The 5th Workshop on Theoretical Computer Science
Hiroshima, September 2009 (WTCS2009)

第5回情報科学ワークショップ

Satoshi Fujita

Sayaka Kamei

序言

本論文集は，2009 年 9 月 6 日～8 日に広島市国際青年会館において開催された第 5 回情報科学ワークショップの発表原稿をまとめたものである．

本ワークショップは，研究室以上の議論と研究会(LA)以下のフォーマルさを基本理念として，日本の並列／分散計算の研究者がお互いを徹底的に切りまくる「ちゃんばら大会」を開こうという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），長浜（第 4 回）と毎年 9 月に開催され，今回が 5 回目の開催となる．今回は，名古屋工業大学，大阪大学，岡山大学，九州大学，九州工業大学，広島大学から 54 名の参加者があり，26 件の発表と白熱した議論が行われた．また中野先生の呼びかけによる新規企画である「10000 円ゲーム大会」では，各研究室の予選を勝ち抜いた 12 本のプログラムのダブルエリミ―ション方式での対戦がおこなわれ，広島大学の重本耕司君の作成したプログラムが初代の優勝者となった．今回のワークショップは，これまでの 4 回とは打って変わって市街地の中心での開催となったが，会場の窓からは太田川にはさまれた広島市内の様子や瀬戸内の島々を一望することができ，静かな環境の中で，熱のこもった討論が 3 日間にわたって繰り広げられた．なお，2010 年は名古屋工業大学が主担当となり，名古屋地区で開催される予定である．

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，広島大学の亀井先生には，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．会場の準備・運営に関しては，藤田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．衷心より深く感謝致します．

2009 年 10 月
広島大学　藤田　聡

第5回情報科学ワークショップ in 広島 （2009．9．6－8）

6日 13:00-13:10　開会式

1 長瀧 寛之			(注)タイトル未定のものは(キーワード)
伊藤 靖朗	広島	13:10-13:35	A Hardware-Software Cooperative Approach for the Exhaustive Verification of the Collatz Conjecture
川上 賢介, 重本 耕司	広島	13:35-14:00	FPGA に特化した高速なパイプライン化RSA 暗号回路の実装
仲井 英剛	広島	14:00-14:15	誤差拡散法による集中型ハーフトニング
足立 悦三	広島	14:15-14:30	エッジの曲率を用いた画像のパターンマッチング
三田 尚義	広島	14:30-14:55	組合わせ円マーカーを用いた部品位置および姿勢の同定

2 泉 泰介			
神戸 文香	九州工業	15:10-15:35	膜計算における算術演算，及び，因数分解
野中 良哲	九州	15:35-16:00	グラフの局所情報を利用したランダムウォークのmixing timeについて
溝口 隆	九州	16:00-16:15	頻出項目検出階層型ストリームアルゴリズム
北村 宏大	広島	16:15-16:40	A Biased k-Random Walk to Find Useful Files in Unstructured Peer-to-Peer Networks

3 泉 朋子			
山田 陽介	九州	16:55-17:10	不完全情報渋滞ゲームと近似ナッシュダイナミクス
細田 淳	名古屋工業	17:10-17:25	Publish/Subscribeシステムにおけるオーバーレイネットワーク構成の最適化について
中河 雅弥	広島	17:25-17:40	並列凸包計算アルゴリズムのマルチコアプロセッサ上での評価
滿都呼	広島	17:40-17:55	Parallel Sampling Sorting on the Multicore Processors
門脇 一真	広島	17:55-18:20	A Dynamic User Management in Networked Consumer Electronics via Authentication Proxies

18:20-24:00　自由討論会

7日 7:30-8:30　朝食

4 野中良哲			
中本 浩太	九州工業	9:10-9:25	二連結性を保証する分散センサカバーアルゴリズム
岡本 直樹	九州工業	9:25-9:40	耐攻撃性を考慮したセンサ攻撃アルゴリズムとセンサ配置アルゴリズム
植村 奈緒子	大阪	9:40-10:05	An Experimental Evaluation of Clustering Algorithm using Attractor Selection
馬場 大輔	大阪	10:05-10:20	Multiple Mobile Agents Rendezvous in Trees

＊伊藤 靖朗		10:30-12:00	10000円ゲーム大会

12:00-18:00　自由討論会

18：00-24:00　懇親会、自由討論会(アステールプラザ横　厚生年金会館3階レストラン）

8日 7:30-8:30　朝食

6 大下 福仁			
高田 篤史	大阪	9:10-9:35	トポロジ変化に対して出力の変化数を極小化する全域木更新アルゴリズム
中村 純哉	大阪	9:35-9:50	自己安定性を有した耐ビザンチン故障レプリケーションの検討
羽場 康太郎	名古屋工業	9:50-10:15	自律分散ロボットのモデルの統一的分類とその考察
三浦哲平	名古屋工業	10:15-10:40	自己安定アルゴリズムの多段構成を用いた重み最小全域木構築アルゴリズムについて

7 中村 純哉			
長瀧 寛之	岡山	10:50-11:05	直感的な操作が可能な分散アルゴリズム学習用シミュレータの提案
藤原 啓	大阪	11:05-11:20	アルゴリズム学習向け誤り発見型演習のためのカスタマイズ可能な問題自動生成システム
林 周平	広島	11:20-11:45	An Efficient Web Page Recommendation Based on Preference Footprint to Browsed Pages
中井 亮	大阪	11:45-12:10	Web サービスミドルウェアに対応可能なチェックポイントの作成方法

12:10-12:20　閉会式

目次

A Hardware-Software Cooperative Approach for the Exhaustive Verification of the Collatz Conjecture

Yasuaki Ito and Koji Nakano
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527 JAPAN

Abstract

Consider the following operation on an arbitrary positive number: if the number is even, divide it by two, and if the number is odd, triple it and add one. The Collatz conjecture assert that, starting from any positive number n, repeated iteration of the operations eventually produces the value 1. The main contribution of this paper is to present hardware-software cooperative approach to verify the Collatz conjecture. The key idea of our approach is to sieve numbers n that produces 1 using a circuit implemented on an FPGA. The numbers that fail to be verified by overflow are reported to the host PC. The host PC verifies those numbers using unlimited bits operations by software. We have implemented 24 coprocessors on the Vertex II family FPGA XC2V3000-4. The experimental results show that our hardware-software cooperative approach can verify 2.89×10^9 64-bit numbers per second.
Keywords: *Hardware Algorithm, FPGA Implementation, block RAMs.*

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 (or 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. 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 [2, 3, 8, 9, 10].

The Collatz conjecture is a well-known unsolved conjecture in mathematics [5, 7, 13]. Consider the following operation on an arbitrary positive number:

even operation if the number is even, divide it by two, and

odd operation if the number is odd, triple it and add one.

The Collatz conjecture assert that, starting from any positive number, repeated iteration of the operations eventually produces the value 1. For example, starting from 3, we have the following sequence to produce 1.

$$3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

The main contribution of this paper is to present a hardware-software cooperative approach to verify the Collatz conjecture. More specifically, we have developed a hardware algorithm for exhaustive verification of the Collatz conjecture for a limited number of bits. We use an FPGA to implement this hardware algorithm as coprocessors for this approach. The key idea of our approach is to sieve numbers n verified that it produces 1 using coprocessors on an FPGA connected through PCI-bus. Each coprocessor has a 78-bit architecture and works as follows: First, it receives 32-bit number M from the host PC through the PCI-bus. For each 64-bit number m in $[M \cdot 2^{32}, (M+1) \cdot 2^{32} - 1]$, it verifies if iteration of the operations produce 1 starting from m by repeating the even and odd operations. The coprocessor reports number m to the host PC if the interim value has more than 78 bits and it fails to verify the conjecture. The host PC receives such number m, and perform the iteration of the operations with unlimited number of bits implemented by software. Let us show an example for the reader's benefit. Suppose that, from host PC, a 32-bit number M =0xF1234567 is given to the coprocessor on the FPGA. For each number m from $M \cdot 2^{32}$ (=0xF1234567000000) to $(M+1) \cdot 2^{32} - 1$ (=0xF1234567FFFFFFFF), the coprocessor repeats the even and the odd operations. Starting from m =0xF123456705A1CF67, the iteration of the operations produces an interim number 0xB59C3655454882EE20450CB8 which has more than 78 bits. Thus, the coprocessor detects the overflow of 78-bit

register and reports the 32-bit LSB of m, 0x05A1CF67 to the host PC. The host PC verifies the Collatz conjecture for such m =0xF123456705A1CF67 by software.

This coprocessor approach makes sense because

- the hardware implementation on the FPGA is fast and low power consumption, but the number of bits for the operation is fixed,
- the software implementation on the PC is relatively slow and high power consumption, but the number of bits for the operation is unlimited.

So, our approach finds counter example candidates m of the Collatz conjecture using the hardware implementation, and confirm it for such numbers m using the software implementation. Also, since the circuit for the coprocessor is small enough that we can implement 24 coprocessors on a Vertex II family FPGA XC2V3000-4 [6]. In this way, we can further accelerate the verification by hardware-software cooperative approach.

Figure 1 illustrates our hardware-software cooperative architecture for verifying the Collatz conjecture. We have used a Nallatech Xtreme DSP kit [11], which is a PCI board with Xilinx VirtexII family FPGA XC2V3000-4, We have implemented 24 coprocessor working independently in the FPGA. The FPGA and the host PC are connected though the PCI-bus. The host PC works as a server, which requests to verify the Collatz conjecture for 2^{32} numbers from $M \cdot 2^{32}$ to $(M+1) \cdot 2^{32} - 1$ for some 32-bit integer M to each coprocessor. The experimental results show that, for 64-bit numbers, each coprocessor can verify 1.20×10^8 numbers per second. Also, we can verify 2.89×10^9 numbers per second using our hardware-software cooperative architecture using 24 coprocessors.

In our previous paper [1], we have shown an architecture for the verification of the Collatz conjecture. This paper showed an implementation on a Xilinx Spartan-3AN family FPGA XC3S700AN-5, which uses 831 slices, seven 18-bit embedded multipliers, two 18Kbit block RAMs. The simulation and the timing analysis results for XC3S700AN-5 show that this architecture can verify 2^{16} numbers in 421μs. Thus, it can verify 1.55×10^8 numbers per second. However, it uses 128-bit registers, and does not check the overflow. Our new idea is to improve this architecture to work as a coprocessor, and to implement 24 coprocessors in a Virtex II family FPGA XC2V3000-4.

This paper is organized as follows. Section 2 shows basic ideas to accelerate the verification of the Collatz conjecture. In Section 3, we show a basic hardware algorithm for the verification on the FPGA. Section 4 shows how we have implemented our hardware algorithm as a coprocessor including the interface between the host PC and the FPGA, and how the software in the host PC works. Sections 5 and 6 show the performance analysis and and the experimental results. Finally, Section 7 offers the conclusions.

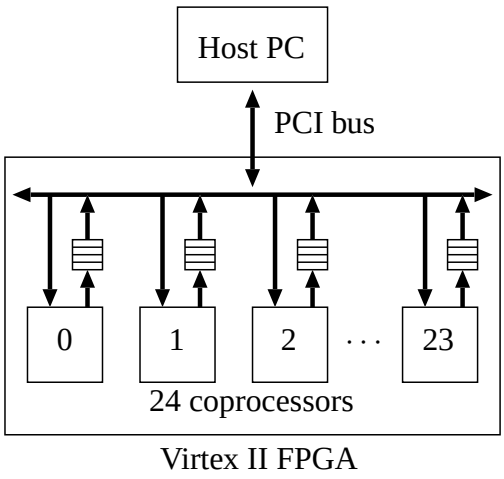


Figure 1. Our hardware-software cooperative architecture for verifying the Collatz conjecture

2 Accelerating the verification of the Collatz conjecture

The main purpose of this section is to show a hardware algorithm for accelerating the verification of the Collatz conjecture. The basic ideas of acceleration are shown in [7, 13].

The first idea is to terminate the operations before the iteration produces 1. Suppose that we have already verified that the Collatz conjecture is true for numbers less than n, and we are now in position to verify it for number n. Clearly, if we repeatedly execute the operations for n until the value is 1, then we can confirm that the conjecture is true for n. Instead, if the value becomes n' for some n' less than n, then we can guarantee that the conjecture is true for n because it has been proved to be true for n'. Thus, it is not necessary to repeat this operation until the value is 1, and we can terminate the iteration when, for the first time, the value is less than n.

The second idea is to perform several operations in one step. Consider that we want to perform the operations for n and let n_L and n_H be the least significant two bits and the remaining bits of n. In other words, $n = 4n_H + n_L$ holds. Clearly, the value of n_L is either 00, 01, 10, or 11. We can perform the several operations for n based on n_L as follows:

$n_L = 00$**:** Since two even operations are applied, the result-

ing number is n_H.

$n_L = 01$**:** First, odd operation is applied and the resulting number is $(4n_H + 1) \cdot 3 + 1 = 12n_H + 4$. After that, two even operations are applied, and we have $3n_H + 1$.

$n_L = 10$**:** First, even operation is performed and we have $2n_H + 1$. Second, odd operation is applied and we have $(2n_H + 1) \cdot 3 + 1 = 6n_H + 4$. Finally, by even operation, the value is $3n_H + 2$.

$n_L = 11$**:** First, odd operation is applied and we have $(4n_H + 3) \cdot 3 + 1 = 12n_H + 10$. Second, by even operation, the value is $6n_H + 5$. Again, odd operation is performed and we have $(6n_H+5)\cdot 3+1 = 18n_H+16$. Finally, by even operation, we have $9n_H + 8$.

For example, if $n_L = 11$ then we can obtain $9n_H + 8$ by applying 4 operations, odd, even, odd, and even operations in turn. Let A and B be tables as follows:

	A	B
00	1	0
01	3	1
10	3	2
11	9	8

Using these tables, we can perform the following table operation, which emulates several odd and even operations:

table operation For least significant two bits n_L and the remaining most significant bits n_H of the value, the new value is $A[n_L] \cdot n_H + B[n_L]$.

Let us extend the table operation for least significant two bits to more bits. For a number n and d (≥ 2), let n_L and n_H be the least significant d bits, that is, $n = 2^d n_H + n_L$. We call d is *the base bits*. Let the current value of n is $an_H + b$. Initially, $a = 2^d$ and $b = n_L$. We repeatedly perform the following rules for a and b.

even rule If both a and b are even, then divide them by two.

odd rule If b is odd, then triple a, and triple b and add one.

These two rules are applied until no more rules can be applied, that is, until a is odd and b is even. It should be clear that, even and odd rules correspond to even and odd operations of the Collatz conjecture. If i even rules and j odd rules applied, then the value of a is $2^{d-i}3^j$. Thus, exactly d even rules are applied until the termination. After the termination, we can determine the value of elements in tables A and B such that $A[n_L] = a$ and $B[n_L] = b$. Using tables A and B, we can perform the table operation for d bits n_L, which involves d even operations and zero or more odd operations. In this way, we can accelerate the operation of the Collatz conjecture. In our previous paper [1], we have implemented for various numbers of bits of n_L. Our implementation results show that the performance is well balanced when the number of bits of n_L is 10.

The third idea to accelerate the verification of the Collatz conjecture is to skip numbers n such that we can guarantee that the resulting number is less than n after the table operation. For example, suppose we are using two bit table and $n_H > 0$. If $n_L = 00$ then the resulting value is n_H, which is less than n. Thus, we can skip the table operation for n if $n_L = 00$. If $n_L = 01$ then the resulting value is $3n_H + 1$, which is always less than $n = 4n_H + 1$, and we can skip the table operation. Similarly, if $n_L = 10$ then we can skip the table operation. On the other hand $n_L = 11$ then the resulting value is $9n_H + 8$, which is always larger than n. Therefore, the Collatz conjecture is guaranteed to be true whenever $n_L \neq 11$, because it has been verified true for numbers less than n. Consequently, we need to execute the table operation for number n such that $n_L = 11$.

We can extend this idea for general case. For least significant d bits n_L, we say that n_L is not *mandatory* if the value of a is less than 2^d at some moment while even and odd rules are repeatedly applied. We can skip the verification for non mandatory n_L. The reason is as follows: Consider that for number n, we are applying even and odd rules. Initially, $a = 2^d$ and $b \leq 2^d - 1$ hold. Thus, while even and odd rules are applied, $a > b$ always hold. Suppose that $a \leq 2^d - 1$ holds at some moment while the rules are applied. Then, the current value of n is

$$an_H + b < an_H + a \leq (2^d - 1)n_H + a = 2^d n_H < n.$$

It follows that, the value is less than n when the corresponding even and odd operations are applied. Therefore, we can omit the verification for numbers that have no mandatory least significant bits.

For least significant d bit number, we use table S to store the mandatory least significant bits. Let s_d be the number of such mandatory least significant bits. Using these table, we can write a verification algorithm as follows:

for $m_H = 1$ **to** $+\infty$ **do**
  for $i = 0$ **to** $s_d - 1$ **do**
    begin
      $m_L = S[i]$;
      $n = m = 2^d m_H + m_L$;
      while($n \geq m$**) do**
        begin
          Let n_L be the least significant d bits and
            n_H be the remaining bits.
          $n = A[n_L] \cdot n_H + B[n_L]$;
        end
    end

For the benefit of readers, we show A, B, and S for 4 base bits in Table 1. From $s_4 = 3$, we have 3 mandatory least significant bits out of 16.

Table 1. Tables A, B, and S for least significant 4 bits.

	A	B	S
0000	1	0	0111
0001	9	1	1011
0010	9	2	1111
0011	9	2	-
0100	3	1	-
0101	3	1	-
0110	9	4	-
0111	27	13	-
1000	3	2	-
1001	27	17	-
1010	3	2	-
1011	27	20	-
1100	9	8	-
1101	9	8	-
1110	27	26	-
1111	81	80	-

3 An Architecture for the verification on an FPGA

The main purpose of this section is to show an architecture for verifying the Collatz conjecture using tables A, B, and S. The preliminary results of FPGA implementation are shown in our previous paper [1].

Let d be the base bits. Each of the tables A and B can be stored in a ROM of size $2^d \times a_d$, where a_d is the number of bits necessary to store the values of A. From Table 1, $a_4 = 7$ because the maximum value is $81 < 2^7$. To store table S, a ROM of size $s_d \times d$ is used. Let w be the maximum number of bits of n. Clearly, n_H has $w - d$ bits and n_L has d bits. Thus, to compute the product $A[n_L] \cdot n_H$, we use an $a_d \times (w - d)$-bit multiplier. After that, to compute the sum $A[n_L] \cdot n_H + B[n_L]$, a w-bit a_d-bit adder is used.

The ROM can be implemented by block RAMs in the FPGA. Block RAMs in most FPGAs including Virtex II Family FPGA used in our implementation support synchronous read and synchronous write [6]. In the synchronous operation, the address bits given to an address port is written in an internal address register at the rising edge of the clock. After that, the data specified by the address register can be read from the data output port. Thus, one clock cycle necessary to read the data in a synchronous read block RAM. A naive implementation of the ROM using the block RAM needs two clock cycles to update the value of n_H and n_L by next value $A[n_L] \cdot n_H + B[n_L]$. One clock cycle is necessary to read the value of $A[n_L]$ and $B[n_L]$ stored in the block RAMs. After that, additional one clock cycle is necessary to store that value of $A[n_L] \cdot n_H + B[n_L]$ to registers for n_H and n_L. As we are going to show next, the update of n_H and n_L can be done in one clock cycle by careful implementation.

Figure 2 shows an illustration of our architecture for verifying the Collatz conjecture, which performs the update of n_H and n_L in every clock cycle. The address registers for tables A and B are used to store n_L as in Figure 2, while the value of n_H is stored in a register. Two counters are used to store the current values of m_H and i. The value of $S[i]$ is computed from i and stored in m_L. In the same time, $S[i]$ is also stored in the address register for tables A and B. From the value of the address register, $A[n_L]$ and $B[n_L]$ are obtained. After that, $A[n_L] \cdot n_H + B[n_L]$ is computed using a multiplier and an adder. This value is compared with the value of m stored in the register. If $n \geq m$ then the resulting value is stored in the register for n_H and the address register for n_L. Otherwise these registers are updated by next values of m_H and m_L. In this way, the values of n_H and n_L can be updated in one clock cycle.

4 Our hardware-software cooperative approach

The main purpose of this section is to present the details of implementation of our hardware-software operative approach.

We have used 24 coprocessors as illustrated in Figure 1. We modify the architecture in Figure 2 to use as a coprocessor in Figure 1. The readers should refer Figure 3 for illustrating the interface between the host PC and our coprocessor architecture. Our coprocessor architecture takes 10 base bits, and thus, tables A and B has 1024 16-bit words each. It has 64 mandatory least significant bits out of 1024 least significant bits. Thus, the counter for table S needs 6 bits to indicate one of the 64 mandatory least significant bits. Each of the coprocessors receives 32-bit integer M from the host PC, and performs the exhaustive verification for the Collatz conjecture for numbers from $M \cdot 2^{32}$ to $(M + 1) \cdot 2^{32} - 1$. In other words, the verification is performed for 64-bit numbers. This is reasonable because working projects for the Collatz conjecture are currently checking 60-bit numbers [12] and 63-bit numbers [4]. Suppose that the coprocessor find that the interim value is overflow for the initial value m. The coprocessor reports the least significant 32-bit of m through the buffer with four 32-bit registers if the overflow is detected. After the host PC receives this 32-bit value, it verifies the Collatz conjecture for m.

Figure 4 illustrates a part of the circuit to compute $A[n_L] \cdot N_H + B[n_L]$ and to detect the overflow. The value of n_H is stored in 68-bit register, and that of n_L is stored in the block RAMs for tables A and B. Note that, since our architecture

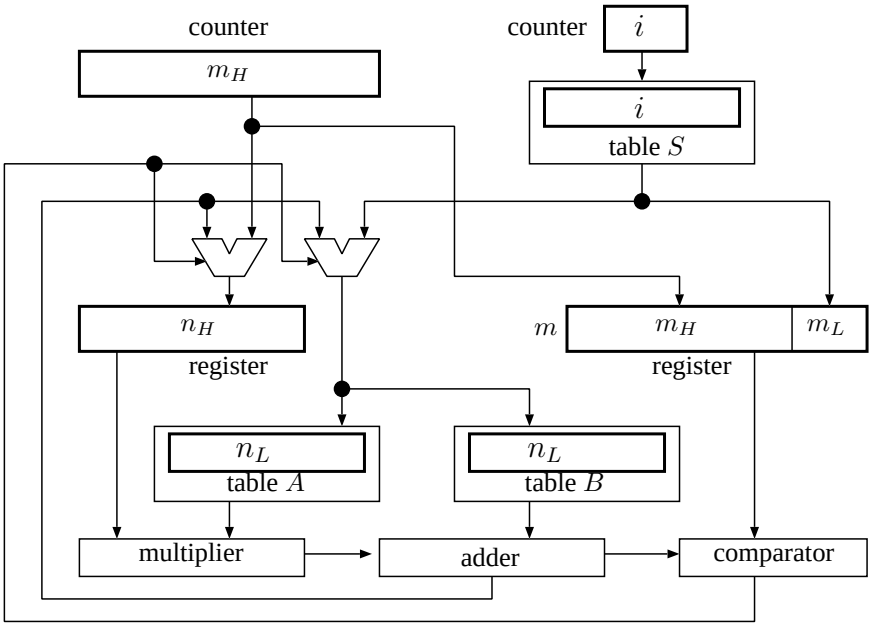


Figure 2. Our architecture for verifying the Collatz conjecture

has 10 base bits, n_L has 10 bits and the output values $A[n_L]$ and $B[n_L]$ of tables A and B have 16 bits. The product $A[n_L] \cdot n_H$ is computed by a 68×16-bit multiplier. The sum $A[n_L] \cdot n_H + B[n_L]$ is computed by an $84 + 16$-bit adder. If the resulting value of the sum is larger than 2^{78}, the overflow is detected. If this is the case, the least significant 32 bits of the current value of m is transferred to the buffer in Figure 3.

The host PC works as a server of the coprocessors as follows: Suppose that we need to verify the Collatz conjecture for all 64-bit numbers. The host PC sends $0, 1, 2, \ldots, 23$, to the 24 coprocessors respectively. The host PC always checks the buffers of the coprocessors. Consider that the host PC finds a 32-bit number m' stored in one of the buffers of the coprocessor and the host PC has sent request of 32-bit number M to the coprocessor. It follows that the coprocessor detect the overflow for the initial value $m = M \cdot 2^{32} + m'$. Thus, the host start start the verification for m using the unlimited bits operations. If the host PC finds that one of the coprocessors finishes the verification for 2^{32} numbers, it sends next number 24 to the processor. Similarly, if the host PC finds a processor that finishes the verification, it sends next numbers 25, 26, ..., $2^{32} - 1$.

The unlimited bits operations by software are implemented by a linked list with each element taking a 32-bit unsigned integer. Hence, a linked list with n elements can represent $32n$-bit integer. Since n is unlimited, the linked list can represent any large number. Further, it is not difficult to implement the even and the odd operations for a linked list in an obvious way. In particular, by the odd operation, we may need more number of bits. If this is the case, an element with 32-bit number is added to the linked list to increase the number of bits.

5 The performance analysis

Let us evaluate the hardware resources in the FPGA to implement coprocessors. A coprocessor uses a 68×16-bit multiplier. The Virtex II FPGA has 18×18-bit multipliers, which support the multiplication of two 17-bit unsigned integers. Thus, the product of 68-bit and 16-bit unsigned integers can be implemented using four 18×18-bit multipliers. Tables A and B have 1024 16-bit words each. Hence, each of them can be implemented in one 18k-bit block RAM in the Virtex II FPGA. Table S has 64 10-bit words, which can also be implemented in one 18k-bit block RAM. Thus, each coprocessor uses three 18k-bit block RAMs for tables A, B, and S. Further, since the block RAM in the Virtex II FPGA is dual port, that is, it has two address ports and can be read the data of two addresses, which can be distinct, in the same time. Hence, two coprocessors can share the three 18k-bit block RAMs for three tables. Consequently, 24 coprocessors uses 96 18×18-bit multipliers and 36 18k-bit block RAMs. Since our target is XC2V3000 has 96 18×18-bit multipliers and 96 18k-bit block RAMs, our implementation is feasible.

Let us evaluate how often we have the overflow. Suppose that the even and the odd operations are performed for a

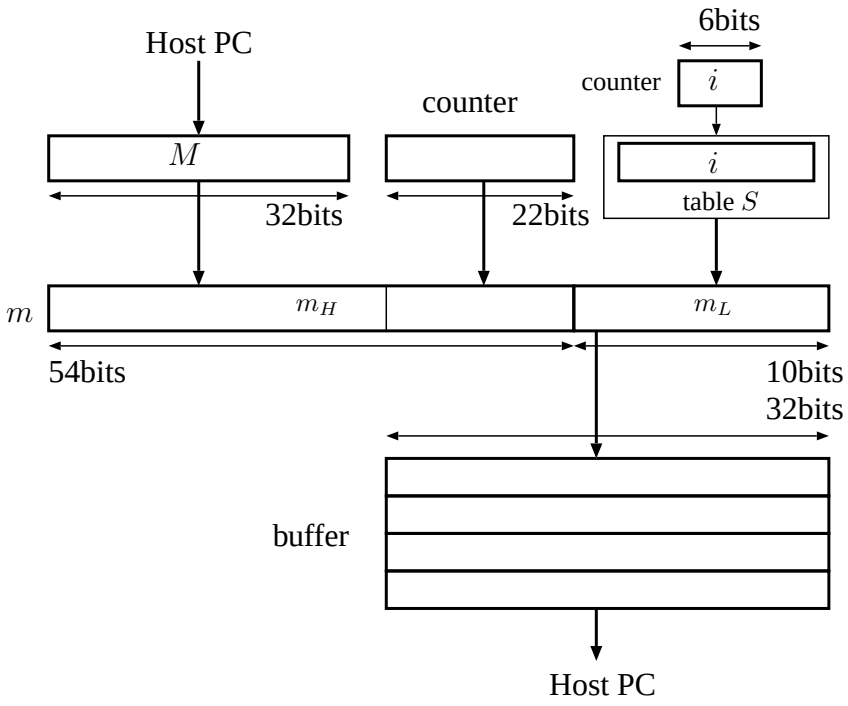


Figure 3. The interface between the host PC and the coprocessor

p-bit number n. Note that $2^{p-1} \leq n < 2^p$ holds. Let $M(n)$ denote the maximum number until n produces, for the first time, a number less than n during the operations. For example, $M(3) = 16$ holds. Suppose that we use b-bit architecture for the even and the odd operations. The verification of n is overflow if $M(n) \geq 2^b$. Let $V(p, b)$ be the set of such number n, that is $V(p, b) = \{n \mid 2^{p-1} \leq n < 2^p \text{ and } M(n) \geq 2^b\}$. Then, the ratio $R(p, b)$ of the overflow of p-bit numbers on b-bit architecture is

$$R(p, b) = \frac{|V(p, b)|}{2^{p-1}}.$$

Let us clarify the ratio $R(p, b)$ for $p = 58, 59, \ldots, 64$, and $b = 66, 67, \ldots, 90$. For this purpose, we generate a p-bit number 10^8 times at random, and see if $M(n) \geq 2^b$. Figure 5 shows the ratio $R(p, b)$ of overflow for a p-bit integer on the b-bit architecture. This figure shows that the ratio $R(p, b)$ rapidly decreases by increasing the value of b. Recall that the parameters $p = 64$ and $b = 78$ are used in our coprocessors. From the figure, we can see $R(64, 78) = 2.9 \times 10^{-5}$, which is small enough.

