pinotti, J. L. Schwing, and A. Y. Zomaya. Scalable hardware-algorithms for binary prefix sums. *IEEE Trans. on Parallel and Distributed Systems*, 11(8):838–850, August 2000.

[8] C. McIvor, M. McLoone, and J. McCanny. FPGA Montgomery multiplier architectures - a comparison. In *Proc. of Field-Programmable Custom Computing Machines*, pages 279 – 282, 2004.

[9] P. V. A. Mohan. Fast algorithm for implementation of montgomery's modular multiplication technique. *Circuit System Signal Processing*, 23(6):463–478, 2004.

[10] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[11] K. Nakano and E. Takamichi. An image retrieval system using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):811–818, May 2003.

[12] K. Nakano and Y. Yamagishi. Hardware n choose k counters with applications to the partial exhaustive search. *IEICE Trans. on Information & Systems*, 2005.

[13] B. Parhami. *Computer Arithmetic - Algorithm and Hardware Designs*. Oxford University Press, 2000.

[14] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120 – 126, 1978.

非同期ブロックRAMの同期ブロックRAMへの変換について

広島大学　伊藤 靖朗

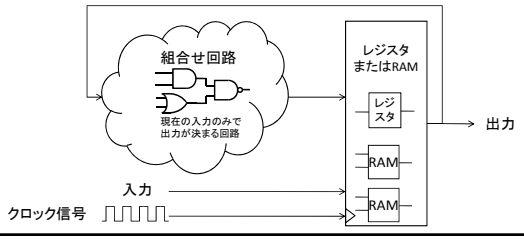
目的

- FPGAで回路設計する際に、FPGA内部で利用可能な同期読出しブロックRAMが用いられる
- 同期読出しブロックRAM
  - 読出しのタイミングを考慮する必要があり、設計が複雑になる

- 非同期読出しブロックRAMを利用して設計された回路から同期読出しブロックRAMを用いる回路に体系的に変換する方法を提案
  - 設計が容易な非同期読出しブロックRAMを用いて回路を設計可能に

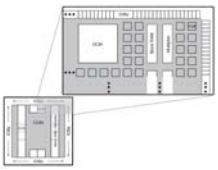
対象とする回路

- 完全同期式回路
  - 全体で共有するクロック信号を利用して動作のタイミングをあわせる回路
    - すべてのレジスタ・RAMは共通のクロック信号のタイミングでのみ値を更新
    - 設計が容易



FPGA(Field Programmable Gate Array)

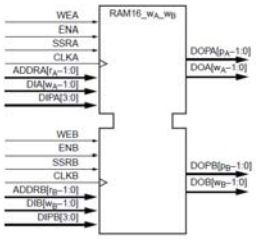
- FPGAの構成
  - 4入力1出力のルックアップテーブル
    - 任意の論理ゲートを作成可能
  - フリップフロップ
  - ブロックRAM
    - 容量が大きい（数百k～数Mbits ）
    - デュアルポート
    - 同期書込み、同期読出し
  - 乗算器
  - 入出力バッファ
  - デジタル クロック マネージャ
    - クロック管理機構



ブロックRAM

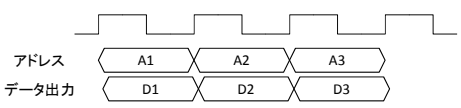
- 特徴
  - 一般的なFPGAに搭載
  - 容量
    - 数百k～数Mbits
  - デュアルポート
    - アドレス、データ入力、データ出力等の入出力を2組使用可能
  - 同期書込み
  - **同期読出し**

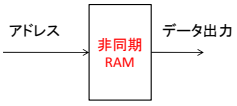
現在一般的に使われているFPGAに非同期読出しRAMは搭載されていない



非同期読出しRAM

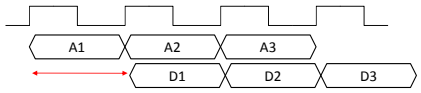
- 非同期読出し
  - アドレスを指定した後、少し遅れてデータ出力される



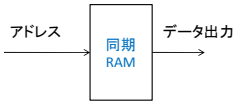


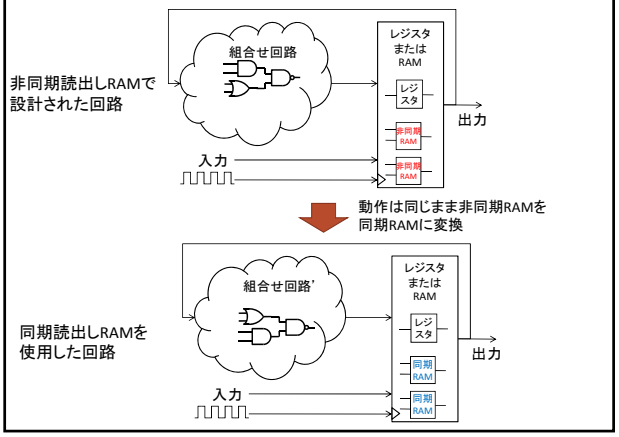
同期読出しRAM

- 同期読出し
  - アドレスを指定した次のクロックの立ち上がりでデータが出力される



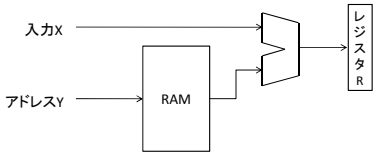
1サイクル遅れて出力





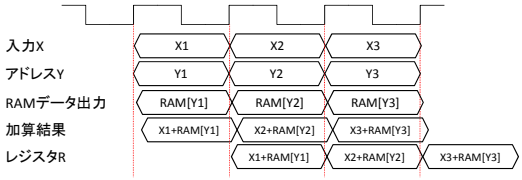
同期読み出しRAMが扱いにくい理由

- 例：入力値Xにアドレスy番地の値を加算し、レジスタRに格納する回路
  - R ← X + RAM[Y]

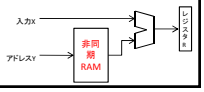


同期読み出しRAMが扱いにくい理由

- 非同期読み出しRAMの場合

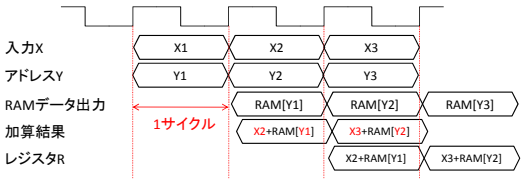


入力の次のサイクルでレジスタに結果が格納

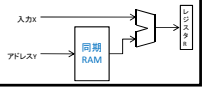


同期読み出しRAMが扱いにくい理由

- 同期読み出しRAMの場合

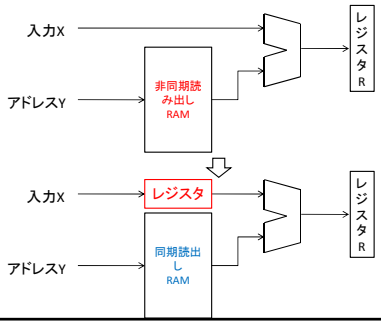


XとYのタイミングがずれて加算される
入力から2サイクル後にレジスタに格納される



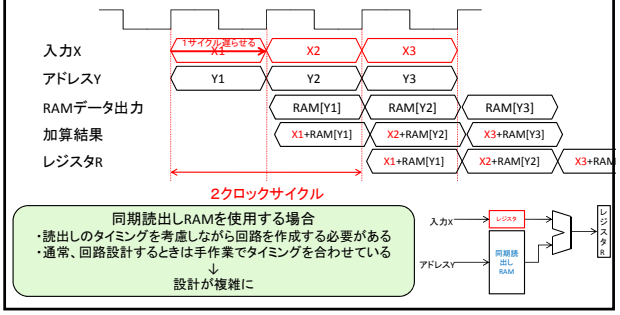
同期読み出しが扱いにくい理由

- 解決方法
  - 入力値Xを一旦レジスタに格納して、1クロックサイクル遅らせることによってタイミングを合わせる



同期読み出しが扱いにくい理由

- 同期読み出しRAMで、入力Xを1クロックサイクル遅延させた場合



同期読出しRAMを使用する場合
・読出しのタイミングを考慮しながら回路を作成する必要がある
・通常、回路設計するときは手作業でタイミングを合わせている
↓
設計が複雑に

非同期読出しRAM→同期読出しRAM変換

- 回路中のブロックRAMを非同期読出しとして設計
  - 設計が容易

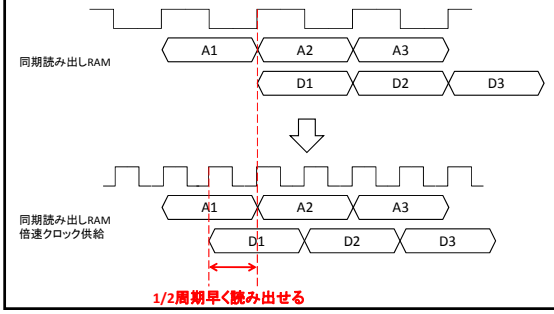
非同期同期読出し→同期読出し

- 同期読出しのブロックRAMを用いた回路
  - 回路の振る舞いはそのまま
  - 動作速度を低下させたくない
  - 回路リソースを増やしたくない

・倍速クロック法
・レジスタ挿入法
・バイパス法

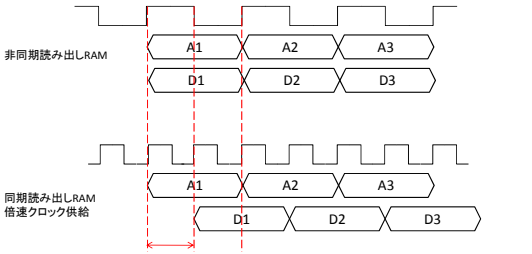
非同期RAM→同期RAM変換（倍速クロック法）

- RAMに供給するクロックの周波数を倍にする



非同期RAM→同期RAM変換（倍速クロック法）

- 非同期読出しRAMとのタイミングの比較

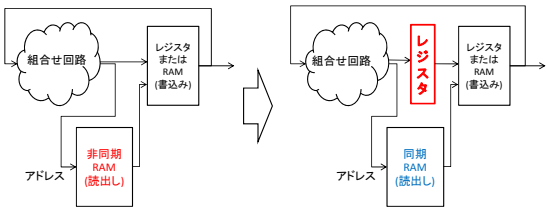


非同期RAM→同期RAM変換（倍速クロック法）

- 倍速クロック法
  - 倍速のクロックをRAMに供給するだけで実現可能
  - 2倍の周波数のクロックが必要
    - 現在のFPGAにはクロック管理機構が搭載されており、倍速のクロックが供給可能
  - RAMの性能に大きく依存
    - 回路の動作周波数の上限はRAMの動作周波数の1/2
  - RAMからの読出しに1/2サイクル時間必要

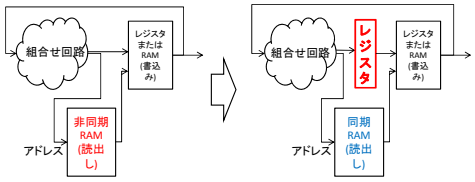
非同期RAM→同期RAM変換（レジスタ挿入法）

- RAMの読み出しと組合せ回路の出力が同じタイミングになるように、組合せ回路の出力すべて、または一部にレジスタを挿入



非同期RAM→同期RAM変換（レジスタ挿入法）

- レジスタ挿入法
  - レジスタを挿入するだけで変換可能
  - 実行に必要なサイクル数が2倍に
    - 回路の動作周波数を半分にすることと実質同じ
  - 多くの回路リソース（レジスタ）が必要

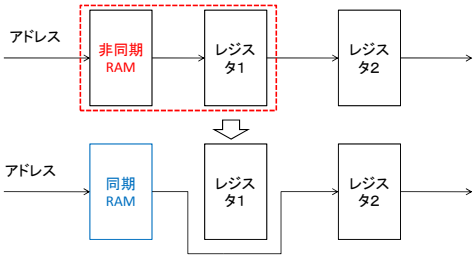


非同期RAM→同期RAM変換（バイパス法）

- バイパス法
  - RAMの前段または後段のレジスタをスキップすることにより非同期RAMを同期RAMで実現
    - 1サイクル遅れて読み出されても非同期読み出しRAMと同じタイミングで動作可能
  - ある種、回路の先読みを行う
    - RAMが1個の場合
    - RAMが複数ある場合
    - 変換ができない場合