6 Experimental results

We have implemented and evaluated the performance of our hardware-software cooperative approach on Xilinx Virtex II family FPGA (XC2V3000-4), which has 28672 slices, 96 18-bit multipliers, and 96 18k-bit block RAMs. For logic synthesis, we have used XST with ISE Foundation 10.1. The implemented circuit on the FPGA consists of 24 coprocessors and the interface between Host PC and FPGA. The implementation uses 8848 slices and 96 18-bit multipliers, and 36 18k-bit block RAMs. The timing analysis by ISE Foundation 10.1 reported that our implementation runs in 40.12MHz. Thus, we set the clock frequency the programmable oscillator on the FPGA board to 40MHz.

The computing time is evaluated by verifying the Collatz conjecture for the 64-bit numbers in 240 sections. Each section has 2^{32} numbers in $[M \cdot 2^{32}, (M+1) \cdot 2^{32} - 1]$, where M is a 32-bit number that is generated at random. Therefore, we verified $240 \times 2^{32} \approx 1.0 \times 10^{12}$ 64-bit numbers. For the purpose of showing the goodness of our hardware-software cooperative approach, we have implemented a software approach. We measured the performance of the software approach on an IBM PC-compatible (Xeon X5355 2.66GHz processor with 10GB of memory) using Linux OS (64bit). Since enough size of memory is available in the software approach, we use larger tables A, B, and C. More specifically, the sizes of tables A and B are 65536×26 and the size of table S of sizes 2114×16. The computing time of the software approach is also evaluated by verifying the Collatz conjecture for the same 64-bit numbers in 240 sections.

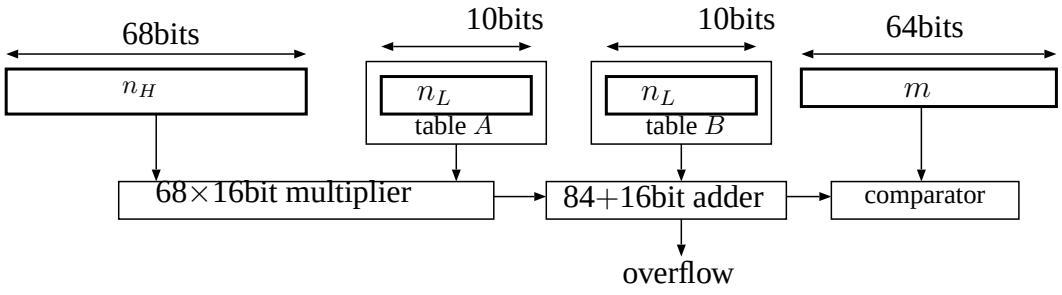


Figure 4. The circuit to compute $A[n_L] \cdot N_H + B[n_L]$ and to detect the overflow

Table 2 shows the computing time of our hardware-software cooperative approach and its corresponding software approach for verifying approximately 1.0×10^{12} 64-bit numbers. As shown in the table, a speed-up factor of more than 15 is experimented and our hardware-software cooperative approach verified 2.89×10^9 numbers per second. In the hardware-software cooperative approach, there are 5656255 overflow-numbers. The number of overflow is much smaller than the number of verified numbers, and those numbers are verified by software on the host PC. Therefore, almost numbers are sieved using the circuit on the FPGA.

7 Conclusions

We have presented a hardware-software cooperative approach that verifies the Collatz conjecture. The key idea of our approach is to sieve numbers n verified that it produces 1 using a circuit on an FPGA. The numbers that fail to be verified by overflow are reported to the host PC. The host PC verify those numbers using unlimited bits operations by software.

We have implemented 24 coprocessors on the Vertex II family FPGA XC2V3000-4. The experimental results show that our hardware-software cooperative approach can verify 2.89×10^9 numbers per second, and attains a speed-up factor of more than 15 over the software approach.

References

[1] F. An and K. Nakano. An architecture for veryfing collatz conjecture using an fpga. In *Proc. of the International Conference on Applications and Principles of Informatin Science*, pages 375–378, 2009.

[2] 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.

[3] 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.

[4] T. O. e Silva. Computational verification of the 3x+1 conjecture. http://www.ieeta.pt/ tos/3x+1.html.

[5] T. O. e Silva. Maximum excursion and stopping time record-holders for the $3x + 1$ problem: Computational results. *Mathematics of Computation*, 68(225):371–384, Jan. 1999. Up to date computational results can be found at `http://www.ieeta.pt/~tos/3x+1.html`.

[6] X. Inc. *Virtex-II Platform FPGAs: Complete Data Sheet*, 2003.

[7] J. C. Lagarias. The 3x+1 problem and its generalizations. *The American Mathematical Monthly*, 92(1):3–23, 1985.

[8] K. Nakano and E. Takamichi. An image retrieval system using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):811–818, May 2003.

[9] K. Nakano and K. Wada. Integer summing algorithms on reconfigurable meshes. *Theoretical Computer Science*, 197:57–77, 1998.

[10] K. Nakano and Y. Yamagishi. Hardware n choose k counters with applications to the partial exhaustive search. *IEICE Trans. on Information & Systems*, 2005.

[11] Nallatech. *Xtreme DSP Development Kit User Guide*, 2002.

[12] E. Roosendaal. On the 3x + 1 problem. http://www.ericr.nl/wondrous/index.html.

[13] E. W. Weisstein. Collatz problem. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/CollatzProblem.html.

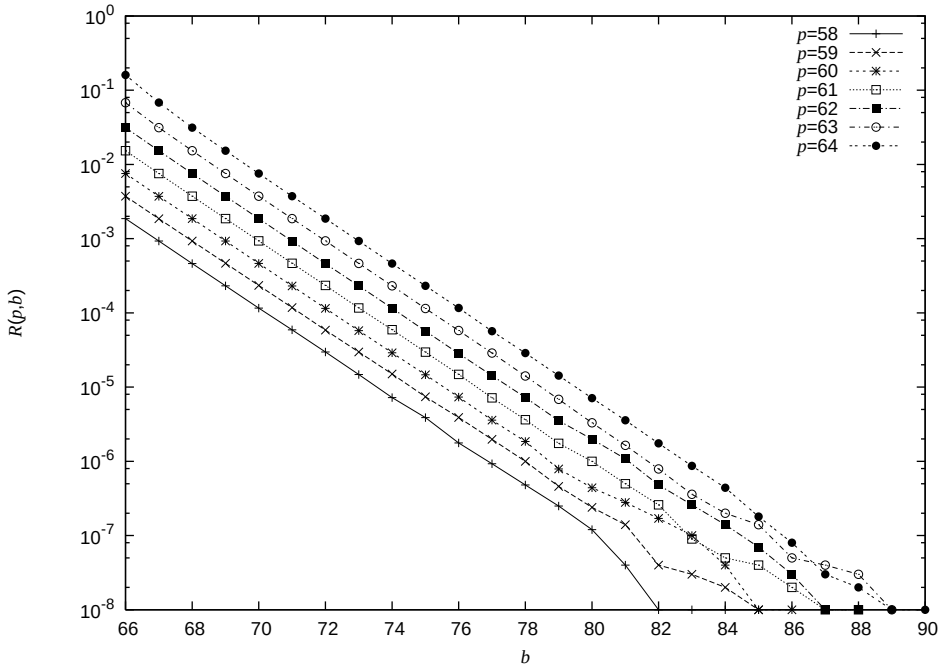


Figure 5. The ratio $R(p, b)$ of overflow for a p-bit integer on the b-bit architecture

Table 2. The computing time for verifying Collatz conjecture

	Computing Time	The number of verified numbers per second	Speed-up
Software approach	5517.67 sec	1.87×10^8	–
Hardware-software cooperative approach	357.17 sec	2.89×10^9	15.45

FPGA に特化した高速なパイプライン化 RSA 暗号回路の実装

川上 賢介　　重本 耕司　　中野 浩嗣
(広島大学大学院 工学研究科 情報工学専攻)

1　はじめに

FPGA は再構成可能なハードウェア素子であり，ハードウェア記述言語によって記述した回路データをダウンロードすることで，容易に回路の書き換えを行うことができる．本研究では，RSA 暗号で用いられるべき乗剰余演算回路を FPGA に実装した．本手法のアルゴリズムは Xilinx Virtex-5 ファミリ FPGA をターゲットとし，組み込みの信号処理エレメントである DSP48E と組み込みのメモリであるブロック RAM を効率的に用いる．具体的には，べき乗剰余演算を高速化するアルゴリズムとして知られているモンゴメリ乗算に対して，DSP48E を用いたパイプライン化，冗長表現演算，ブロック RAM を用いたテーブル参照の 3 つのアイデアを適用することで高速化を図った．本研究のアルゴリズムでは 1024bit のべき乗剰余演算を 1.96[Mbit/s] で実行することが可能であり，従来の研究と比べ優れた結果を得られた．

2　RSA 暗号

RSA 暗号 [2] は公開鍵暗号方式を実現するために広く用いられているアルゴリズムで，べき乗剰余演算を用いて暗号化を行う．ある平文 P を暗号化するためには，適切に選択した法 M と暗号鍵 E を用いて，暗号文 $C = P^E \mathrm{mod} M$ を求める．逆に，暗号文を復号化する際には，復号鍵 D を用いて，同様に $P = C^D \mathrm{mod} M$ を求める．実用上，現在では RSA 暗号の安全性を保つためには 1024bit 以上の巨大なオペランドが必要であり，剰余演算を含む RSA 暗号は計算に非常に時間がかかる．そこで，ハードウェアを用いて高速にべき乗剰余演算を行うことが可能なモンゴメリ乗算が利用されている．

2.1　モンゴメリ乗算

モンゴメリ乗算 [1] は，べき乗剰余演算の計算に必要な乗算剰余演算を高速に行うアルゴリズムである．モンゴメリ乗算は，剰余演算の性質 $(X + kM) \bmod M = X \bmod M$ を利用して，剰余演算をシフト演算と加算，乗算に置き換えることで高速化を行う．

以下に，radix-2^r のモンゴメリ乗算アルゴリズムを示す．ここで，dr は法 M のビット幅であり，r はモンゴメリ乗算の基数が 2^r であることを示す．モンゴメリ乗算アルゴリズムでは，3 行目，4 行目，5 行目の演算を d 回繰り返す．下位ビットを 0 にするために加算する値 k ををを計算することで剰余演算を置き換えながら，乗算，加算，シフト演算を繰り返す．最終的な U_d の値は $2M > U_d > M$ の値となるので，$U_d > M$ となった場合は M を減算することで，最終的な解を求める．

- Algorithm1 -
Input: X, Y, M
$\{X = (X_{d-1}, ..., X_1, X_0),\ Y = (Y_{d-1}, ..., Y_1, Y_0)\}$
Output: $U_d = X \cdot Y \cdot 2^{-dr} \bmod M$
1. $U_0 \leftarrow 0$
2. **for** $i = 0$ **to** $d - 1$ **do**
3. $\quad U_{i+1} \leftarrow X \cdot Y_i + U_i$
4. $\quad k_i \leftarrow (U_{i+1} \cdot (-M^{-1}))[r - 1 : 0]$
5. $\quad U_{i+1} \leftarrow (U_{i+1} + k_i \cdot M)\ /\ 2^r$
6. **end for**
7. **if** $(U_d > M)$ **then** $U_d \leftarrow U_d - M$

2.2　RSA 暗号アルゴリズム

モンゴメリ乗算を用いて R ビットのべき乗剰余演算の解を求めるアルゴリズムを以下に示す．ここでは，右向きバイナリ法によるアルゴリズムを示す．アルゴリズムでは，まずモンゴメリ領域への写像を行い，その後乗算剰余演算を繰り返し，最終的にモンゴメリ領域から値を戻すことによって，解を求めている．アルゴリズム中の下線はモンゴメリ乗算を行う処理であることを示す．

- Algorithm2 -
Input: P, E
$\{E = (E_{R-1}, ..., E_1, E_0)\}$
Output: $C = P^E \bmod M$
1. $C \leftarrow 2^R \bmod M$;
2. $P \leftarrow \underline{P \cdot (2^{2R} \bmod M) \cdot 2^{-R} \bmod M}$;
3. **for** $i = R - 1$ **downto** 0 **do**
4. $\quad C \leftarrow \underline{C \cdot C \cdot 2^{-R} \bmod M}$;
5. $\quad$ if $E_i = 1$ then $C \leftarrow \underline{C \cdot P \cdot 2^{-R} \bmod M}$;
6 **end for**
7. $C \leftarrow \underline{C \cdot 1 \cdot 2^{-R} \bmod M}$;

3　提案手法

本研究では，上記の RSA 暗号アルゴリズムについて改善を行い，FPGA 向けの高速な RSA 暗号回路アルゴリズムを実装した．以下に詳細を示す．

3.1　冗長表現演算

2.1 節で見たように，モンゴメリ乗算では加算，乗算を行う必要がある．RSA 暗号の要件から，演算のオペランドが非常に大きくなるため，ハードウェアに実装する際には，キャリーによる遅延が非常に大きくなってしまう．そこで，本手法では，オペランドを r ビット毎にブロックに分割して，それぞれを独立に計算することでキャリー遅延を軽減させる，冗長表現手法 [3] を用いた．

冗長表現演算では，キャリーの伝搬を保持するために，分割した r ビットのブロックに 2 ビットの冗長ビットを付加することで，ブロック毎に独立に加算，乗算などの演算を行うことが可能であり，これらの手法をモンゴメリ乗算に適用することが可能である [3].

3.2　ブロック RAM を用いたテーブル参照

2.1 節で見たように，モンゴメリ乗算アルゴリズムでは，$k_i M$ の値を解に足し合わせることによって，剰余演算をシフト演算に置き換えている．この $k_i M$ の値は，Algorithm 1 の 3 行目で求めた U_{i+1} の値から一意に決定することができる．そこで，FPGA の組み込みメモリであるブロック RAM に kM の値を格納することで，Algorithm 1 の 4 行目の計算を省き，高速化を図った．radix-2^r のモンゴメリ乗算では，r ビット毎に計算を行うので，U_i の下位 r ビットから kM の値を決定する．この場合，アドレス幅 r，データ幅 dr のブロック RAM が必要となる．

さらに，本手法では FPGA のデュアルポートブロック RAM の特性を利用してブロック RAM の容量を削減し，アドレス幅を $r/2$ とするアルゴリズムを用いている．アルゴリズムの詳細は過去に発表した [3] に示す．これにより，Algorithm 1 の 4 行目の演算を以下

のように書き換えることができる．ブロック RAM を用いた関数 f により，k_iM の値が計算される．

4. $k_iM \leftarrow f(U_{i+1}[r-1:0])$;

3.3 モンゴメリ乗算回路のパイプライン化

RSA 暗号アルゴリズムをハードウェア化する際には，モンゴメリ乗算の回路遅延がボトルネックとなる．本手法では，モンゴメリ乗算の演算部分をパイプライン化することで高速化を図った．高速化を図る必要がある処理は，Algorithm1 の for 文内の処理である．図 1 の左図に，Algorithm1 の演算部分の回路を示す．回路は，冗長表現演算で分割された r ビットの 1 ブロックを示し，X_i, Y_j を格納するレジスタから途中結果 U_{i+1} を格納するまでに複数の加算器と乗算器を通過している．同期式回路では，クロックの動作周波数は最大パスに合わせて設計されるので，この部分が最大パスとなり，回路の動作周波数が低下してしまう．

そこで，本手法では図 1 の右図のように各演算器間にレジスタを挟むことでパイプライン化を行い，1 クロックで行う処理を減らし，動作周波数を向上させている．また，図 1 中に示すように，Xilinx Virtex-5 ファミリの積和演算エレメントである DSP48E を用いて高速化と回路規模の削減を図っている．DSP48E は様々な信号処理用に拡張することが可能であり、積和演算や乗算、論理演算など様々な演算を実行できる。また、DSP48E 内のパイプラインレジスタを用いることで高速な演算処理を実行することが可能となる。本手法では図 2 のような回路となるように DSP48E のパラメータを設定し，モンゴメリ乗算中で積和演算器として利用している．本手法ではモンゴメリ乗算のパイプライン化に合わせて、DSP48E 内のパイプラインレジスタを利用し、動作周波数を向上させている．

パイプライン化モンゴメリ乗算回路を効率的に用いるために，2.2 節で述べた RSA 暗号回路アルゴリズムの拡張を行った．拡張した回路では，6 段にパイプライン化したモンゴメリ乗算回路を用いて，6 つの異なるオペランドを計算する．これにより，平文 $P0, ..., P5$ を入力し暗号文 $C0, ..., C5$ を得る処理を 1 回の RSA 暗号処理で実行できるため，スループットの向上が可能である．

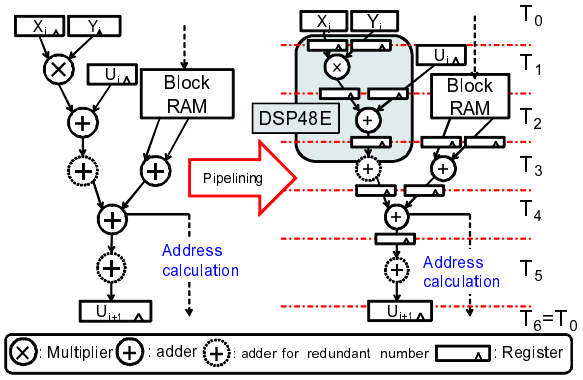


図 1　モンゴメリ乗算のパイプライン化

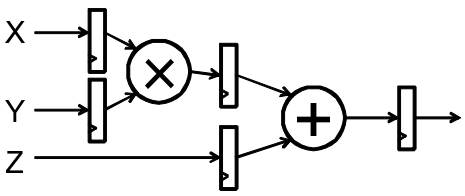


図 2　DSP48E の積和演算器としての利用

4　実験結果と考察

提案するアルゴリズムを Xilinx Virtex-5 ファミリ FPGA である xc5vfx100t-2ff1738 に実装して評価した．ハードウェアアルゴリズムの記述には Verilog HDL を用いた．表 1 にパイプライン化 RSA 暗号回路の実装結果，表 2 にパイプライン化前の RSA 暗号回路 [3] の実装結果について示す．ここでは，128bit から 1024bit までの RSA 暗号回路を実装して評価を行った．SLICE は FPGA のプログラマブルロジックのリソースの使用量を示す．

パイプライン化 RSA 暗号回路では，解を求めるまでの実行時間 (レイテンシ) は増加したが，スループットは 3 倍程度増加している．また，ブロック RAM を用いて途中結果の格納を行っているため，ブロック RAM の使用量は増加したが，SLICE は小さくなっている．両方の回路において冗長表現手法を用いているので，ビット数が増加しても動作周波数はほぼ一定である．

本手法と同様に，[4] では，Xilinx Virtex-4 ファミリ FPGA の信号処理エレメントである DSP48 を用いた高速なべき乗剰余演算を実装している．[4] の手法では，1024 ビットのべき乗剰余演算において実行時間は 1071[μs] で，スループットは 0.96[Mbit/s] となっている．本手法と比較すると，実行時間は [4] の手法が優れているが，スループットは本手法が優れていることがわかる．

表 1　パイプライン化 RSA 暗号回路の実装結果

ビット幅	128	256	512	1024
SLICE	638	1212	2550	5036
DSP48E	8	16	32	64
ブロック RAM	17	32	63	125
動作周波数 [MHz]	293.2	288.4	285.7	264.3
クロックサイクル数	17552	59640	217544	828264
実行時間 [us]	59.85	206.8	761.6	3133
スループット [Mbit/s]	12.83	7.43	4.03	1.96

表 2　パイプライン化前の RSA 暗号回路の実装結果

ビット幅	128	256	512	1024
SLICE	699	1355	2810	5683
DSP48E	8	16	32	64
ブロック RAM	4	8	15	29
動作周波数 [MHz]	86.80	86.52	86.24	86.20
クロックサイクル数	2846	9782	35942	137414
実行時間 [us]	32.79	113.1	416.8	1594
スループット [Mbit/s]	3.9	2.26	1.23	0.64

5　まとめ

本研究では，Xilnx Virtex-5 ファミリ FPGA に特化した高速な RSA 暗号回路を提案し，関連研究と比較して優れたスループットが得られた．今後は，さらなる高速化を目指す．

参考文献

[1] P. L.Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519-521, 1985.

[2] 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.

[3] K. Shigemoto, K. Kawakami, and K. Nakano. Accelerating montgomery modulo multiplication for redundant radix-64k number system on the FPGA using dual-port block RAMs. In *Proc. of International Conference On Embedded and Ubiquitous Computing(EUC)*, pages 44–51, 2008.

[4] D. Suzuki. How to maximize the potential of FPGA resources for modular exponentiation. In P. Paillier and I. Verbauwhede, editors, *Proc. 9th International Workshop on Cryptographic Hardware in Embedded Systems (CHES'07)*, number 4727 in Lecture Notes in Computer Science, pages 272– 288, Springer-Verlag, 2007.

誤差拡散法による集中型ハーフトニング

仲井 英剛　　中野 浩嗣
(広島大学大学院 工学研究科 情報工学専攻)

1　はじめに

コンピュータ上のディジタル画像は，連続した値を持つ連続階調画像として表現される．一方で，ディジタル画像を印刷するには，印刷機はドットを打つか打たないかの二種類の表現しかできないため，「ディジタルハーフトニング」と呼ばれる技術によって，擬似的に元の連続階調画像を表現してやる必要がある．

ディジタルハーフトニングについて，次のように定義する．$N \times M$ の連続階調画像 $G = g(i,j)$，$(0 \leq g(i,j) \leq 1)$ を入力として，同サイズの二値画像 $B = b(i,j)$，$b(i,j) = \{0,1\}$ に変換する．ここで，$(0 \leq i \leq M-1, 0 \leq j \leq N-1)$，“0” = 白，“1” = 黒 として，$g(i,j)$ は値が 1 に近づく程黒色に近づく．図 1 に示す連続階調画像に対するハーフトニングの例を図 2 に示す．これまでに，様々なハーフトニングの手法が提案されており，本研究ではその中でも最もポピュラーな手法の 1 つである誤差拡散法 [1] に注目している．

ハーフトニングによる二値画像を印刷する際，出力される画像が「ドットゲイン」や「ドットロス」といった印刷時の影響を受けてしまい，意図する印刷結果を得ることができない．ドットゲインは，インクのにじみや光学的特性によって，意図するよりもドットが太ってしまうもの，反対にドットロスは，うまくインクが用紙に付着せず，ドットが欠けてしまう現象のことをいう (図 3)．この解決策の 1 つとして，複数のドットを固めて打つ，「集中型ハーフトニング」と呼ばれる方法がある．集中型ハーフトニングは，複数のドットを固めて打つ (クラスタ化) ことで，ドットゲインやドットロスの影響に強い出力画像を生成できる．

誤差拡散法による既存の集中型ハーフトニングに，Feedback Error-Diffusion (EDODF) [2] がある．処理済みの近傍ピクセルの結果をしきい値処理にフィードバックすることで，集中型ハーフトニングを実現している．しかし，この手法の欠点として，画像品質がパラメータに大きく依存し，高品質な画像を生成することが難しいことがあげられる．そのため，品質を高めるための改良案もいくつか提案されている [3]．本研究では，EDODF に代わる，誤差拡散法による全く新しい集中型ハーフトニングのアルゴリズムを提案する．

2　誤差拡散法

誤差拡散法は，ある一定のしきい値と入力画像の各ピクセル値を比較し，出力を決定した際に生じる誤差を未処理の近傍ピクセルに拡散する操作を繰り返すことで出力画像を生成する．

R. Floyd, L. Steinberg[1] による誤差拡散法のアルゴリズムを以下に示す．今，入力画像 G に対して処理誤差を拡散した画像 G' について，現在参照中のピクセル値を $g'(i,j)$，しきい値を t(一般的には 0.5) とすると，出力画像 $b(i,j)$ は以下の式 (1) で表わされる．

$$b(i,j) = \begin{cases} 1 & \text{if } g'(i,j) \geq t \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

ここで，量子化誤差 $e(i,j)$ は式 (2) で計算される．

$$e(i,j) = b(i,j) - g'(i,j) \quad (2)$$

図 1　入力画像 G

図 2　出力画像 B

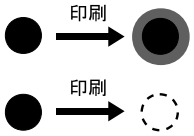


図 3　印刷時の影響

	●	7/16
3/16	5/16	1/16

図 4　Floyd & Steinberg 型

誤差 $e(i,j)$ を式 (3) に従い，近傍ピクセルに誤差を拡散する．

$$g'(i+k, j+l) \leftarrow g'(i+k, j+l) - \omega_{k,l} \cdot e(i,j) \quad (3)$$

$\omega_{k,l}$ は誤差拡散フィルタである．入力画像 $g(i,j)$ に対して，誤差の拡散を行った画像 $g'(i,j)$ は式 (4) で与えられる．

$$g'(i,j) = g(i,j) - \sum_{k,l} \omega_{k,l} \cdot e(i-k, j-l) \quad (4)$$

このように，しきい値処理により発生した量子化誤差を，拡散フィルタに従って近傍ピクセルに拡散させることで，入出力画像間の平均輝度値を保つことができる．[1] では, 図 4 の拡散フィルタが提案されている．

3　提案手法

3.1　アルゴリズム

提案手法では，しきい値処理の際に発生した誤差を，注目ピクセルから少し離れたピクセルに拡散させることによって，クラスタを生成する．

図 5 のように，一般的な誤差拡散法では隣接ピクセルに誤差を拡散していくが，提案手法は，図 6 のように，隣接ピクセルに誤差を拡散させずに，少し離れたピクセルに誤差を拡散していく．隣接ピクセル同士は近い輝度値を持つ可能性が高く，誤差を拡散させないことで，しきい値処理によって同じ色に二値化される可能性が高くなり，クラスタを生成する．

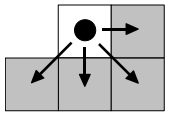


図 5　一般的なフィルタ

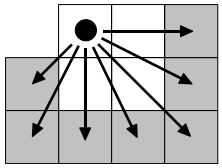
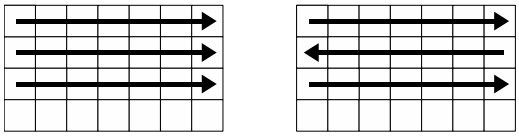


図 6　提案手法



(a) raster scan　　(b) serpentine scan

図 7　走査方法

提案手法において，フィルタ処理以外は，一般的な誤差拡散法と同様となる．画像の走査方向は通常図 7(a) の raster scan が使用されるが，提案手法では，図 7(b) の serpentine scan を使用する．これにより，等方性なクラスタを生成することができ，画像品質を高くすることができる．

4　計算機実験

4.1　フィルタの提案

集中型ハーフトニングを実現する，図 8 に示す 2 つのフィルタを提案する．注目ピクセル (● のピクセル) を含めた 2×2 の領域には，誤差を拡散させない．

	●	0	2/14
2/14	0	0	2/14
1/14	3/14	2/14	1/14
	1/14		

(a) 4×4 フィルタ

		●	0	5/56	4/56
1/56	5/56	0	0	4/56	3/56
	3/56	5/56	4/56	5/56	3/56
	3/56	4/56	3/56	3/56	1/56

(b) 4×6 フィルタ

図 8　提案する拡散フィルタ

図 9–11 に，lena 画像 (256×256) に対して，既存手法 (EDODF) と提案手法による出力画像を示す．EDODF に比べ，提案手法は高品質な画像を生成している．図 12 は，図 11 の一部を拡大したもので，クラスタ同士が重なりあい，画像を形成しているのが分かる．

図 13 は，ramp 画像 (512×64) に対して，それぞれの手法を用いて生成した画像である．EDODF が横方向のクラスタが多いのに対して，提案手法は，様々な方向にクラスタが生成されていることが分かる．

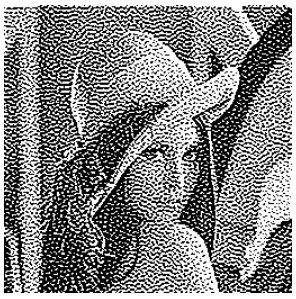


図 9　EDODF

図 10　4×4 フィルタ

図 11　4×6 フィルタ

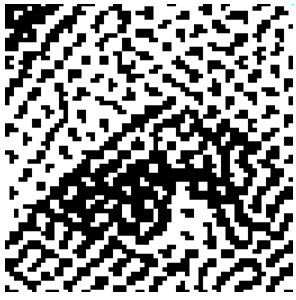


図 12　図 11 の一部を拡大

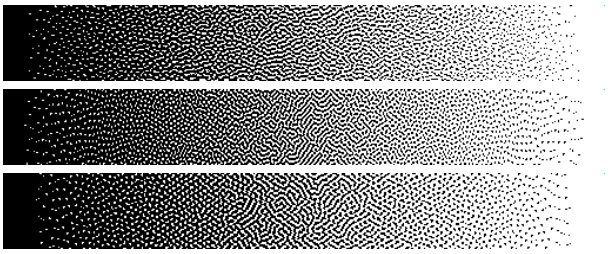


図 13　上から，EDODF，4×4 フィルタ，4×6 フィルタ

4.2　考察

図 8 に示したフィルタは，以下の項目に対して条件を満たすように設計されている．

(i) 誤差を拡散させない領域のサイズ
注目ピクセルを含め 2×2 とする．このサイズによってクラスタのサイズを大きく変更することができる．この領域が 3×3 以上場合，クラスタを構成しないピクセル数が増えてしまい，画像品質の低下を招く．

(ii) 拡散係数の対称性
図 8 のそれぞれのフィルタは，太線によって 5 つの領域に分割している．注目ピクセルを含む領域を中心とし，それぞれ他の領域は左下，下，右下，右の領域とする．ここで，左下と下の領域の拡散係数の合計値と，右下と右の領域の拡散係数の合計値が同じとなるよう設計している．これら 2 つの値が同じでない場合，出力画像へのアーティファクトの原因となる．

(iii) 斜め方向と縦横方向の係数割合
(ii) と同様に，分割された領域に注目する．右と右下のそれぞれの領域の拡散係数の合計値に着目する．この係数値の比は (右 : 右下) = 4 : 3 となっている．この値を変更することで，クラスタの形状を変更できる．右方向の拡散係数を増やすことで，縦方向に伸びるクラスタを，右下方向の拡散係数を増やすことで，斜め方向に伸びるクラスタを生成する．4 : 3 程度が等方的なクラスタを持ち，高品質な出力画像を生成できる．

4×4, 4×6 フィルタを比較すると，両方のフィルタが誤差を拡散させない近傍領域が 2×2 であるのにもかかわらず，4×6 フィルタによる画像の方がクラスタサイズが大きくなっているのが分かる．これは，4×6 フィルタの方が注目ピクセルからより遠くに多くの誤差を拡散しているためと考えられる．

5　まとめ

本研究では，既存手法に代わる新たな誤差拡散法による集中型ハーフトニングのアルゴリズムを提案した．提案手法では，一般的な誤差拡散法から誤差拡散フィルタを変更することによって，クラスタを持つ高品質な画像を生成することができた．

参考文献

[1] R. Floyd and L. Steinberg,“An adaptive algorithm for spatial grayscale,” *Proc. Soc. Image Display,* vol. 17, no. 2, pp. 75–77, 1976.

[2] R. Levien, “Output dependent feedback in error diffusion halftoning,” *Proc. IS&T Imaging Science and Technology,* vol. 1, pp. 115–118, May 1992.

[3] Pingshan Li and Jan P. Allebach, “Clustered Minority Pixel Error Diffusion,” *J. Opt. Soc. Am. A*, vol. 21, pp. 1148–1160, July 2004.

エッジの曲率を用いた画像のパターンマッチング

足立 悦三 (広島大学大学院 工学研究科)

1　はじめに

近年ロボットビジョンや，工場のオートメーション化など多くの分野で高速な画像認識手法が必要とされている．

最も一般的な手法としては，輝度相関を用い，ウィンドウ操作によって類似する部分画像領域を検出する手法が存在する．しかしこの手法は毎回ウィンドウ内の全画素について比較を行うため，検出対象の物体が傾いている場合には計算時間が大きくなる．また，輝度情報に基づいた比較は物体同士の重なりによって一部が隠れているような場合や，背景として検出対象の物体と類似した輝度領域が多く存在する場合においては検出が困難である．

また，様々な特徴量を機械学習によって組み合わせ，特定の検出対象に適した検出器を作成し，これを画像認識に用いる手法も存在する．[1] しかしこの手法は検出対象の画像を大量に用いる必要があり，学習にも多大な時間を必要とする．そのため，新たな検出対象への変更が容易ではない．

本研究では，回転に対し不変なエッジの局所的な曲率を特徴として利用した高速な画像認識手法を提案する．エッジの情報を用いるため，輝度変化が大きい場合であっても安定した検出が期待できる．また，局所的な特徴を組み合わせているため，検出対象の物体の一部が隠れているような場合においても検出が可能である．

2　検出手順

本研究での検出処理は 2 段階に分けて行われる．処理手順を図 1 に示す．

まず事前に登録されたテンプレート画像からエッジの曲率を計算する．エッジの曲率はエッジ画素ごとに得られ，エッジの角の先端部分で最も大きな値を示し，直線部分や曲線部分では低い値を示す．図 1 の曲率値グラフで濃い色で示された山形の区間がエッジの角の部分に対応する．この十分に大きな曲率値を持つグラフ区間をエッジの角の部分の特徴として抽出し，データベースを作成しておく．

検出の 1 段階目は，探索画像からも同様に抽出した曲率値グラフの区間とデータベースとのマッチングである．(図中:マッチング 1) これにより類似した形状をもつ角のペア集合をマッチ区間集合として得る．2 段階目はこのマッチ区間集合から複数のマッチ区間を選択し，幾何学的条件によるマッチングを行う．(図中:マッチング 2) これにより誤検出を排除し，検出対象の位置と姿勢を決定する．

ここで本研究での物体のエッジ検出には Canny Edge Detector を利用した．Canny Edge Detector はノイズに対して頑強で，連続した 1 画素幅のエッジを取得することができるため提案手法での利用に適している．

提案手法ではエッジの角に対応する曲率値の大きなグラフ区間を複数用いる．そのため検出対象の物体のエッジに角が少なく，単調な曲線ばかりで構成された物体の検出には適さない．

2.1　曲率とエッジ方向

連続したエッジ点列から曲率とエッジ方向を取得する．図 2 の様に注目画素から n 番目の近傍画素へのベクトル

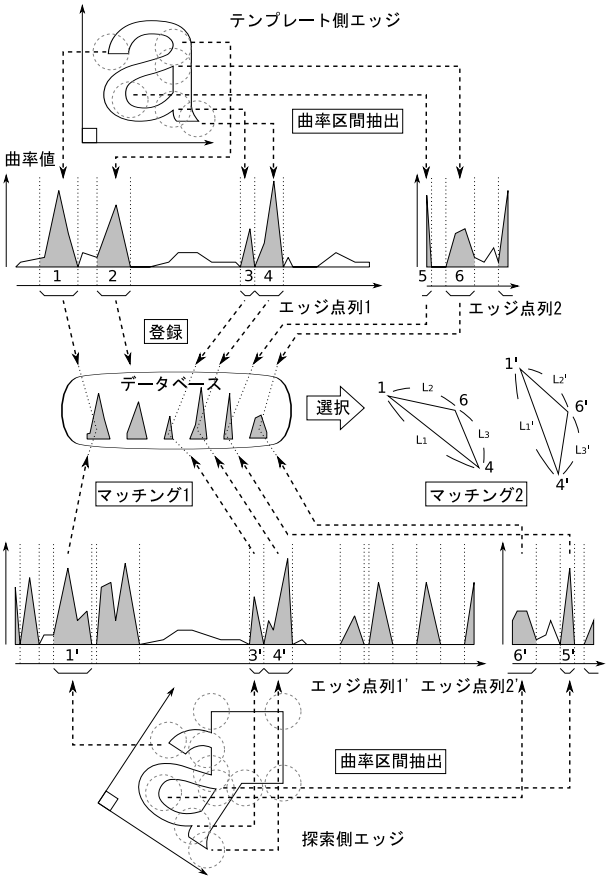


図 1　局所的な曲率変化を用いたマッチング

をそれぞれ $\vec{p_1}, \vec{p_2}$ とする．図 2 は $n = 3$ の場合の例である．

この注目画素における曲率は，$\vec{p_1}$ と $\vec{p_2}$ のなす角 $\theta(0 \leq \theta \leq \pi)$ を用いて計算する．曲がりが大きい時に値が大きくなるよう，曲率を $\pi - \theta$ と定義する．また $\vec{p_1}$ と $\vec{p_2}$ の合計ベクトルの傾き ϕ をエッジ方向として定義する．

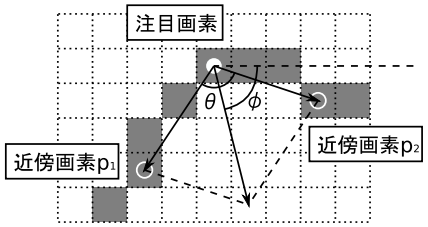


図 2　3 番目の近傍画素へのベクトル

2.2　Peak 区間の抽出

得られた曲率値グラフをもとに，高い曲率値を持ったグラフ区間を Peak 区間として抽出する．Peak 区間のエッジは実際の形状として明確な角を持ち，区間内最大曲率値を示す画素位置によって角の先端位置を特定できる．

Peak 区間の抽出には 2 つの閾値を用る．閾値はそれぞれ t_{high}, t_{low} で，$t_{high} > t_{low}$ とする．Peak 区間の抽出例を図 3 に示す．Peak 区間の条件は，区間内の極大値が t_{high} 以上かつ，極小値が t_{low} 以下となっていることである．この条件によって図 3 では 4 つの Peak 区間が抽出されている．

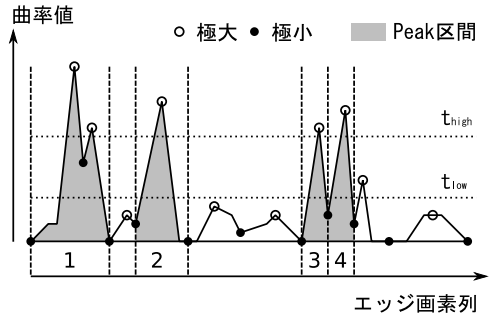


図 3　Peak 区間の抽出

2.3　Peak 区間同士のマッチング

Peak 区間同士のマッチングは，区間内最大曲率値の位置による比較位置合わせを行った上で曲率グラフの類似性を評価する．（図 4）ここでは類似性の評価式として式 (1) を用い，$error$ が閾値以下の組をマッチ区間として選択する．

$$error = \frac{\sum |v_1 - v_2|}{\sum \max(v_1, v_2)} \tag{1}$$

ここで v_1, v_2 は，それぞれ位置合わせ後に同じ位置となった曲率値である．

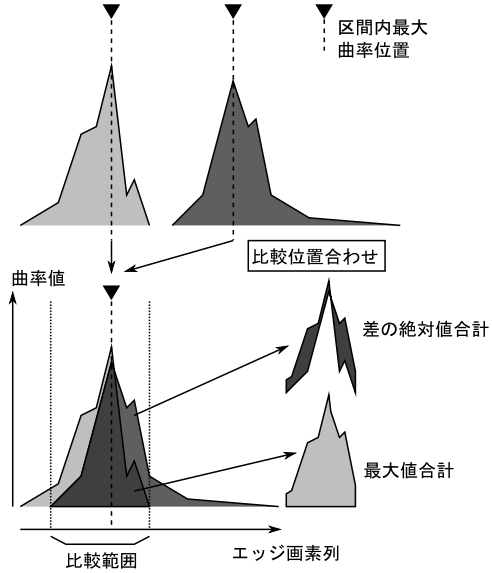


図 4　Peak 区間の比較

2.4　幾何学的マッチング

Peak 区間同士のマッチングで得られたマッチ区間 3 つから成る三角形の幾何学的特徴によってマッチングを行う．本研究で用いた条件は，3 辺の辺長比率，倍率，そして相対的なエッジ方向である．

幾何学的マッチングで得られた 3 点の対応より，検出対象の位置と姿勢が決定できる．ただし，この際 3 点が同一直線上に存在するような場合は正しく姿勢を決定できないため，そのような 3 点は選択しない．

ここで，幾何学的マッチングを行うためには選択した Peak 区間に対応する画素位置を決定する必要がある．これには Peak 区間内で最大となる曲率値を示す画素位置を採用する．曲率値が最大の画素位置は実際の形状として角の先端位置であると考えられ，安定した位置決定が期待できる．

3　検出結果

以下に本手法を用いた検出結果を示す．検出対象の物体にはハサミを用いた．図 5 はテンプレートに用いたエッジ画像である．抽出された Peak 区間に対応する位置を円で，エッジ方向を直線で示している．

図 6 は探索画像のエッジ画像で，検出された位置にテンプレート画像の矩形を表示している．探索画像には検出対象以外の物体も含まれており，互いに重なり合っている．図 6 では検出対象の物体を正しい位置と姿勢で検出している．

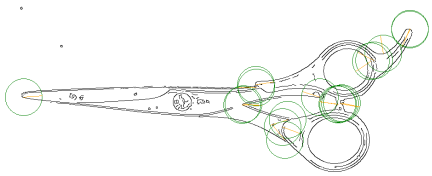


図 5　テンプレートエッジ

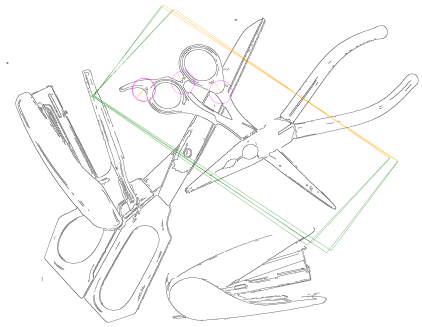


図 6　検出結果

4　まとめ

エッジの局所的な曲率変化の特徴を利用することで，目的の形状の大まかな位置と姿勢を検出することができた．

提案手法では幾何学的マッチングに 3 つの Peak 区間を利用しているため，平均的計算時間は全マッチ区間数を n，テンプレート側の Peak 区間数を m としたとき $O(m^3 \frac{n}{m}^3)$ となる．よって得られるマッチ区間数によって計算時間が大きく変化するという問題がある．この問題に対しては，マッチ区間ごとの投票操作によって解決する予定である．

参考文献

[1] 隆義山下，弘亘藤吉，“特定物体認識に有効な特徴量 (チュートリアル, アンビエント環境知能),” 電子情報通信学会技術研究報告. PRMU, パターン認識・メディア理解，vol.108，no.327，pp.221-236，20081120.

組合わせ円マーカーを用いた部品位置および姿勢の同定

三田 尚義　　中野 浩嗣
(広島大学大学院 工学研究科 情報工学専攻)

1　はじめに

近年，産業用ロボットなどを用いた工場の自動化が盛んに行われる中で，対象物の位置および姿勢を高精度かつ高速に認識する手法が求められている．

中でもビンピッキングと呼ばれる工程は、自動化が非常に困難である．ビンピッキングとは箱などにバラバラに積まれた部品から，対象となる部品を順次取り出してくる作業である．従来の手法では，単純な形状の物体や整列配置された物体を対象にしていた．

そこで本研究では，対象部品の形状を限定することなく，バラ積みされた物体から，対象物体の位置および姿勢を同定することを目的としている．いくつかの円を組合わせたマーカーを提案し，そのマーカーを元にバラ積み部品の中から対象となる部品の位置および姿勢を同定する方法を示している．

2　マーカーを用いた部品位置同定

2.1　部品位置同定までの流れ

本手法では，1 台のカメラから得られる 1 枚のバラ積み部品画像と対象部品のモデルデータを元にした，位置および姿勢の同定を目指している．部品位置および姿勢の同定までの流れを図 1 に示す．

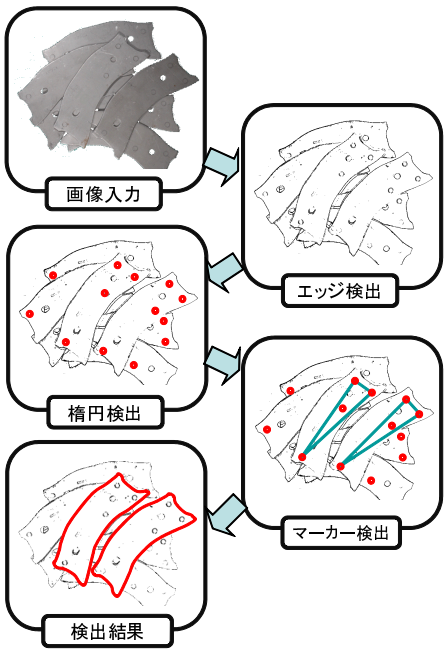


図 1　部品位置同定方法の流れ

まずカメラからバラ積み部品の画像を取得しエッジ検出を行う．次に，検出したエッジ画像から楕円の中心を検出し，検出した楕円中心群の中から組合わせ円マーカーを構成する組合わせを求める．検出したマーカーを元に後述の P3P 問題を解くことで，部品位置および姿勢を同定する．求めた位置および姿勢から部品輪郭を求めて，入力画像と比較する事で正しく同定できているかを確認する．

2.2　組合わせ円マーカー

本手法で用いる組合わせ円マーカー (図 2) について説明する．画像上の 3 つの特徴点を検出することができれば，後述の P3P 問題を解くことで部品の位置および姿勢を同定する事が可能である．そこでマーカーには位置同定のための特徴点として 3 つの円 $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ を配置している．

さらに 3 つの円を一意に識別するための目印点 Gw を重み付き重心の位置に配置している．Gw の座標は式 (1) のように定義する．ただし，a,b,c は互いに異なる自然数とする．

$$G_w = \left(\frac{ax_1 + bx_2 + cx_3}{a+b+c}, \frac{ay_1 + by_2 + cy_3}{a+b+c} \right) \quad (1)$$

目印点 Gw を追加したことにより，マーカー検出を行う際，3 つの特徴点を一意に識別する事が出来るようになった．

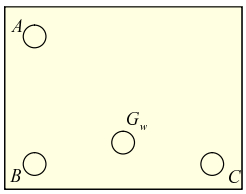


図 2　組合わせ円マーカー

3　P3P 問題

P3P(Perspective 3-Points) 問題 [1],[2] とは，画像中の 3 つの特徴点の座標と 3 次元のモデルデータの座標から対象物体の 3 次元位置を推定する方法である．ここでは入力画像中の特徴点座標を $\mathbf{s_i} = (s_{ix}, s_{iy})^t$，モデルデータの座標系での特徴点座標を $\mathbf{w_i} = (w_{ix}, w_{iy}, w_{iz})^t$，カメラ座標系での特徴点座標を $\mathbf{c_i} = (c_{ix}, c_{iy}, c_{iz})^t$ とする $(i = 1, 2, 3, 4)$.

3 次元位置を推定するということは，図 3 で示しているモデルデータの座標系からカメラ座標系への変換行列を求めるということになる．

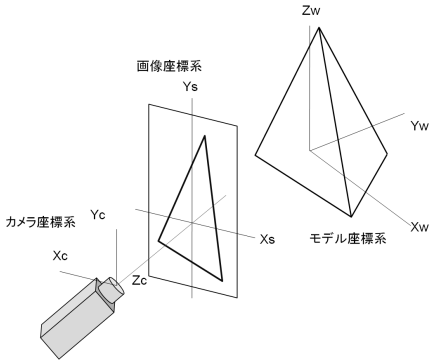


図 3　各座標系の関係

つまり，P3P 問題とは 3 つの特徴点の対応関係から，$\mathbf{c_i} = \mathbf{M}\mathbf{w_i}$ となるような変換行列 $\mathbf{M}$ を求める問題である．また，P3P 問題では最大 4 個の解が求まることが知られている．

4 部品位置同定アルゴリズム

4.1 Step1(エッジ検出)

本手法では，あらかじめ入力画像から Canny 法という手法を用いてエッジ画像を生成しておく．Canny 法では安定したエッジが検出でき，さらにエッジピクセルの総数も他の手法に比べて少なくなるので，高精度かつ高速に画像認識を行う事ができる．

4.2 Step2(楕円検出)

円は傾いても楕円にしか変化しないので，まずはエッジ画像から楕円を検出する．以下に今回使用した楕円検出アルゴリズムの概略を述べる．

まず，エッジ画像からエッジピクセルを探索し，最初に見つかったエッジピクセルを注目点とする．注目点から 15 度ごとにエッジを探索しそれぞれ最初に見つかった点を探索点とする．注目点と 12 個の探索点のうちの 4 点，計 5 点の座標を式 (2)(3) に代入し，連立方程式を解くことで，楕円の中心を推定する．

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \quad (2)$$

$$A + C = 1 \quad (3)$$

次に，さらに精度を高めるために，先ほど推定した楕円の中心から，10 度ごとにエッジを探索し，エッジピクセルを 36 点選択する．得られた 36 点を用いて最小二乗法により最終的な楕円を推定する．

この手法により高速かつ高精度な楕円検出が可能である．

4.3 Step3(組合わせ円マーカーの検出)

組合わせ円マーカーをエッジ画像から検出する方法を示す．楕円検出により検出された中心座標を順に探索し，以下の 2 つの条件を満たすものをマーカーとして検出する．

1. 各楕円の中心を結んでできる $\triangle ABC$ の辺の長さがそれぞれ許容範囲内
2. 重み付き重心 $G_w\left(\frac{ax_1+bx_2+cx_3}{a+b+c}, \frac{ay_1+by_2+cy_3}{a+b+c}\right)$ の位置に，楕円の中心が存在

一つ目の条件により，明らかに大きさの異なるマーカーの候補を排除する．また，二つ目の条件から，選択した三角形がマーカーであるかを判定すると共に，三角形の各頂点 A, B, C を識別することができる．

4.4 Step4(3 次元部品位置同定)

検出された組合わせ円マーカーの特徴点 A, B, C の座標を元に部品の位置および姿勢を同定する．ここでは Fischler と Bolles の解法 [2] を用いる．Fischler と Bolles の解法では画像上の 3 特徴点 $\mathbf{s_i}$ とモデルデータ上の 3 特徴点 $\mathbf{w_i}(i = 1, 2, 3)$，さらに $\mathbf{w_i}$ と $\mathbf{w_j}(i \neq j)$ 間の距離を用いることで，最大 4 個の解 $\mathbf{M_k}(k = 1, 2, 3, 4)$ が求まる．反復最適化を行わずに解析的な数値解を求めている．

さらに，求めた $\mathbf{M_k}$ をそれぞれモデルデータの各点に適用し部品輪郭を推定する．推定部品輪郭と実際の入力のエッジ画像を比較する事で 4 個の解の中から最適な $\mathbf{M}$ を求める．$\mathbf{M}$ はモデルデータからカメラ座標系への変換行列を表すので，対象とする部品の位置および姿勢が同定できたことになる．

5 実験結果

以下に実際の部品を撮影した画像に対して，部品位置同定を行った結果を示す．

図 4 は対象となる部品を 2 つと異なる形状の部品 1 つが含まれた入力画像である．部品表面に組合わせ円マーカーを配置している．図 5 では入力画像に対して Canny 法を用いてエッジ検出を行い，さらに楕円検出を行った結果を示している．組合わせ円マーカーを構成する全ての円が検出されていることが確認できる．図 6 では検出した楕円を元に，組合わせ円マーカーを検出した結果を示している．全ての円が見えているマーカーに関しては，正しく検出できている．最後に図 7 では，求めた変換行列 $\mathbf{M}$ を元に部品の輪郭を描いている．正しく部品の位置および姿勢が同定できていることが確認できた．

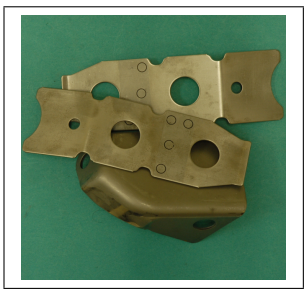


図 4 入力画像

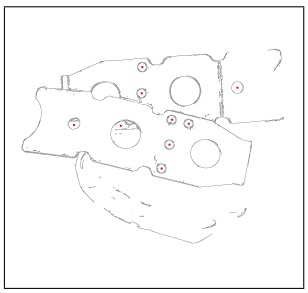


図 5 楕円検出結果

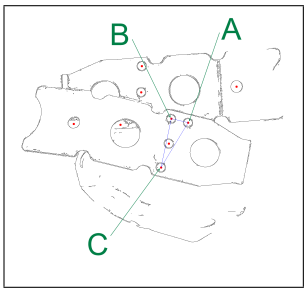


図 6 マーカー検出結果

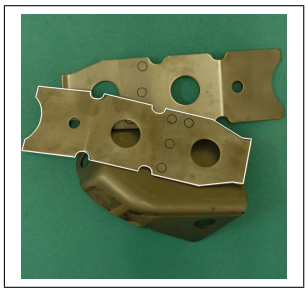


図 7 部品位置同定結果

6 まとめ

本研究では，組合わせ円マーカーを用いることで任意形状の部品位置および姿勢を同定する方法について提案した．組合わせ円マーカーを用いることで，P3P 問題を解く上で必要な対応点を容易に検出する事が可能となった．

実際の撮影画像に対して行った実験でも，一番上の部品を検出し，その位置および姿勢を正しく同定できることを確認した．

参考文献

[1] 大隈隆史，竹村治雄，“拡張現実感システムのための画像からの実時間カメラ位置姿勢推定，” 電子情報通信学会論文誌 ’99/10 Vol. J82-D-II No.10, 1999.

[2] Robert M.Haralick, Chung-Nan Lee, Karsten Ottenberg, Michael Nolle, “Review and Analysis of Solutions of the Three Point Perspective Pose Estimation Problem,” International Journal of Computer Vsion, 13, 3, 331-356, 1994.

膜計算における算術演算，及び，因数分解

神戸 文香　　藤原 暁宏
九州工業大学　大学院情報工学府
a-kambe@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，加算，多入力加算，乗算，因数分解を実行するアルゴリズムを提案する．まず，膜計算において加算を実行するアルゴリズムを示す．このアルゴリズムは，2 つの m ビットの 2 進数の加算を，$O(m^2)$ 種類のオブジェクトを用いて，$O(\log m)$ ステップで実行するものである．次に，膜計算において多入力加算を実行するアルゴリズムを示す．このアルゴリズムは，m 個の $2m$ ビットの 2 進数を，2 個の $2m$ ビットの 2 進数へ束ねた後，加算を実行するもので，$O(m^2)$ 種類のオブジェクトを用いて $O(\log m)$ ステップで実行するものである．また，膜計算において乗算を実行するアルゴリズムを示す．このアルゴリズムは，2 つの m ビットの 2 進数の乗算を，$O(m^2)$ 種類のオブジェクトを用いて $O(\log m)$ ステップで実行するものである．更に，因数分解を実行するアルゴリズムを示す．このアルゴリズムは 2 つの素数 p，q の積からなる正の整数 c に対し，因数分解を実行し，解として 2 つの素数 p，q を出力するもので，$O(m^2)$ 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，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[5, 6] 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 完全問題を解くアルゴリズム [3, 4, 7, 8] が提案されており，また，論理演算，加算を実行するアルゴリズム [2] も提案されている．

このような基本演算に関するアルゴリズムの一つとして，文献 [1] において，多入力加算，乗算，そして因数分解を実行するより高速なアルゴリズムが提案されている．加算アルゴリズムは，n 個の m ビットの 2 進数の加算を求めるものであり，$O(m)$ ステップで実行可能となっている．乗算アルゴリズムは，2 個の m ビットの 2 進数の積を求めるものであり，$O(m \log m)$ ステップで実行可能となっている．また，因数分解アルゴリズムは，2 つの素数 p，q の積からなる 2 進数に対し，因数分解を行い，解として p を出力するものであり，加算と乗算のアルゴリズムを用いて，$O(m \log m)$ ステップで実行可能となっている．

本研究では，この膜計算において，加算，多入力加算，乗算，因数分解を実行するアルゴリズムを提案する．まず，膜計算において加算を実行するアルゴリズムを示す．このアルゴリズムは，2 個の m ビットの 2 進数の加算を $O(\log m)$ ステップで実行するものである．なお，文献 [2] において提案されている加算アルゴリズムは，$O(1)$ ステップで実行可能としているが，進化規則において利用可能なオブジェクトのサイズを $O(m)$ としている．一方，本研究で提案する加算アルゴリズムの進化規則において利用可能なオブジェクトのサイズは $O(1)$ であり，より現実的なオブジェクトのサイズとなっている．

また，乗算アルゴリズムで用いる多入力加算を実行するアルゴリズムを示す．このアルゴリズムは，m 個の $2m$ ビットの 2 進数を，2 個の $2m$ ビットの 2 進数へ束ねた後，加算を実行するもので，$O(m^2)$ 種類のオブジェクトを用いて $O(\log m)$ ステップで実行するものである．

次に，膜計算において乗算を実行するアルゴリズムを示す．このアルゴリズムは，2 個の m ビットの 2 進数の乗算を，$O(m^2)$ 種類のオブジェクトを用いて $O(\log m)$ ステップで実行するものである．

最後に，因数分解を実行するアルゴリズムを示す．このアルゴリズムは 2 つの素数 p，q の積からなる正の整数 c の入力に対し，因数分解を実行し，解として 2 つの素数 p，q を出力するもので，$O(m^2)$ 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

本節では，膜計算の基本要素である膜構造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は，生

物の細胞等をモデル化したものである．図 1 に，基本的な植物の細胞の概念図を示す．

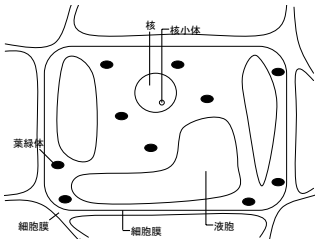


図 1: 植物の細胞

図 1 に示す様に，生物の細胞では，細胞壁等の膜で区切られた空間に細胞膜等の膜が繰り返し含まれている．更に，細胞膜で区切られた空間に液胞や核などが存在し，それぞれの区切られた膜という空間において，生命活動 (計算) が行われていると考えられる．この構造をモデル化したものが**膜構造**である．

ここで，膜計算で用いられる膜構造の例を図 2 に示す．

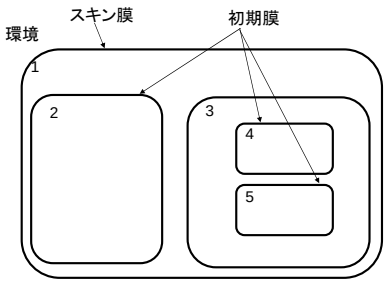


図 2: 膜構造

この膜構造は階層的にとらえることができる．最も外側の膜 (それ以上外側に膜がない膜) を**スキン膜**といい，図 1 の細胞における細胞壁に対応する膜である．逆に，最も内側の膜 (それ以上内側に膜がない膜) を**初期膜**といい，植物の細胞における核小体や液胞に対応する膜である．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 2 の膜構造は，以下のように表される．

$$[\,[\,]_2\,[\,[\,]_4\,[\,]_5\,]_3\,]_1$$

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．

$$[\,[\,]_2\,[\,[\,]_5\,[\,]_4\,]_3\,]_1$$

また，各膜は，電気的極性 $e \in \{-, 0, +\}$ を持つ．例えば，図 2 の膜 5 が $+$ という極性を持つ場合以下のように記述する．

$$[\,[\,]_2\,[\,[\,]_4\,[\,]_5^+\,]_3\,]_1$$

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．オブジェクトとは，図 1 の細胞における葉緑体に対応するものでり，生命活動 (計算) に重要な要素である．例えば，図 2 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．

$$[\,[\,]_2\,[\,[\,]_4\,[\,a\,]_5\,]_3\,]_1$$

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．これは，図 1 の細胞における光合成や細胞分裂に対応するものである．植物は光合成により，二酸化炭素と水から糖類 (炭水化物) を生成する．これは，植物内のオブジェクト (二酸化炭素や水) が別のオブジェクト (糖類) に進化したことを意味する．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，2.1 節で説明した膜の性質を厳密に定義した計算モデルとして，文献 [3] 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．