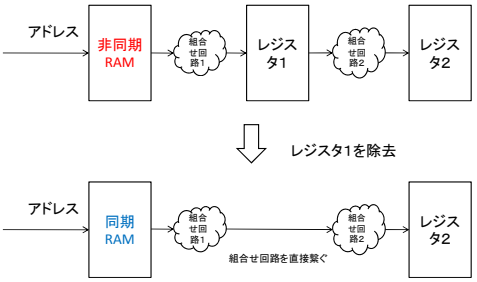
非同期RAM→同期RAM変換（バイパス法）

- 後段にレジスタが存在する場合



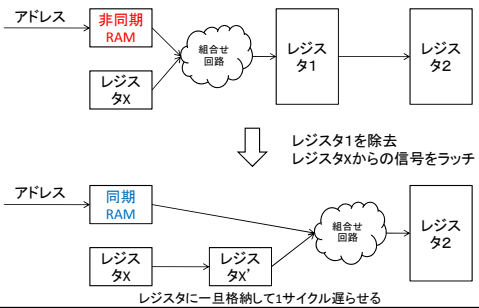
非同期RAM→同期RAM変換（バイパス法）

- RAMとレジスタ1、レジスタ1とレジスタ2の両方の間に組合せ回路が存在する場合



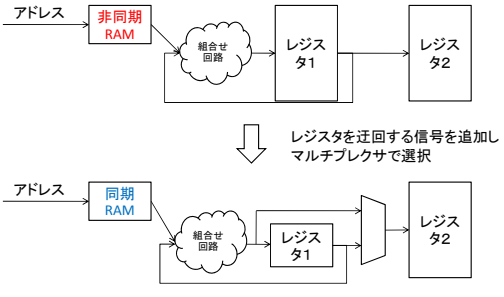
非同期RAM→同期RAM変換（バイパス法）

- RAMとレジスタ1の間に組合せ回路が存在し、かつ、その組合せ回路がRAMとレジスタからの入力に依存する場合



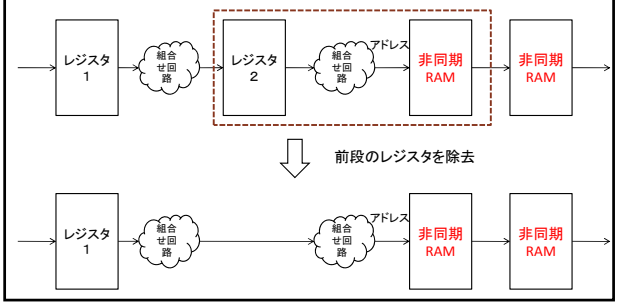
非同期RAM→同期RAM変換（バイパス法）

- 後段のレジスタを利用する場合



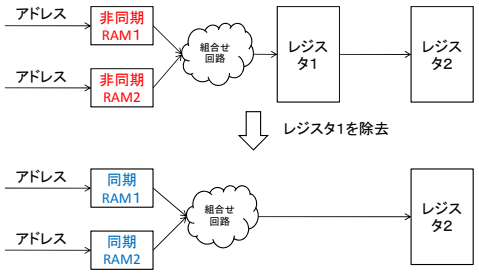
非同期RAM→同期RAM変換（バイパス法）

- 後段のレジスタをスキップできない場合

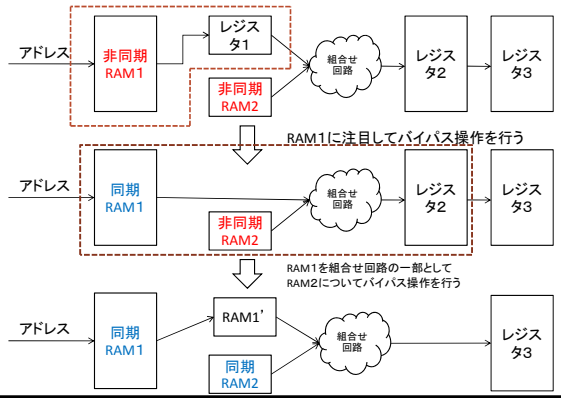


非同期RAM→同期RAM変換（バイパス法）

- 複数のRAMが存在する場合

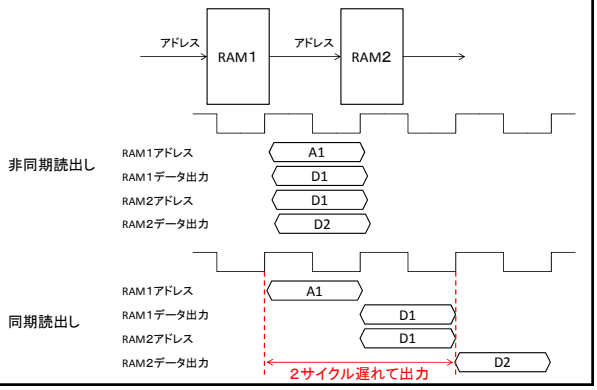


複数のRAMが存在する場合
⇒RAM毎に非同期RAMから同期RAMへ変換



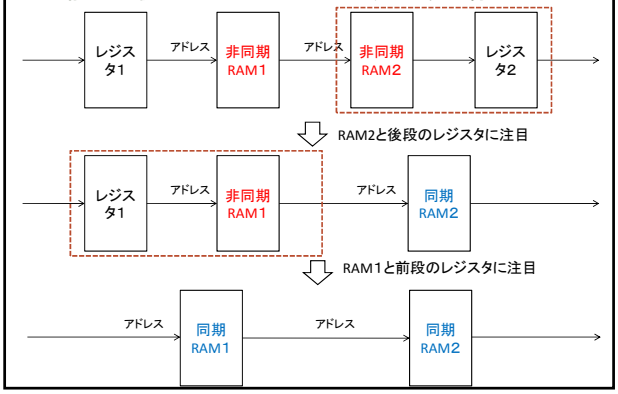
RAMが連続する場合

- RAMの出力を別のRAMのアドレスに入力



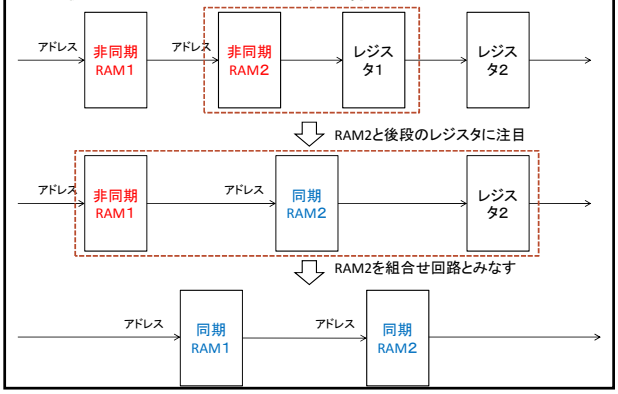
RAMが連続する場合

- 前段と後段にレジスタが1個ずつある場合



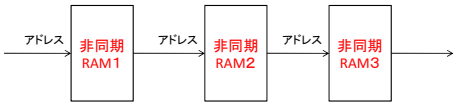
RAMが連続する場合

- 後段にレジスタが2個ある場合

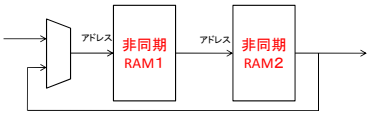


バイパス法が適用できない場合

- RAMが3つ連続する場合



- ループする場合



スキップすべきレジスタが存在しない

非同期RAM→同期RAM変換（バイパス法）

- バイパス法
  - 長所
    - リソース（レジスタ）の数をあまり変更することなく実現可能
  - 短所
    - 変換できない場合がある
      - RAMが3個連続する場合
      - RAMが連続しループする場合

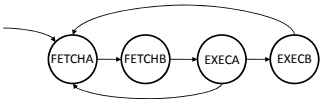
応用例（TinyCPUの状態数削減）

- TinyCPU
  - VerilogHDLで記述したシンプルなスタックアーキテクチャのCPU
    - 同期RAMで設計
    - 1命令4状態

  - 1命令1状態に状態数を削減
    - 手順1．非同期RAMを用いて設計
    - 手順2．同期RAMを用いた回路に変換
      - バイパス法を適用
      - 1命令1状態を維持したまま変換

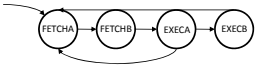
応用例（TinyCPUの状態数削減）

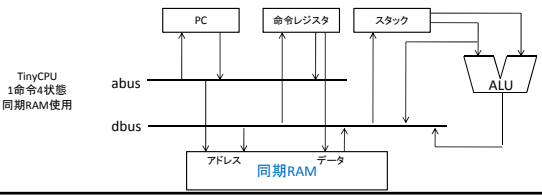
- TinyCPU
  - 1命令4状態（CPI=4）※CPI(Cycles Per Instruction)
    - 命令フェッチ×2サイクル、実行×2サイクル
      - ただし、ロード命令（メモリ読出し）以外の実行は1サイクル

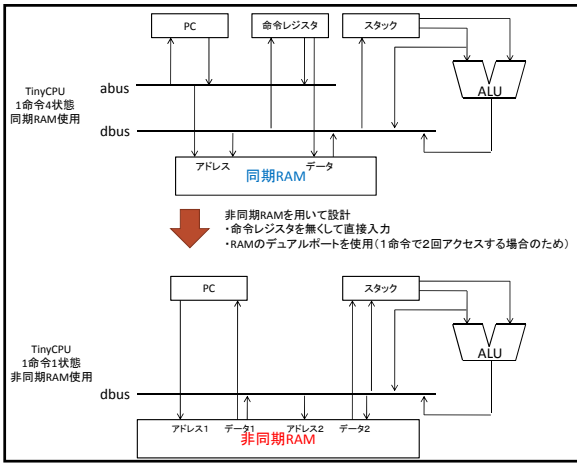


TinyCPUの動作例

- PUSH命令
  - RAMの指定したアドレスからデータを読出しスタックに格納
    - 1．PCからRAMにアドレスを指定
    - 2．命令レジスタに格納
    - 3．RAMにアドレスを指定
    - 4．スタックに格納



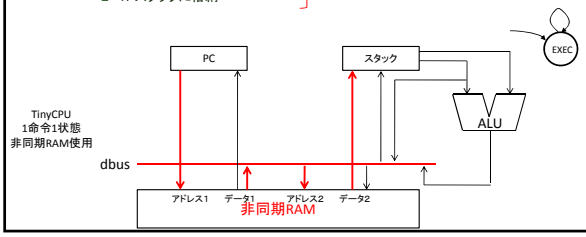




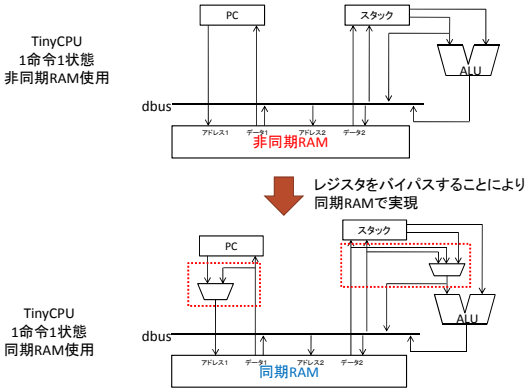
TinyCPU（非同期RAM使用）の動作例

- PUSH命令
  - RAMの指定したアドレスからデータを読出しスタックに格納
    1. PCからRAMにアドレスを指定
    2. 命令レジスタに格納
    3. RAMにアドレスを指定
    4. スタックに格納

非同期RAMを用いているので
1クロックサイクルで実行可能



非同期読出しRAM→同期読出しRAMへ変換



性能評価

性能評価
・ターゲットFPGA
・Xilinx社Spartan-3A(XC3S700A)
・70万ゲート相当
・360KbitsブロックRAM
・論理合成ツール
・Xilinx社ISE Foundation 10.1

	TinyCPU 1命令4状態バージョン	TinyCPU 1命令1状態バージョン
フリップフロップ数	115	159
ルックアップテーブル数	698	732
ブロックRAM数（18kBits）	4	4
最大動作周波数	80MHz	74MHz

動作周波数がほとんど変わらないので、性能は約4倍に

まとめ

- 非同期読出しブロックRAMを利用して設計された回路から同期読出しブロックRAMを用いる回路に体系的に変換する方法を提案
  - 同期読出しブロックRAMを意識せずに、設計が容易な非同期読出しブロックRAMを用いて回路を設計可能に
- 今後の課題
  - 自動変換ツールの作成
    - ハードウェア記述言語（VerilogHDL、VHDL等）で記述された回路を変換

共有メモリシステムにおける調停木スキップ相互排除アルゴリズム

鈴木　健司†　　井上美智子†　　藤原　秀雄†

† 奈良先端科学技術大学院大学 情報科学研究科
E-mail: †{tsuyoshi-s,kounoe,fujiwara}@is.naist.jp

あらまし　共有メモリシステムにおいて遠隔のメモリ参照回数である RMR 計算量に関して効率の良いアルゴリズムを提案する．N プロセスの相互排除アルゴリズムでは 1 回の資源獲得と資源解放に当たり，最悪時の RMR 計算量が $\Theta(logN)$ であることが知られている．本稿では同時に処理を行うプロセス数が増加した場合に効率の良いアルゴリズムを提案する．

キーワード　相互排除問題，共有メモリシステム，RMR(Remote Memory Reference)，調停木

1. はじめに

相互排除問題は，複数のプロセスがある資源を排他的に使用するために解く問題で，分散協調問題における基本問題の一つである．各プロセスが idle，entry，critical，exit の 4 セクションを繰り返し実行する時，相互排除アルゴリズムは critical セクションを実行するプロセスが高々1 プロセスであることを保証する．

共有メモリシステムで動作する分散アルゴリズムの評価尺度として遠隔メモリ参照 (Remote Memory Reference，以降 RMR) の回数である RMR 計算量が知られている．相互排除アルゴリズムの RMR 計算量とは 1 プロセスが 1 回の critical セクションの実行の前後に行う entry，exit セクションでの RMR の回数である．RMR を削減することによって通信コストと計算時間を削減することが可能となる．read，write 操作のみを使用する共有メモリモデルにおける N プロセスの相互排除問題について最悪時の RMR 計算量に関する既知の結果を述べる．

Yang ら [5] は RMR 計算量 $O(logN)$，空間計算量 $O(NlogN)$ であるアルゴリズムを提案した．彼らは 2 プロセスの相互排除問題を定数回の RMR で解くアルゴリズムを提案し，それを二分木状に配置することで N プロセスの相互排除アルゴリズムを解いている．

Kim ら [4] は $O(logN)$ の RMR 計算量を維持したまま空間計算量を $O(N)$ に改善した．この空間計算量は最適である．

Anderson ら [1] は同時に entry セクションを実行するプロセス数を表す競合度:k に RMR 計算量が依存する競合度アダプティブなアルゴリズムを提案している．彼らの計算量は $O(min(k, logN))$ であり，競合度が少ない時に効率的なアルゴリズムである．

相互排除問題の下界の証明も行われており，Anderson ら [2] は $\Omega(\frac{logN}{loglogN})$ を証明した．さらに Attiya ら [3] は $\Omega(logN)$ であることを証明した．これによって Yang ら [5] の相互排除アルゴリズムは最悪時評価において RMR 計算量が最適であることが示された．

我々は RMR 計算量の期待値に対して効率の良いアルゴリズムを提案する．アルゴリズムを使用する側にとって最悪値よりも期待値評価の方が有用な場合が多い．提案アルゴリズムは最悪時 RMR 計算量は $O(logN)$ であるが同時に処理を行うプロセス数が多い場合 RMR 計算用の期待値が $O(1)$ に近づくことを示す．

以降 2 節では使用するモデルについて述べ，3 節で先行研究である Yang ら [5] のアルゴリズムの概要を説明する．そして 4 節で提案アルゴリズムを説明して 5 節で正当性の証明をする．6 節で評価を行い，最後にまとめを行う．

2. モデル

2.1 共有メモリシステム

共有メモリシステムは非同期に処理を行う複数のプロセスとそれらがアクセス可能な共有メモリからなる．共有メモリへの操作はこのメモリへの読み出し (read) と書き込み (write) の 2 つとする．

以下に取り扱う二種類の計算機モデルのメモリアクセスについて説明する．

2.1.1　分散共有メモリ (Distributed Shared Memory)

分散共有メモリ (Distributed Shared Memory，以降 DSM) は共有メモリがシステム内のローカル共有メモリに分散しているモデルである．各プロセスはローカル共有メモリ上の変数にはローカル操作を行い，他プロセスのローカル共有メモリ上の変数には遠隔操作を行う．

2.1.2　キャッシュコヒーレント (Cache Coherent) モデル

キャッシュコヒーレント (Cache Coherent，以降 CC) モデルは各プロセスがキャッシュを持ち共有変数のコピーを管理するモデルであり，コヒーレンスプロトコルにより，キャッシュの一貫性が保証される．共有変数のアクセスはキャッシュが共有変数の最新の値を維持していればローカル操作となる．最新の値でなければ遠隔操作となり，キャッシュの値が最新になる．

2.2 相互排除アルゴリズム

相互排除アルゴリズムではプロセスが次の 4 つのセクションを順に繰り返し実行する．

idle　資源を使用しない
entry　資源を獲得する
critical　資源を使用する
exit　資源を解放する

ここで idle セクションは任意の時間とし，critical セクションは任意の有限時間で実行が完了する．相互排除アルゴリズムでは以下を満たすよう entry，exit セクションを設計する．

a）相互排除である

critical セクションを同時に実行するプロセスは高々一つである．

b）スタベーションフリーである

あるプロセスが entry セクションを実行すればいずれそのプロセスは critical セクションを実行する．

2.3 RMR 計算量の評価

相互排除アルゴリズムの評価を共有変数への遠隔操作である RMR の回数で評価する．共有変数への遠隔操作はローカル操作と比較して通信コストが高く処理時間も長い．よってアルゴリズムの性能は遠隔操作の回数に支配されると考えられる．相互排除アルゴリズムの RMR 計算量は 1 プロセスが critical セクションを実行する時に前後で実行する entry，exit セクションでの遠隔操作で評価する．

3. 先行研究

3.1 Yang ら ['95]

Yang ら [5] のアルゴリズム (以降 YA) では RMR 計算量が定数である 2 プロセスの相互排除アルゴリズム (以降 2PME-YA) を提案し，さらに 2 分木状に配置した調停木を用いて N プロセスの相互排除問題を RMR 計算量 $O(logN)$ で解いている．

3.1.1 2 プロセスの相互排除アルゴリズム (2PME-YA)

図 1 に 2 つのプロセス p，q の 2PME-YA の p のコードを示す．q についてのコードは命令文の p，q を入れ替えたものに対応する．2PME-YA は $C[p]$, $C[q]$, $P[p]$, $P[q]$, T の 5 つの共有変数を用いる．

$C[p]$, $C[q]$ は entry セクションの実行を開始したことを示す変数で初期値が -1 で entry，critical セクションでは $C[p]$ は p, $C[q]$ は q である．$P[p]$, $P[q]$ は critical セクションを実行するための優先度を表し，$0, 1, 2$ の順に優先度が上がる．p, q が同時に entry セクションを実行すると $C[p] = p$ かつ $C[q] = q$ となる．この時，共有変数 T を先に更新した方が優先的に critical セクションを実行する．プロセスが 2PME-YA の entry セクションを終了することを 2PME-YA を獲得すると呼ぶ．exit セクションで相手プロセス q が entry セクションを実行中であれば，q の優先度を上げるために $P[q]$ を更新する．プロセスがこの更新を行うことを 2PME-YA を解放すると呼ぶ．