本稿では，進化規則としては，文献 [3] 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．

$$[\ a\ ]_h^{e_1} \rightarrow [\ b\ ]_h^{e_2}$$

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，

$$[\ a\ ]_h^0 \rightarrow [\ b\ ]_h^0$$

という規則の場合は，

$$a \rightarrow b$$

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．

$$a\ [\ ]_h^{e_1} \rightarrow [\ b\ ]_h^{e_2}$$

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．

$$[\ a\ ]_h^{e_1} \rightarrow b\ [\ ]_h^{e_2}$$

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．

$$[\ a\ ]_h^e \rightarrow b$$

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．

$$[\ a\ ]_h^{e_1} \rightarrow [\ b\ ]_h^{e_2} [\ c\ ]_h^{e_3}$$

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，**最大並列則**と**非決定性**という進化規則の 2 つの特徴について説明する．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．

本節の最後に，例として簡単な P システム Π を示す．この例では，P システム Π を以下のように定義する．

$$\Pi = (O, \mu, \omega_1, \omega_2, R_1, R_2, i_{in}, i_{out})$$

また，Π を構成するそれぞれの集合を以下のように定義する．

- $O = \{a, c(0), c(1), c(2), c(3), c(4)\}$
- $\mu = [\ [\ ]_2\ ]_1$
- $\omega_1 = \phi$
- $\omega_2 = \{ac(0)\}$
- $R_1 = \phi$
- $R_2 = \{a \rightarrow aa, [\ a]_2^+ \rightarrow a[\ ]_2^+, c(0) \rightarrow c(1), c(1) \rightarrow c(2), c(2) \rightarrow c(3), [c(3)]_2 \rightarrow [c(4)]_2^+\}$
- $i_{in} = 2$
- $i_{out} = 1$

この P システムの動作を以下に示す．この P システムの初期状態は以下の通りである．

$$[\ [ac(0)]_2\ ]_1$$

上記の膜構造に適用できる進化規則は，$a \rightarrow aa$，及び，$c(0) \rightarrow c(1)$ である．最大並列則により，この 2 つの規則が並列に適用され，状態は以下のように遷移する．

$$[\ [a^2c(1)]_2\ ]_1$$

上記の膜構造に適用できる進化規則は，$a \rightarrow aa$，及び，$c(1) \rightarrow c(2)$ である．また，オブジェクト a は 2 つ存在するので，それぞれに進化規則が適用され，状態は以下のように遷移する．

$$[\ [a^4c(2)]_2\ ]_1$$

上記の膜構造にも，$a \to aa$，及び，$c(2) \to c(3)$ が適用され，状態は以下のように遷移する．

$$[\ [a^8 c(3)]_2\ ]_1$$

上記の膜構造に適用できる進化規則は，$a \to aa$，及び，$[c(3)]_2 \to [c(4)]_2^+$ である．よって，状態は以下のように進遷移する．

$$[\ [a^{16} c(4)]_2^+\ ]_1$$

これにより，膜 2 が $+$ に帯電したため，進化規則 $[a]_2^+ \to a[\ ]_2^+$ のみが適用され，膜 2 の中に存在するすべてのオブジェクト a が膜 1 に移動する．よって，状態は以下のように遷移する．

$$[a^{16}[c(4)]_2^+\ ]_1$$

上記の膜構造に適用できる進化規則がなく，これ以上遷移することができないので計算は終了し，出力膜に存在するオブジェクト a^{16} が出力となる．出力からわかるように，この P システムは入力されたオブジェクト a を 16 倍にして出力するものである．

3　2 進数を表すデータ表現

本研究では，文献 [2] 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 [2] 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．更に，$V_{i,j} (\in \{0,1\})$ は，各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．

なお，本研究において，ビット数を表す m は 2 のべき乗とする．

4　加算アルゴリズム

本節では，2 個の m ビットの 2 進数の加算を実行するアルゴリズムを示す．なお，文献 [1] では，$O(m)$ ステップで実行可能な加算アルゴリズムが提案されている．また，文献 [2] においては，$O(1)$ ステップで実行可能な加算アルゴリズムが提案されているが，進化規則に利用可能なオブジェクトのサイズの数を $O(m)$ としている．一方，本研究で提案する加算アルゴリズムでは，進化規則に利用可能なオブジェクトのサイズを $O(1)$ としながら，$O(\log m)$ ステップで実行可能としている．

以下では，まず，アルゴリズムの入出力を定義し，次に，アルゴリズムの概要，アルゴリズムの動作，そして，アルゴリズムの計算量を示す．

4.1　入出力

入力 アドレス s, t に格納された 2 つの m ビットの 2 進数を表すオブジェクトの集合を入力とする．
なお，これらのオブジェクトは入力として加算膜に与えられるものとする．また，$V_{s,m-1} = V_{t,m-1} = 0$ とする．

$$\{\langle A_s, B_{m-1}, V_{s,m-1} \rangle, \langle A_s, B_{m-2}, V_{s,m-2} \rangle, \cdots, \langle A_s, B_0, V_{s,0} \rangle\}$$

$$\{\langle A_t, B_{m-1}, V_{t,m-1} \rangle, \langle A_t, B_{m-2}, V_{t,m-2} \rangle, \cdots, \langle A_t, B_0, V_{t,0} \rangle\}$$

出力 アドレス $answer$ に格納された m ビットの 2 進数を表すオブジェクトの集合を出力とする．すなわち，$V_{answer} = V_s + V_t$ となる．

$$\{\langle A_{answer}, B_{m-1}, V_{answer,m-1} \rangle, \langle A_{answer}, B_{m-2}, V_{answer,m-2} \rangle, \vdots \langle A_{answer}, B_0, V_{answer,0} \rangle\}$$

4.2　アルゴリズムの概要

本研究で示す 2 入力 m ビットの 2 進数の加算の操作を説明する．まず，アドレス s,t に格納された m ビットの 2 進数を表わすオブジェクトを，加算膜に入力として与える．なお，説明を簡単にするために，$V_{s,m-1} = V_{t,m-1} = 0$ と仮定する．

$$\{\langle A_s, B_{m-1}, V_{s,m-1} \rangle, \langle A_s, B_{m-2}, V_{s,m-2} \rangle, \cdots, \langle A_s, B_0, V_{s,0} \rangle\}$$

$$\{\langle A_t, B_{m-1}, V_{t,m-1} \rangle, \langle A_t, B_{m-2}, V_{t,m-2} \rangle, \cdots, \langle A_t, B_0, V_{t,0} \rangle\}$$

加算を実行するアルゴリズムは，3 つのステップからなる．Step 1 において，$V_{X,j} = V_{s,j} \oplus V_{t,j}$ と $V_{Y,j} = V_{s,j} \wedge V_{t,j}$ を計算する‡．また，加算膜の初期膜に存在する不要なオ

‡この操作は論理回路における半加算器の動作と同じである．

ブジェクトを削除する．次に，Step 2 において，桁上がりが起こる部分を表す値V_Z を生成する．最後に，Step 3 において，$V_{answer,j} = V_{X,j} \oplus V_{Z,j}$ を計算し，出力する．

以下に加算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．

(1-1) 与えられた入力 V_s と V_t より，$V_{X,j} = V_{s,j} \oplus V_{t,j}$ と $V_{Y,j} = V_{s,j} \wedge V_{t,j}$ を計算する．なお，$V_{X,j}$ は桁上がりを除いた合計，$V_{Y,j}$ は桁上がりを表している．

(1-2) 加算膜の初期膜に存在するオブジェクト $\langle A_i, A_j \rangle$ $(0 \leq j \leq m-1,\ i < j)$ に対し，不要なものを削除する．ここで，$\langle A_i, A_j \rangle$ とは，加算膜に初期状態から存在し，加算において起こる桁上がりの部分を示すオブジェクトである．Step 2 においてオブジェクト $\langle A_i, A_j \rangle$ が存在することは，i ビット目から j ビット目まで桁上がりが起きることを表している．

また，$i \leq k \leq j$ において，$\langle A_X, B_k, 0 \rangle$ が存在する場合は，$\langle A_i, A_j \rangle$ を不必要とみなす．これは，i から j の間に桁上がりが伝播しないことを示すためである．

(1-3) カウンターをリセットすることで，Step 1 を終了とする．

Step 2 V_X において桁上がりが起こる部分を表す値V_Z を生成する．

(2-1) 桁上がりが起こる各ビット j に対して，桁上がりを表す値 $V_{Z,j} = 1$ とする．なお，V_Z は常に，すべての値が 1 になることはない．これは，0 ビット目には前の桁からの桁上がりが起こり得ないため，必ず 0 になるためである．

(2-2) $V_{Z,j} = 1$ が存在しない各ビット j について，$V_{Z,j} = 0$ とする．

また，(2-2) の操作終了とともに加算膜を $+$ に帯電させる．

Step 3 V_X と V_Z を用いて，$V_{answer,j} = V_{X,j} \oplus V_{Z,j}$ を計算する．計算後は $V_{answer,j}$ を外部へ出力することで，加算を終了する．

以下に加算を実行する P システム Π_{add} を示す．

$$\Pi_{add} = (O, \mu, \omega_{add}, R_{add})$$

ここで，Π_{add} を構成するそれぞれの集合は以下のようになる．

- $O = \{\langle A_s, B_j, V_{s,j} \rangle, \langle A_t, B_j, V_{t,j} \rangle, \langle A_{answer}, B_j, V_{answer,j} \rangle,$
  $\langle A_X, B_j, V_{X,j} \rangle, \langle A_Y, B_j, V_{Y,j} \rangle, \langle A_Z, B_j, V_{Z,j} \rangle$
  $\mid 0 \leq j \leq m-1,$
  $V_{s,j}, V_{t,j}, V_{answer,j}, V_{X,j}, V_{Y,j}, V_{Z,j} \in \{0,1\}\}$
  $\cup \{\langle FA1_j \rangle, \langle FA2_j \rangle \mid 0 \leq j \leq m-1\}$
  $\cup \{\langle CA_j(h) \rangle \mid 0 \leq j \leq m-1, 0 \leq h \leq \log m\}$
  $\cup \{\langle A_i, A_j \rangle \mid 0 \leq i < j, 0 \leq j \leq m-1\}$
- $\mu = [\ ]_{add}$
- $\omega_{add} = \{\langle A_s, B_j, V_{s,j} \rangle, \langle A_t, B_j, V_{t,j} \rangle, \langle A_i, A_j \rangle$
  $\mid 0 \leq i < j, 0 \leq j \leq m-1,$
  $V_{s,j}, V_{t,j} \in \{0,1\}\}$
- $R_{add} = R_1 \cup R_2 \cup R_3$
  - $R_1 = R_{1.1} \cup R_{1.2} \cup R_{1.3}$
  - $R_{1.1} = \{$
    $\langle A_s, B_j, 0 \rangle \langle A_t, B_j, 0 \rangle$
    $\rightarrow \langle A_X, B_j, 0 \rangle \langle A_X, B_j, 0 \rangle \langle A_Y, B_j, 0 \rangle$
    $\langle FA1_j \rangle \langle CA_j(0) \rangle,$
    $\langle A_s, B_j, 0 \rangle \langle A_t, B_j, 1 \rangle$
    $\rightarrow \langle A_X, B_j, 1 \rangle \langle A_X, B_j, 1 \rangle \langle A_Y, B_j, 0 \rangle$
    $\langle FA1_j \rangle \langle CA_j(0) \rangle,$
    $\langle A_s, B_j, 1 \rangle \langle A_t, B_j, 0 \rangle$
    $\rightarrow \langle A_X, B_j, 1 \rangle \langle A_X, B_j, 1 \rangle \langle A_Y, B_j, 0 \rangle$
    $\langle FA1_j \rangle \langle CA_j(0) \rangle,$
    $\langle A_s, B_j, 1 \rangle \langle A_t, B_j, 1 \rangle$
    $\rightarrow \langle A_X, B_j, 0 \rangle \langle A_X, B_j, 0 \rangle \langle A_Y, B_j, 1 \rangle$
    $\langle FA1_j \rangle \langle CA_j(0) \rangle \mid 0 \leq j \leq m-1\}$
  - $R_{1.2} = \{$
    $\langle CA_j(h) \rangle \rightarrow \langle CA_j(h+1) \rangle$
    $\mid 0 \leq j \leq m-1, 0 \leq h \leq \log m - 1\}$
  - $R_{1.3} = \{R_{1.3.1} \cup R_{1.3.2}\}$
    - $R_{1.3.1} = \{$
      $\langle A_i, A_j \rangle \langle A_X, B_k, 0 \rangle$
      $\rightarrow \langle A_X, B_k, 0 \rangle \langle A_X, B_k, 0 \rangle$
      $\mid 0 \leq i < j, 0 \leq j \leq m-1, i < k < j\}$
    - $R_{1.3.2} = \{$
      $\langle CA_j(\log m) \rangle \langle FA1_j \rangle$
      $\rightarrow \langle CA_j(0) \rangle \langle FA2_j \rangle \mid 0 \leq j \leq m-1\}$
  - $R_2 = R_{2.1} \cup R_{2.2}$
  - $R_{2.1} = \{$
    $\langle A_X, B_j, 0 \rangle \langle A_i, A_k \rangle \langle A_Y, B_i, 1 \rangle \langle FA2_j \rangle$
    $\rightarrow \langle A_X, B_j, 0 \rangle \langle A_X, B_j, 0 \rangle \langle A_Y, B_i, 1 \rangle$
    $\langle A_Y, B_i, 1 \rangle \langle A_Z, B_k, 1 \rangle$
    $\mid 0 \leq i < j, 0 < j \leq m-1, i \leq k \leq j\}$
  - $R_{2.2} = \{$
    $[\langle CA_j(\log m) \rangle \langle FA2_j \rangle]_{add}$
    $\rightarrow [\langle A_Z, B_j, 0 \rangle]^+_{add} \mid 0 \leq j \leq m-1\}$
  - $R_3 =$
    $[\langle A_X, B_j, 0 \rangle \langle A_Z, B_j, 0 \rangle]^+_{add}$
    $\rightarrow \langle A_{answer}, B_j, 0 \rangle,$
    $[\langle A_X, B_j, 0 \rangle \langle A_Z, B_j, 1 \rangle]^+_{add}$
    $\rightarrow \langle A_{answer}, B_j, 1 \rangle,$
    $[\langle A_X, B_j, 1 \rangle \langle A_Z, B_j, 0 \rangle]^+_{add}$
    $\rightarrow \langle A_{answer}, B_j, 1 \rangle,$
    $[\langle A_X, B_j, 1 \rangle \langle A_Z, B_j, 1 \rangle]^+_{add}$
    $\rightarrow \langle A_{answer}, B_j, 0 \rangle \mid 0 \leq j \leq m-1\}$
  - $R_4 = \{$
    $[\langle A_{answer}, B_j, V_{answer,j} \rangle]_{add}$
    $\rightarrow \langle A_{answer}, B_j, V_{answer,j} \rangle [\ ]_{add}$
    $\mid 0 \leq j \leq m-1, V_{answer} \in \{0,1\}\}$

ここで，アルゴリズム中における各オブジェクトの役割を説明する．

- V_X: V_s と V_t の桁上がりなしの合計を示すオブジェクト
- V_Y: V_s と V_t の桁上がりを示すオブジェクト
- V_Z: V_X において桁上がりが起こる部分を表わすオブジェクト
- $\langle A_i, A_j \rangle$: i ビット目から j ビット目まで桁上がりが伝播することを表すオブジェクト
- $FA1, FA2(Flag\ for\ Addition)$: フラグとなるオブジェクト
- $CA(Counter\ for\ Addition)$: カウンタを表すオブジェクト

4.3 アルゴリズムの動作

本節では，膜計算において，加算を実行する具体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，2 進数 0110 と 0011 の加算を実行する例を示し，以下のオブジェクトが加算膜に入力として与えられるものとする．

$$\langle A_s, B_3, 0\rangle, \langle A_s, B_2, 1\rangle, \langle A_s, B_1, 1\rangle, \langle A_s, B_0, 0\rangle,$$
$$\langle A_t, B_3, 0\rangle, \langle A_t, B_2, 0\rangle, \langle A_t, B_1, 1\rangle, \langle A_t, B_0, 1\rangle$$

Step 1 加算を行うための準備を行う．この膜計算モデルの初期状態は，図 3 である．

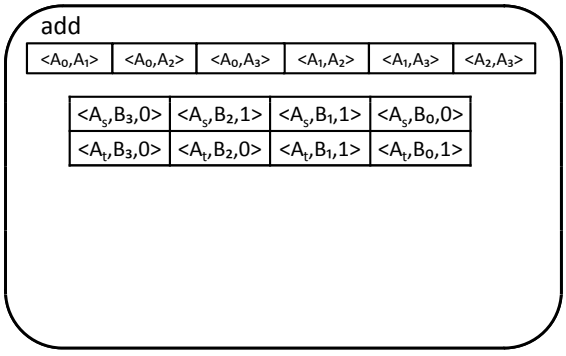


図 3: (1-1) 開始時における膜の状態

(1-1) 入力 V_s と V_t より，$V_{X,j} = V_{s,j} \oplus V_{t,j}$ と $V_{Y,j} = V_{s,j} \wedge V_{t,j}$ を計算する．また，V_X は Step 3 において再び用いるため，2 個生成する．同時に，(1-2) への移行を示す $FA1$ とカウンタである $CA(0)$ を各ビットに生成する．図 3 の状態に対して，以下の進化規則が適用され，図 4 の状態に遷移する．

$$\begin{aligned}
&\langle A_s, B_0, 0\rangle\langle A_t, B_0, 1\rangle \\
&\quad \rightarrow \quad \langle A_X, B_0, 1\rangle\langle A_X, B_0, 1\rangle\langle A_Y, B_0, 0\rangle \\
&\qquad\quad \langle FA1_0\rangle\langle CA_0(0)\rangle \\
&\langle A_s, B_1, 1\rangle\langle A_t, B_1, 1\rangle \\
&\quad \rightarrow \quad \langle A_X, B_1, 0\rangle\langle A_X, B_1, 0\rangle\langle A_Y, B_1, 1\rangle \\
&\qquad\quad \langle FA1_1\rangle\langle CA_1(0)\rangle \\
&\langle A_s, B_2, 1\rangle\langle A_t, B_2, 0\rangle \\
&\quad \rightarrow \quad \langle A_X, B_2, 1\rangle\langle A_X, B_2, 1\rangle\langle A_Y, B_2, 0\rangle \\
&\qquad\quad \langle FA1_2\rangle\langle CA_2(0)\rangle \\
&\langle A_s, B_3, 0\rangle\langle A_t, B_3, 0\rangle \\
&\quad \rightarrow \quad \langle A_X, B_3, 0\rangle\langle A_X, B_3, 0\rangle\langle A_Y, B_3, 0\rangle \\
&\qquad\quad \langle FA1_3\rangle\langle CA_3(0)\rangle
\end{aligned}$$

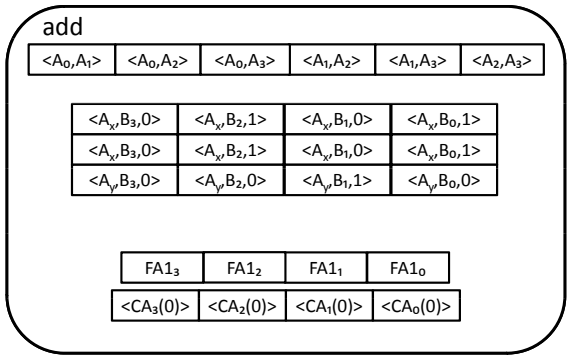


図 4: (1-2) 開始時における膜の状態

(1-2) 不要な $\langle A_0, A_3\rangle, \langle A_0, A_2\rangle$ を $\langle A_X, B_1, 0\rangle$ を用いて削除する．
例えば，$\langle A_0, A_3\rangle$ は，X において，桁上がりが 0 ビット目から 3 ビット目まで伝播することを示すオブジェクトである．しかし，X の 1 ビット目に 0 が存在することは，0 ビット目から 3 ビット目まで桁上がりが伝播しないことを示すため，$\langle A_0, A_3\rangle$ を不要とし，削除する．$\langle A_0, A_2\rangle$ も同様である．
この削除操作は，この例においては，消去するオブジェクト $\langle A_0, A_3\rangle, \langle A_0, A_2\rangle$ に対し，$\langle A_X, B_1, 0\rangle$ が 2 つ存在するため，1 ステップで完了する．前述の状態に対して，以下の進化規則が適用され，図 5 の状態へ進化する．

$$\begin{aligned}
\langle A_0, A_2\rangle\langle A_X, B_1, 0\rangle &\rightarrow \langle A_X, B_1, 0\rangle\langle A_X, B_1, 0\rangle \\
\langle A_0, A_3\rangle\langle A_X, B_1, 0\rangle &\rightarrow \langle A_X, B_1, 0\rangle\langle A_X, B_1, 0\rangle \\
\langle CA_0(0)\rangle &\rightarrow \langle CA_0(1)\rangle \\
\langle CA_1(0)\rangle &\rightarrow \langle CA_1(1)\rangle \\
\langle CA_2(0)\rangle &\rightarrow \langle CA_2(1)\rangle \\
\langle CA_3(0)\rangle &\rightarrow \langle CA_3(1)\rangle
\end{aligned}$$

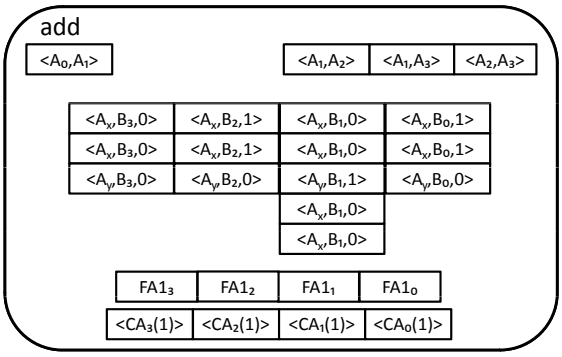


図 5: (1-2) における膜の状態

更に，図 5 の状態に対して，以下の進化規則が適用され，図 6 の状態へ進化する．

$$\begin{aligned}\langle CA_0(1)\rangle &\rightarrow \langle CA_0(2)\rangle\\ \langle CA_1(1)\rangle &\rightarrow \langle CA_1(2)\rangle\\ \langle CA_2(1)\rangle &\rightarrow \langle CA_2(2)\rangle\\ \langle CA_3(1)\rangle &\rightarrow \langle CA_3(2)\rangle\end{aligned}$$

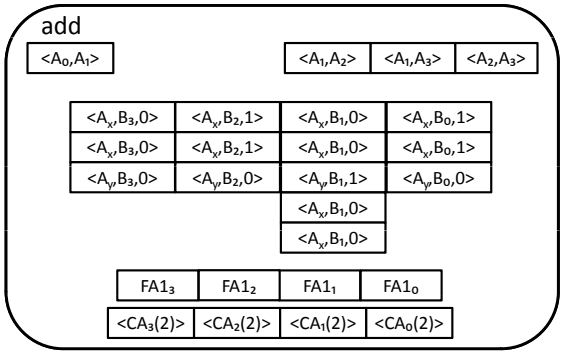


図 6: (1-3) 開始時における膜の状態

(1-3) $CA_0(2), CA_1(2), \cdots, CA_3(2)$ と $FA1_0, FA1_1, \cdots, FA1_3$ を用いることで，Step 1 を終了する．なお，$FA2$ は Step 1 実行後であることを示すオブジェクトである．また，カウンタは 0 にリセットする．前述の状態に対して，以下の進化規則が適用され，図 7 の状態へ進化する．

$$\begin{aligned}\langle CA_0(2)\rangle\langle FA1_0\rangle &\rightarrow \langle CA_0(0)\rangle\langle FA2_0\rangle\\ \langle CA_1(2)\rangle\langle FA1_1\rangle &\rightarrow \langle CA_1(0)\rangle\langle FA2_1\rangle\\ \langle CA_2(2)\rangle\langle FA1_2\rangle &\rightarrow \langle CA_2(0)\rangle\langle FA2_2\rangle\\ \langle CA_3(2)\rangle\langle FA1_3\rangle &\rightarrow \langle CA_3(0)\rangle\langle FA2_3\rangle\end{aligned}$$

Step 2 V_X において桁上がりが起こる部分を表す Z を生成する．

(2-1) $\langle A_1, A_2\rangle$ を用いて，$V_{Z,2} = 1$ を生成する．同時に，不足するオブジェクト $V_{Y,1}$ と $V_{X,3}$ の生成も行う．この例では，前述の状態において，以下の進化規則が適用され，図 8 の状態へ遷移する．

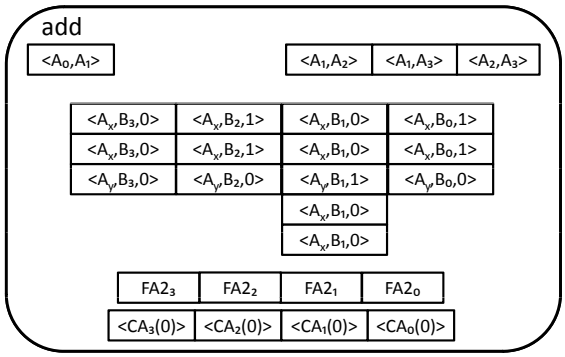


図 7: Step 2 開始時における膜の状態

$$\begin{aligned}&\langle A_X, B_3, 0\rangle\langle A_1, A_2\rangle\langle A_Y, B_1, 1\rangle\langle FA2_2\rangle\\ &\quad\rightarrow \langle A_X, B_3, 0\rangle\langle A_X, B_3, 0\rangle\langle A_Y, B_1, 1\rangle\\ &\qquad\langle A_Y, B_1, 1\rangle\langle A_Z, B_2, 1\rangle\\ &\qquad\langle CA_0(0)\rangle \rightarrow \langle CA_0(1)\rangle\\ &\qquad\langle CA_1(0)\rangle \rightarrow \langle CA_1(1)\rangle\\ &\qquad\langle CA_2(0)\rangle \rightarrow \langle CA_2(1)\rangle\\ &\qquad\langle CA_3(0)\rangle \rightarrow \langle CA_3(1)\rangle\end{aligned}$$

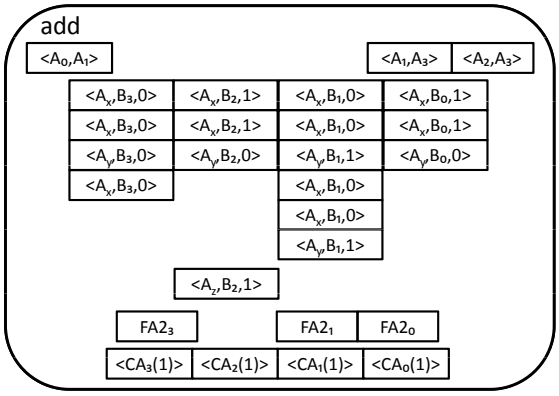


図 8: (2-1) における膜の状態

更に，$\langle A_1, A_3\rangle$ を用いて，$V_{Z,3} = 1$ を生成する．図 8 の状態に対して，以下の進化規則が適用され，図 9 の状態へ進化する．

$$\begin{aligned}&\langle A_X, B_3, 0\rangle\langle A_1, A_3\rangle\langle A_Y, B_1, 1\rangle\langle FA2_3\rangle\\ &\quad\rightarrow \langle A_X, B_3, 0\rangle\langle A_X, B_3, 0\rangle\langle A_Y, B_1, 1\rangle\\ &\qquad\langle A_Y, B_1, 1\rangle\langle A_Z, B_3, 1\rangle\\ &\qquad\langle CA_0(1)\rangle \rightarrow \langle CA_0(2)\rangle\\ &\qquad\langle CA_1(1)\rangle \rightarrow \langle CA_1(2)\rangle\\ &\qquad\langle CA_2(1)\rangle \rightarrow \langle CA_2(2)\rangle\\ &\qquad\langle CA_3(1)\rangle \rightarrow \langle CA_3(2)\rangle\end{aligned}$$

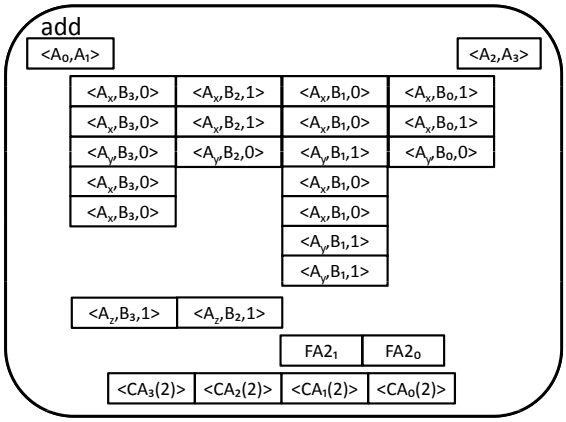


図 9: (2-2) 開始時における膜の状態

(2-2) (2-1) の操作実行後に得るカウンタ $CA_0(2), CA_1(2)$ と $FA2_0, FA2_1$ を用いて，$Z_0 = Z_1 = 0$ を生成する．また，加算膜を $+$ に帯電させることで，Step 2 を終了とする．前述の状態において，以下の進化規則が適用され，図 10 の状態へ遷移する．

$$\begin{aligned}\langle CA_0(2)\rangle\langle FA2_0\rangle &\rightarrow [\langle A_Z, B_0, 0\rangle]^+_{add}\\ \langle CA_1(2)\rangle\langle FA2_1\rangle &\rightarrow [\langle A_Z, B_1, 0\rangle]^+_{add}\end{aligned}$$