2PME-YA では p は $C[p]$, $C[q]$, $P[q]$, T に q は $C[p]$，$C[q]$, $P[p]$，T にそれぞれ定数回の操作を行う．よって DSM では $P[p]$ を p のローカル変数，$P[q]$ を q のローカル変数 ($C[p]$, $C[q]$, T はどちらかのローカル変数) とすれば RMR 計算量が定数となる．

Algorithm 1 2PME-YA for Process p

```
shared variable
    C[p, q] : integer initially −1
    T : integer
    P[p, q] : (0, 1, 2) initially 0

 1: while true do
 2:     idle section
 3:     C[p] := p;                                  ▷ entry section 開始
 4:     T := p;
 5:     P[p] := 0;
 6:     if C[q] ≠ −1 then
 7:         if T = p then
 8:             if P[q] = 0 then
 9:                 P[q] := 1;
10:             end if
11:             await P[p] > 0
12:             if T = p then
13:                 await P[p] > 1
14:             end if
15:         end if
16:     end if                                      ▷ entry section 終了
17:     critical section
18:     C[p] := −1;                                 ▷ exit section 開始
19:     if T ≠ p then
20:         P[q] := 2;
21:     end if                                      ▷ exit section 終了
22: end while
```

図 1　Algorithm 1 2PME-YA for process p

また，p は 11 行目，13 行目で $P[p]$ の値が変わるまで $P[p]$ の値の参照を繰り返す．CC モデルでは共有変数の値が最新である限り値の参照はローカル操作であるので RMR 計算量は定数となる．

3.1.2 N プロセスの相互排除アルゴリズム (YA)

YA では 2PME-YA を葉数 $N/2$ の完全二分木状に配置して N プロセスの相互排除問題を解く．但し，根以外の 2PME-YA は entry,exit セクションのみからなる．各プロセスは識別子に対応した葉の 2PME-YA から実行を開始し，ある 2PME-YA の entry セクションを終了すると親の 2PME-YA の entry セクションを実行する．そして根の entry セクションを終了すると critical セクションを実行する．critical セクションを終了すると根から葉に向かって exit セクションを実行する．

各 2PME-YA は RMR が定数で処理可能なので，完全二分木に 2PME-YA を配置することで N プロセスの相互排除問題を $O(logN)$ で解くことができる．このような機能を果たす完全二分木を調停木 (arbitration tree) と呼ぶ．

4. 提案アルゴリズム

4.1 基本アイデア

これまで提案されている多くの N プロセスの相互排除アルゴリズムでは 2 プロセスの相互排除アルゴリズムを $logN$ 段の完全二分木状に配置し，プロセスが葉から根まで 2 プロセスの相互排除アルゴリズムの entry セクションを順に実行している．我々は調停木を最後まで登らずに critical セクションを実行することで計算量の削減を行うアルゴリズムを提案する．

提案アルゴリズム (以降 SIF) は図 2 のような葉数 $N/2$ の完全 2 分木状の調停木と大きさ N の待ち配列からなる．調停木の各ノードは 2PME-YA(Entry2PME，Exit2PME) とスキップ判定処理，待ち配列への追加処理 (AddtoArray) からなる．各プロセスは調停木の葉から根に向かって Entry2PME とスキッ

プ判定処理を実行することで critical セクションの実行を目指す．以降 SIF の 2 プロセスの相互排除アルゴリズムを 2PME と呼ぶ．

各 Entry2PME，Exit2PME はそれぞれ 2PME-YA の entry セクション，exit セクションと同一であり，一方のプロセスが entry セクションを終了し，exit セクションを実行を開始していなければ他方のプロセスは entry セクションの実行を終了できない．

SIF では critical セクションを終了したプロセスは exit セクションにおいて entry セクションで移動した木の経路をたどることで経路上で待機しているプロセスを見つけ，そのプロセスを待ち配列に追加する．

図 2 はアルゴリズムの entry セクションにおけるプロセスの流れを示したものである．entry セクションでプロセスは exit セクションを実行する他プロセスに発見されないと木を登るが，発見されると待ち配列へ追加され，待機をする．待ち配列への追加と呼び出しは FIFO(First In,First Out) で行う．これにより，待ち配列に追加されたプロセスはいずれ呼び出され，スタベーションフリーを実現している．さらに相互排除であることを保証するために木の根の上に新たに 2PME-YA を追加する．これにより木を登ってきたプロセスと木をスキップしたプロセス間で 2 プロセスの相互排除を行わせ，最終的に critical セクションを実行するプロセスを高々一つにしている．図 4 にアルゴリズムを示す．簡単のため N を 2 のべき乗数とする．

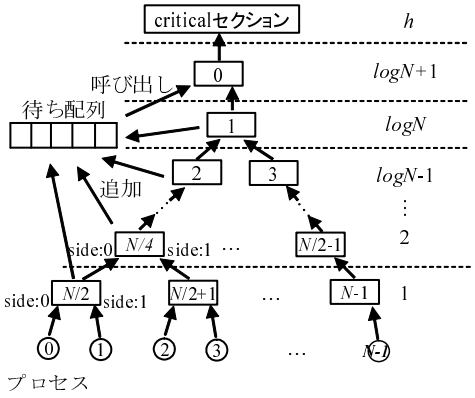


図 2　entry セクション時のプロセスの流れ

4.2　アルゴリズムの概要

4.2.1　共有変数

図 3 にアルゴリズムが使用する共有変数を示す．配列 T, C, P は YA と同じ目的で使用する．W は待ち配列である．W は -1 に初期化されており，プロセスの識別子または，現在の処理の状態を示すために用いる．配列 $Added$ はプロセスが待ち配列に追加されたか，さらにスキップの許可が与えられたかを表す．$Added[p]$ が 1 の時はプロセス p が追加された状態で，2 の時は呼び出されスキップの許可が与えられた状態である．$Add, Call$ はそれぞれプロセスが待機する待ち配列の末尾と先頭を表す．DSM の場合，$P[p], Added[p]$ をプロセス p のローカル変数とする．

```
ID : process identifier
const
    L : logN
shared variables
    T[0..N − 1] : integer
    C[0..N − 1][0, 1] : integer initially − 1
    P[1..logN + 1][0..N − 1] : (0, 1, 2) initially 0
    W[0..N − 1] : integer initially − 1
    Added[0..N − 1] : (0, 1, 2) initially 0
    Add : integer initially N − 1
    Call : integer initially N − 1
```

図 3　共有変数一覧

4.2.2　entry セクション

entry セクションは図 4 の 3 行目から 19 行目までである．3 行目から 14 行目までの for ループにおいて h は木の段数を表し，葉 (1 段目) から根 ($logN$ 段目) まで順に 2PME の処理を行う．途中で待ち配列に追加されれば待ち配列を経由して $node$ 値 0 の 2PME(9 行目) へスキップし entry セクションを終了する．各プロセスは図 2 に示すようにプロセス ID に応じた葉ノードの 2PME の side(0 または 1) から実行を開始する．以降 $node$ 値が n の 2PME を 2PME(n)，それに加えて $side$ が s の 2PME を 2PME(n, s) と表記する．

6,9,16 行目の Entry2PME，28,32 行目の Exit2PME は YA の entry，exit セクションと同じ処理を行う．同じ $node$, $side$ で Entry2PME を実行するプロセスが同時に高々1 つであるならば，Entry2PME の実行を終了し同じ $node$ の Exit2PME の実行を開始していないプロセスは同時に高々1 つであることが保証されている．

7 行目で $Added[ID]$ が 1 の場合，プロセスは待ち配列に追加された状態となり，8 行目で $Added[ID]$ の値が 2 以上になれば $2PME(0, 0)$ の entry セクションを実行する．一方，待ち配列に追加されなかったプロセスは $2PME(0, 1)$ の entry セクションを実行する．

4.2.3　exit セクション

exit セクションは図 4 の 21 行目から 38 行目までである．exit セクションでは共有変数である待ち配列のインデックスを排他的に更新するために，YA の exit セクションの処理 (Exit2PME) の前に待ち配列への追加 (AddtoArray)，待ち配列からの呼び出し判定 (DecisionCall) を行う．最後に呼び出し判定が真ならば呼び出し処理を行い，待ち配列に追加されたプロセスに木をスキップする許可を与える．以下に待ち配列への追加，呼び出し処理，YA の exit セクションの処理をそれぞれ説明する．

a）　待ち配列への追加 (AddtoArray)

exit セクションの最初に AddtoArray 関数 (図 6) で待ち配列へプロセス追加する．AddtoArray を実行するプロセスは entry セクションで通過した 2PME をスキップしたノードから葉まで順にたどり，それらの 2PME の entry セクションを実行するプロセスの存在を確認する．この時にプロセスが存在すればプロセスを発見したと呼び，そのプロセスを待ち配列に追加する．

b）　呼び出しの判定 (DecisionCall)

exit セクションを実行するプロセスは待ち配列への追加を行った後，待ち配列の先頭のプロセスを呼び出して Entry2PME(0,0) を実行させるかどうかを判定する．

呼び出し判定では自身が 2PME(0) を $side = 0$，つまり待ち配列側から実行した場合，または 2PME(0,0) の exit セクションの処理を終了していないプロセスが存在しない場合に待ち配列の先頭のプロセスの呼び出しを決定する．

呼び出し判定を終了したプロセスは YA の exit セクションの処理 (図 8) を実行後，判定したプロセスを呼び出す．

c） YA の exit セクションの処理 (Exit2PME)

待ち配列への追加，呼び出し判定は待ち配列のインデックスを排他的に更新する必要があったために 2PME の解放を行う前に行った．その後，exit セクションを実行するプロセスは自身が獲得した 2PME を解放する．Exit2PME(図 8) は YA の exit セクションの処理と同じもので，プロセスは自身が 2PME を獲得した順と逆順に Exit2PME を実行して 2PME を解放する．

5. 証　　明

SIF の正当性を証明する．証明の中で使用する表記法を説明する．n@p(Funciton) はプロセス p の関数 Function の n 行目を意味する．n@(Funciton) は関数 Function の n 行目を意味する．[n_1@p(Funciton),n_2@p(Funciton)] は n_1@p(Funciton) の実行から n_2@p(Funciton) の実行までを意味する．

5.1 相互排除である

まず使用する補題を示す．

補題 1. プロセス p が 27@p(Mutual Exclusion) を実行終了時のシステム状況まで [20@(Mutual Exclusion),27@(Mutual Exclusion)] が排他的に行われ，かつ 22@p(Mutual Exclusion) で $callflg = true$，かつ p によって呼び出されたプロセス q が [20@q(Mutual Exclusion),27@q(Mutual Exclusion)] を排他的に行うならば，q が 27@q(Mutual Exclusion) の実行を終了するまで 22@(Mutual Exclusion) で $callflg = true$ となるような他のプロセスは存在しない．

証明. n を任意の自然数とし，任意の実行において n 番目に 22@(Mutual Exclusion) で $callflg = true$ となったプロセスに対して補題 1 が成り立つことを数学的帰納法によって示す．

$n = 1$ の時，最初に 22@(Mutual Exclusion) で $callflg = true$ となったプロセスを p_1 とし，p_1 が呼び出したプロセスを q_1 とする．q_1 が 27@q_1(Mutual Exclusion) の実行を終了するまでに他のプロセス r が 22@r(Mutual Exclusion) で $callflg = true$ となると仮定する．そのような r の中で最初に $callflg = true$ となったプロセスを r' とする．[20@p_1(Mutual Exclusion),27@p_1(Mutual Exclusion)] を終了後，$W[0] = -2$ が成り立つ．この値は q_1 が 11@q_1(DecisionCall) を実行するまで更新されない．r' が $side = 1$ とすると 5@r'(DecisionCall) で $W[0] = -1$ となり，矛盾する．従って r' は $side = 0$ である．この時，r' を呼び出したプロセス s が存在するので s は $callflg = true$ となり矛盾する．従って q_1 が 28@q_1(Mutual Exclusion) の実行を開始するまでに 22@(Mutual Exclusion) で $callflg = true$ となるようなプロセスは存在しない．

$n = k$ の時，k 番目に 22@(Mutual Exclusion) で $callflg =$

Algorithm 2 Mutual Exclusion

```
private variable
  h, node, side, tmpcall, rival : integer
  skip : integer initially −1
  callflg : boolean

 1: while true do
 2:   idle section
 3:   for h := 1 to L do                          ▷ entry section 開始
 4:     node := ⌊(N+ID)/2^h⌋;
 5:     side := ⌊(N+ID)/2^(h−1)⌋ mod 2;
 6:     Entry2PME(node, side, h)
 7:     if Added[ID] > 0 then
 8:       await Added[ID] > 1
 9:       Entry2PME(0, 0, L + 1)
10:       skip := h;
11:       side := 0;
12:       break;
13:     end if
14:   end for
15:   if skip = −1 then
16:     Entry2PME(0, 1, L + 1)
17:     skip := L;
18:     side := 1;
19:   end if                                       ▷ entry section 終了
20:   critical section
21:   AddtoArray(skip)    ▷ exit section 開始, 待ち配列への追加処理
22:   callflg := DecisionCall(side)                ▷ 呼び出し判定処理
23:   if callflg = true then
24:     tmpcall := Call
25:     rival := W[tmpcall];
26:     W[tmpcall] := −2;
27:   end if
28:   Exit2PME(0, side, L + 1)      ▷ アルゴリズム YA の exit section(28〜33 行目)
29:   for h := skip down to 1 do
30:     node := ⌊(N+ID)/2^h⌋;
31:     side := ⌊(N+ID)/2^(h−1)⌋ mod 2;
32:     Exit2PME(node, side, h)
33:   end for
34:   if callflg = true ∧ Added[rival] = 1 then    ▷ 呼び出し処理
35:     Added[rival] := 2;
36:   end if
37:   skip := −1;
38:   Added[ID] := 0;                              ▷ exit section 終了
39: end while
```

図 4　Algorithm 2 Mutual Exclusion

Algorithm 3 Entry2PME
(*node* **:integer**,*side* **:integer**,*h* **:integer**)

```
private variable
   rival : integer

 1: C[node][side] := ID;
 2: T[node] := ID;
 3: P[h][ID] := 0;
 4: rival := C[node][1 − side];
 5: if rival ≠ −1 then
 6:     if T[node] = ID then
 7:         if P[h][rival] = 0 then
 8:             P[h][rival] := 1;
 9:         end if
10:         await P[h][ID] > 0
11:         if T[node] = ID then
12:             await P[h][ID] > 1
13:         end if
14:     end if
15: end if
```

図 5　Algorithm 3 Entry2PME

Algorithm 4 AddtoArray(*skip*: **integer**)

```
private variable
   h, node, rival, tmpadd : integer

 1: tmpadd := Add;
 2: for h := skip down to 1 do
 3:     node := ⌊(N+ID)/2^h⌋;
 4:     rival := T[node];
 5:     if rival ≠ ID then
 6:         tmpadd := (tmpadd + 1) mod N;
 7:         if Added[rival] = 0 then
 8:             Added[rival] := 1;
 9:             W[tmpadd] := rival;
10:         end if
11:     end if
12: end for
13: Add := tmpadd;
```

図 6　Algorithm 4 AddtoArray

Algorithm 5 DecisionCall(*side*: **integer**)

```
private variable
   tmpcall, precall, rival : integer
   callflg : boolean initially false

 1: precall := Call;
 2: tmpcall := (precall + 1) mod N;
 3: rival := W[tmpcall];
 4: if rival ≧ 0 then
 5:     if side = 0 ∨ (side = 1 ∧ W[precall] = −1) then
 6:         Call := tmpcall;
 7:         callflg := true;
 8:     end if
 9: end if
10: if side = 0 then
11:     W[precall] := −1;
12: end if
13: return(callflg);
```

図 7　Algorithm 5 DecisionCall

Algorithm 6 Exit2PME
(*node* **:integer**,*side* **:integer**,*h* **:integer**)

```
private variable
   rival : integer

1: C[node][side] := −1;
2: rival := T[node];
3: if rival ≠ ID then
4:     P[h][rival] := 2;
5: end if
```

図 8　Algorithm 6 Exit2PME

$true$ となったプロセスを p_k とする．p_k が [20@p_k(Mutual Exclusion),27@p_k(Mutual Exclusion)] を排他的に行い，かつ 22p_k@(Mutual Exclusion) で $callflg = true$ の場合，p_k によって呼び出されたプロセス q_k が 27@q_k(Mutual Exclusion) の実行を終了するまで 22@(Mutual Exclusion) で $callflg = true$ となるような他のプロセスは存在しないと仮定する．

$n = k + 1$ の時，$k + 1$ 番目に 22@(Mutual Exclusion) で $callflg = true$ となったプロセスを p_{k+1} とし，p_{k+1} が呼び出したプロセスを q_{k+1} とする．q_{k+1} が 27@q_{k+1}(Mutual Exclusion) の実行を終了するまでに他のプロセス t が 22@t(Mutual Exclusion) で $callflg = true$ となると仮定する．そのような t の中で最初に $callflg = true$ となったプロセスを t' とする．[20@p_{k+1}(Mutual Exclusion),27@p_{k+1}(Mutual Exclusion)] を終了後，$W[i] = -2$ が成り立つ．i は q_{k+1} が書かれた W のインデックスである．$W[i]$ は q_{k+1} が 11@q_{k+1}(DecisionCall) を実行するまで更新されない．t' が $side = 1$ とすると 5@t'(DecisionCall) で $W[i] = -1$ となるので矛盾する．従って t' は $side = 0$ である．この時 t' を呼び出したプロセス u が存在し，u は $callflg = true$ となる．つまり u は p_{k+1} より前に 22@(Mutual Exclusion) を実行している．u の後に $callflg = true$ となった p_{k+1} が存在し，かつ [20@u(Mutual Exclusion),27@u(Mutual Exclusion)] を排他的に行うので，仮定より u が呼び出した t' は 28@t'(Mutual Exclusion) 以降を実行している．よって矛盾し，q_{k+1} が 28@q_{k+1}(Mutual Exclusion) の実行を開始するまでに 22@(Mutual Exclusion) で $callflg = true$ となったプロセスは存在しない．

従って補題 1 は成り立つ．　□

各プロセス p は entry セクションで調停木のノードを葉から 2PME(1) に向かって順に獲得しようとする．この時 $Added[p]$ の値が常に 0 であれば p は YA の entry セクションと同じ動作を行う．従って証明の中で Yang ら [5] の補題を利用する．

YA に対して以下の条件が成り立つ．

条件　1. 任意の 2PME(n) の任意の $side$:s において [1@(Entry2PME(n, s)),1@(Exit2PME(n, s))] は同時に高々 1 プロセスが行う．

SIF の調停木は YA の entry セクション，exit セクションを含んでいる．従って 2PME(0) 以外の 2PME に対して条件 1 が成り立つ．

従って 2PME(0) が条件 1 を満たせば SIF は相互排除である．$side = 1$ については YA の critical セクションに対応しており，YA が相互排除を満たしているため条件 1 は成立する．従って 2PME(0,0) に対して条件 1 が成り立つことを示す．

補題　2. 2PME(0,0) において [9@(Mutual Exclusion), 27@(Mutual Exclusion)] は同時に高々 1 プロセスが行う．

証明. 任意のプロセス p の任意の実行に対して以下の 2 つの条件が満たされれば補題は成立する．

ME 1. 9@p(Mutual Exclusion) を実行する時，$side = 0$ で

[9@(Mutual Exclusion),27@(Mutual Exclusion)] を行うプロセスは存在しない.

ME 2. $side = 0$ で [9@p(Mutual Exclusion),27@p(Mutual Exclusion)] を行う間，9@(Mutual Exclusion) を実行するプロセスは存在しない.

n を任意の自然数とし，任意の実行において n 番目に Mutual Exclusion の 9 行目を実行したプロセスが ME1，2 を満たすことを数学的帰納法によって示す.

$n = 1$ のとき，p_1 が最初に Mutual Exclusion の 9 行目を実行したプロセスとする.

(1) ME1 について

2PME(0,0) を実行したプロセスは存在しないので ME1 は成立する.

(2) ME2 について

プロセス p_1 が $side = 0$ で [9@p_1(Mutual Exclusion), 27@p_1(Mutual Exclusion)] を行う間，9@q(Mutual Exclusion) を実行するプロセス q が存在すると仮定する．そのような q の中で最初に 9@q(Mutual Exclusion) を実行したプロセスを q' とする．q' が 9@q'(Mutual Exclusion) を実行するまでは 2PME(0) を $side = 0$ で実行するのは p のみなので $side = 0, 1$ 共に条件 1 を満たし [20@(Mutual Exclusion),27@(Mutual Exclusion)] は排他的に行われる．この排他処理を行ったプロセスの集合を S とする．p_1, q' が 9@(Mutual Exclusion) を実行するには $Added[p_1] \geqq 2$,$Added[q'] \geqq 2$ となることが必要である．よってプロセス r, s が存在し ($r, s \in S$)，r, s はそれぞれ 35@(Mutual Exclusion) で $Added[p_1]$,$Added[q']$ に 2 を書いており，r, s の $callflg = true$ である．ここで r, s が共に $callflg = true$ となる以前のシステム状況を考える．この時，27@(Mutual Exclusion) を実行したプロセスは S に含まれ，p_1, q' は共に 27@(Mutual Exclusion) の実行を終了していないので，補題 1 よりこの後に r, s が共に $callflg = true$ となるような遷移は存在しない．よって矛盾し，ME2 は成り立つ.

$n = k$ の時 k 番目に 9@(Mutual Exclusion) を実行したプロセス p_k が ME1，ME2 を満たすと仮定する.

$n = k + 1$ の時 p_{k+1} が $k + 1$ 番目に 9@(Mutual Exclusion) を実行したプロセスとする.

(1) ME1 について

p_k が ME2 を満たすので p_{k+1} は ME1 が成り立つ.

(2) ME2 について

プロセス p_{k+1} が $side = 0$ で [9@p_{k+1}(Mutual Exclusion), 28@p_{k+1}(Mutual Exclusion)] を行う間，9@t(Mutual Exclusion) を実行するプロセス t が存在すると仮定する．そのような t の中で最初に 9@t(Mutual Exclusion) を実行したプロセスを t' とする．9@t'(Mutual Exclusion) を実行する前に [20@(Mutual Exclusion),27@(Mutual Exclusion)] を実行した任意のプロセス u は ME1 が成り立ち，仮定より ME2 が成り立つ．従って [20@u(Mutual Exclusion),27@u(Mutual Exclusion)] は排他的に行われる．u からなる集合を S' と置く．p_{k+1}, t' が 9@(Mutual Exclusion) を実行するには $Added[p_{k+1}] \geqq 2$,$Added[t'] \geqq 2$ となる必要がある．よってプロセス v, w が存在し ($v, w \in S'$)，v, w はそれぞれ 35@(Mutual Exclusion) で $Added[p_{k+1}]$,$Added[t']$ に 2 を書いており，v, w の $callflg = true$ である．ここで v, w が共に $callflg = true$ となる前のシステム状況を考える．この時，27@(Mutual Exclusion) を実行したプロセスは S' に含まれ，p_{k+1}, t' は共に 27@(Mutual Exclusion) の実行を終了していないので，補題 1 よりこの後に v, w が共に $callflg = true$ となるような遷移は存在しない．よって矛盾し，ME2 は成り立つ.

従って任意のプロセスの任意の実行において ME1，ME2 が成立し，補題 2 は成立する． □

2PME(0) が条件 1 を満たすので SIF は相互排除である.

定理 1. SIF は相互排除である.

5.2 スタベーションフリーである

SIF がスタベーションフリーであることを示す．そのためには entry セクション内の各待機処理がいずれ終了することを示せばよい．既に相互排除であることを証明したので以下の性質を利用する.

性質 1 任意の実行において [20@(Mutual Exclusion), 27@(Mutual Exclusion)] を行うプロセスは同時に高々1 つである.

entry セクション内の待機処理は以下である.

待機 1. 10@(Entry2PME)(**await** $P[h][ID] > 0$)

待機 2. 8@(Mutual Exclusion)(**await** $Added[ID] > 1$)

待機 3. 12@(Entry2PME)(**await** $P[h][ID] > 1$)

参考文献 [5] より

補題 3. 待機 1 はいずれ終了する.

が YA において成り立つ．待機 1 は同じ $node$ に対する Entry2PME の処理が重なった時に起こる．この時プロセスは 2PME-YA と同じ振る舞いをするので補題 3 が SIF においても成り立つ.

次に待機 2 がいずれ終了することを示すために使用する補題を示す.

補題 4. プロセス p が T に p を書いた後に $Added[p]$ に 2 が書かれると 8@p(Mutual Exclusion) を終了するまで $Added[p]$ は更新されない.

証明. $Added[p]$ の更新は

更新 0. 38@(Mutual Exclusion) で 0 に (p による初期化)

更新 1. 8@(AddtoArray) で 1 に (待ち配列に追加)

更新 2. 35@(Mutual Exclusion) で 2 に (呼び出し処理)

である.

更新 2 の後に p が 8@p(Mutual Exclusion) を終了するまで更新 0 が起こらないことを示す．更新 2 は $Added[p] = 1$ の時に行われるので更新 1 が以前に行われている．更新 1 の対象となるプロセスは entry セクション時に通った 2PME 内で T に識別子を書いたプロセスのみである．さらに更新 1 は YA の

exit セクションの処理の前に行う．従って更新 1 が行われる時，p は Entry2PME 内の処理を行っている．さらに更新 1 の後に $Added[p] = 1$ となり，この間に更新 0 の処理は p 自身しか行えないので 7@p(Mutual Exclusion) が真となり，更新 2 が起こるまで 8@p(Mutual Exclusion) を終了することはできない．従って更新 2 の後に p が 8@p(Mutual Exclusion) を終了するまで更新 0 は起こらない．

次に更新 2 の後に更新 1 が起こらないことを示す．更新 1 は $Added[p]$ が 0 の場合に 1 に更新し，更新 2 は $Added[p]$ が 1 の場合に 2 に更新する．性質 1 より，AddtoArray は排他的に実行されるので更新 1 が $Added[p]$ を 1 に更新した後に更新 0 が起こらない限り，$Added[p]$ に対してさらに更新 1 は起こらない．更新 2 の後に 8@(Mutual Exclusion) を終了するまで更新 0 が起こらないことは既に示されている．更新 2 は更新 1 の後にしか起こらず，8@p(Mutual Exclusion) を終了するまで更新 0 が起こらないので更新 2 の後に p が 8@p(Mutual Exclusion) を終了するまで更新 1 は起こらない．

従って更新 2 の後に p が 8@p(Mutual Exclusion) を終了するまで更新 0，1 は起こらない． □

補題 5. W に追加されたプロセス p はいずれ待機 2 を終了する．

証明. n を任意の自然数とする．任意の実行において n 番目に W に追加されたプロセスがいずれ待機 2 を終了することを数学的帰納法によって証明する．

$n = 1$ の時，実行の最初に W に追加されたプロセスを p_1 とする．p_1 の前に 2PME(0,0) を実行したプロセスは存在しない．従って W に p_1 を追加したプロセス q は 2PME(0,1) を実行する．q は 9@q(AddtoArray) で $W[0]$ に p_1 を書く．その後，性質 1 より q は DecisionCall まで排他的に処理を行い，5@q(DecisionCall) の実行時に $side = 1$ で かつ $W[N-1]$ は初期値の -1 なので真になり $callflg = true$ となる．その後 q は 25@q(Mutual Exclusion) で $W[0]$ から p_1 を得る．そして 35@q(Mutual Exclusion) で $Added[p_1]$ に 2 を書く．補題 4 より最初に W に追加されたプロセス p_1 はいずれ待機 2 を終了する．

次に $n = k$ の時，実行の k 番目に W に追加されたプロセス p_k はいずれ待機 2 を終了すると仮定する．

$n = k+1$ の時，実行の $k+1$ 番目に W 追加されたプロセスを p_{k+1} とし，p_{k+1} が追加された W のインデックスを i，その 1 つ前のインデックスを i_{pre} とする．性質 1 より AddtoArray は排他的に実行されるので $W[i_{pre}]$ には k 番目に追加された p_k が書かれている．

$W[i]$ に p_{k+1} を書いた時，$W[i_{pre}]$ の値が

(i) -1 である時 (すなわち初期化済みである)

p_{k+1} を W に追加する前に p_k は 11@p_k(DecisionCall) で $W[i_{pre}]$ に -1 を書いている．p_k は W への追加処理である AddtoArray が既に終了しているので p_{k+1} を W に追加せずに exit セクションを終了する．その後，W に p_{k+1} が追加されるのでこの処理を行ったプロセス r が存在し，性質 1 より r が AddtoArray 終了後は r が DecisionCall を排他的に実行する．従って r が 5@r(DesicionCall) を実行した時に $W[i_{pre}]$ が -1 なので真となり，$callflg = true$ となる．r は 25@r(Mutual Exclusion) で $W[i]$ から p_{k+1} を得る．その後，r は 35@r(Mutual Exclusion) で $Added[p_{k+1}]$ を 2 にし，補題 4 より p_{k+1} はいずれ待機 2 を終了する．

(ii) -2 である時 (すなわち $W[i_{pre}]$ に追加されたプロセスは処理の途中である) または，p_{k+1} である時 (すなわち $W[i_{pre}]$ に追加されたプロセスは呼び出される前である)

仮定より k 番目に W に追加されたプロセス p_k はいずれ待機 2 を終了する．p_k は既に呼び出されたので，この時 $Call = i_{pre}$ である．p_k はいずれ 2PME(0) の entry セクションを実行する．2PME(0) は 2PME-YA と同一なのでスタベーションフリーが成立し，p_k はいずれ DecisionCall を実行する．

この時まで 5@(DecisionCall) で真となる他のプロセスが存在しない場合，$Call$ は更新されず，$Call = i_{pre}$ である．p_k は $side = 0$ なので 5@p_k(DecisionCall) が真となり，$callflg = true$ となる．p_k はその後 25@p_k(Mutual Exclusion) で $W[i]$ から p_{k+1} を得て 35@p_k(Mutual Exclusion) で $Added[p_{k+1}]$ を 2 とする．補題 4 より p_{k+1} はいずれ待機 2 を終了する．

一方，5@(DecisionCall) で真となった他のプロセスが存在した場合，そのプロセスの中で最初に DecisionCall を実行したプロセス r が 25@r(Mutual Exclusion) で $W[i]$ から p_{k+1} を得るので 35@r(Mutual Exclusion) で $Added[p_{k+1}]$ を 2 とする．補題 4 より p_{k+1} はいずれ待機 2 を終了する．

よって p_{k+1} はいずれ待機 2 を終了し，W に追加された任意のプロセスはいずれ待機 2 を終了する． □

次に補題 4，5 を用いて待機 2 についての補題を示す．

補題 6. 待機 2 はいずれ終了する．

証明. プロセス p が 8@p(Mutual Exclusion) を実行しているならば 7@p(Mutual Exclusion) が真であった．つまり $Added[p]$ に 1 または 2 の値が書かれた．