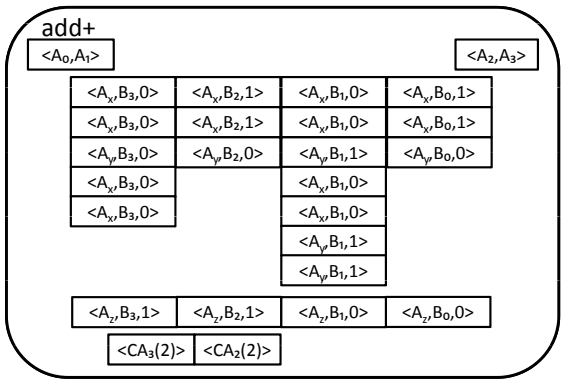


図 10: Step 3 開始時における膜の状態

Step 3 X と Z の排他的論理和を実行し，V_{answer} を生成する．この例では，前述の状態において，以下の進化規則が適用され，図 11 の状態へと遷移する．

$$\begin{aligned}[\langle A_X, B_3, 0\rangle\langle A_Z, B_3, 1\rangle]^+_{add} &\rightarrow \langle A_{answer}, B_3, 1\rangle\\ [\langle A_X, B_2, 1\rangle\langle A_Z, B_2, 1\rangle]^+_{add} &\rightarrow \langle A_{answer}, B_2, 0\rangle\\ [\langle A_X, B_1, 0\rangle\langle A_Z, B_1, 0\rangle]^+_{add} &\rightarrow \langle A_{answer}, B_1, 0\rangle\\ [\langle A_X, B_0, 1\rangle\langle A_Z, B_0, 0\rangle]^+_{add} &\rightarrow \langle A_{answer}, B_0, 1\rangle\end{aligned}$$

更に，V_{answer} を外部へ出力し，計算を終了する．この例では，前述の状態において，以下の進化規則が

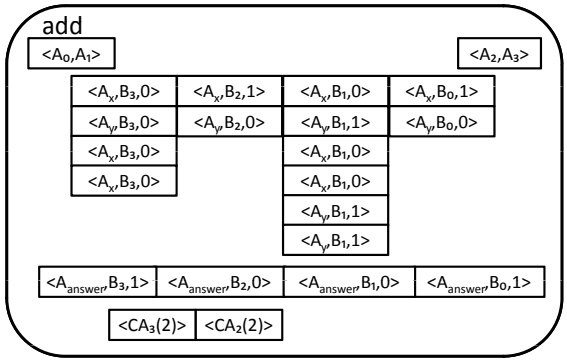


図 11: Step 3 における膜の状態

適用され，加算を終了とする．

$$\begin{aligned}[\langle A_{answer}, B_0, 1\rangle]_{add} &\rightarrow \langle A_{answer}, B_0, 1\rangle[\,]_{add}\\ [\langle A_{answer}, B_1, 0\rangle]_{add} &\rightarrow \langle A_{answer}, B_1, 0\rangle[\,]_{add}\\ [\langle A_{answer}, B_2, 0\rangle]_{add} &\rightarrow \langle A_{answer}, B_2, 0\rangle[\,]_{add}\\ [\langle A_{answer}, B_3, 1\rangle]_{add} &\rightarrow \langle A_{answer}, B_3, 1\rangle[\,]_{add}\end{aligned}$$

4.4 アルゴリズムの計算量

加算を実行する P システム Π_{add} は，$O(m^2)$ 個の進化規則の適用によりアルゴリズムが終了するので，以下の定理が得られる．

定理 1 2 個の m ビットの 2 進数を入力として，加算を実行する P システム Π_{add} は，$O(m^2)$ 種類のオブジェクト，$O(1)$ 個の膜，及び，$O(m^2)$ 個の進化規則を用いて，$O(\log m)$ ステップで実行可能である． □

5 多入力加算アルゴリズム

本節で示す多入力加算アルゴリズムは，m 個の $2m$ ビットの 2 進数の加算を行うものであり，前節で提案した加算アルゴリズムを用いる．つまり，多入力加算膜で m 個あるアドレスを 2 個まで束ね，その後前節で提案した加算膜において 2 入力の加算を行うことで，多入力加算を可能にする．

以下では，まず，アルゴリズムの入出力を定義し，次に，アルゴリズムの概要，アルゴリズムの動作，そして，アルゴリズムの計算量を示す．

5.1 入出力

入力 アドレス $Z, Z+1, \cdots, Z+(m-1)$ に格納された m 個の $2m$ ビットの 2 進数を表すオブジェクトの集合を入力とする．なお，これらのオブジェクトは，多入力加算膜に入力として与えられるものとする．

$$\{\langle A_Z, B_{2m-1}, V_{Z,2m-1}\rangle, \langle A_Z, B_{2m-2}, V_{Z,2m-2}\rangle,$$

$$\cdots, \langle A_Z, B_0, V_{Z,0}\rangle\}$$
$$\{\langle A_{Z+1}, B_{2m-1}, V_{Z+1,2m-1}\rangle, \langle A_{Z+1}, B_{2m-2}, V_{Z+1,2m-2}\rangle,$$
$$\cdots, \langle A_{Z+1}, B_0, V_{Z+1,0}\rangle\}$$
$$\vdots$$
$$\{\langle A_{Z+(m-1)}, B_{2m-1}, V_{Z+(m-1),2m-1}\rangle,$$
$$\langle A_{Z+(m-1)}, B_{2m-2}, V_{Z+(m-1),2m-2}\rangle,$$
$$\cdots, \langle A_{Z+(m-1)}, B_0, V_{Z+(m-1),0}\rangle\}$$

出力 アドレス $answer$ に格納された $2m+2$ ビットの 2 進数を表すオブジェクトの集合を出力とする．

$$\{\langle A_{answer}, B_{2m+1}, V_{answer,2m+1}\rangle,$$
$$\langle A_{answer}, B_{2m}, V_{answer,2m}\rangle,$$
$$\vdots,$$
$$\langle A_{answer}, B_0, V_{answer,0}\rangle\}$$

5.2 アルゴリズムの概要

本研究で提案する多入力加算の概要を説明する．

まず，m 個の $2m$ ビットの 2 進数を，多入力加算膜に入力として与える．

$$\{\langle A_Z, B_{2m-1}, V_{Z,2m-1}\rangle, \langle A_Z, B_{2m-2}, V_{Z,2m-2}\rangle,$$
$$\cdots, \langle A_Z, B_0, V_{Z,0}\rangle\}$$
$$\{\langle A_{Z+1}, B_{2m-1}, V_{Z+1,2m-1}\rangle, \langle A_{Z+1}, B_{2m-2}, V_{Z+1,2m-2}\rangle,$$
$$\cdots, \langle A_{Z+1}, B_0, V_{Z+1,0}\rangle\}$$
$$\vdots$$
$$\{\langle A_{Z+(m-1)}, B_{2m-1}, V_{Z+(2m-1),m-1}\rangle,$$
$$\langle A_{Z+(m-1)}, B_{2m-2}, V_{Z+(2m-1),m-2}\rangle,$$
$$\cdots, \langle A_{Z+(m-1)}, B_0, V_{Z+(m-1),0}\rangle\}$$

多入力加算を実行するアルゴリズムは，4 つのステップからなる．Step 1 において，各ビット毎に m 個存在するオブジェクトを多くとも 2 個に束ねる．次に，Step 2 において，2 個に束ねられたオブジェクトをアドレス s, t に格納する．更に，Step 3 において，前章で提案した加算膜へオブジェクト V_s, V_t を入力として与え，加算膜は加算結果である V_{answer} を多入力加算膜へ出力する．最後に，加算膜から得た V_{answer} を外部へ出力して，計算終了とする．

以下に多入力加算を行うアルゴリズムの概要を示す．

Step 1 各ビット毎に m 個存在するオブジェクトを多くとも 2 個に束ねる．

(1-1) まず，各アドレス，各ビットに 0 が存在する場合，後の計算では不要なものなので削除する．また，以降の操作において，アドレス $Z, Z+1, \cdots, Z+(m-1)$ におけるオブジェクトについて，j ビット目に 1 が存在するという情報のみあればよいので，各ビットをアドレスから開放する．このとき，j ビット目のオブジェクトを $\langle B_j, V_j\rangle$ と表すこととする．

(1-2) $0 \leq j \leq 2m-1$ において，j ビット目に 1 が 2 つ以上存在する場合，2 つの 1 を反応させ，$j+1$ ビット目に 1 つの 1 を生成する．この操作を各ビットで並列に行う．この操作を $\log m + 1$ 回行うことで，各ビットに m 個存在するオブジェクトは多くとも 2 個になる．これは，$\log m + 1$ 回後に，各ビットに存在する 1 の個数が多くとも 2 個になるということであり，詳しい証明は 5.4 節において行う．

(1-3) 多入力加算膜を $+$ に帯電させることにより，Step 1 の終了を表す．

Step 2 Step 1 において，各ビットに多くとも 2 つ存在するオブジェクトをアドレス s, t に格納する．なお，$2m+2$ ビットあるアドレス s, t は元々多入力加算膜に存在しているものとし，初期状態で持つ値はすべて 0 とする．

(2-1) j ビット目に 2 つの 1 が存在する場合，アドレス t の $j+1$ ビット目に 1 を入力し，2 つの 1 を削除する．

(2-2) j ビット目に 1 つの 1 が存在する場合，アドレス s の j ビット目に 1 を入力する．

Step 3 加算膜に V_s, V_t を入力として与えることで，V_s と V_t の加算を行い，計算結果として V_{answer} が多入力加算膜に出力される．

Step 4 加算膜からの出力として V_{answer} を得ると，V_{answer} を外部へ出力し，計算終了とする．

以下に多入力加算を実行する P システム Π_{m_add} を示す．

$$\Pi_{m\_add} = (O, \mu, \omega_{m\_add}, R_{m\_add})$$

- $O = \{\langle A_{Z+i}, B_j, V_{Z+i,j}\rangle, \langle B_j, V_j\rangle$
  $\mid 0 \leq i \leq m-1, 0 \leq j \leq 2m-1,$
  $V_{Z+i,j} \in \{0,1\}\}$
  $\cup \{\langle A_s, B_j, V_{s,j}\rangle, \langle A_t, B_j, V_{t,j}\rangle,$
  $\langle A_{answer}, B_j, V_{answer,j}\rangle$
  $\mid 0 \leq j \leq 2m+1,$
  $V_{s,j}, V_{t,j}, V_{answer,j} \in \{0,1\}\}$
  $\cup \{\langle FM_j\rangle \mid 0 \leq j \leq 2m+1\}$
  $\cup \{\langle CM_j(h)\rangle$
  $\mid 0 \leq j \leq 2m+1, 0 \leq h \leq \log m + 4\}$
- $\mu = [\ [\ ]_{add}\ ]_{m\_add}$
- $\omega_{m\_add} = \{\langle A_{Z+i}, B_j, V_{Z+i,j}\rangle, \langle A_s, B_j, 0\rangle,$
  $\langle A_t, B_j, 0\rangle, \langle CM_j(0)\rangle$
  $\mid 0 \leq i \leq m-1, 0 \leq j \leq 2m+1,$
  $V_{Z+i,j}, V_{s,j}, V_{t,j} \in \{0,1\}\}$
- $R_{m\_add} = R_1 \cup R_2 \cup R_3$
  - $R_1 = R_{1.1} \cup R_{1.2} \cup R_{1.3} \cup R_{1.4}$

· $R_{1.1} = \{$
$[\langle A_{Z+k}, B_j, 0\rangle]_{m\_add} \to [\ ]_{m\_add}$,
$\langle A_{Z+k}, B_j, 1\rangle \to \langle B_j, 1\rangle$
$\mid 0 \le k \le m-1, 0 \le j \le 2m-1\}$

· $R_{1.2} = \{$
$\langle B_j, 1\rangle\langle B_j, 1\rangle \to \langle B_{j+1}, 1\rangle$
$\mid 0 \le j \le 2m-1\}$

· $R_{1.3} = \{$
$\langle CM_j(h)\rangle \to \langle CM_j(h+1)\rangle$
$\mid 0 \le j \le 2m+1, 0 \le h \le \log m+1\}$

· $R_{1.4} = \{$
$[\langle CM_j(\log m+2)\rangle]_{m\_add}$
$\to [\langle CM_j(\log m+3)\rangle]^+_{m\_add}$
$\mid 0 \le j \le 2m+1\}$

– $R_2 = R_{2.1} \cup R_{2.2}$

· $R_{2.1} = R_{2.1.1} \cup R_{2.1.2}$

* $R_{2.1.1} = \{$
$[\langle B_j, 1\rangle\langle B_j, 1\rangle\langle A_t, B_{j+1}, 0\rangle]^+_{m\_add}$
$\to [\langle A_t, B_{j+1}, 1\rangle]^+_{m\_add} \mid 0 \le j \le 2m+1\}$

* $R_{2.1.2} = \{$
$[\langle CM_j(\log m+3)\rangle]^+_{m\_add}$
$\to [\langle CM_j(\log m+4)\rangle\langle FM_j\rangle]^+_{m\_add}$
$\mid 0 \le j \le 2m+1\}$

· $R_{2.2} = R_{2.2.1} \cup R_{2.2.2}$

* $R_{2.2.1} = \{$
$[\langle B_j, 1\rangle\langle A_s, B_j, 0\rangle\langle FM_j\rangle]^+_{m\_add}$
$\to [\langle A_s, B_j, 1\rangle]^+_{m\_add} \mid 0 \le j \le 2m+1\}$

* $R_{2.2.2} = \{$
$[\langle CM_j(\log m+4)\rangle]^+_{m\_add}$
$\to [\langle CM_j(\log m+5)\rangle]^+_{m\_add}$
$\mid 0 \le j \le 2m+1\}$

– $R_3 = \{$
$[\langle A_s, B_j, V_{s,j}\rangle\langle A_t, B_j, V_{t,j}\rangle\langle CM_j(\log m+5)\rangle]^+_{m\_add}$
$\to [\ [\langle A_s, B_j, V_{s,j}\rangle\langle A_t, B_j, V_{t,j}\rangle]_{add}]_{m\_add}$
$\mid 0 \le j \le 2m+1, V_{s,j}, V_{t,j} \in \{0,1\}\}$

– $R_4 = \{$
$[\langle A_{answer}, B_j, V_{answer,j}\rangle]_{m\_add}$
$\to \langle A_{answer}, B_j, V_{answer,j}\rangle[\ ]_{m\_add}$
$\mid 0 \le j \le 2m+1, V_{answer,j} \in \{0,1\}\}$

ここで，アルゴリズム中における各オブジェクトの役割を説明する．

- V_s, V_t: 初期膜に存在する，各ビット毎に多くとも 2 個にまとめた値を格納するオブジェクト
- FM($Flag\ for\ Multi\ addition$): (2-1) 実行後であることを示すオブジェクト
- CM($Counter\ for\ Multi\ addition$): 初期膜に存在するカウンタを表すオブジェクト

5.3 アルゴリズムの動作

本節では，膜計算において，多入力加算を実行する具体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，以下の 4 入力 8 ビットの 2 進数が，多入力加算膜の入力として与えられるものとする．

$\{\langle A_Z, B_7, 0\rangle, \langle A_Z, B_6, 0\rangle, \langle A_Z, B_5, 0\rangle, \langle A_Z, B_4, 0\rangle,$
$\langle A_Z, B_3, 1\rangle, \langle A_Z, B_2, 1\rangle, \langle A_Z, B_1, 1\rangle, \langle A_Z, B_0, 1\rangle$

$\langle A_{Z+1}, B_7, 0\rangle, \langle A_{Z+1}, B_6, 0\rangle, \langle A_{Z+1}, B_5, 0\rangle, \langle A_{Z+1}, B_4, 1\rangle,$
$\langle A_{Z+1}, B_3, 1\rangle, \langle A_{Z+1}, B_2, 1\rangle, \langle A_{Z+1}, B_1, 1\rangle, \langle A_{Z+1}, B_0, 0\rangle$

$\langle A_{Z+2}, B_7, 0\rangle, \langle A_{Z+2}, B_6, 0\rangle, \langle A_{Z+2}, B_5, 1\rangle, \langle A_{Z+2}, B_4, 1\rangle,$
$\langle A_{Z+2}, B_3, 1\rangle, \langle A_{Z+2}, B_2, 1\rangle, \langle A_{Z+2}, B_1, 0\rangle, \langle A_{Z+2}, B_0, 0\rangle$

$\langle A_{Z+3}, B_7, 0\rangle, \langle A_{Z+3}, B_6, 1\rangle, \langle A_{Z+3}, B_5, 1\rangle, \langle A_{Z+3}, B_4, 1\rangle,$
$\langle A_{Z+3}, B_3, 1\rangle, \langle A_{Z+3}, B_2, 0\rangle, \langle A_{Z+3}, B_1, 0\rangle, \langle A_{Z+3}, B_0, 0\rangle\}$

Step 1 各ビット毎に 4 個ずつ存在するオブジェクトを多くとも 2 個に束ねる．この膜計算モデルの初期状態は，図 12 である．

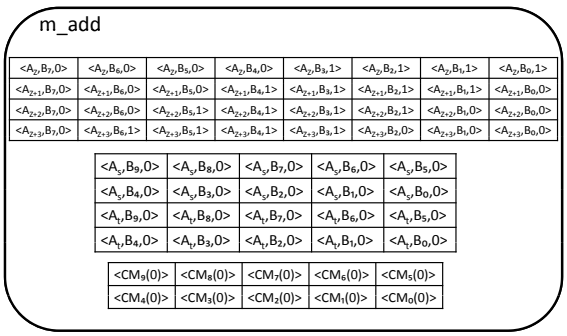


図 12: (1-1) 開始時における膜の状態

(1-1) 各ビットに 1 が存在する場合，アドレスから解放する．同時に各ビットに 0 が存在する場合，それらのオブジェクトを削除する．また，$R_{1.3}$ が適用され，カウンタが増加する．まず，以下の進化規則が適用され，図 13 の状態へと遷移する．

$$
\begin{aligned}
[\langle A_Z, B_4, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_Z, B_5, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_Z, B_6, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_Z, B_7, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_{Z+1}, B_0, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_{Z+1}, B_5, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_{Z+1}, B_6, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_{Z+1}, B_7, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_{Z+2}, B_0, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_{Z+2}, B_1, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}\\
[\langle A_{Z+2}, B_6, 0\rangle]_{m\_add} &\to [\ ]_{m\_add}
\end{aligned}
$$

$$
\begin{aligned}
[\langle A_{Z+2}, B_7, 0\rangle]_{m\_add} &\rightarrow [\,]_{m\_add}\\
[\langle A_{Z+3}, B_0, 0\rangle]_{m\_add} &\rightarrow [\,]_{m\_add}\\
[\langle A_{Z+3}, B_1, 0\rangle]_{m\_add} &\rightarrow [\,]_{m\_add}\\
[\langle A_{Z+3}, B_2, 0\rangle]_{m\_add} &\rightarrow [\,]_{m\_add}\\
[\langle A_{Z+3}, B_7, 0\rangle]_{m\_add} &\rightarrow [\,]_{m\_add}
\end{aligned}
$$

$$
\begin{aligned}
\langle A_Z, B_0, 1\rangle &\rightarrow \langle B_0, 1\rangle\\
\langle A_Z, B_1, 1\rangle &\rightarrow \langle B_1, 1\rangle\\
\langle A_Z, B_2, 1\rangle &\rightarrow \langle B_2, 1\rangle\\
\langle A_Z, B_3, 1\rangle &\rightarrow \langle B_3, 1\rangle\\
\langle A_{Z+1}, B_1, 1\rangle &\rightarrow \langle B_1, 1\rangle\\
\langle A_{Z+1}, B_2, 1\rangle &\rightarrow \langle B_2, 1\rangle\\
\langle A_{Z+1}, B_3, 1\rangle &\rightarrow \langle B_3, 1\rangle\\
\langle A_{Z+1}, B_4, 1\rangle &\rightarrow \langle B_4, 1\rangle\\
\langle A_{Z+2}, B_2, 1\rangle &\rightarrow \langle B_2, 1\rangle\\
\langle A_{Z+2}, B_3, 1\rangle &\rightarrow \langle B_3, 1\rangle\\
\langle A_{Z+2}, B_4, 1\rangle &\rightarrow \langle B_4, 1\rangle\\
\langle A_{Z+2}, B_5, 1\rangle &\rightarrow \langle B_5, 1\rangle\\
\langle A_{Z+3}, B_3, 1\rangle &\rightarrow \langle B_3, 1\rangle\\
\langle A_{Z+3}, B_4, 1\rangle &\rightarrow \langle B_4, 1\rangle\\
\langle A_{Z+3}, B_5, 1\rangle &\rightarrow \langle B_5, 1\rangle\\
\langle A_{Z+3}, B_6, 1\rangle &\rightarrow \langle B_6, 1\rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle CM_0(0)\rangle &\rightarrow \langle CM_0(1)\rangle\\
\langle CM_1(0)\rangle &\rightarrow \langle CM_1(1)\rangle\\
&\vdots\\
\langle CM_9(0)\rangle &\rightarrow \langle CM_9(1)\rangle
\end{aligned}
$$

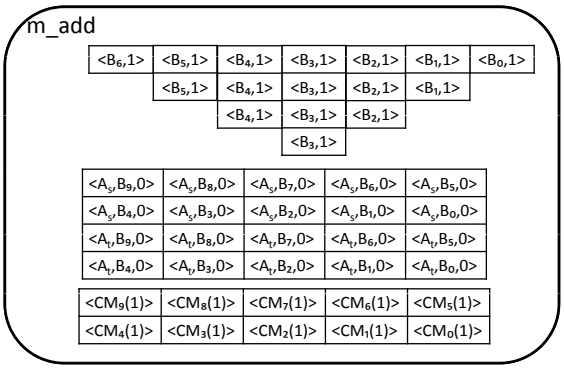


図 13: (1-1) における 1 ステップ目の膜の状態

(1-2) 各ビットに 1 が 2 つ以上ある場合，2 つの 1 を反応させ，次のビットに 1 つの 1 を生成する．また，$R_{1.3}$ が適用され，カウンタが増加する．この操作は，$\log_2 4 + 1 = 3$ 回行う．まず，以下の進化規則が適用され，図 14 の状態へと遷移する．

$$
\langle B_1, 1\rangle\langle B_1, 1\rangle \rightarrow \langle B_2, 1\rangle
$$

$$
\begin{aligned}
\langle B_2, 1\rangle\langle B_2, 1\rangle &\rightarrow \langle B_3, 1\rangle\\
\langle B_3, 1\rangle\langle B_3, 1\rangle &\rightarrow \langle B_4, 1\rangle\\
\langle B_4, 1\rangle\langle B_4, 1\rangle &\rightarrow \langle B_5, 1\rangle\\
\langle B_5, 1\rangle\langle B_5, 1\rangle &\rightarrow \langle B_6, 1\rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle CM_0(1)\rangle &\rightarrow \langle CM_0(2)\rangle\\
\langle CM_1(1)\rangle &\rightarrow \langle CM_1(2)\rangle\\
&\vdots\\
\langle CM_9(1)\rangle &\rightarrow \langle CM_9(2)\rangle
\end{aligned}
$$

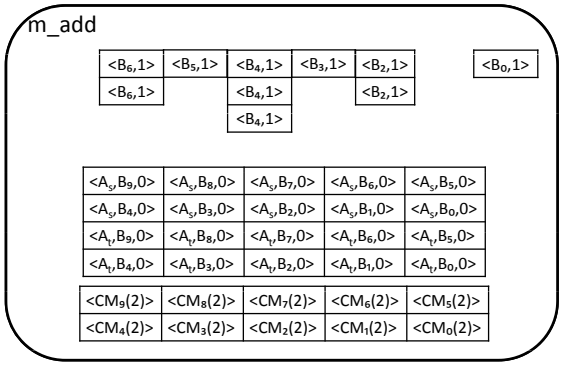


図 14: (1-2) における 1 ステップ目の膜の状態

更に，図 14 に，以下の進化規則が適用され，図 15 の状態に遷移する．

$$
\begin{aligned}
\langle B_2, 1\rangle\langle B_2, 1\rangle &\rightarrow \langle B_3, 1\rangle\\
\langle B_4, 1\rangle\langle B_4, 1\rangle &\rightarrow \langle B_5, 1\rangle\\
\langle B_6, 1\rangle\langle B_6, 1\rangle &\rightarrow \langle B_7, 1\rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle CM_0(2)\rangle &\rightarrow \langle CM_0(3)\rangle\\
\langle CM_1(2)\rangle &\rightarrow \langle CM_1(3)\rangle\\
&\vdots\\
\langle CM_9(2)\rangle &\rightarrow \langle CM_9(3)\rangle
\end{aligned}
$$

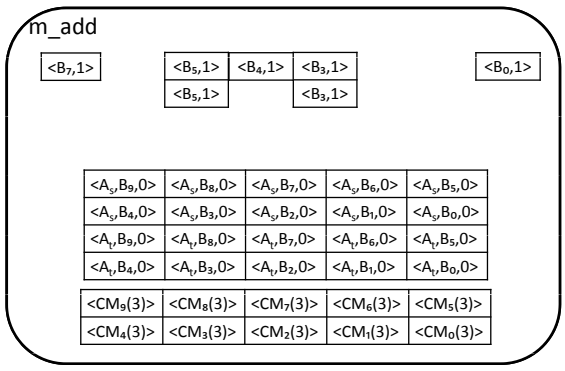


図 15: (1-2) における 2 ステップ目の膜の状態

更に，図 15 に，以下の進化規則が適用され，図 16 の状態に遷移する．

$$\begin{aligned}
\langle B_3,1\rangle\langle B_3,1\rangle &\rightarrow \langle B_4,1\rangle\\
\langle B_5,1\rangle\langle B_5,1\rangle &\rightarrow \langle B_6,1\rangle\\
[\langle CM_0(3)\rangle]_{m_add} &\rightarrow [\langle CM_0(4)\rangle]^+_{m_add}\\
[\langle CM_1(3)\rangle]_{m_add} &\rightarrow [\langle CM_1(4)\rangle]^+_{m_add}\\
&\vdots\\
[\langle CM_9(3)\rangle]_{m_add} &\rightarrow [\langle CM_9(4)\rangle]^+_{m_add}
\end{aligned}$$

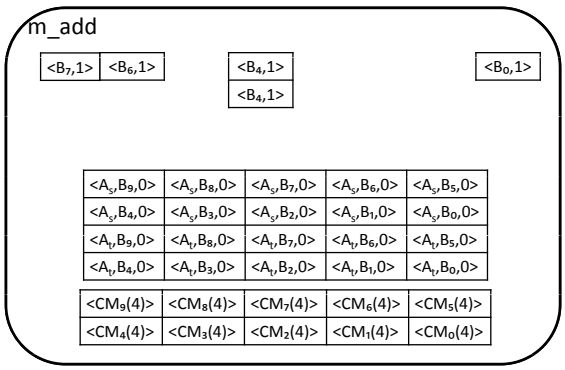


図 16: (1-3) 開始時における膜の状態

(1-3) (1-2) において生成された $CM(4)$ を用いて，多入力加算膜を帯電させ，Step 1 を終了とする．この例では，図 16 において，以下の進化規則を適用し，図 17 の状態に遷移する．

$$\begin{aligned}
[\langle CM_0(4)\rangle]_{m_add} &\rightarrow [\langle CM_0(5)\rangle]^+_{m_add}\\
[\langle CM_1(4)\rangle]_{m_add} &\rightarrow [\langle CM_1(5)\rangle]^+_{m_add}\\
&\vdots\\
[\langle CM_9(4)\rangle]_{m_add} &\rightarrow [\langle CM_9(5)\rangle]^+_{m_add}
\end{aligned}$$

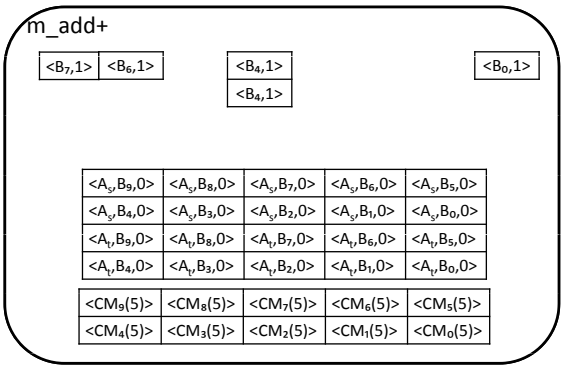


図 17: Step 2 開始時における膜の状態

Step 2 各ビットに多くとも 2 つ存在するオブジェクトをアドレス s,t に格納する．なお，10 ビット分のアドレス s,t の 2 進数を表すオブジェクトは元々多入力加算膜に存在しているものとし，初期状態で持つ値はすべて 0 とする．