$Added[p]$ に 2 が書かれた場合は補題 4 より成立する．

$Added[p]$ に 1 が書かれた場合は $Added[p]$ に 1 を書く処理の前に W に p を追加しているので補題 5 より成立する．

よって補題 6 は成り立つ． □

最後に待機 3 についての補題を示す．ここで使用する補題を示す．参考文献 [5] より

補題 7. 同じ $node$ 値で同時に 2 つ以上のプロセスが待機 3 を実行することはない．

が YA において成り立つ．SIF において待機 3 は Entry2PME で起こり，Exit2PME によって終了する．従って SIF も補題 7 が成り立つ．

補題 8. 待機 3 はいずれ終了する．

証明. 背理法によって証明する．12@(Entry2PME) で待機し続けているプロセスが存在すると仮定する．12@(Entry2PME)

で待機をしている 2PME の中で最小の *node* 値の 2PME を 2PME(min) とする．補題 7 より 12@(Entry2PME(min)) で待機しているプロセスが一意に決まるのでこれを p とする．p は 12@p(Entry2PME(min)) を実行している時，先に T に値を書いた他のプロセス q によって待機させられている．仮定より q は 12@q(Entry2PME(min)) で待機していない．また q がその他の待機処理をしている場合でも補題 3，6 よりいずれ終了することが示されている．従って q はいずれ exit セクションを実行し，q は p の待機しているノードに対して 4@q(Exit2PME(min)) を実行して p のスピン変数 P を 2 に更新する．この処理によって p は待機を終了する．これは仮定に反する．よって矛盾し補題 8 は成立する． □

補題 3，6，8 より SIF はスタベーションフリーである．

定理 2. SIF はスタベーションフリーである．

6. 評　　価

6.1 最悪時評価

SIF で行う各処理における最悪時の RMR 計算量は，プロセスが待ち配列で待機を行ってから木をスキップするまでの処理が $O(1)$，待ち配列への追加処理が $O(logN)$，呼び出し判定処理が $O(1)$ である．よって SIF の最悪の RMR 計算量は YA と同じ $O(logN)$ である．

6.2 平均時評価

提案アルゴリズムは entry セクションを同時に実行するプロセスが増加すると，木をスキップする頻度が増加し計算量を大幅に削減することが可能である．スキップする頻度は相互排除アルゴリズムを同時に実行するプロセスの混み具合に依存する．本節ではこの混み具合を考慮した計算量の評価を行う．

待ち行列理論 [6] を利用して平均時評価を行う．M/M/1(1) 型はシステムのサービス時間，客の到着間隔が共に指数時間分布で，同時にサービスを受けられる人数が 1 人，待ち行列を作らないモデルである．このモデルでの待ち行列理論では客の平均到着密度を λ[個/時間]，システムの平均サービス時間を $1/\mu$[時間/個] とするとシステムのサービスに関して平衡状態時に以下が成り立つ．

サービス中である確率:

$$P_{ocpy} = \frac{\lambda}{\lambda + \mu} \tag{1}$$

サービス中でない確率:

$$P_{empty} = \frac{\mu}{\lambda + \mu} \tag{2}$$

各 2PME から始まるような処理を M/M/1(1) 型のサービスモデルに適用してアルゴリズムの評価を行う．

d） サービスモデル

サービスを各 2PME の各 *side* の entry セクションの 2@(Entry2PME) の実行から exit セクションの対応する 2PME の 4@(AddtoArray) の実行までと定義し，プロセスをそのサービスを受ける対象とする．

葉から始まるサービスについて，平均サービス時間を $1/\mu_1$[時間/プロセス] とする．さらに各プロセス毎の平均到着密度を λ_1[プロセス/時間] とする．プロセスがある 2PME に到着するとはその 2PME のサービスを開始する，つまり Entry2PME の 2 行目を実行することを表す．また葉の 2PME から始まるサービスを終了したプロセスは任意のタイミングで idle セクションの実行を終了し，entry セクションの実行を開始する．2PME のサービス時間は critical セクションの実行時間に依存し，到着間隔は，idle セクションに依存する．本稿ではプロセスの到着，サービスの終了が共にランダム，すなわちプロセスの到着間隔，サービス時間間隔が指数時間分布に従う場合を解析する．

図 9 のように entry セクションを実行するプロセスは最初に 1 段目の葉から始まるサービスを開始し，木をスキップするまで 2 段目以降のサービスを順に開始する．exit セクションにおいてプロセスはスキップを行った段のサービスから葉のサービスまで順にサービスを終了する．

各 2PME では $side = 0, 1$ の 2 つのサービスが開始される．プロセスがある 2PME のサービスを開始する時に相手 *side* のサービスが既に開始されているか否かでプロセスのスキップの有無が決定する．

各段でプロセスが待ち配列に追加される確率をもとめるために各段のサービスモデルの平均サービス時間，平均到着密度をもとめる．これらの値は図 10 のように根から葉まで順に計算を行うことでもとまる．以下にその計算方法を示す．

まず $k(k = 2, ..., logN)$ に対し k 段目の 2PME の到着密度を求める．プロセス p がある 2PME に到着した時，その段の 2PME の相手 *side* のプロセスがまだサービス中でない場合，p はその 2PME で発見されずに $k+1$ 段目の 2PME に到着する．

従って k 段目の平均到着密度の期待値は

$$\lambda_k = 2\lambda_{k-1} P_{empty,k-1} \tag{3}$$

と表される．

次に k 段目の平均サービス時間を求める．$k-1$ 段目に到着してから k 段目に到着するまでの時間 (待ち配列に追加されない時の Entry2PME の平均処理時間) を m_1[時間/プロセス]，k 段目のサービスが終了してから $k-1$ 段目のサービスが終了するまでの時間 (1 つの 2PME に対する AddtoArray の処理時間) を m_2[時間/プロセス] とする．m_1 と m_2 の和をを m[時間/プロセス] と置くと k 段目の平均サービス時間は

$$\frac{1}{\mu_k} = \frac{1}{\mu_{k-1}} - m \tag{4}$$

となる．

式 (3),(4) を葉のサービスから根のサービスへ順に適用し各段の到着密度，平均サービス時間を求め，さらにその値を式 (1),(2) に代入することで各段でプロセスが発見される確率を求めることができる．

$k(k = 1, ..., logN - 1)$ に対し k 段目でプロセスが発見される確率 $P_{stop,\,k}$ をもとめる．k 段目でプロセスが発見される時，

葉から $k-1$ 段目までの 2PME で発見されない．この確率は

$$P_{pass,\ k-1} \equiv P_{empty,\ 1} \cdot P_{empty,\ 2} \cdot \ldots \cdot P_{empty,\ k-1} \qquad (5)$$

となる ($P_{pass,\ 0} = 1$ とする)．さらに k 段目で待ち配列に登録されるので

$$P_{stop,\ k} \equiv P_{ocpy,\ k} \cdot P_{pass,\ k-1} \qquad (6)$$

と表される．

最後に計算量の期待値を求める．k 段目で発見されたプロセスは k 段までの 2PME と 2PME(0) に対する entry セクション，exit セクションの処理を行うので 1 つの 2PME に対する entry セクション，exit セクションの RMR の回数を c[回/プロセス] と置くと k 段目で待ち配列に登録されたプロセスの RMR 計算量は c(k+1) の計算量となる．

また，2PME(1) まで到着したプロセスは計算量が $c(logN+1)$ で一定である．

よって RMR 計算量の期待値は

$$\sum_{k=1}^{logN-1} P_{stop,\ k} \cdot c(k+1) \ + P_{pass,\ logN} \cdot c(logN+1) \quad (7)$$

となる．

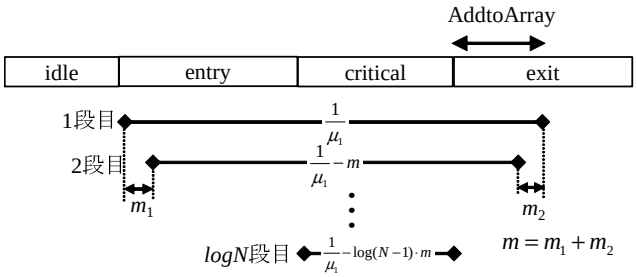


図 9　アルゴリズム SIF と各段のサービスの関係

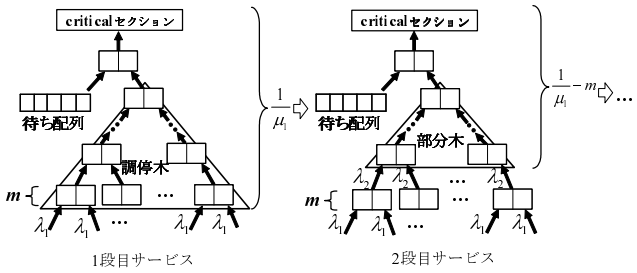


図 10　到着密度，平均サービス時間の再帰的計算

6.3　ケーススタディ

式 (7) を適用した計算量を示す．表 1 の値を基として，いくつかのパラメータを変化させて RMR 計算量の変化の傾向を考察する．

図 11 は平均到着密度を変化させた時の RMR 計算量の期待値の変化である．到着密度が低い時は 17 の計算量が必要であるが，密度が増加するにつれて計算量が減少し，2 に収束している．この結果より到着密度が増加するにつれて計算量が減少することが分かる．これは到着密度が増加することでシステムが混み，プロセスがスキップする頻度が増加したためだと考えられる．

表 1　パラメーター一覧

項目	値
プロセス数	65536
最下段の到着密度 [プロセス/時間]	0.001
システムの平均サービス時間 [時間/プロセス]	1000
m [時間/プロセス]	0.001
c [回/プロセス]	1

図 12 はシステムの平均サービス時間を変化させた時の結果である．平均サービス時間が増加するにつれて計算量が減少することが分かる．これはサービス時間が増加することで相互排除の実行が他プロセスと重なりやすくなり，スキップの頻度が増加したと考えられる．

図 13 はプロセス数を変化させた時の計算量の比較である．到着密度が低い時は各結果は全て $logN+1$ の計算量が必要となっている．一方，到着密度が高い時は各結果が 2 に収束している．これより，プロセスが発見される確率が高い時は計算量の期待値はプロセス数に依存しないことが分かる．

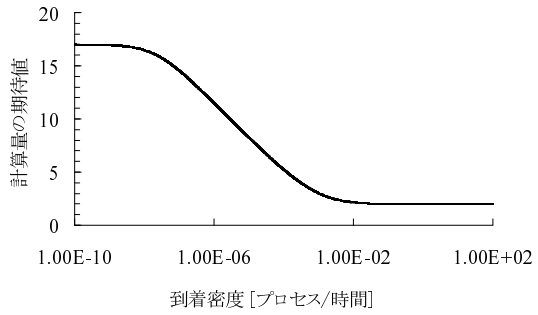


図 11　到着密度による計算量の変化

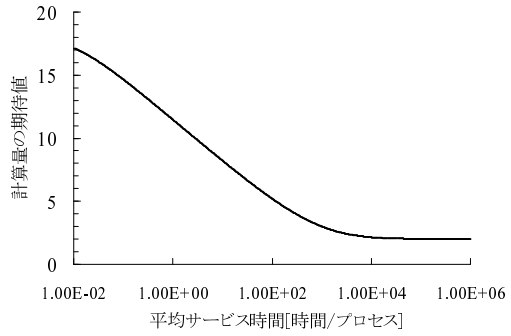


図 12　平均サービス時間による計算量の変化

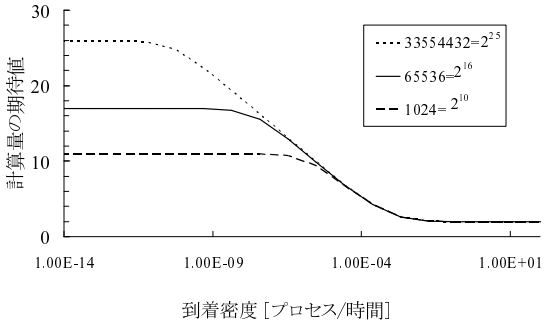


図 13　プロセス数による計算量の比較

7. ま と め

混み具合によって RMR 計算量の期待値が従来アルゴリズムよりも削減可能な相互排除アルゴリズムを提案した．本アルゴリズムはプロセス数が少ない場合は従来と同じ $O(logN)$ の計算量だが，混み合ってくると $O(1)$ に近づいていく．つまり同時に相互排除を実行するプロセスが多い時に有効である．この性質が既存の相互排除アルゴリズムとの大きな違いであり利点でもある．

提案アルゴリズムのシステムが使用する空間計算量は $O(NlogN)$ である．しかし，Kim ら [4] の提案手法を適用可能で適用後の空間計算量は $O(N)$ となる．

本稿では各プロセスの idle，critical セクションの時間，処理速度の平均値が同じ状況での解析を行ったが，今後は様々な状況において本アルゴリズムの有効性を検証する必要がある．また本アルゴリズムと参考文献 [1] のような競合度アダプティブなアルゴリズムを組み合わせて，プロセスの混み具合が少ない時，多い時の両方において高速に計算が可能なアルゴリズムを提案する事が今後の課題である．

文　献

[1] J.H. Anderson and Y.J. Kim. Adaptive Mutual Exclusion with Local Spinning. *Distributed Computing: 14th International Conference, DISC 2000, Toledo, Spain, October 2000: Proceedings*, 2000.

[2] J.H. Anderson and Y.J. Kim. An improved lower bound for the time complexity of mutual exclusion. *Distributed Computing*, Vol. 15, No. 4, pp. 221–253, 2002.

[3] H. Attiya, D. Hendler, and P. Woelfel. Tight RMR Lower Bounds for Mutual Exclusion and Other Problems. *Proceedings of the fourtieth annual ACM symposium on Theory of computing*, pp. 217–226, 2008.

[4] Y.J. Kim and J.H. Anderson. A space-and time-efficient local-spin spin lock. *Information Processing Letters*, Vol. 84, No. 1, pp. 47–55, 2002.

[5] J.H. Yang and J.H. Anderson. A fast, scalable mutual exclusion algorithm. *Distributed Computing*, Vol. 9, No. 1, pp. 51–60, 1995.

[6] 森村英典, 大前義次. 応用待ち行列理論. 日科技連出版社, 1975.

モバイルP2Pネットワークにおける ノードの密度を考慮した資源複製法の検討

長谷川力也，山内由紀子，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

概要　近年モバイルアドホックネットワーク上で Peer to Peer 通信を実現するモバイル P2P ネットワーク (MP2P) が注目されている．MP2P において情報資源の共有は重要なアプリケーションであるが，ノードの移動や端末の通信能力制限のため有線 P2P ネットワークの資源探索手法を適用することは困難である．ノードの移動を考慮した資源探索手法として，資源の複製を各ノードが保持する手法が用いられるが，複製配布時のネットワーク負荷を十分に考慮したものとはいえない．本研究では，通信回数を削減した効率の良い資源共有を実現するため，あるノードからの情報発信が無線通信範囲内のすべてのノードに伝達することに着目し，局所的にノードが密な部分でのみ資源の複製を配布する．任意のノードの密度が把握できるという条件の元で行った予備実験により，密度を考慮した複製を行うことで資源複製の通信回数が削減できることを示す．

1　背景と目的

近年，無線端末の急速な普及により移動無線端末 (以降ノードとよぶ) で構成された P2P ネットワークであるモバイル P2P ネットワーク (MP2P) が注目されている．固定計算機が有線接続された従来の P2P ネットワークと比べ，MP2P では通信帯域や電池寿命など制約が多く大規模な計算には不向きである．MP2P における代表的なアプリケーションとして各ノードの保持しているファイルなどの情報を共有する資源共有システムがある．資源共有システムにおいては，各ノードに分散配置された資源のうち特定の資源を見つけ出す探索問題が重要な問題であり，P2P ネットワークにおいてさまざまな手法が提案されている．MP2P においては，さらにノードの移動に伴うネットワークトポロジの更新やネットワークの切断を考慮する必要があり，有線 P2P で有効であった手法が適用できるとは限らない．[1] ではネットワークの切断に対処するため，あるノードが要求して特定のノードから獲得した資源は複製され，一定の複製周期ごとに周囲のノードが参照し必要に応じて複製を作成する．これによりノードの移動によるネットワークの切断が発生しても高い確率で資源を獲得することができる．しかし [1] では，資源の複製は要求時にまず作成されるため，あらかじめ複数のノードに複製を配布しておくことで資源要求時の転送コストを抑え，より効率的に複製を配布することが可能であると考えられる．

本研究では，資源要求時の資源転送コストを低下させるため，資源複製の配布回数を削減しつつ多くのノードに複製を保持させる．都市部など場所に応じてノードの密度が異なる環境 [2] を想定し，ノードの密度が高い場所でブロードキャスト通信を用いることで一度の通信で多くのノードに転送し資源複製のコストを抑えつつ資源獲得確率を高める．

2　システムモデル

ノード p は自身を中心とする半径 R の円内に存在するノードと通信可能である．R を無線通信距離とよぶ．各ノードの無線通信距離は一定であると仮定する．ノード p の無線通信距離内に存在するノードを p の隣接ノードとよび，p の隣接ノード数 N_p を p の密度とよぶ．ノード p が隣接ノードと通信する際には，無線通信距離内の特定の 1 ノードとのみ通信するユニキャスト通信と，無線通信距離内のすべてのノードと通信するブロードキャスト通信がある (図 1)．ノード p から隣接ノードへのユニキャスト送信とブロードキャスト送信に消費電力の差はないと仮定する [3]．ノードは HELLO パケットとよばれるサイズの小さいパケットを用いて隣接ノードの存在を確認する．HELLO パケットの転送は瞬間的に隣接ノードに届くものと仮定する．

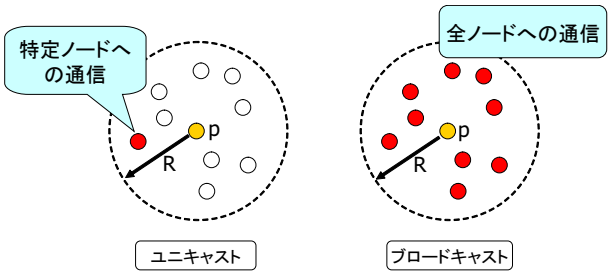


図 1: ユニキャストとブロードキャスト

3　資源複製手法

各ノードはネットワーク中を移動しながらノードの密度ができるだけ高くなる部分を探し，一定の複製周期ごとに，現在の場所の密度が高いと判断した場合自身の持っている資源を隣接ノードに送信し資源を複製させる．密度の高低の判断には自身の付近の密度情報と，ネットワーク全体の密度情報を引数とする関数を閾値として用いる．

3.1 密度の推定

HELLO パケットの利用

各ノードはネットワーク中を移動しながら t 秒ごとに HELLO パケットを送信しあうことで密度を把握する．各ノードが密度の変化を記録しておくことで現在の位置が密度の高い場所か低い場所かを判断する．十分長い時間が経過すると，ノードはネットワークの広い範囲での密度を記録できるのでこれを用いてネットワーク全体の密度を計算し，複製の閾値として用いる．ノード p が時刻 $[0,t]$ に受信した HELLO パケットの数を $N_p(t)$ とするとき，ノード p が計算するネットワーク全体の推定密度を

$$deg_p = \frac{N_p(t_1) + \cdots N_p(t_m)}{m}$$

と定義する．このとき m はノード p がネットワークの広い領域を移動するために十分大きい値とする．

k ホップ先情報の利用

HELLO パケットを利用する方法では，各ノードが計算したネットワーク全体の推定密度が実際のネットワークの密度，すなわち各ノードの密度の平均値に近づくためにはノードが広い領域を移動するまでに長い時間を必要とする．そこで各ノードが計算した密度を数ホップ先まで転送することで，各ノードは自身から数ホップ離れたノードの密度を把握することができるので，ノードの移動を待たずにネットワークの広い範囲の密度を計算できる．ノード p が k ホップ先から時刻 $[0,t]$ に受信した密度情報を含む HELLO パケットの数を $N_p^k(t)$ とするとき，ノード p における時刻 t でのネットワーク全体の推定密度を

$$deg_p(t) = \frac{N_p^1(t) + \cdots N_p^k(t)}{k \text{ ホップ以内のノード数}}$$

と定義する．そしてネットワーク全体の推定密度を

$$deg_p = \frac{deg_p(t) + \cdots deg_p(t_m)}{m}$$

と定義する．このとき m は HELLO パケットの転送ホップ数 k に応じて変化させる必要がある．つまり k が十分に大きいとき m は小さい値でよいが k が小さいときは 1 ホップの HELLO パケットを用いたときと同様に m をある程度大きな値とする必要がある．HELLO パケットの転送ホップ数による差を表 1 に示す．

3.2 資源複製

各ノードは一定の複製周期 $T(>t)$ ごとに自分の密度とネットワーク全体の密度を比較し，現在の場所がネットワーク中で密度が高い場所と判断したときのみ隣接ノードに資源を転送し複製する．これにより，密度を考慮せずに資源を転送した場合に比べ一度の転送回数で多くのノードに資源の複製を保持させることができる．ノード p では T 秒ごとにネットワーク全体の推定密度 deg_p と自身の密度 N_p を比較し，(1) $deg_p < \delta(N_p)$ が成り立つときのみ資源を複製する．この条件により各ノードに資源を配布するための通信回数の削減を実現する．δ は閾値を決定するための関数でありネットワークに応じて適切に設定する必要がある．

表 1: HELLO パケットの転送ホップ数による差

	1 ホップ	k ホップ
コスト	小	大
推定時間	大	小
ノードの移動	必要	不要

3.3 複製の抑制

複製周期ごとに式 (1) が成り立つかどうか確認し，成り立つ場合常に複製を続けると十分な時間が経てば全てのノードに資源の複製が作成されることになる．資源へのアクセス可能性を高めるためにはある程度の複製が作成されれば十分であり，適切な段階で資源複製を抑制する必要がある．式 (1) に加える複製の条件として以下が考えられる．

- ホップ数による制限
  各ノードが保持している資源の情報を HELLO パケットに含めて転送する．各ノードは自分の持っている資源の複製が k ホップ先まで存在しないときに限り複製を作成する．
- ネットワークの領域被膜による制限
  資源の複製を持っているノードの無線通信範囲に，ネットワーク全体のノードのうちどれくらいのノードが存在するか計算し，一定の割合を超えると複製を停止する．

4 評価指標

資源複製の有効性を計る評価指標として以下のものを用いる．

- 資源獲得コスト
  資源を要求したノードから資源を保持しているノードまでの最短ホップ数
- 資源獲得確率
  資源を要求したノードのがどれくらいの割合で資源を獲得できたか
- 資源が複製されるまでにかかる時間

5 シミュレーション実験

密度を考慮した資源複製手法が有効であることを検証するため，特定のノード s が単一の資源をネットワーク全体に配布する状況を設定し，複製回数に応じて資源の複製を受け取るノード数の変化

を確認するシミュレーション実験を行った．実験では任意の時間での各ノードの密度が把握可能であると設定し，複製周期ごとに，常に複製を実行した場合と，式 (1) が成り立つ場合のみ複製を実行する場合の 2 通りを比較した．

5.1 シミュレーション設定

各ノードはランダムウェイポイントモデルに従って移動するものとし，フィールドサイズ $1000m \times 1000m$，ノード数 300，$R = 100m$，平均速度 $5.5m/s$，停止時間 0 秒，資源複製の実行周期を平均 10 秒とした．密度を考慮した複製では，ノード s の密度 N_s が，全ノードのうち密度の高いものから 20% のノードの密度の平均よりも大きいときのみ資源複製を実行するものとした．

5.2 実験結果

図 2 に複製回数に応じたノードの資源獲得率を示す．密度を考慮していない場合 95% のノードが資源を獲得するために 240 回の資源複製が必要であるが，密度を考慮した場合は 74 回の資源複製が必要であり，複製回数が削減できていることがわかる．

図 3 に経過時間に応じたノードの資源獲得率を示す．95% のノードが資源を獲得するためには，密度を考慮した場合，密度を考慮していない場合に比べて倍程度の時間が必要となり，即時性の求められる状況では密度を考慮した複製が十分に有効とはいえないことがわかる．

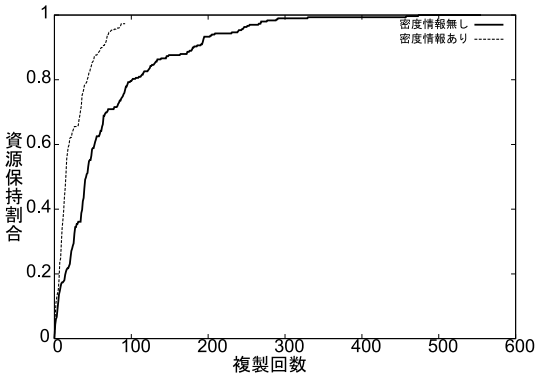


図 2: 複製回数による資源獲得率の変化

6 まとめと今後の課題

本稿では，MP2P における資源複製においてノードの密度を考慮した資源複製法を検討した．シミュレーションにより，提案手法を用いることで必要な時間は増えるが資源複製の回数が削減可能であることを示した．

今後の課題として，複数の資源を考えた場合の資源獲得率および資源獲得コストによる評価が必要

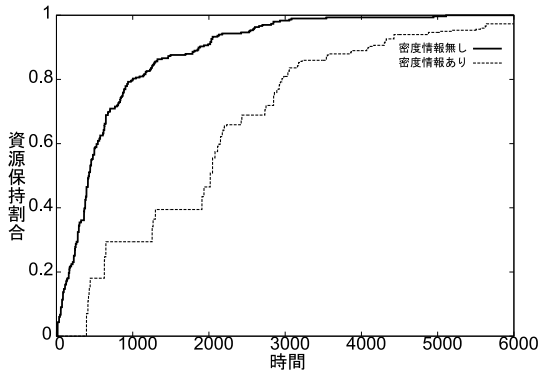


図 3: 経過時間による資源獲得率の変化

である．資源獲得コストによる評価を考えると，十分に複製された資源よりもあまり複製されていない資源を優先的に複製する必要があるため，密度情報以外にも各ノードが保持している資源の情報なども閾値として利用する必要があると考えている．また現状では密度の推定が正しく行えているという前提での実験しか行えていないため，HELLO パケットを用いた密度推定の可否の検証，ネットワークの状態に応じて HELLO パケットの転送ホップ数の変更などが必要であると考える．

7 謝辞

本研究の一部は，文部科学省グローバルＣＯＥプログラム（研究拠点形成費）の研究助成，日本学術振興会科学研究費補助金（基盤研究 (B)19300017, (B)17300020, 20300012，若手研究 (B)18700059），および，（財）栢森情報科学振興財団助成金によるものである．

参考文献

[1] T. Hara. Effective replica allocation in ad hoc networks for improving data accessibility. *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 3:1568–1576 vol.3, 2001.

[2] S. Bohacek and V. Sridhara. The graphical properties of MANETs in urban environments. *The Forty-Second Annual Allerton Conference on Communication, Control, and Computing*, 2004.

[3] L.M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 3:1548–1557 vol.3, 2001.

Acceleration of Byzantine Fault Tolerance by Parallelizing Agreements

Junya Nakamura[†], Tadashi Araragi[††], and Shigeru Masuyama[†††]

[†] Graduate School of Information Science and Technology,
Osaka University, Japan
[††] NTT Communication Science Laboratories,
Nippon Telegraph and Telephone Corporation, Japan
[†††] Department of Knowledge-based Information Engineering,
Toyohashi University of Technology, Japan

Author's e-mail address: junya-n@ist.osaka-u.ac.jp

Abstract

We propose a new method for Byzantine Fault Tolerance(BFT) on asynchronous distributed systems. BFT realizes a reliable system against Byzantine failures and is usually solved by executing agreements for requests repeatedly. Our method parallelizes the agreement process and introduces a new agreement phase for processing order of agreed requests. We show the correctness of our method and compare the performance with an existing method and a simple parallelizing method analytically. The result indicates the simple parallelizing method has poor performance and our method has good performance under highly-loaded environments.

Keywords: Byzantine Fault Tolerance, state machine replication, Byzantine agreement, asynchronous

1 Introduction

Recently, several kinds of network services on the Internet, such as e-commerce and e-government, have emerged, and further development of these services is demanded. On the other hand, crackers' attacks and software errors, called Byzantine failure, in such services have become more serious. If an attack against a critical service that handles personal data or bank account information succeeded, this could cause severe damage to users. Meanwhile, the software size of such services is large and the system logic is complicated, and thus there is a high possibility that the service has software errors. Therefore, it is important to make a distributed system for these services robust and to avoid Byzantine failures.

Server replication is one of the most effective approaches to solve this problem. In the replication approach, the behavior of the server is assumed to be determined by the currently processed request and the series of requests processed before it. Then, some server replicas are run individually on different host machines, and clients multicast their requests submitted to the original server to all of the replicas. Since the orders of arriving requests are different among the replicas, they have to reach agreement on the order of processing the requests to synchronize their behaviors. Then, even if some replicas become faulty due to a cracker's intrusion, the other replicas can mask the undesired effect.

In this paper, we propose a new method to accelerate the server replication by parallelizing agreement on the order of processing requests.

In general, agreement on the order of processing requests is achieved by agreement for a set of requests (this approach is called "destination agreement" in [6]). Replicas agree with a set of requests submitted by clients and repeat the agreement until the service is terminated. All requests sent by clients are eventually contained in an agreed set by some mechanism. When a replica agrees with a set of requests, it arranges the requests in a lexicographical order and processes them accordingly. For Byzantine agreement on a set of requests, many fast protocols have been proposed [5, 9, 3].

We accelerate the replication by concurrently executing agreements on a set of requests, where we treat the existing fast agreement protocols on a set of requests as black boxes. A naive approach could run the agreement protocols in parallel and process their agreed sets in the order they were started. However, the performance would not be good because a replica has to wait for all previous agreements to finish before it processes the agreed set. In our parallelizing method, replicas make another agreement on which agreed sets are processed first without waiting for all of the agreements to finish.

Our proposed method shortens the interspaces between the time when some agreed set is decided and the time when the next agreed set is decided. In the asynchronous distributed systems we are aiming at, like the Internet, each communication link's speed or each replica's processing speed is different, and the timing of finishing the corresponding agreement can greatly differ among the replicas. In such an environment, our method has a great effect on acceleration. In this paper, we prove that our method satisfies the correctness of replications and show its efficiency, based on theoretical analysis, compared with an ordinary sequential method (non-parallel) and a naive parallel method.

1.1 Related Work

Few attempts have been made to improve replication performance by running multiple agreement protocols in parallel, as our method does.

ABC[3] and RITAS[5, 9] are fast agreement protocols for a set of requests, which are instances of a black box in our method, and we abstract their characteristics and employ them for performance evaluation of our method. These protocols are designed for asynchronous distributed systems and can resist Byzantine failures. They decompose one agreement for a set of requests to multiple binary agreements. ABC theoretically realizes fast agreement at the cost of a high-level cryptosystem. In fact, the expected number of rounds to reach agreement is constant in ABC. RITAS doesn't employ a high-level cryptosystem, and the expected number of rounds is $O(f)$, where f is the maximum number of possible Byzantine failure systems. The number of binary agreements employed in an agreement protocol for a set of requests is an important factor for evaluating our method.

Previous works[1, 2, 10, 4] have explored asynchronous binary Byzantine agreement protocols used for set-of-requests agreement protocols. The binary agreement protocols toss a coin per round and solve agreement by repeating rounds. These protocols are classified into two classes based on how the coin is tossed: shared coin toss[10, 4] or local coin toss[1, 2]. Although the shared coin toss scheme can reach agreement with probability 0.5, it needs a high computation cost for the cryptosystem. On the other hand, the local coin toss scheme doesn't need a cryptosystem, but the probability is very low at $2^{-(n-f)}$. The superiority in terms of total performance by either of these two approaches depends on the environment[8].