(2-1) j ビット目に 2 つの 1 を表すオブジェクトが存在するときに，2 つの 1 を用いて，アドレス t の j ビット目を 1 にし，1 を表すオブジェクト 2 つを削除する．ここでは，4 ビット目に，2 つの 1 を表すオブジェクトが存在するので，アドレス t の 5 ビット目に 1 を格納する．また，$CM(5)$ を用いて，(2-1) 実行後であることを表す FM と，カウンタである $CM(6)$ を各ビットに生成する．図 17 において，以下の進化規則を適用し，図 18 の状態に遷移する．

$$[\langle B_4,1\rangle\langle B_4,1\rangle\langle A_t,B_5,0\rangle]^+_{m\_add} \rightarrow [\langle A_t,B_5,1\rangle]^+_{m\_add}$$

$$\begin{aligned}
[\langle CM_0(5)\rangle]^+_{m_add} &\rightarrow [\langle CM_0(6)\rangle\langle FM_0\rangle]^+_{m_add}\\
[\langle CM_1(5)\rangle]^+_{m_add} &\rightarrow [\langle CM_1(6)\rangle\langle FM_1\rangle]^+_{m_add}\\
&\vdots\\
[\langle CM_9(5)\rangle]^+_{m_add} &\rightarrow [\langle CM_9(6)\rangle\langle FM_9\rangle]^+_{m_add}
\end{aligned}$$

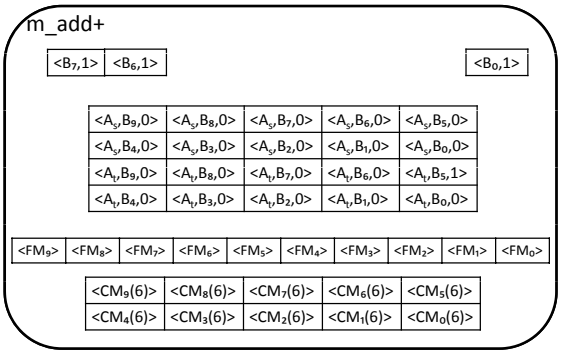


図 18: (2-2) 開始時における膜の状態

(2-2) j ビット目に存在する 1 つの 1 と，(2-1) において生成した FM を用いて，アドレス s の j ビット目に 1 を生成する．ここでは，0 ビット目と 5 ビット目と 7 ビット目に 1 つの 1 が存在するので，$V_{s,0}$，$V_{s,5}$，及び，$V_{s,7}$ に 1 を格納する．同時に，$CM(6)$ を $CM(7)$ に進化させる．図 18 において，以下の進化規則を適用し，図 19 の状態に遷移する．

$$\begin{aligned}
&[\langle B_0,1\rangle\langle A_s,B_0,0\rangle\langle FM_0\rangle]^+_{m_add}\\
&\quad\rightarrow [\langle A_s,B_0,1\rangle]^+_{m_add}\\
&[\langle B_6,1\rangle\langle A_s,B_6,0\rangle\langle FM_6\rangle]^+_{m_add}\\
&\quad\rightarrow [\langle A_s,B_6,1\rangle]^+_{m_add}\\
&[\langle B_7,1\rangle\langle A_s,B_7,0\rangle\langle FM_7\rangle]^+_{m_add}\\
&\quad\rightarrow [\langle A_s,B_7,1\rangle]^+_{m_add}\\
&[\langle CM_0(6)\rangle]^+_{m_add} \rightarrow [\langle CM_0(7)\rangle]^+_{m_add}\\
&[\langle CM_1(6)\rangle]^+_{m_add} \rightarrow [\langle CM_1(7)\rangle]^+_{m_add}\\
&\vdots\\
&[\langle CM_9(6)\rangle]^+_{m_add} \rightarrow [\langle CM_9(7)\rangle]^+_{m_add}
\end{aligned}$$

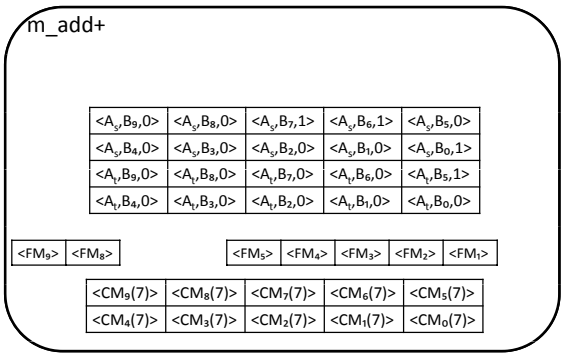


図 19: Step 3 開始時における膜の状態

Step 3 $CM(6)$ を用いて，多入力加算膜が含む加算膜に V_s, V_t を入力として与えることで，V_s, V_t の加算を行う．図 19 において，以下の進化規則を適用し，図 20 の状態に遷移する．

$$
\begin{aligned}
&[\langle A_s, B_0, 1\rangle\langle A_t, B_0, 0\rangle\langle CM_0(7)\rangle]^+_{m\_add} \\
&\quad\rightarrow\quad [\,[\langle A_s, B_0, 1\rangle\langle A_t, B_0, 0\rangle]_{add}]_{m\_add} \\
&[\langle A_s, B_1, 0\rangle\langle A_t, B_1, 0\rangle\langle CM_1(7)\rangle]^+_{m\_add} \\
&\quad\rightarrow\quad [\,[\langle A_s, B_1, 0\rangle\langle A_t, B_1, 0\rangle]_{add}]_{m\_add} \\
&\qquad\vdots \\
&[\langle A_s, B_9, 0\rangle\langle A_t, B_9, 0\rangle\langle CM_9(7)\rangle]^+_{m\_add} \\
&\quad\rightarrow\quad [\,[\langle A_s, B_9, 0\rangle\langle A_t, B_9, 0\rangle]_{add}]_{m\_add}
\end{aligned}
$$

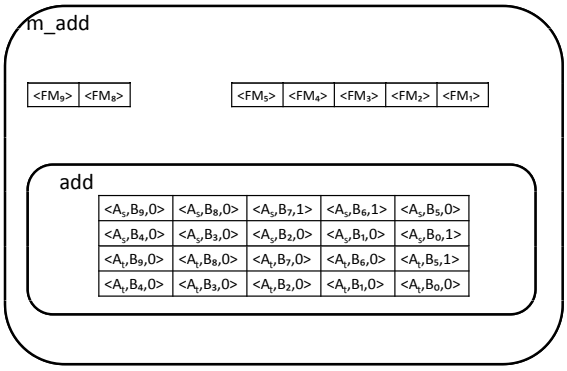


図 20: Step 3 における膜の状態

次に，加算膜が V_s と V_t を入力として得ると，計算結果として V_{answer} を多入力加算膜へ出力する．そのときの状態が図 21 である．

Step 4 多入力加算膜において，加算膜からの出力として V_{answer} を得ると，V_{answer} を外部へ出力する．図 21 において，以下の進化規則を適用されることで，多入力加算を終了する．

$$[\langle A_{answer}, B_0, 1\rangle]_{m\_add} \quad\rightarrow\quad \langle A_{answer}, B_0, 1\rangle[\,]_{m\_add}$$

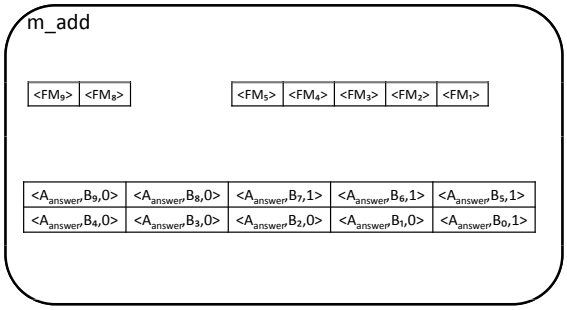


図 21: Step 4 開始時における膜の状態

$$
\begin{aligned}
[\langle A_{answer}, B_1, 0\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_1, 0\rangle[\,]_{m\_add} \\
[\langle A_{answer}, B_2, 0\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_2, 0\rangle[\,]_{m\_add} \\
[\langle A_{answer}, B_3, 0\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_3, 0\rangle[\,]_{m\_add} \\
[\langle A_{answer}, B_4, 0\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_4, 0\rangle[\,]_{m\_add} \\
[\langle A_{answer}, B_5, 1\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_5, 1\rangle[\,]_{m\_add} \\
[\langle A_{answer}, B_6, 1\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_6, 1\rangle[\,]_{m\_add} \\
[\langle A_{answer}, B_7, 1\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_7, 1\rangle[\,]_{m\_add} \\
[\langle A_{answer}, B_8, 0\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_8, 0\rangle[\,]_{m\_add} \\
[\langle A_{answer}, B_9, 0\rangle]_{m\_add} &\rightarrow \langle A_{answer}, B_9, 0\rangle[\,]_{m\_add}
\end{aligned}
$$

5.4 アルゴリズムの計算量

5.2 節において，(1-1) の各ビットに存在する 2 つの 1 を次のビットの 1 とする過程で，$\log_2 m + 1$ ステップ後に，各ビットの 1 の個数は多くとも 2 つになることを示す．

補題 1 *P* システム Π_{m_add} の進化規則 $R_{1.2}$ を $\log_2 m + 1$ 回適用することにより，各ビットの *1* を表すオブジェクトの個数は高々 *2* となる．

(証明)
$R_{1.2}$ に進化規則適用前の各ビットの *1* の個数は多くとも m 個である．ここで，進化規則 $R_{1.2}$ を *1* 回適用後の各ビットの *1* の個数について考える．まず，P システム Π_{m_add} の進化規則 $R_{1.2}$ により，各桁に存在する *2* つの *1* が反応して，次の桁の *1* つの *1* となる．このことより，m 個の *1* が存在する場合，次の桁への繰り上がりは多くとも $\frac{m}{2}$ 個である．また，各桁において，m が奇数の場合は *1* 個，偶数の場合は *0* 個の *1* が残ることになる．よって，*1* ステップ後の各ビットの *1* の個数は，多くとも $\frac{m}{2} + 1$ である．同様にして考えると，各ステップにおいて，k 個の *1* が存在する場合，次のステップでの各ビットの *1* の個数は，$\frac{k}{2} + 1$ となり，n ステップ後の各ビットの *1* の個数を A_n とすると以下の式が成り立つ．

$$A_{n+1} \leq \left(\frac{A_n}{2}\right) + 1$$

この漸化式を解くと[§]，以下のようになる．

$$A_n \leq (m-2)\left(\frac{1}{2}\right)^{n-1} + 2$$

ゆえに，$n = \log_2 m + 1$ ステップ後の各ビットの *1* の個数 $A_{\log_2 m+1}$ は以下のようになる．

$$A_{\log_2 m+1} \leq (m-2)\left(\frac{1}{2}\right)^{\log_2 m} + 2$$

$$= \frac{m-2}{m} + 2$$

ここで，$0 < \frac{m-2}{m} < 1$ より，

$$0 \leq A_{\log_2 m+1} < 3$$

となる．

したがって，A_n は整数なので，$\log_2 m + 1$ ステップ後に各ビットの *1* の個数は多くとも *2* つである． □

以上のことを踏まえ，多入力加算を実行する P システム Π_{m_add} は，$O(m^2)$ 個の進化規則の適用によりアルゴリズムが終了するので，以下の定理が得られる．

定理 2 m 個の $2m$ ビットの 2 進数を入力とし，多入力加算を実行する P システム Π_{m_add} は，$O(m^2)$ 種類のオブジェクト，$O(1)$ 個の膜，及び，$O(m^2)$ 個の進化規則を用いて，$O(\log m)$ ステップで実行可能である． □

6 乗算アルゴリズム

本節は，2 個の m ビットの 2 進数の乗算を実行するアルゴリズムを示す．文献 [1] により提案されている乗算アルゴリズムでは，$O(m \log m)$ ステップで実行可能となっているが，本研究では，$O(\log m)$ ステップで実行可能なアルゴリズムを提案する．

6.1 入出力

入力 アドレス p, q に格納された 2 個の m ビットの 2 進数を表すオブジェクトの集合を入力とする．なお，これらのオブジェクトは，乗算膜に入力として与えられるものとする．

$$\{\langle A_p, B_{m-1}, V_{p,m-1}\rangle, \langle A_p, B_{m-2}, V_{p,m-2}\rangle, \cdots, \langle A_p, B_0, V_{p,0}\rangle\}$$

$$\{\langle A_q, B_{m-1}, V_{q,m-1}\rangle, \langle A_q, B_{m-2}, V_{q,m-2}\rangle, \cdots, \langle A_q, B_0, V_{q,0}\rangle\}$$

出力 アドレス $answer$ に格納された $2m+2$ ビットの 2 進数を表すオブジェクトの集合を出力とする．

$$\{\langle A_{answer}, B_{2m+1}, V_{answer,2m+1}\rangle, \langle A_{answer}, B_{2m}, V_{answer,2m}\rangle,$$
$$\cdots, \langle A_{answer}, B_0, V_{answer,0}\rangle\}$$

[§] $B_n = A_n - 2$ とおくと，$B_{n+1} \leq \frac{B_n}{2}$ であり，$B_n \leq (m-2)(\frac{1}{2})^{n-1}$ となる．したがって，$A_n \leq (m-2)(\frac{1}{2})^{n-1} + 2$ となる．

6.2 アルゴリズムの概要

本研究で提案する乗算アルゴリズムの概要を説明する．なお，本研究で提案する乗算膜は，前節で提案した多入力加算膜を用いて加算を行う．多入力加算膜は加算結果 V_{answer} を乗算膜に出力し，乗算膜は V_{answer} を外部へ出力することで計算を終了する．

まず，2 個の m ビットの 2 進数を，乗算膜に入力として与える．

$$\{\langle A_p, B_{m-1}, V_{p,m-1}\rangle, \langle A_p, B_{m-2}, V_{p,m-2}\rangle, \cdots, \langle A_p, B_0, V_{p,0}\rangle\}$$

$$\{\langle A_q, B_{m-1}, V_{q,m-1}\rangle, \langle A_q, B_{m-2}, V_{q,m-2}\rangle, \cdots, \langle A_q, B_0, V_{q,0}\rangle\}$$

乗算を実行するアルゴリズムは，4 つのステップからなる．Step 1 において，Step 2 において用いるオブジェクトを生成する．次に，Step 2 において，各ビット毎の乗算を行い，Step 3 においては，Step 2 において計算した乗算結果を多入力加算膜へ出力する．多入力加算膜では加算が行われ，加算結果を乗算膜へ出力する．最後に，Step 4 において，多入力加算膜から出力された答えを外部へ出力し，計算を終了する．

以下に乗算を行うアルゴリズムの概要を示す．なお，本アルゴリズムは，図 22 に示す筆算を元にしている．

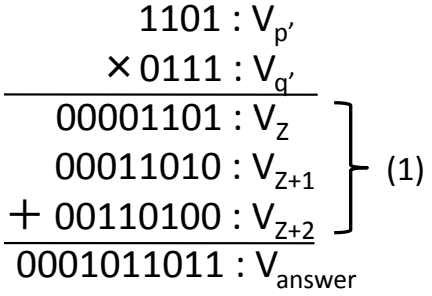


図 22: 筆算による乗算の計算方法

Step 1 以下のサブステップを実行することによって，Step 2 への下準備を行う．

(1-1) 入力 p, q より以下の 4 種類のオブジェクトを生成する．

- $V_{p'}, V_{q'}$: V_p と V_q の値を保持するオブジェクト (図 22 の $V_{p'}, V_{q'}$ に相当する)
- V_Z, V_{Z+1}: 積の結果を入力するオブジェクト (図 22 の (1) に相当する)
- FP: (1-2) において用いるオブジェクト
- CP: カウンタを表すオブジェクト

(1-2) Step 2 において m 個のオブジェクト $V_{p'}, V_{q'}$ と，$V_{Z+2}, V_{Z+3}, \cdots, V_{Z+(m-1)}$ を用いるので，Step 1 で生成したオブジェクトから，これらのオブジェクトを複製する．

(1-3) 膜を + に帯電させることで，Step 1 を終了とする．

Step 2 各 $V_{p',j}$ に対して，$V_{q'}$ との積を計算する．

(2-1) 各 $V_{p',j}$ について，$V_{q',0}, V_{q',1}, \cdots, V_{q',m-1}$ との積を計算し，その積を V_{Z+j} に保存する．

(2-2) 乗算膜を $-$ に帯電することで，Step 2 を終了とする．

Step 3 多入力加算膜に $V_Z, V_{Z+1}, \cdots, V_{Z+(m-1)}$ を入力として与える．加算後は，計算結果として V_{answer} が乗算膜に出力される．

Step 4 多入力加算膜より出力される V_{answer} を外部へ出力することで，計算を終了する．

以下に乗算を実行する P システム Π_{prod} を示す．

$$\Pi_{prod} = (O, \mu, \omega_{prod}, R_{prod})$$

- $O = \{\langle A_{Z+i}, B_j, V_{Z+i,j}\rangle \mid 0 \le i \le m-1, 0 \le j \le 2m-1, V_{Z+i,j} \in \{0,1\}\} \cup \{\langle A_p, B_j, V_{p,j}\rangle, \langle A_q, B_j, V_{q,j}\rangle, \langle A_{p'}, B_j, V_{p,j}\rangle, \langle A_{q'}, B_j, V_{q,j}\rangle \mid 0 \le j \le m-1, V_{p,j}, V_{q,j} \in \{0,1\}\} \cup \{\langle A_{answer}, B_j, V_{answer,j}\rangle \mid 0 \le j \le 2m+1, V_{answer,j} \in \{0,1\}\} \cup \{\langle FP_{i,j}\rangle \mid 1 \le i \le m-1, 0 \le j \le 2m-1\} \cup \{\langle CP(h)\rangle \mid 0 \le h \le \log m + 1\}$
- $\mu = [\,[\,[\,]_{add}\,]_{m\_add}\,]_{prod}$
- $\omega_{prod} = \{\langle A_p, B_j, V_{p,j}\rangle, \langle A_q, B_j, V_{q,j}\rangle \mid 0 \le j \le m-1, V_{p,j}, V_{q,j} \in \{0,1\}\}$
- $R_{prod} = R_1 \cup R_2 \cup R_3 \cup R_4$
  - $R_1 = R_{1.1} \cup R_{1.2} \cup R_{1.3}$
    - $R_{1.1} = \{$
      $\langle A_p, B_0, V_{p,0}\rangle\langle A_q, B_0, V_{q,0}\rangle \to \langle A_{p'}, B_0, V_{p,0}\rangle\langle A_{p'}, B_0, V_{p,0}\rangle \langle A_{q'}, B_0, V_{q,0}\rangle\langle A_{q'}, B_0, V_{q,0}\rangle\langle CP(0)\rangle \langle A_Z, B_0, 0\rangle\langle A_{Z+1}, B_0, 0\rangle\langle FP_{1,0}\rangle \langle A_Z, B_m, 0\rangle\langle A_{Z+1}, B_m, 0\rangle\langle FP_{1,m}\rangle,$
      $\langle A_p, B_j, V_{p,j}\rangle\langle A_q, B_j, V_{q,j}\rangle \to \langle A_{p'}, B_j, V_{p,j}\rangle\langle A_{p'}, B_j, V_{p,j}\rangle \langle A_{q'}, B_j, V_{q,j}\rangle\langle A_{q'}, B_j, V_{q,j}\rangle \langle A_Z, B_j, 0\rangle\langle A_{Z+1}, B_j, 0\rangle\langle FP_{1,j}\rangle \langle A_Z, B_{j+m}, 0\rangle\langle A_{Z+1}, B_{j+m}, 0\rangle\langle FP_{1,j+m}\rangle \mid 1 \le j \le m-1, V_{p,0}, V_{q,0}, V_{p,j}, V_{q,j} \in \{0,1\}\}$
    - $R_{1.2} = R_{1.2.1} \cup R_{1.2.2} \cup R_{1.2.3}$
      - $R_{1.2.1} = \{ \langle A_{p'}, B_j, V_{p,j}\rangle\langle A_{q'}, B_j, V_{q,j}\rangle \to \langle A_{p'}, B_j, V_{p,j}\rangle\langle A_{p'}, B_j, V_{p,j}\rangle \langle A_{q'}, B_j, V_{q,j}\rangle\langle A_{q'}, B_j, V_{q,j}\rangle \mid 0 \le j \le 2m-1, V_{p,j}, V_{q,j} \in \{0,1\}\}$
      - $R_{1.2.2} = \{ \langle A_{Z+i}, B_j, 0\rangle\langle FP_{i,j}\rangle \to \langle A_{Z+2i}, B_j, 0\rangle\langle A_{Z+2i+1}, B_j, 0\rangle \langle FP_{2i,j}\rangle\langle FP_{2i+1,j}\rangle \mid 1 \le i \le \frac{m}{2}, 0 \le j \le 2m-1\}$
      - $R_{1.2.3} = \{ \langle CP(h)\rangle \to \langle CP(h+1)\rangle \mid 0 \le h \le \log m - 2\}$
    - $R_{1.3} = \{ [\langle CP(\log m - 1)\rangle]_{prod} \to [\langle CP(\log m)\rangle]^+_{prod}\}$
  - $R_2 = R_{2.1} \cup R_{2.2}$
    - $R_{2.1} = \{ [\langle A_{p'}, B_i, 1\rangle\langle A_{q'}, B_j, 1\rangle\langle A_{Z+j}, B_{i+j}, 0\rangle]^+_{prod} \to [\langle A_{Z+j}, B_{i+j}, 1\rangle]^+_{prod} \mid 0 \le i \le m-1, 0 \le j \le 2m-1\}$
    - $R_{2.2} = \{ [\langle CP(\log m)\rangle]^+_{prod} \to [\langle CP(\log m + 1)\rangle]^+_{prod}\}$
    - $R_{2.3} = \{ [\langle CP(\log m + 1)\rangle]^+_{prod} \to [\,]^-_{prod}\}$
  - $R_3 = \{ [\langle A_{Z+i}, B_j, V_{Z+i,j}\rangle]^-_{prod} \to [\,[\langle A_{Z+i}, B_j, V_{Z+i,j}\rangle]_{m\_add}]_{prod} \mid 0 \le i \le m-1, 0 \le j \le 2m-1, V_{Z+i,j} \in \{0,1\}\}$
  - $R_4 = \{ [\langle A_{answer}, B_j, V_{answer,j}\rangle]_{prod} \to \langle A_{answer}, B_j, V_{answer,j}\rangle[\,]_{prod} \mid 0 \le j \le 2m+1, V_{answer,j} \in \{0,1\}\}$

ここで，アルゴリズム中における各オブジェクトの役割を説明する．

- $V_{p'}, V_{q'}$: 入力 V_p と V_q の値を保持するオブジェクト
- $V_Z, V_{Z+1}, \cdots, V_{Z+(m-1)}$: 積の結果を入力するオブジェクト
- $FP(Flag\ for\ Production)$: (1-2) において用いるオブジェクト
- $CP(Counter\ for\ Production)$: カウンタを表すオブジェクト

6.3 アルゴリズムの動作

本節では，膜計算において，乗算を実行する具体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，2 進数 101 と 111 の乗算を実行する例を示す．なお，本研究ではビット数は 2 のべき乗としているが，総オブジェクト数が，$m = 2$ のときは少なく，$m = 4$ のときは多いため，いずれも説明には不適当となる．このため，例外として $m = 3$ とし，説明することとする．よって，以下のオブジェクトが乗算膜に入力として与えられるものとする．

$$\langle A_p, B_2, 1\rangle, \langle A_p, B_1, 0\rangle, \langle A_p, B_0, 1\rangle,$$

$$\langle A_q, B_2, 1\rangle, \langle A_q, B_1, 1\rangle, \langle A_q, B_0, 1\rangle$$

Step 1 Step 2 において用いるオブジェクトの生成を行う．

(1-1) この膜計算モデルの初期状態は，図 23 である．

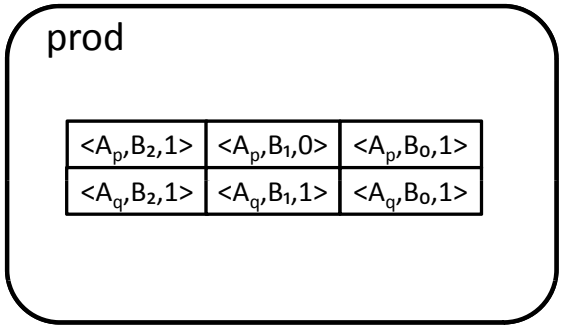


図 23: (1-1) 開始時における膜の状態

入力 V_p と V_q を用いて，V_p と V_q の値を保持する $V_{p'}$ と $V_{q'}$，6 ビット持つ V_Z と V_{Z+1}，更に FP と $CP(0)$ を生成する．なお，FP は V_{Z+1} から V_{Z+2} までの各アドレスに 6 ビットずつ生成する．また，V_Z, V_{Z+1} は値はすべて 0 として生成する．図 23 の状態に対して，以下の進化規則が適用され，図 24 の状態に遷移する．

$$\begin{aligned}
&\langle A_p, B_0, 1\rangle\langle A_q, B_0, 1\rangle \\
&\quad\to\quad \langle A_{p'}, B_0, 1\rangle\langle A_{p'}, B_0, 1\rangle \\
&\qquad\quad \langle A_{q'}, B_0, 1\rangle\langle A_{q'}, B_0, 1\rangle\langle CP(0)\rangle \\
&\qquad\quad \langle A_Z, B_0, 0\rangle\langle A_{Z+1}, B_0, 0\rangle\langle FP_{1,0}\rangle \\
&\qquad\quad \langle A_Z, B_3, 0\rangle\langle A_{Z+1}, B_3, 0\rangle\langle FP_{1,3}\rangle \\
&\langle A_p, B_1, 0\rangle\langle A_q, B_1, 1\rangle \\
&\quad\to\quad \langle A_{p'}, B_1, 0\rangle\langle A_{p'}, B_1, 0\rangle \\
&\qquad\quad \langle A_{q'}, B_1, 1\rangle\langle A_{q'}, B_1, 1\rangle \\
&\qquad\quad \langle A_Z, B_1, 0\rangle\langle A_{Z+1}, B_1, 0\rangle\langle FP_{1,1}\rangle \\
&\qquad\quad \langle A_Z, B_4, 0\rangle\langle A_{Z+1}, B_4, 0\rangle\langle FP_{1,4}\rangle \\
&\langle A_p, B_2, 1\rangle\langle A_q, B_2, 1\rangle \\
&\quad\to\quad \langle A_{p'}, B_2, 1\rangle\langle A_{p'}, B_2, 1\rangle \\
&\qquad\quad \langle A_{q'}, B_2, 1\rangle\langle A_{q'}, B_2, 1\rangle \\
&\qquad\quad \langle A_Z, B_2, 0\rangle\langle A_{Z+1}, B_2, 0\rangle\langle FP_{1,2}\rangle \\
&\qquad\quad \langle A_Z, B_5, 0\rangle\langle A_{Z+1}, B_5, 0\rangle\langle FP_{1,5}\rangle
\end{aligned}$$

(1-2),(1-3) Step 2 の計算段階において，V'_p と V'_q は少なくとも $m = 3$ 個必要とするため，このステップで p' と q' の増加を行う．同時に，V_{Z+1} と FP を用いて，V_{Z+2}, V_{Z+3} を生成する．また，カウンター $CP(0)$ を用いて，膜を $+$ に帯電することで，Step 1 を終了する．図

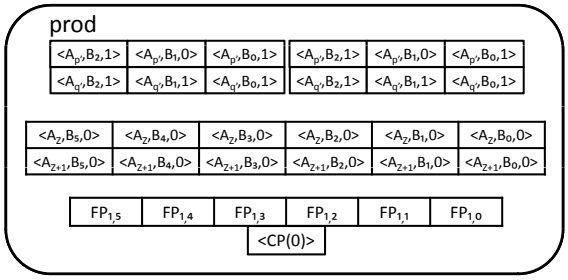


図 24: (1-2) 開始時における膜の状態

24 の状態に対して，以下の進化規則が適用され，図 25 の状態に遷移する．

$$\begin{aligned}
&\langle A_{p'}, B_0, 1\rangle\langle A_{q'}, B_0, 1\rangle \\
&\quad\to\quad \langle A_{p'}, B_0, 1\rangle\langle A_{p'}, B_0, 1\rangle\langle A_{q'}, B_0, 1\rangle\langle A_{q'}, B_0, 1\rangle \\
&\langle A_{p'}, B_1, 0\rangle\langle A_{q'}, B_1, 1\rangle \\
&\quad\to\quad \langle A_{p'}, B_1, 0\rangle\langle A_{p'}, B_1, 0\rangle\langle A_{q'}, B_1, 1\rangle\langle A_{q'}, B_1, 1\rangle \\
&\langle A_{p'}, B_2, 1\rangle\langle A_{q'}, B_2, 1\rangle \\
&\quad\to\quad \langle A_{p'}, B_2, 1\rangle\langle A_{p'}, B_2, 1\rangle\langle A_{q'}, B_2, 1\rangle\langle A_{q'}, B_2, 1\rangle \\
&\qquad \langle A_{Z+1}, B_0, 0\rangle\langle FP_{0,0}\rangle \\
&\qquad\quad\to\quad \langle A_{Z+2}, B_0, 0\rangle\langle A_{Z+3}, B_0, 0\rangle \\
&\qquad\qquad \langle FP_{2,0}\rangle\langle FP_{3,0}\rangle \\
&\qquad \langle A_{Z+1}, B_1, 0\rangle\langle FP_{1,0}\rangle \\
&\qquad\quad\to\quad \langle A_{Z+2}, B_1, 0\rangle\langle A_{Z+3}, B_1, 0\rangle \\
&\qquad\qquad \langle FP_{2,1}\rangle\langle FP_{3,1}\rangle \\
&\qquad\qquad\qquad\vdots \\
&\qquad \langle A_{Z+1}, B_5, 0\rangle\langle FP_{1,5}\rangle \\
&\qquad\quad\to\quad \langle A_{Z+2}, B_5, 0\rangle\langle A_{Z+3}, B_5, 0\rangle \\
&\qquad\qquad \langle FP_{2,5}\rangle\langle FP_{3,5}\rangle \\
&[\langle CP(0)\rangle]_{prod} \quad\to\quad [\langle CP(1)\rangle]^+_{prod}
\end{aligned}$$

Step 2 各 $V_{p',j}$ に対して，$V_{q'}$ との積を計算する．

(2-1) Step 1 において，生成した $V_{p'}$，$V_{q'}$，及び V_Z，V_{Z+1}，V_{Z+2} を用いて，$V_{p',i} \times V_{q',j} = 1$ の部分を計算し，$V_{Z+j,i+j}$ に 1 を入力する．この操作は p' と q' がそれぞれ 3 個以上存在するため，多くとも $3^2 = 9$ 通りある $V_{p',i} \times V_{q',j} = 1$ のすべてのパターンを網羅することになるので，1 ステップあれば完了する．図 25 の状態に対して，以下の進化規則が適用され，図 26 の状態に遷移する．

$$\begin{aligned}
&[\langle A_{p'}, B_0, 1\rangle\langle A_{q'}, B_0, 1\rangle\langle A_Z, B_0, 0\rangle]^+_{prod} \\
&\qquad\to\quad [\langle A_Z, B_0, 1\rangle]^+_{prod} \\
&[\langle A_{p'}, B_0, 1\rangle\langle A_{q'}, B_2, 1\rangle\langle A_Z, B_2, 0\rangle]^+_{prod} \\
&\qquad\to\quad [\langle A_Z, B_2, 1\rangle]^+_{prod}
\end{aligned}$$

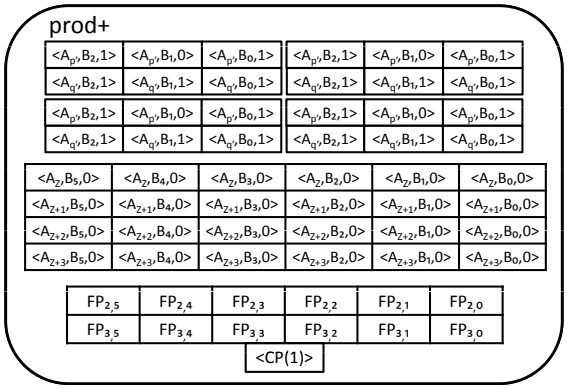


図 25: (1-2) における膜の状態

$$\vdots$$

$$[\langle A_{p'}, B_2, 1\rangle\langle A_{q'}, B_2, 1\rangle\langle A_{Z+2}, B_4, 0\rangle]^+_{prod} \rightarrow [\langle A_{Z+2}, B_4, 1\rangle]^+_{prod}$$

$$[\langle CP(1)\rangle]^+_{prod} \rightarrow [\langle CP(2)\rangle]^+_{prod}$$

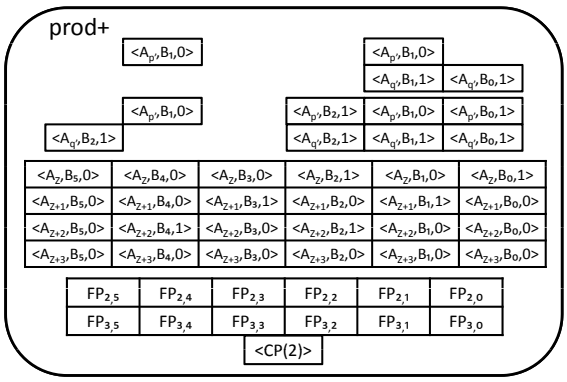


図 26: (2-2) 開始時における膜の状態

(2-2) (2-1) 実行後に得る $CP(2)$ を用いて，乗算膜を $-$ に帯電し，Step 2 を終了する．図 26 の状態に対して，以下の進化規則が適用され，図 27 の状態に遷移する．

$$[\langle CP(2)\rangle]^+_{prod} \rightarrow [\,]^-_{prod}$$

Step 3 多入力加算膜に V_Z, V_{Z+1}, V_{Z+2} を入力として与えることで，加算を実行する．このとき，乗算膜の帯電も解く．図 27 の状態に対して，以下の進化規則が適用される．

$$\begin{aligned}
[\langle A_Z, B_0, 1\rangle]^-_{prod} &\rightarrow [\,[\langle A_Z, B_0, 1\rangle]_{m_add}]_{prod} \\
[\langle A_Z, B_1, 0\rangle]^-_{prod} &\rightarrow [\,[\langle A_Z, B_1, 0\rangle]_{m_add}]_{prod} \\
&\vdots \\
[\langle A_Z, B_5, 0\rangle]^-_{prod} &\rightarrow [\,[\langle A_Z, B_5, 0]_{m_add}]_{prod}
\end{aligned}$$

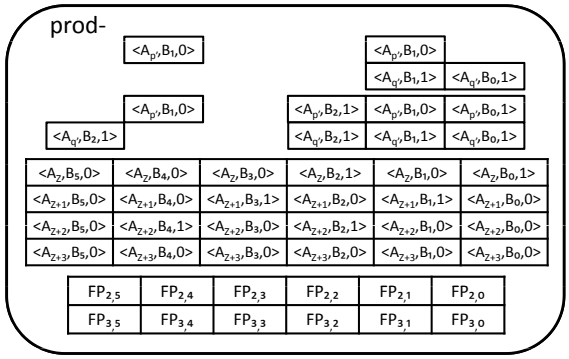


図 27: Step 2 終了時における膜の状態

$$\begin{aligned}
[\langle A_{Z+1}, B_0, 0\rangle]^-_{prod} &\rightarrow [\,[\langle A_{Z+1}, B_0, 0\rangle]_{m_add}]_{prod} \\
[\langle A_{Z+1}, B_1, 1\rangle]^-_{prod} &\rightarrow [\,[\langle A_{Z+1}, B_1, 1\rangle]_{m_add}]_{prod} \\
&\vdots \\
[\langle A_{Z+1}, B_5, 0\rangle]^-_{prod} &\rightarrow [\,[\langle A_{Z+1}, B_5, 0\rangle]_{m_add}]_{prod} \\
\\
[\langle A_{Z+2}, B_0, 0\rangle]^-_{prod} &\rightarrow [\,[\langle A_{Z+2}, B_0, 0\rangle]_{m_add}]_{prod} \\
[\langle A_{Z+2}, B_1, 0\rangle]^-_{prod} &\rightarrow [\,[\langle A_{Z+2}, B_1, 0\rangle]_{m_add}]_{prod} \\
&\vdots \\
[\langle A_{Z+2}, B_5, 0\rangle]^-_{prod} &\rightarrow [\,[\langle A_{Z+2}, B_5, 0\rangle]_{m_add}]_{prod}
\end{aligned}$$

Step 4 多入力加算膜において，V_Z, V_{Z+1}, V_{Z+2} の加算が実行され，乗算膜に V_{answer} が出力される．そのときの状態が図 28 である．乗算膜において，V_{answer} を得ると，V_{answer} を外部へ出力することで，計算終了とする．図 28 の状態に対して，以下の進化規則が適用されることで，乗算を終了とする．

$$\begin{aligned}
[\langle A_{answer}, B_0, 1\rangle]_{prod} &\rightarrow \langle A_{answer}, B_0, 1\rangle[\,]_{prod} \\
[\langle A_{answer}, B_1, 1\rangle]_{prod} &\rightarrow \langle A_{answer}, B_1, 1\rangle[\,]_{prod} \\
[\langle A_{answer}, B_2, 0\rangle]_{prod} &\rightarrow \langle A_{answer}, B_2, 0\rangle[\,]_{prod} \\
[\langle A_{answer}, B_3, 0\rangle]_{prod} &\rightarrow \langle A_{answer}, B_3, 0\rangle[\,]_{prod} \\
[\langle A_{answer}, B_4, 0\rangle]_{prod} &\rightarrow \langle A_{answer}, B_4, 0\rangle[\,]_{prod} \\
[\langle A_{answer}, B_5, 1\rangle]_{prod} &\rightarrow \langle A_{answer}, B_5, 1\rangle[\,]_{prod} \\
[\langle A_{answer}, B_6, 0\rangle]_{prod} &\rightarrow \langle A_{answer}, B_6, 0\rangle[\,]_{prod} \\
[\langle A_{answer}, B_7, 0\rangle]_{prod} &\rightarrow \langle A_{answer}, B_7, 0\rangle[\,]_{prod}
\end{aligned}$$

6.4 アルゴリズムの計算量

乗算を実行する P システム Pi_{prod} は，$O(m^2)$ 個の進化規則の適用によりアルゴリズムが終了するので，以下の定理が得られる．

定理 3 2 個の m ビットの 2 進数を入力とし，乗算を実行する P システム Π_{prod} は，$O(m^2)$ 種類のオブジェクト，$O(1)$ 個の膜，及び，$O(m^2)$ 個の進化規則を用いて，$O(\log m)$ ステップで実行可能である． □

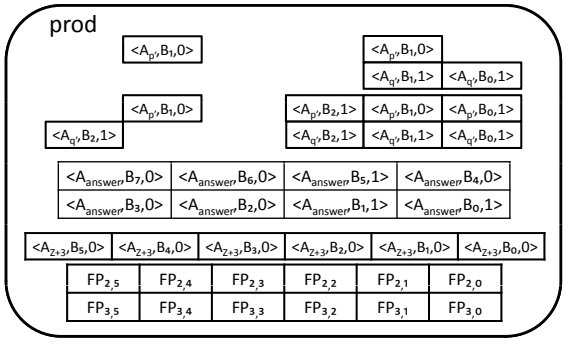


図 28: Step 4 開始時における膜の状態

7 因数分解アルゴリズム

本節では，m ビットの 2 進数の因数分解を実行するアルゴリズムを示す．文献 [1] により提案されている因数分解アルゴリズムでは，$O(m \log m)$ ステップで実行可能となっているが，本研究では前節で提案した乗算アルゴリズムを用いて，$O(m)$ で実行可能としている．まず，P システムの入出力を定義し，次に，アルゴリズムの概要を説明し，最後にアルゴリズム動作，及び，アルゴリズムの計算量を示す．

7.1 入出力

入力 アドレス c に格納され，素数 p, q の積からなる m ビットの 2 進数を表すオブジェクトの集合を入力とする．これらのオブジェクトは，スキン膜に入力として与えられる．

$$\{\langle A_c, B_{m-1}, V_{c,m-1}\rangle, \langle A_c, B_{m-2}, V_{c,m-2}\rangle, \cdots, \langle A_c, B_0, V_{c,0}\rangle\}$$

出力 アドレス p, q に格納された $m-1$ ビットの 2 進数を表すオブジェクトの集合を出力とする．

$$\{\langle A_p, B_{m-2}, V_{p,m-2}\rangle, \langle A_p, B_{m-3}, V_{p,m-3}\rangle, \cdots, \langle A_p, B_0, V_{p,0}\rangle\}$$

$$\{\langle A_q, B_{m-1}, V_{q,m-2}\rangle, \langle A_q, B_{m-2}, V_{q,m-3}\rangle, \cdots, \langle A_q, B_0, V_{q,0}\rangle\}$$

7.2 アルゴリズムの概要

本研究で示す因数分解アルゴリズムの概要を説明する．まず，m ビットの 2 進数 c を，スキン膜に入力として与える．

$$\{\langle A_c, B_{m-1}, V_{c,m-1}\rangle, \langle A_c, B_{m-2}, V_{c,m-2}\rangle, \cdots, \langle A_c, B_0, V_{c,0}\rangle\}$$

因数分解を実行するアルゴリズムは，4 つのステップからなる．Step 1 において，膜の分割を繰り返し，因数として可能性のあるすべての整数の組み合わせを表すオブジェクトを含む膜を生成する．次に，Step 2 において，各膜に存在する因数候補である V_p と V_q の乗算を実行する．なお，この乗算には，前章で提案した乗算アルゴリズムを用いる．更に，Step 3 において，計算結果 $V_p \times V_q = V_{answer}$ と入力 V_c が等しいか判別する．そして，等しければ，Step 4 において因数候補解 V_p，V_q を外部へと出力し，因数分解を終了する．

以下に因数分解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って，すべての因数候補を作成する．

(1-1) 入力 V_c より，膜 2 を生成する．このとき，後の照合段階において用いる $2m$ ビットの V_c を生成する．なお，$m \leq i \leq 2m-1$ において，$V_{c,i} = 0$ とする．また，$2m$ ビット分生成するのは，計算結果である V_{answer} が $2m$ ビット分存在するためである．

(1-2) 膜 2 の分割を行うことにより，因数として可能性のあるすべての整数の組み合わせを表すオブジェクトを生成する．

(1-3) すべての膜 2 を $+$ に帯電させる．その操作により，Step 1 を終了とする．

Step 2 すべての膜 2 において，因数候補 p, q を乗算膜へ入力として与える．乗算膜において，$V_p \times V_q = V_{answer}$ が計算され，計算が終了すると，V_{answer} を膜 2 に出力する．

Step 3 各膜 2 において以下の操作を行う．

Step 2 において得る V_{answer} と入力 V_c を照合する．

(3-1) V_{answer} と V_c の比較を行い，$V_{answer} \neq V_c$ のとき，$F3$ を生成する．

(3-2) 膜 1 に結果出力を行う．$F3$ が存在するとき，膜を $-$ に帯電することで，適当な解が存在しないことを示す．$F3$ が存在せず，膜帯電が行われなければ，膜 2 を分解することで，因数候補解 p, q を出力する．

Step 4 (3-2) において出力された因数候補解 p, q を，スキン膜から外部へ出力することで，因数分解を終了する．

以下に因数分解を実行する P システム Π_{fact} を示す．

$$\Pi_{fact} = (O, \mu, \omega_1, \omega_2, R_1, R_2)$$

ここで，Π_{fact} を構成するそれぞれの集合は以下のようになる．

- $O = \{\langle A_c, B_j, V_{c,j}\rangle \mid 0 \leq j \leq 2m-1, V_{c,j} \in \{0,1\}\}$
  $\cup \{\langle A_p, B_j, V_{p,j}\rangle, \langle A_q, B_j, V_{q,j}\rangle$
  $\mid 0 \leq j \leq m-2, V_{p,j}, V_{q,j} \in \{0,1\}\}$
  $\cup \{\langle F1_j\rangle, \langle F2_j\rangle, \langle F3\rangle, \langle F4\rangle \mid 0 \leq j \leq m-1\}$
  $\cup \{\langle C(0)\rangle, \langle C(1)\rangle\}$

- $\mu = [\,[\,[\,[\,[\,]_{add}\,]_{m\_add}\,]_{prod}\,]_2\,]_1$
- $\omega_1 = \{\langle A_c, B_j, V_{c,j}\rangle \mid 0 \le j \le m-1, V_{c,j} \in \{0,1\}\}$
- $\omega_2 = \{\langle A_c, B_j, V_{c,j}\rangle, \langle F1_j\rangle \mid 0 \le j \le 2m-1, V_{c,j} \in \{0,1\}\}$
- $R_1 = R_{1.1} \cup R_{1.2}$
  - $R_{1.1} = R_{1.1.1} \cup R_{1.1.2}$
    - $R_{1.1.1} = \{ [\langle A_c, B_0, V_{c,0}\rangle]_1 \to [\,[\langle A_c, B_0, V_{c,0}\rangle\langle A_c, B_m, 0\rangle\langle F1_{m-1}\rangle]_2\,]_1 \mid V_{c,0} \in \{0,1\}\}$
    - $R_{1.1.2} = \{ [\langle A_c, B_j, V_{c,j}\rangle]_1 \to [\,[\langle A_c, B_j, V_{c,j}\rangle\langle A_c, B_{j+m}, 0\rangle]_2]_1 \mid 1 \le j \le m-1, V_{c,j} \in \{0,1\}\}$
  - $R_{1.2} = \{ [\langle A_p, B_j, V_{p,j}\rangle\langle A_q, B_j, V_{q,j}\rangle]_1 \to \langle A_p, B_j, V_{p,j}\rangle\langle A_q, B_j, V_{q,j}\rangle[\,]_1 \mid 0 \le j \le m-2, V_{p,j}, V_{q,j} \in \{0,1\}\}$
- $R_2 = R_{2.1} \cup R_{2.2} \cup R_{2.3}$
  - $R_{2.1} = R_{2.1.1} \cup R_{2.1.2} \cup R_{2.1.3}$
    - $R_{2.1.1} = \{ [\langle F1_j\rangle]_2 \to [\langle A_p, B_{j-1}, 0\rangle\langle A_q, B_{j-1}, 0\rangle\langle F1_{j-1}\rangle]_2 [\langle A_p, B_{j-1}, 1\rangle\langle A_q, B_{j-1}, 0\rangle\langle F2_{j-1}\rangle]_2 [\langle A_p, B_{j-1}, 1\rangle\langle A_q, B_{j-1}, 1\rangle\langle F1_{j-1}\rangle]_2 \mid 1 \le j \le m-1\}$
    - $R_{2.1.2} = \{ [\langle F2_j\rangle]_2 \to [\langle A_p, B_{j-1}, 0\rangle\langle A_q, B_{j-1}, 0\rangle\langle F2_{j-1}\rangle]_2 [\langle A_p, B_{j-1}, 0\rangle\langle A_q, B_{j-1}, 1\rangle\langle F2_{j-1}\rangle]_2 [\langle A_p, B_{j-1}, 1\rangle\langle A_q, B_{j-1}, 0\rangle\langle F2_{j-1}\rangle]_2 [\langle A_p, B_{j-1}, 1\rangle\langle A_q, B_{j-1}, 1\rangle\langle F2_{j-1}\rangle]_2 \mid 1 \le j \le m-1\}$
    - $R_{2.1.3} = \{[\langle F1_0\rangle]_2 \to [\,]_2^+, [\langle F2_0\rangle]_2 \to [\,]_2^+\}$
  - $R_{2.2} = \{ [\langle A_p, B_j, V_{p,j}\rangle\langle A_q, B_j, V_{q,j}\rangle]_2^+ \to [\,[\langle A_p, B_j, V_{p,j}\rangle\langle A_q, B_j, V_{q,j}\rangle]_{prod} \langle A_p, B_j, V_{p,j}\rangle\langle A_q, B_j, V_{q,j}\rangle]_2 \mid 0 \le j \le m-2, V_{p,j}, V_{q,j} \in \{0,1\}\}$
  - $R_{2.3} = R_{2.3.1} \cup R_{2.3.2} \cup R_{2.3.3}$
    - $R_{2.3.1} = \{ \langle A_c, B_j, 0\rangle\langle A_{answer}, B_j, 0\rangle \to \langle C(0)\rangle, \langle A_c, B_j, 0\rangle\langle A_{answer}, B_j, 1\rangle \to \langle F3\rangle, \langle A_c, B_j, 1\rangle\langle A_{answer}, B_j, 0\rangle \to \langle F3\rangle, \langle A_c, B_j, 1\rangle\langle A_{answer}, B_j, 1\rangle \to \langle C(0)\rangle \mid 0 \le j \le 2m-1\}$
    - $R_{2.3.2} = R_{2.3.2.1} \cup R_{2.3.2.2}$
      - $R_{2.3.2.1} = \{[\langle F3\rangle]_2 \to [\,]_2^-\}$
      - $R_{2.3.2.2} = \{ \langle C(0)\rangle \to \langle C(1)\rangle \mid 0 \le j \le 2m-1\}$
    - $R_{2.3.3} = \{ [[\langle C(1)\rangle]_2]_1 \to [\,]_1 \mid 0 \le j \le 2m-1\}$

ここで，アルゴリズム中における各オブジェクトの役割を説明する．

- V_p, V_q: V_c の因数候補を表すオブジェクト
- $F1, F2$: 因数候補 V_p, V_q の生成に用いるオブジェクト
- $F3$: $V_{answer} \neq V_c$ であることを表すオブジェクト
- $C(Counter)$: カウンタを表すオブジェクト

7.3 アルゴリズムの動作

本節では，膜計算において，因数分解を実行する具体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，2 進数 110 の因数分解を実行する例を示す．なお，本研究ではビット数を表す m は 2 のべき乗としているが，$m = 2$ のとき膜 2 は 3 個，$m = 4$ のとき膜 2 は 64 個と，いずれも説明には不適当である．このため，例外として，$m = 3$ とし，説明することとする．よって，以下のオブジェクトが膜 1 に入力として与えられるものとする．

$$\langle A_c, B_2, 1\rangle, \langle A_c, B_1, 1\rangle, \langle A_c, B_0, 0\rangle$$

Step 1 以下のサブステップを実行することによって，膜 1 に因数候補を持つ膜 2 を生成する．

(1-1) この膜計算モデルの初期状態は，図 29 である．

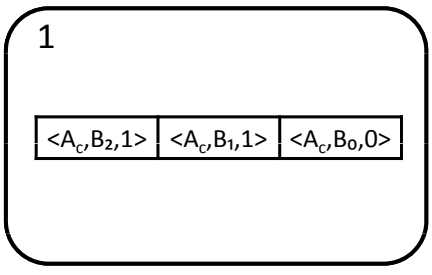


図 29: (1-1) 開始時における膜の状態

与えられた入力 A_c より，膜 2 を生成する．このとき，後の照合段階において用いる，$2 \times 3 = 6$ ビットの A_c を生成する．なお，3 ビット目から 5 ビット目までの値は，すべて 0 とする．また，(1-2) において用いる $F1$ を生成する．図 29 の状態に対して，以下の進化規則が適用され，図 30 の状態に遷移する．

$$[\langle A_c, B_0, 0\rangle]_1 \to [\,[\langle A_c, B_0, 0\rangle\langle A_c, B_3, 0\rangle\langle F1_2\rangle]_2\,]_1$$
$$[\langle A_c, B_1, 1\rangle]_1 \to [\,[\langle A_c, B_1, 1\rangle\langle A_c, B_4, 0\rangle]_2]_1$$
$$[\langle A_c, B_2, 1\rangle]_1 \to [\,[\langle A_c, B_2, 1\rangle\langle A_c, B_5, 0\rangle]_2]_1$$

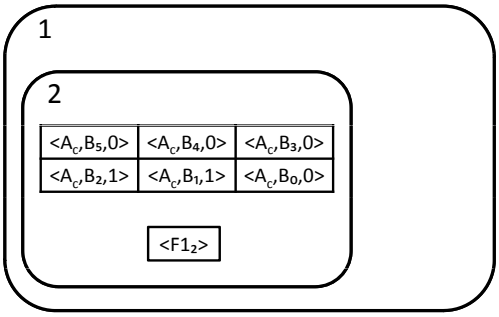


図 30: (1-2) 開始時における膜の状態

(1-2) 膜 2 において，膜分割を行い，すべての因数候補 p, q を生成する．このとき，p が q よりも常に大きくなり，かつすべての候補を網羅的に生成する．この操作は，常に $V_q \leq V_p$ となるように V_p, V_q を生成する $F1$ と，$V_p \leq V_q$ のパターンも含めて V_p, V_q を生成する $F2$ を用いることで可能となる．まず，$F1_2$ を用いて，$V_{p,1}$ と $V_{q,1}$ を生成する．図 30 の状態に対して，以下の進化規則が適用され，図 31 の状態に遷移する．

$$
\begin{aligned}
[\langle F1_2\rangle]_2 \quad \rightarrow \quad & [\langle A_p, B_1, 0\rangle\langle A_q, B_1, 0\rangle\langle F1_1\rangle]_2 \\
& [\langle A_p, B_1, 1\rangle\langle A_q, B_1, 0\rangle\langle F2_1\rangle]_2 \\
& [\langle A_p, B_1, 1\rangle\langle A_q, B_1, 1\rangle\langle F1_1\rangle]_2
\end{aligned}
$$

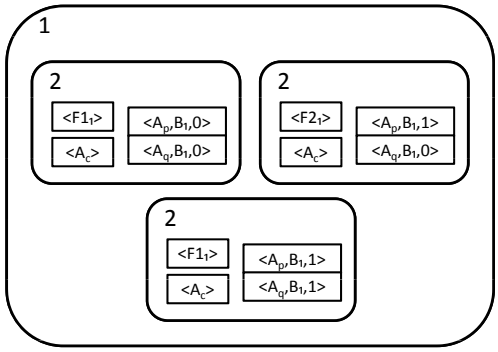


図 31: (1-2) における膜の状態

更に，$V_{p,1} = V_{q,1}$ が存在する膜には $F1_1$ を用いる．しかし，$V_{p,1} = 1, V_{q,1} = 0$ が存在する膜には，V_p はこれ以降 V_q より小さくなることはないので，$F2$ を用いる．こうすることで，p が q よりも常に大きくなり，かつすべての候補を網羅的に生成することができる．図 31 の状態に対して，以下の進化規則が適用され，図 32 の状態に遷移する．

$$
\begin{aligned}
[\langle F1_1\rangle]_2 \quad \rightarrow \quad & [\langle A_p, B_0, 0\rangle\langle A_q, B_0, 0\rangle\langle F1_0\rangle]_2 \\
& [\langle A_p, B_0, 1\rangle\langle A_q, B_0, 0\rangle\langle F2_0\rangle]_2 \\
& [\langle A_p, B_0, 1\rangle\langle A_q, B_0, 1\rangle\langle F1_0\rangle]_2
\end{aligned}
$$

$$
\begin{aligned}
[\langle F1_1\rangle]_2 \quad \rightarrow \quad & [\langle A_p, B_0, 0\rangle\langle A_q, B_0, 0\rangle\langle F1_0\rangle]_2 \\
& [\langle A_p, B_0, 1\rangle\langle A_q, B_0, 0\rangle\langle F2_0\rangle]_2 \\
& [\langle A_p, B_0, 1\rangle\langle A_q, B_0, 1\rangle\langle F1_0\rangle]_2 \\
[\langle F2_1\rangle]_2 \quad \rightarrow \quad & [\langle A_p, B_0, 0\rangle\langle A_q, B_0, 0\rangle\langle F2_0\rangle]_2 \\
& [\langle A_p, B_0, 0\rangle\langle A_q, B_0, 1\rangle\langle F2_0\rangle]_2 \\
& [\langle A_p, B_0, 1\rangle\langle A_q, B_0, 0\rangle\langle F2_0\rangle]_2 \\
& [\langle A_p, B_0, 1\rangle\langle A_q, B_0, 1\rangle\langle F2_0\rangle]_2
\end{aligned}
$$

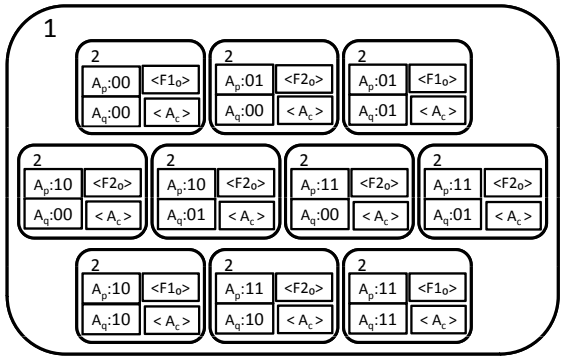


図 32: (1-3) 開始時における膜の状態

(1-3) (1-2) 実行後に得る $F1_0$，$F2_0$ を用いて，すべての膜 2 を $+$ に帯電させる．この操作により，Step 1 を終了とする．図 32 の状態に対して，以下の進化規則が適用され，図 33 の状態に遷移する．

$$
\begin{aligned}
[\langle F1_0\rangle]_2 \quad \rightarrow \quad & [\,]_2^+ \\
[\langle F2_0\rangle]_2 \quad \rightarrow \quad & [\,]_2^+
\end{aligned}
$$

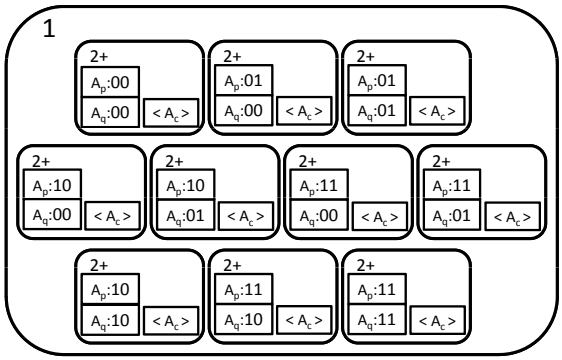


図 33: Step 2 開始時における膜の状態

Step 2 (1-2) において生成した因数候補 p, q を，乗算膜へ入力として与える．なお，後の出力段階において，因数候補 p, q を出力するために，p と q のコピーを膜 2 に生成する．同時に，膜 2 の帯電を解く．図 33 の状態に対して，以下の進化規則がすべての膜 2 に

おいて適用される．

$$
\begin{aligned}
&[\langle A_p, B_2, V_{p,2}\rangle\langle A_q, B_2, V_{q,2}\rangle]_2^+ \\
&\quad\rightarrow\quad [[\langle A_p, B_2, V_{p,2}\rangle\langle A_q, B_2, V_{q,2}\rangle]_{prod} \\
&\qquad\quad \langle A_p, B_2, V_{p,2}\rangle\langle A_q, B_2, V_{q,2}\rangle]_2 \\
&[\langle A_p, B_1, V_{p,1}\rangle\langle A_q, B_1, V_{q,1}\rangle]_2^+ \\
&\quad\rightarrow\quad [[\langle A_p, B_1, V_{p,1}\rangle\langle A_q, B_1, V_{q,1}\rangle]_{prod} \\
&\qquad\quad \langle A_p, B_1, V_{p,1}\rangle\langle A_q, B_1, V_{q,1}\rangle]_2 \\
&[\langle A_p, B_0, V_{p,0}\rangle\langle A_q, B_0, V_{q,0}\rangle]_2^+ \\
&\quad\rightarrow\quad [[\langle A_p, B_0, V_{p,0}\rangle\langle A_q, B_0, V_{q,0}\rangle]_{prod} \\
&\qquad\quad \langle A_p, B_0, V_{p,0}\rangle\langle A_q, B_0, V_{q,0}\rangle]_2
\end{aligned}
$$