1.2 Organization

This paper is organized as follows. The following section defines the system model and the target problem, and explains the existing replication approach using agreement protocols. Section 3 presents our proposed parallelized method for replication and section 4 proves its correctness. Section 5 analytically evaluates three methods; the non-parallel method, the naively parallelized method, and the proposed parallelized method. Finally section 6 concludes the paper.

2 Model

2.1 System Model

We use an asynchronous distributed systems model, where each process's CPU power and speed of communication links between processes are unbounded. Furthermore, a process cannot know when a reply to its messages arrives. A process has a local clock, but their clocks are not synchronized, i.e. every process's clock indicates a different time.

There are finite processes in a distributed system. Every pair of two processes is connected by communication links directly, and a process can exchange information only by message passing. We assume that communication links are reliable channels, i.e. messages sent by non-faulty processes must be delivered to destination processes eventually, and no such message is lost. A process can identify the originating process of a delivered message, and even a malicious process cannot send an impersonating message to a non-faulty process.

Some processes may fail during protocol execution. Here, we adopt the Byzantine failure model, the most general failure model. A Byzantine process can behave in a way that arbitrarily deviates from protocol specifications, e.g. stop processes, omit messages, and submit invalid messages. Some Byzantine faulty processes can collaborate with others to break down a server system. In this paper, we assume that clients are always non-faulty and only replicas can fail. Replicas behaving based on protocol specifications are called "correct replicas," while other replicas are called "faulty replicas." Hereafter, we assume that at most f replicas may fail.

2.2 State Machine Replication

Here, we describe our target problem, i.e. state machine replication. State machine replication is an approach to realize fault tolerance for the server in a server/client model. A server is expressed as a state machine, and the internal state of a state machine is determined by the initial state, processed requests, and the processing order of the requests deterministically. The role of a server is replicated to n replicas, and they run the state machine on distinct hosts independently.

A client submits a request to all replicas to require the server to execute certain tasks. When a replica executes the request, it then replies to the client with the execution result. The client accepts the result when it receives the $f+1$ same results from different replicas. This is because, since at most f replicas can fail, f invalid results possibly arrive. A replica confirms that there is at least one true result submitted by a correct replica when it collects the $f + 1$ same result.

A protocol that solves state machine replication must satisfy the following two requirements;

Safety All correct replicas process all requests submitted by clients in the same order.

Liveness A client eventually accepts the result of any request it has submitted.

2.3 Replication by Agreement for a Set of Requests

This section explains the existing approach that realizes replication by employing an agreement protocol for a set of requests and shows what requirements the agreement protocol should satisfy for the replication.

A replica attributes an identifier to each agreement by increasing numbers when the agreement protocol starts. With this identifier, a replica controls the order of processing agreed sets of requests. Every replica maintains a set of unprocessed requests, ur, and it adds a newly arriving request to ur. Replication is done as follows:

1. Start an agreement protocol of ID i with an input ur as a candidate of the agreement.
2. When the agreement of ID i terminates, process the requests in the agreed set that have not yet been processed, in lexicographical order.
3. Remove the processed requests from ur.
4. $i := i + 1$.
5. Go back to 1.

To guarantee the requirements of state machine replication, the agreement protocol for "request set consensus" must satisfy the four requirements listed below. The input and the output of the protocol is a set of requests, respectively. Hereafter, we denote an execution of the agreement protocol for ID i, whose input is v, by $RSC(i, v)$.

RSC Agreement All correct processes output the same set of requests.

RSC Validity The output set is a subset of the union of the inputs of all correct processes.

RSC Integrity A request contained in the inputs of all correct replicas is also contained in the output.

RSC Termination Every correct process eventually outputs a set of requests.

We call the period from when an execution of the agreement protocol begins to when the next execution begins an "interval."

3 Parallelizing Agreements for a Set of Requests

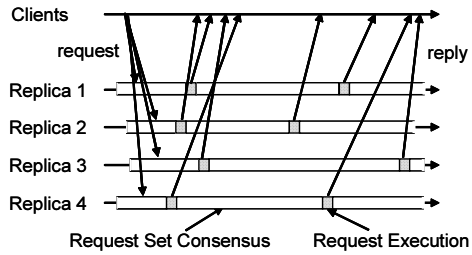
3.1 Overview

The idea of our proposed parallelizing method is simple. In our method, each replica maintains an *interval* with a timer, and when the timer expires, it runs a new execution of the agreement protocol. When an execution of the agreement protocol finishes, it starts a multi-valued agreement protocol whose input is the ID of the agreement just finished to determine the processing order among the agreed sets. The requirement of the multi-valued agreement protocol is shown in section 3.2. Note that, since because of asynchronous environment, the finishing time of the agreement protocol of the same ID may be different among replicas and the processing order may not be equal to the order of ID numbers.

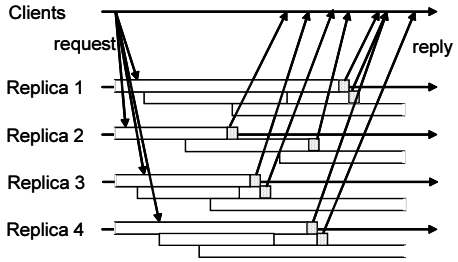
Three executions of replication based on different methods: the ordinary non-parallelized method, the naively parallelized method, and our proposed parallelized method are illustrated in figure 1 (a)-(c). The non-parallelized method runs only one execution of agreement at a time in a replica, and there are few chances to process requests and the processing efficiency is low, when a large number of requests are issued at the same time. The naively parallelized method runs multiple agreements concurrently, but it cannot process requests until previously started agreements are finished. The proposed parallelized method runs multi-valued agreement to decide the processing order of an agreed set when an agreement for a set of requests finishes. By executing multi-valued agreement, all correct replicas can process the requests in the agreed set consistently. Then, the chance to process requests grows and it can realize efficient replication.

3.2 Multi-valued Agreement Protocol

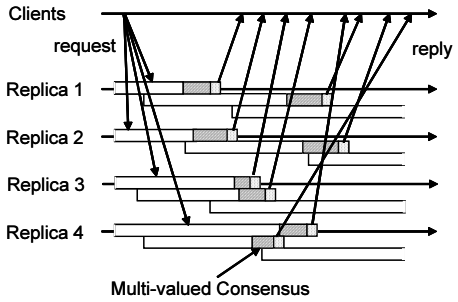
A multi-valued agreement protocol used in our parallelized method is a randomized protocol that gets a multi-value as input and outputs a multi-value. The domain of the input and the output value is a countably infinite set. A replica chooses an element of it and proposes the element as an agreement candidate. Hereafter, we denote an execution of the multi-valued agreement protocol for ID i, whose input is



(a) Non-parallelized method



(b) Naively parallelized method



(c) Proposed parallelized method

Figure 1: Execution examples of three methods

v, as $MVC(i, v)$. We assume that the agreement protocol satisfies the following three properties;

MVC Agreement All correct processes output the same value.

MVC Validity The output value is an input of some process.

MVC Termination Every correct process eventually outputs the agreed value.

3.3 Proposed Method

The proposed parallelized method is shown in algorithm 1. ur is a set of unprocessed requests arrived from clients. When a new request arrives from a client, lines 15-17 are executed and the request is added to ur. ur is also used as a candidate for RSC(request set consensus protocol), and requests contained in some agreed sets are removed from ur. pr is a set of previously processed requests, and it is used to prevent requests from being executed again. AV and SN are arrays of agreed values of RSC and agreed values of MVC(multi-valued consensus) respectively. rsc_id_queue is a queue used to store the ID of RSC, whose agreed set has not yet been executed, and the front of it is proposed as an agreement candidate for MVC.

The protocol consists of two parts: the first part for executing the agreement protocol (lines 18-38) and the second part for processing requests, and these are executed independently. The second part simply repeats the processing of requests in the agreed set of RSC whose ID is $SN[sn]$, from $sn = 1$ in sequence.

The execution of RSC is controlled by intervals, and when timer T expires and ur is not empty, a replica begins a new execution of RSC. The length of an interval, i.e. the time set to timer T, is a parameter denoted as I. The shorter I, the more executions of RSC run in parallel. When some agreed set is decided, all elements of the agreed set are removed from ur. This is because the request is contained in some agreed set, which guarantees that the request must process eventually. When the RSC

whose ID is *nowid* finishes, *nowid* is added to *rsc_id_queue* and starts a new execution of MVC. Since the input of MVC is the front of *rsc_id_queue*, possibly the front is not equal to *nowid*. Here the replica repeats agreement until the output of some execution of MVC obtains *nowid*. By doing this, the proposed method guarantees that the agreed set $AV[nowid]$ must be processed.

Algorithm 1 Proposed parallelized method

```
 1: begin initialization
 2:     ur := ∅; {set of unprocessed requests}
 3:     pr := ∅; {set of processed requests}
 4:     AV := empty; {array of integer, initially each element is ⊥}
 5:     SN := empty; {array of integer, initially each element is ⊥}
 6:     rsc_id_queue := empty; {queue of RSC ID}
 7:     rscid := 1; {ID for RSC}
 8:     mvcid := 1; {ID for MVC}
 9:     sn := 1; {sequence number}
10:     start timer T;
11:     start task T1;
12: end initialization
13:
14: when a request r arrives then
15:     ur := ur ∪ {r};
16: end when
17:
18: when ur ≠ ∅ and timer T expires then
19:     start timer T;
20:     nowid := rscid;
21:     rscid := rscid + 1;
22:     execute RSC(nowid, ur);
23:     let the agreed value be rs;
24:     ur := ur \ rs;
25:     AV[nowid] := rs;
26:     enqueue nowid into rsc_id_queue;
27:     while ∀i, SN[i] ≠ nowid
28:         cid := the front of rsc_id_queue;
29:         nid := mvcid;
30:         mvcid := mvcid + 1;
31:         execute MVC(nid, cid);
32:         let the agreed value be tid;
33:         if rsc_id_queue contains tid then
34:             remove tid from rsc_id_queue;
35:         end if
36:         SN[nid] := tid;
37:     end while
38: end when
39:
40: task T1
41:     loop
42:         sn := sn + 1;
43:         wait until SN[sn] ≠⊥;
44:         target := SN[sn];
45:         wait until AV[target] ≠⊥;
46:         RS := AV[target];
47:         for all r ∈ RS \ pr in some deterministic order do
48:             execute the request r and reply the result to the client;
49:         end for
50:         pr := pr ∪ RS;
51:     end loop
52: end task
```

4 Correctness

Here we show that the proposed parallelized method satisfies the requirements of state machine replication, i.e. Safety and Liveness.

Safety. The processing order of requests is determined by agreed values of RSC and MVC. The RSC Agreement and MVC Agreement guarantee that the values of all agreed values are the same among all correct replicas, and RSC Validity ensures that all agreed values of RSC do not contain any invalid requests. RSC Termination and MVC Termination also ensure that each execution of RSC and MVC terminates eventually. Therefore, Safety is satisfied. □

Liveness. Assume there exists a request r that has never been processed. Such a request is eventually delivered to all correct replicas, and there must be an execution of RSC where every correct replica contains r in its input. Let the ID of the execution be id. By RSC Integrity, the agreed set must also contain r. Then id is added to rsc_id_queue and MVC begins. In this phase, if id does not get some agreed value of MVC for a long time, the front of rsc_id_queue of all correct replicas gets id. By MVC Validity, the agreed value of such execution also must be id. Accordingly, this ensures that r is processed eventually. □

5 Evaluation

5.1 Evaluation Model

In this section, we introduce the analytical model on which our evaluation of the method is based on. In the analysis, we target any given request r and analyze the time L from when r arrives at some replica to when replicas process r. Although the arrival time of the request r or the finishing time of the agreement protocol may be different among replicas, we suppose the difference is very small and ignore it. Therefore, We don't have to discuss the configuration of individual replica but consider only the configuration of the entire system.

In an asynchronous distributed model, agreement problem cannot be solved deterministically, even if only one crash process exists[7]. Therefore, the agreement protocols we assume are randomized ones, and their running time is a random variable.

Existing agreement protocols for a set of requests realize one agreement for the set of requests by decomposing to multiple binary agreements. For instance, RITAS executes $f+1$ binary agreements, and ABC executes a constant number of agreements. Let t be how many times a binary agreement protocol is executed.

An execution of most binary agreement protocols[5, 9, 3] consists of rounds, and each round involves one coin-tossing. If this coin-tossing succeeds, the execution reaches agreement and it can terminate. Here we assume such a protocol, where we let r be the length of one round and p be the probability to reach agreement in a round. The expected running time can be calculated by p and R, and, by using this, the expected running time of RSC (denoted as E) and MVC (denoted as M) are also calculated. I denotes the length of interval of the naively parallelized method and the proposed parallelized method, and m denotes the average number of concurrently running $RSCs$. m is not an independent parameter and is expressed by the ratio of E and I ($m = E/I$).

Our evaluation compares the three methods, i.e. the non-parallelized method,

the naively parallelized method, and the proposed parallelized method. We assume that all correct replicas receive the request r at time T, and from here we start the discussion. By RSC Integrity, the output of the execution of RSC that is initiated next must contain r. In the analysis of these methods, the length from T to when the next execution of RSC is initiated (denoted as A) and the length from when the execution finishes to when the request r is processed (denoted as B) are important factors. Hereafter, we explain how A and B are expressed in these methods.

The non-parallelized method models an execution of the existing replication approach described in section 2.3. A replica runs only one instance of an agreement protocol, i.e. it doesn't run in parallel. When an agreed set is decided, it processes the requests contained in the set that have not yet been processed in lexicographical order. Then, it begins the next run of the agreement protocol whose input is a set of requests that have not yet been processed. This process is executed repeatedly.

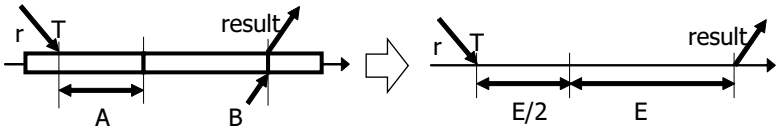


Figure 2: Modeling of the non-parallelized method

The execution of the non-parallelized method is modeled as shown in the left-side of figure 2. Since it runs RSC sequentially, A is half the expected running time of RSC. It can process requests immediately, and $B = 0$.

The naively parallelized method is similar to the proposed parallelized method. The difference is how it works when an agreed set is decided. The proposed parallelized method decides the processing order of the agreed set by executing MVC; on the other hand, the naively parallelized method simply waits until all previously initiated agreements terminates. Then, the order of processing the agreed sets coincides with the increasing order of IDs of agreement, which is commonly shared among replicas.

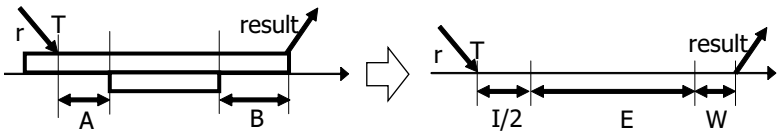


Figure 3: Modeling of the naively parallelized method

The execution of the naively parallelized method is modeled as shown in figure 3. The method begins a new execution of the agreement protocol when an interval ends, and A is equal to $I/2$. B is the time when the last execution initiated before the target execution finishes (denoted as W, see appendix for the details of W).

The proposed parallelized method is explained in section 3. The execution is modeled as shown in figure 4. A is the same as the naively parallelized method. B is the length taken by MVC initiated when an execution of RSC finishes. If we assume that the target id gets the agreed value of the first MVC execution, B is M.

5.2 Results

The parameters used in the evaluation are the probability to reach agreement in one round in binary agreement, p, how many binary agreements RSC is composed of, t, the expected running time of RSC, E, and the length of interval, I. However, since

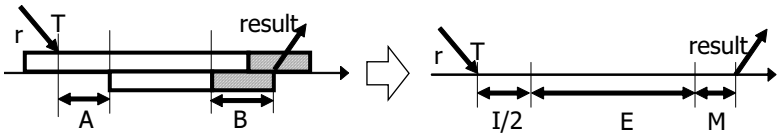


Figure 4: Modeling of the proposed parallelized method

only the ratio of E and I is used in the performance evaluation, we use the parameter m instead of E and I. Thus we consider p, t, and m. The measure of the evaluation is the sum of A (the length from T to when the next execution of RSC is initiated) and B (the length from when the execution finishes to when the request r is processed).

First, we observe the influence of t. p is fixed at 0.3. The results where $t = 2, 3, 4$ are shown in figure 5.

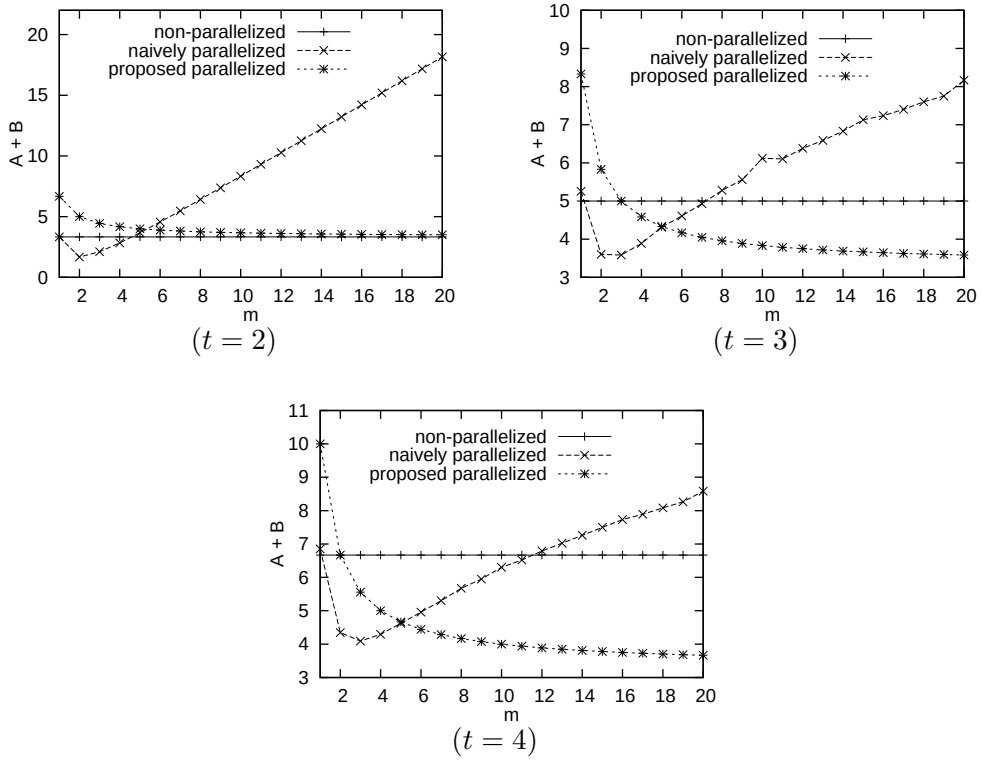
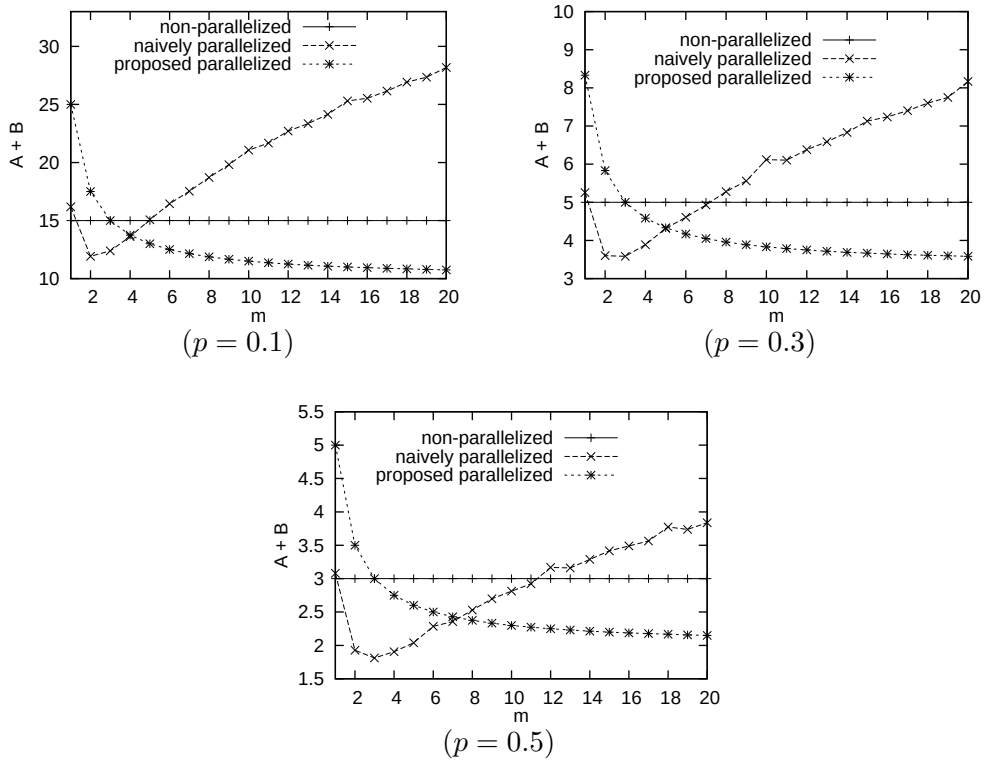


Figure 5: Influence of t

The more binary agreements RSC is composed of, the better the performance of the naively parallelized method and that of the proposed parallelized method become. In all cases where t is more than 2, the proposed method is the best and this tendency gets clearer with increasing t. The naively parallelized method performs poorly when m is large. This indicates that a simple and easy parallelization cannot realize fast replication, which requires more complex techniques like the proposed method. The fact that m is large means that I is relatively small with E, or E is relatively large. The former case is where a request load is high, and the latter case is where it uses the protocol whose one round is long.

Next, we observe the influence of the probability p when t is fixed at 3. Figure 6 shows the results where $p = 0.1, 0.3, 0.5$. The size of p does not change the relative order of the non-parallelized method and the proposed parallelized method. The naively

parallelized method shows a good performance with a large p, and the area where the naively parallelized method has the best performance also increases. Nevertheless, the proposed parallelized method is the best when m is large.



($p = 0.1$) ($p = 0.3$) ($p = 0.5$)

Figure 6: Influence of p

5.3 Summary

The proposed parallelized method has better performance than the non-parallelized method when the request load is high. In particular, in the cases where RSC is composed of 3 or more binary agreements and where the average number of concurrently running $RSCs$ is large, the performance is very good. However, the simple and easy parallelization cannot realize sufficient performance, and performance degrades as the number of concurrently running $RSCs$ increases.

For example, in the case of RITAS, t becomes large in proportion to the number of possibly faulty replicas. The proposed parallelized method is suitable for such large distributed systems.

The performance of the proposed method is not affected by the probability of reaching agreement, p. There are two approaches for binary agreement; shared coin toss and local coin toss. Shared coin toss realizes $p = 0.5$, and its theoretical performance is the best. But it needs a cryptosystem with a high computational cost. Although the probability of local coin toss is $2^{-(n-f)}$ and thus poor, it can be used at low computational cost. The evaluation of these schemes was done by Moniz[8]. As our results indicate, the proposed parallelized method can realize good performance irrespectively of p.

6 Conclusion

In this paper, we proposed a method to increase the efficiency of processing requests. This improvement was accomplished by an approach that has not been previously attempted. The proposed method tries to increase efficiency in two ways; 1) shortening the time until some agreed set is decided by executing an agreement protocol for a set of requests in parallel, and 2) decreasing the waste time until a replica is able to process requests by changing the processing order of agreed sets dynamically. The proposed method can use existing agreement protocols without any modification, and a user can easily realize good performance with little implementation cost.

The correctness of the proposed method was proven in section 4 and the analytical evaluation was done in section 5. The evaluation shows that the proposed method has good performance under high request load or large distributed systems, e.g. a system that has to withstand many faulty replicas. On the other hand, the performance of the naively parallelization was poor, and this indicates the need for more complex techniques like the proposed method.

Acknowledgement

The authors would like to thank Prof. Masuzawa and Prof. Kakugawa of Osaka University for meaningful discussions for improving this paper.

References

[1] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, New York, NY, USA, 1983. ACM Press.

[2] G. Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, New York, NY, USA, 1984. ACM Press.

[3] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. *Lecture Notes in Computer Science*, 2139:524+, 2001.

[4] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 123–132, New York, NY, USA, 2000. ACM Press.

[5] M. Correia, N. F. Neves, and P. Verissimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82–96, 2006.

[6] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, 2004.

[7] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[8] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. Experimental comparison of local and shared coin randomized consensus protocols. In *Proceedings of the 27th IEEE Symposium on Reliable Distributed Systems. Leeds, UK, October 2006*, 2006.

[9] H. Moniz, N. F. Neves, M. Correia, and P. Veríssimo. Randomized intrusion-tolerant asynchronous services. In *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pages 568–577, jun 2006.

[10] M. O. Rabin. Randomized byzantine generals. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 403–409, 1983.

A Modeling Executions of Agreement Protocols

A.1 Agreement Protocol for a Set of Requests

An agreement protocol for a set of requests (denoted as RSC) is composed of t binary agreement protocols. The round number of one binary agreement relies on geometrical distribution, and the probability to reach agreement at round x is $P(x) = p(1-p)^{x-1}$ for $x \geq 1$. Since the expected round number is $1/p$, the expected running time is R/p. Then the probability distribution of RSC is the sum of t geometrically distributed random variables, which is $P_t(x) = {}_{x-1}C_{t-1}\ p^t(1-p)^{x-t}$ for $x \geq t$. The expected running time is $E = tR/p$.

The wait time W of the naively parallelized method is calculated as follows. Here we assume that $m+1$ $RSCs$ are running concurrently, and the i-th execution was initiated at the initiated time of $(i-1)$-th execution plus interval I, for $1 < i \leq m+1$. We also suppose that the agreed set of the $(m+1)$-th execution contains the target request at the first time, and we estimate the wait time in this case. When the k-th execution has just finished at round x, $P(k,x)$, the probability where it is the last execution among m executions, is expressed as the product of the probability that the other executions have finished before,

$$P(k,x) = \prod_{l=1,l\neq k}^{m} \sum_{i=1}^{x+\lfloor (k-l)I/R \rfloor} P_t(i). \tag{1}$$

When such a k-th execution finishes at a round n and the $(m+1)$th execution finishes at a round i, the wait time of this case is

$$kI + nR - ((m+1)I + iR). \tag{2}$$

Then W is expressed as

$$W = \sum_{k=1}^{m} \sum_{i=1}^{\infty} \sum_{n=\lceil (m+1-k)I/R \rceil}^{\infty} \{kI + nR - ((m+1)I + iR)\}\, P_t(n) P(k,n) P_t(i). \tag{3}$$

A.2 Agreement Protocol for Multi-value

A multi-valued agreement protocol is composed of one binary agreement; here, the expected number of rounds is $1/p$ and the expected running time is R/p. Thus, the expected running time of a multi-valued agreement protocol, M is

$$M = R/p. \tag{4}$$

局所的な辺移動によるリングの構成可能性について

泉 泰介[1]　泉 朋子[2]　大下 福仁[3]

複数のエージェントがリンクにて接続されている系を考える．ここで，各エージェントは自身が張っているリンクについて，近隣の情報を元に接続先を変更することが可能であるとする．このような系において，システム全体をある特定のトポロジに収束させるアルゴリズムの実現可能性について考察する．本研究では，トポロジをリングに収束させるオブリビアスアルゴリズムを提案し，そのステップ計算量が $O(n^2)$ であることを示す．

[1]名古屋工業大学工学研究科, 愛知県名古屋市昭和区御器所町, 466-8555
[1]名古屋工業大学産学官連携センター, 愛知県名古屋市昭和区御器所町, 466-8555
[3]大阪大学大学院情報科学研究科, 大阪府豊中市待兼山町 1-3, 560-8531

A Note on Certificate Dispersal Problems

Tomoko Izumi[1], Taisuke Izumi[1], Hirotaka Ono[2], Koichi Wada[1]

[1]Nagoya Institute of Technology, [2] Kyushu University

We consider a mobile network, where each node u has a private key $pri.u$ and a public key $pub.u$. Users themselves create their public and private keys. In this network, in order for a node u to send a message m to a node v securely, u needs to know $pub.v$ to encrypt the message with it, denoted by $pub.v < m >$. If a node u knows the public key $pub.v$ of another node v in the network, then the node u can issue a certificate from u to v that identifies $pub.v$. A certificate from a node u to a node v is of the following form: $pri.u < u, v, pub.v >$. The certificate is encrypted by using $pri.u$ and it contains three items: (i) the identity of the certificate issuer u, (ii) the identity of the certificate subject v, and (iii) the public key of the certificate subject $pub.v$.

Any node who knows $pub.u$ can use it to decrypt the certificate from u to v for obtaining $pub.v$. When a node u wants to obtain the public key of another node v, u acquires a sequence of certificates $pri.u < u, v_0, pub.v_0 >, pri.v_0 < v_0, v_1, pub.v_1 >, \ldots pri.v_\ell, v_\ell, v, pub.v >$ which are stored in either u or v.

All certificates issued by nodes in a network can be represented by a directed graph, called *a certificate graph*, denoted by $G = (V, E)$. We define a *dispersal* D of a directed graph $G = (V, E)$ as a family of sets of edges indexed by V, where $D = \{D_v \subseteq E | v \in V\}$. We define a request for a certificate graph G as a reachable ordered pair of nodes in G, denoted by (u, v). A set of requests, denoted by R, is called *full* if all reachable pairs of nodes in G are contained in R. We call a dispersal D of a directed graph $G = (V, E)$ *satisfies* a set of requests R, if for any request (u, v) in R, there exists a path from u to v in $D_u \cup D_v$, where D_u and D_v are dispersal of u and v. Let D be a dispersal of G satisfying R. The *cost* of dispersal D, denoted by $c.D$, is the sum of cardinalities of each dispersal in D. A dispersal D of G satisfying R is optimal if and only if for any other dispersal D' satisfies R, $c.D \leq c.D'$.

MINIMUM CERTIFICATE DISPERSAL PROBLEM(MCD) is defined as follows:

INPUT: A directed graph $G = (V, E)$ and a set of requests R

OUTPUT: A dispersal D of G satisfying R with the minimum cost.

It has been shown that MCD is NP-complete, even if the input graph is restricted to a strongly connected one. Also a polynomial-time 2-approximation algorithm can be constructed for strongly connected graphs. Moreover, it can be shown that this algorithm outputs optimal dispersals for complete graphs, trees, rings and Cartesian product of graphs.

In general, MCD is NP-complete for strongly connected graphs. A remaining question is whether MCD remains NP-complete for bidirectional graphs(or undirected graphs) and/or full requests or not.

Can the approximation ratio of MCD algorithms can be improved? That is, can a polynomial-time α-approximation algorithm such that $\alpha < 2$ be constructed for any directed graph?

The 2-approximation algorithm shown in [1] becomes optimal one for complete graphs, trees, rings hypercubes(more generally, Cartesian product of graphs). For some useful graph classes, such as chordal graphs or interval graphs, can we construct an optimal MCD algorithm?

References

[1] H. Zheng, S. Omura, K. Wada: A 2-approximation algorithm for minimum certificate dispersal problems, Proceedings of the 16th Australasian Workshop on Combinational Algorithms, 384-394 (2005).