各乗算膜において，入力 p, q を得ると，$p \times q$ が計算される．計算結果として V_{answer} を膜 2 に出力されると，図 34 の状態に遷移する．

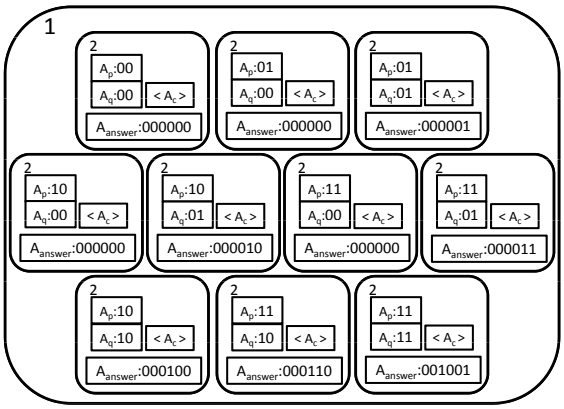


図 34: Step 3 開始時における膜の状態

Step 3 Step 2 実行後に得る V_{answer} が，V_c と等しいか照合する．ここでは，$p = 11$,$q = 10$ の因数候補が存在する膜 2 について詳しく見ていくこととする．このとき，$V_{answer} = 000110$ となり，膜の状態は図 35 となる．

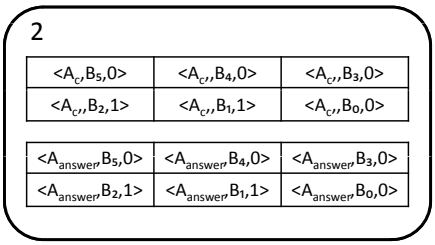


図 35: Step 3 開始時における $p = 11, q = 10$ の候補を持つ膜 2 の状態

(3-1) V_{answer} と A_c との比較を行い，等しいときは，カウンタ $C(0)$ を生成し，異なる時は $F3$ を生成する．この例では，$F3$ は生成されず，$C(0)$ のみ生成される．図 35 の状態に対して，以下の進化規則が適用され，図 36 の状態に遷移する．

$$
\begin{aligned}
\langle A_c, B_5, 0\rangle\langle A_{answer}, B_5, 0\rangle &\rightarrow \langle C(0)\rangle \\
\langle A_c, B_4, 0\rangle\langle A_{answer}, B_4, 0\rangle &\rightarrow \langle C(0)\rangle \\
\langle A_c, B_3, 0\rangle\langle A_{answer}, B_3, 0\rangle &\rightarrow \langle C(0)\rangle \\
\langle A_c, B_2, 1\rangle\langle A_{answer}, B_2, 1\rangle &\rightarrow \langle C(0)\rangle \\
\langle A_c, B_1, 1\rangle\langle A_{answer}, B_1, 1\rangle &\rightarrow \langle C(0)\rangle \\
\langle A_c, B_0, 0\rangle\langle A_{answer}, B_0, 0\rangle &\rightarrow \langle C(0)\rangle
\end{aligned}
$$

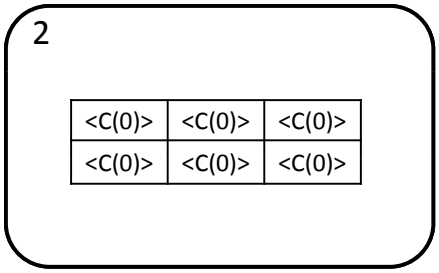


図 36: (3-2) 開始時における膜の状態

(3-2) 膜 1 に結果出力を行う．異なることを示す $F3$ が存在するときは，膜の帯電を行うが，ここでは存在しないので，カウンタのみ進化する．図 36 の状態に対して，以下の進化規則が適用され，図 37 の状態に遷移する．

$$\langle C(0)\rangle \quad\rightarrow\quad \langle C(1)\rangle$$

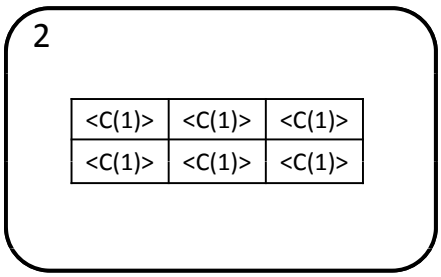


図 37: (3-2) における膜の状態

更に，$C(1)$ を用いて，膜 2 を分解することで，因数候補解 p, q の膜 1 への出力を行う．図 37 の状態に対して，以下の進化規則が適用される．

$$[[\langle C(1)\rangle]_2]_1 \quad\rightarrow\quad [\,]_1$$

同様の操作が膜 2 すべてで行われ，図 38 の状態に遷移する．

Step 4 Step 3 において，膜 1 に出力された V_p, V_q を環境へ出力する．図 38 の状態に対して，以下の進化規則が適用されることで，因数分解を終了する．

$$[\langle A_p, B_1, 1\rangle\langle A_q, B_1, 1\rangle]_1 \rightarrow \langle A_p, B_1, 1\rangle\langle A_q, B_1, 1\rangle[\,]_1$$

$$[\langle A_p, B_0, 1\rangle\langle A_q, B_0, 0\rangle]_1 \rightarrow \langle A_p, B_0, 1\rangle\langle A_q, B_0, 0\rangle[\,]_1$$

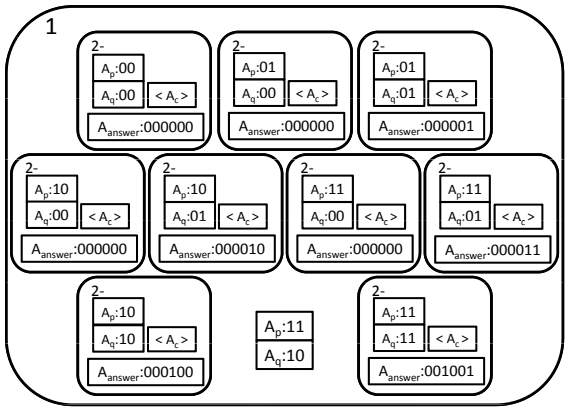


図 38: Step 4 開始時における膜の状態

7.4 アルゴリズムの計算量

因数分解を実行する P システム Π_{fact} は，膜の分割を行う Step 1 の実行に，$O(m)$ ステップを必要とする．また，$O(m^2)$ 個の進化規則の適用によりアルゴリズムが終了するので，以下の定理が得られる．

定理 4 2 つの素数の積を表す m ビットの 2 進数を入力とし，因数分解を実行する P システム Π_{fact} は，$O(m^2)$ 種類のオブジェクト，$O(4^m)$ 個の膜，及び，$O(m^2)$ 個の進化規則を用いて，$O(m)$ ステップで実行可能である．□

8 まとめ

本研究では，膜計算を用いて加算，多入力加算，乗算，及び因数分解を実行するアルゴリズムを示した．提案アルゴリズムにより，加算を，2 個の m ビットの 2 進数を表すオブジェクトを用いて，$O(m^2)$ 種類のオブジェクト，$O(1)$ 個の膜，$O(m^2)$ 個の進化規則を用いて $O(\log m)$ ステップで実行できることを示した．また，多入力加算を，m 個の $2m$ ビットの 2 進数を表すオブジェクトを用いて，$O(m^2)$ 種類のオブジェクト，$O(1)$ 個の膜，$O(m^2)$ 個の進化規則を用いて $O(\log m)$ ステップで実行できることを示した．更に，乗算を，2 個の m ビットの 2 進数を表すオブジェクトを用いて，$O(m^2)$ 種類のオブジェクト，$O(1)$ 個の膜，$O(m^2)$ 個の進化規則を用いて $O(\log m)$ ステップで実行できることを示した．最後に，因数分解については，提案アルゴリズムにより，2 個の素数の積からなる m ビットの 2 進数を表すオブジェクトを用いて，$O(m^2)$ 種類のオブジェクト，$O(1)$ 個の膜，及び，$O(m^2)$ 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，それぞれのアルゴリズ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] C. Zandron A. Leporati and G. Mauri. Solving the factorization problem with p systems. *Journal of-Progeress in Natural Science*, 2007.

[2] A. Fujiwara and T. Tateishi. Computation with aconstant number of steps in membrane computing. *Proceedings of 11th Workshop on Advances inParallel and Distributed Computational Models*, 2009.

[3] 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.

[4] 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.

[5] G. Păum. Introduction to membrane computing. *Applications of Membrane Computing*, 2007.

[6] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, Vol. 61, No. 1, pp. 108–143, 2000.

[7] 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.

[8] 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.

グラフの局所情報を利用したランダムウオークの mixing time について

野中 良哲 * 小野 廣隆 † 山下 雅史 †
* 九州大学大学院システム情報科学府 † 九州大学大学院システム情報科学研究院

概 要

グラフ $G = (V, E)$ 上のランダムウォークとは，グラフ上の頂点をトークンがランダムに遷移するモデルである．すべての隣接頂点へ等確率で遷移するランダムウォークは標準ランダムウォークと呼ばれる．一方，グラフの局所情報を利用することでランダムウォークの hitting time および cover time を高速化する試みとして，隣接頂点の次数を用いたランダムウォークが池田らによって提案されている [4]．また，ランダムウォークの代表的な応用の一つであるマルコフ連鎖モンテカルロにおいてもっともよく用いられる週報の一つであるメトロポリスヘイスティングスアルゴリズム [3] に基づくランダムウォークの一つがメトロポリスウォークである．本稿では，これら 3 つのランダムウォークについて，定常分布へ収束する速さの指標である mixing time の解析を行なう．

1 準備

1.1 グラフ上のランダムウォーク

グラフ $G = (V, E)$ について，$|V| = n, |E| = m$ とする．また，頂点 u に隣接する頂点の集合を $N(u)$ で表し，そのサイズ $|N(u)|$ を u の次数として d_u と書くことにする．

グラフ G 上のランダムウォークとはグラフ上の頂点をトークンが遷移確率行列 $P = (p_{uv})_{u,v \in V}$ にしたがってランダムに遷移していくモデルである．すべての隣接頂点に対して等確率で遷移するランダムウォークを標準ランダムウォークと呼ぶ．

定義 **1.1** (標準ランダムウォーク)

$$p_{uv} = \begin{cases} \frac{1}{d_u} & v \in N(u), \\ 0 & otherwise. \end{cases}$$

標準ランダムウォークでは hitting time および cover time がそれぞれ $O(nm) = O(n^3)$ であることが知られている [1, 2]．一方，グラフの局所情報を利用して hitting time，cover time の観点から高速なランダムウォークを実現しようとする試みとして，池田らは隣接頂点の次数情報を用いたランダムウォークを提案した．

定義 **1.2** (β-ランダムウォーク)

$$p_{uv} = \begin{cases} \frac{d_v^\beta}{\sum_{w \in N(u)} d_w^\beta} & v \in N(u), \\ 0 & otherwise. \end{cases}$$

このランダムウォークは $\beta = \frac{1}{2}$ のとき，hitting time が $O(n^2)$，cover time は $O(n^2 \log n)$ へ改善されることが知られている．[4]

メトロポリス-ヘイスティングスアルゴリズム (MH アルゴリズム)[3] はマルコフ連鎖モンテカルロの代表的手法の一つである．MH アルゴリズムでは，適当な遷移確率行列を基に任意の定常分布を実現する新しいランダムウォークを実現する．標準ランダムウォークに MH アルゴリズムを適用して得られるランダムウォークがメトロポリスウォークである．定常分布 $\pi = (\pi_u)_{u \in V}$ を実現するメトロポリスウォークの遷移確率行列は次にように定義される．

定義 **1.3** (メトロポリスウォーク)

$$p_{uv} = \begin{cases} \frac{1}{d_u} \min \quad \frac{\pi_v d_u}{\pi_u d_v}, 1 \Big\} & v \in N(u) \\ 1 \quad \sum_{x \in N(u)} p_{ux} & u = v \\ 0 & otherwise \end{cases}$$

メトロポリスウォークに関しては hitting time が $O(fn^2)$, cover time が $O(fn^2 \log n)$ でることが著者の過去の研究で分かっている．ここで，$f = \max_{u,v} \pi_u/\pi_v$ である．

1.2 Mixing time

mixing time(混合時間) とは，定常分布への収束の速さを表す指標である．遷移確率行列 P に従うランダムウォークが t ステップ後に各頂点を訪問する確率と定常分布との誤差は total variation distance と呼ばれる．

定義 **1.4** (total variation distance)
V 上の確率行列 P と確率分布 π との t step の時点における total variation distance $\|P^t, \pi\|_{tv}$ は．

$$\|P^t, \pi\|_{tv} = \max_{u \in V} \frac{1}{2} \sum_{v \in V} |P^t_{uv} \quad \pi_v|$$

で定義される．

遷移確率行列 P にしたがうランダムウォークの誤差 ε に対する mixing time $\tau(\varepsilon)$ は total variation distance が ε を下回るまでの遷移数で定義される．

定義 **1.5** (mixing time)

$$\tau(\varepsilon) = \min\{t : \|P^{t'}, \pi\|_{tv} \quad \varepsilon, \forall t' \quad t\}$$

$\tau(\varepsilon)$ が $\log n$ と $\log \varepsilon^{-1}$ の多項式で抑えられるものを,rapidly mixing(高速混合) と呼ぶ. $\log n$ の多項式とは，n 状態を区別するために必要なビット数の多項式という意味である．本稿では標準ランダムウォーク，β-ランダムウォークおよびメトロポリスウォークの mixing time に関する解析を行い，標準ランダムウォークとメトロポリスウォークについて rapidly mixing となるグラフ構造について考察する．

2 mixing time の見積り手法

2.1 第 2 固有値による見積り

mixing time は遷移確率行列の固有値を用いて見積もることができる．遷移確率行列の固有値 $\lambda_0, \lambda_1, \quad , \lambda_{n-1}$ について，任意の $i \quad 2$ に対して $1 = \lambda_0 > |\lambda_1| \quad |\lambda_i$ とする．特に，$Gap(P) = 1 \quad |\lambda_1|$ を *special gap* と呼ぶ．special gap と mixing time に関する定理が知られている [5].

定理 **2.1** $\pi_* = \min_{u \in V} \pi_u$ とする．任意の $\varepsilon > 0$ について，

$$\tau(\varepsilon) \quad \frac{1}{2(1 \quad |\lambda_1|)} \log \quad \frac{1}{2\varepsilon} \quad ,$$
$$\tau(\varepsilon) \quad \frac{1}{1 \quad |\lambda_1|} \log \quad \frac{1}{\pi_* \varepsilon} \quad ,$$

が成り立つ．

2.2 コンダクタンス

辺 $e = (u, u)$ の容量を $c_{uv} = \pi_u p_{uv} = \pi_v p_{vu}$ とする．このとき，頂点の部分集合 $S \subset V$ について

$$\Phi_S = \frac{\sum_{u \in S, v \notin S} c_{uv}}{\pi_S}, \qquad (1)$$

とする．ここで，$\pi_S = \sum_{u \in S} \pi_u$ である．ランダムウォークのコンダクタンス Φ を

$$\Phi = \min_{S : \pi_S \leq 1/2} \Phi_S,$$

と定義する．

定理 **2.2** コンダクタンス Φ を持つランダムウォークについて，

$$\frac{\Phi^2}{2} \quad Gap(P) \quad 2\Phi$$

が成り立つ．

定理 2.1 および定理 2.2 からコンダクタンスによる mixing time の見積もりを得ることができる．

2.3 Canonical Path

ランダムウォークのコンダクタンス Φ を見積もる手法として canonical path を用いる手法がある．頂点の順序対 (u, v) について，x と y を結ぶパス γ_{uv} を一つ選び，それを (u, v) の canonical path とよぶ．全ての順序対に対する canonical path の集合 $\Gamma = \{\gamma_{uv} | (u, v) \in V^2\}$ について，Γ の混雑度 $\rho(\Gamma)$ を

$$\rho(\Gamma) = \max_{e=(u,v)} \frac{1}{\pi_u p_{uv}} \sum_{\gamma_{uv} \ni e} \pi_u \pi_v \qquad (2)$$

と定義する．

定理 2.3 任意の *canonical path* の集合 Γ について

$$\Phi \quad \frac{1}{2\rho(\Gamma)}$$

が成り立つ．

この定理から mixing time の上界を得ることもできるが，Γ の混雑度から直接 mixing time の上界を得ることができる [6]．$\bar{\rho} = \min_\Gamma \rho(\Gamma) l_\Gamma$ とする，ただし l_Γ は Γ に含まれる最長のパスの長さである．

定理 2.4 初期点 u のランダムウォークにおいて以下が成り立つ．

$$\tau(\varepsilon) \quad \bar{\rho}(\log(\pi_u)^{-1} + \log \varepsilon^{-1}).$$

以降の議論のため，パス集合 Γ における極大性を次のように定義する．

定義 2.5 パス集合 Γ の要素 γ_{uv} は，すべての $\gamma_{zw} \in \Gamma \setminus \{\gamma_{uv}\}$ に対して $\gamma_{uv} \nsubseteq \gamma_{zw}$ であるとき極大であるとする．

例えば，図 1 ではパス 1-2-3-4 が極大で他のパスはその部分パスになっている．

Γ における極大なパスの集合を Γ_M とする．また，パス集合 Γ における辺 e の重複度 $\delta_\Gamma(e)$ を

$$\delta_\Gamma(e) = |\{\gamma_{uv} \in \Gamma : e \in \gamma_{uv}\}|$$

とし，その最大値を $\delta_\Gamma = \max_{e \in E} \delta_\Gamma(e)$ とする．例えば，$\delta_{\Gamma_M}((u, v))$ は (u, v) を含む極大なパスの数を表す．

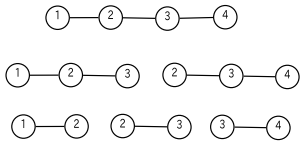


図 1: 極大パス 1-2-3-4 とその部分パス

2.4 枝重みによるランダムウォークの表現

各辺 (u, v) に重み w_{uv} が与えられ，遷移確率および定常分布が

$$p_{uv} = \frac{w_{uv}}{w_u},$$
$$\pi_u = \frac{w_u}{w}.$$

で表されるランダムウォークを考える．ただし，$w_u = \sum_{v \in N(u)} w_{uv}$, $w = \sum_{u \in V} w_u$ である．任意の reversible なランダムウォークはこのような形で表現することが可能であり，$w_{uv} = 1$ とした場合が標準ランダムウォーク，$w_{uv} = 1/\sqrt{d_u d_v}$ とした場合が $\beta = \frac{1}{2}$ の β-ランダムウォークとそれぞれ等価なランダムウォークである．このような表現はコンダクタンスおよび，canonical path の解析と相性がよい．式 (1) に代入すると，

$$\Phi_S = \frac{\sum_{u \in S, v \notin S} w_{uv}/w}{\sum_{u \in S} w_u/w} = \frac{\sum_{u \in S, v \notin S} w_{uv}}{\sum_{u \in S} w_u} \qquad (3)$$

となる．特に $\sum_{u \in S} w_u/w \quad 1/2$ のとき，

$$\frac{2 \sum_{u \in S, v \notin S} w_{uv}}{w} \quad \Phi_S \qquad (4)$$

となる．

また，式 (2) に代入すると，

$$\rho(\Gamma) = \max_{e=(u,v)} \frac{w}{w_{uv}} \sum_{e \in \gamma_{uv}} \frac{w_u}{w} \frac{w_v}{w},$$
$$= \max_{e=(u,v)} \frac{1}{w w_{uv}} \sum_{e \in \gamma'_{uv} \in \Gamma_M} \sum_{i=1}^{k} w_i \sum_{j=k+1}^{l} w_j.$$

ただし，$\gamma'_{uv} := v_1(= x), \quad , v_k(= u), v_{k+1}(= v), \quad , v_l(= y)$ である．

3 標準ランダムウォークの mixing time

標準ランダムウォークは $w_{uv} = 1$ の場合であり，コンダクタンスに関して以下の下界が導かれる．

補題 **3.1** $\sum_{u \in S} \pi_u \quad 1/2$ となる任意の $S \subseteq V$ のコンダクタンスについて以下が成り立つ．

$$\Phi_S \quad \Omega \quad \frac{1}{m} \quad .$$

証明　式 4 へ $w_u = d_u \quad 1, w = 2m$ を代入すると，$\sum_{u \in S} \pi_u \quad 1/2$ となる任意の $S \subseteq V$ について，

$$\Phi_S \quad \frac{1}{m}$$

が得られる． □

この結果と，補題 3.1 および定理 2.1 を組み合わせると，標準ランダムウォークの mixing time に関して $O(n^4(\log n + \log \varepsilon^{-1}))$ という上界を得ることができる．しかし，グラフの構造によってはより小さい上界をもつ場合がある．そこで，今度は canonical path の手法を用いてより詳細な解析を行う．

補題 **3.2** Γ を最短路の集合とすると，以下が成り立つ．

$$\rho(\Gamma) = O \quad \frac{\delta_{\Gamma_M} n^2}{m} \quad .$$

証明　Γ として最短路の集合を考えると，

$$\rho(\Gamma) = \max_{e=(u,v)} \frac{1}{2m} \sum_{e \in \gamma'_{uv} \in \Gamma_M} \sum_{i=1}^{k} d_i \sum_{j=k+1}^{l} d_j, \quad \frac{9\delta_{\Gamma_M} n^2}{2m}.$$

を得る． □

この補題と定理 2.4 から，標準ランダムウォークにおける mixing time について次の上界を得ることができる．

定理 **3.3** Γ を最短路の集合とし，l_Γ を最大の長さとする．任意のグラフにおける誤差 ε に対する mixing time について以下が成り立つ．

$$\tau(\varepsilon) = O \quad \frac{\delta_{\Gamma_M} l_\Gamma n^2}{m} (\log n + \log \varepsilon^{-1}) \quad .$$

この上界はグラフの構造によってかなり差が出るが，$m = \Omega(n^2)$ で $\delta_{\Gamma_M} l_\Gamma$ が $\log n$ の多項式で抑えられるグラフでは rapidly mixing となる．また，定理 3.2 と定理 3.3 の小さい方を上界とすれば，一般のグラフに対する上界が次にように改善される．

系 **1** n 頂点からなる任意のグラフ上における標準ランダムウォークの mixing time について，

$$\tau(\varepsilon) = O(n^{\frac{10}{3}} (\log n + \log \varepsilon^{-1}))$$

が成り立つ．

4 β-ランダムウォークの mixing time

$\beta = \frac{1}{2}$ とする．$w_{uv} = 1/\sqrt{d_u d_v}$ とすると，$\beta = \frac{1}{2}$ の β-ランダムウォークと等価となる．また，重みの総和 w について次の補題が成り立つ．

補題 **4.1** $(u, v) \in E$ について，$w_{uv} = 1/\sqrt{d_u d_v}$ とすると，以下が成り立つ．

$$w = \sum_{u \in V} \sum_{v \in N(u)} w_{uv} \quad n.$$

証明　相加相乗平均の関係から以下の式を得る．

$$\begin{aligned} w &= \sum_{u \in V} \sum_{v \in N(u)} \frac{1}{\sqrt{d_u d_v}}, \\ &\quad \frac{1}{2} \sum_{u \in V} \sum_{v \in N(u)} \quad \frac{1}{d_u} + \frac{1}{d_v} \quad , \\ &= n. \end{aligned}$$

□

補題 **4.2** n 頂点からなる任意のグラフにおける β-ランダムウォークのコンダクタンスについて以下が成り立つ．

$$\Phi = \Omega \quad \frac{1}{n^2} \quad .$$

証明　$w_{uv} = 1/\sqrt{d_u d_v}$ を式 (4) に代入すると，

$$\Phi \quad \frac{2}{nw}.$$

補題 4.1 より，$\Phi = \Omega\left(\frac{1}{n^2}\right)$ が得られる．　□

この補題と，定理 2.1 および定理 2.2 から β-ランダムウォークの mixing time についても $O\left(n^4(\log n + \log \varepsilon^{-1})\right)$ という上界を得る．一方，canonical path の集合 Γ の混雑度に関して解析を行うと，次の補題を導くことができる．

補題 **4.3** 任意の Γ について，以下が成り立つ．

$$\rho(\Gamma) = O\left(\delta_{\Gamma_M} n^2\right).$$

この補題と定理 2.4 から，β-ランダムウォークにおける mixing time について次の上界を得ることができる．

定理 **4.4** 任意のグラフにおける β-ランダムウォークの誤差 ε に対する mixing time $\tau(\varepsilon)$ について，

$$\tau(\varepsilon) = O(\delta_{\Gamma_M} l_\Gamma n^2(\log n + \log \varepsilon^{-1}))$$

が成り立つ．

$\delta_{\Gamma_M} l_\Gamma n^2$ はグラフの構造によって n^2 から n^4 となる．一般のグラフに対する上界は標準ランダムウォークと同じだが，今回の結果からだけではあまりよい上界は得られない．実際は完全グラフのように β-ランダムウォークでも rapidly mixing になる場合が存在する．

5　メトポリスウォークの mixing time

定常分布をすべて等しくした場合を考える．定義から，任意の辺 $e = (u, v)$ について，

$$\pi_u p_{uv} = \pi_v p_{vu} = \max \quad \frac{\pi_u}{d_u}, \frac{\pi_v}{d_v}$$

が成り立つ．式 (1) に代入すると $\Phi = \Omega\left(\frac{1}{n^2}\right)$ が導かれるので，こちらも mixing time に関して $O\left(n^4(\log n + \varepsilon^{-1})\right)$ という上界を得ることができる．また，最短路集合 Γ の混雑度について次の補題が成り立つ．

補題 **5.1** 最短路の集合 Γ の混雑度 $\rho(\Gamma)$ について，$\rho(\Gamma) \quad \delta_{\Gamma_M} l_\Gamma^2$ である．

証明　式 (2) へ代入すると，

$$\rho(\Gamma) = \max_{e=(u,v)} n \max\{d_u, d_v\} \sum_{e \in \gamma_{uv}} \frac{1}{n^2},$$

$$\delta_{\Gamma_M} l_\Gamma^2.$$

□

この補題と定理 2.4 から，メトロポリスウォークの mixing time について次の上界を得ることができる．

定理 **5.2** 任意のグラフにおけるメトロポリスウォークの誤差 ε に対する mixing time $\tau(\varepsilon)$ について，

$$\tau(\varepsilon) = O(\delta_{\Gamma_M} l_\Gamma^3(\log n + \log \varepsilon^{-1}))$$

が成り立つ．

$\delta_{\Gamma_M} l_\Gamma^3$ が $\log n$ の多項式で抑えらるグラフでは rapidly mixing の条件を満たす．

6　まとめ 今後の課題

本研究の目的は，ランダムウォークにおける局所情報の利用と mixing time との関係を明ら

かにすることであり，その第一歩として 3 つのランダムウォークに関する解析を行った．

すべてのランダムウォークについて，コンダクタンスを用いた解析により任意のグラフに対する mixing time の上界が $O(n^4(\log n + \log \varepsilon^{-1}))$ という結果を得た．また，標準ランダムウォーク及びメトロポリスウォークについては canonical path を用いた解析により，rapidly mixing となるための十分条件が得られた．実際，完全グラフにおいては rapidly mixing になっている．一方，β-ランダムウォークでは今回得た結果からだけでは rapidly mixing の条件を満たすグラフの性質を導くことはできない．より詳細な解析は今後の課題である．

参考文献

[1] R.Aleliunas, R.M Karp, R.J. Lipton, L.Lovaasz, and C.Racko , “Random walks, universal traversal sequences, and the complexity of maze problems”, Proc. 20th IEEE Ann.Symposium on Foundations of Computer Science, 218-223, 1979.

[2] D.J.Aldous, “On the time taken by random walks on finite groups to visit every state”, Z.Wahrsch. verw. Gebiete 62 361-1983.

[3] W.K. Hastings, “Monte Carlo sampling methods using Markov chains and their applications,” *Biometrika,* 57, 97–109, 1970.

[4] Satoshi Ikeda, Izumi Kubo, Norihiro Okumoto, Masafumi Yamashita “Impact of Local Topological Information on Random Walks on Finite Graphs”, Proceedings of Thirtieth International Colloquium on Automata,Languages and Programming, 1054-1067,2003.

[5] M.R.Jerrum, L.G. Valiant, and V.V. Vazirani, “Random Generation of Conbinatirial Structures from a Uniform Distirbution”, Theoritical Computer Science, vol. 43, 1986, pp.169-188.

[6] A.J. Sinclair, “Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow, ” Combinatorics, Probability and Computing, vol. 1, 1992, pp.351-370.

頻出項目検出階層型ストリームアルゴリズム

溝口隆 * 小野廣隆 ** 山下雅史 **
(九州大学大学院 システム情報科学府* システム情報科学研究院**)

1 はじめに

入力を蓄積せずに逐次的に時系列データを処理するアルゴリズムをストリームアルゴリズムと呼ぶ. ストリームアルゴリズムは通信ログデータの解析など, 莫大かつ逐次的に得られる時系列データに頻出する項目を検出するためなどに用いられる.

本研究では複数の情報源からデータが階層状に伝播するような通信ネットワーク上でのストリームアルゴリズムについて考察を行う. ネットワークは各葉の深さが全て等しい木状になっている. データは情報源から最下位層のノード (葉) に対して与えられるものとし, データは上位層に対しての一方向のみに送られる. ただし, 各上位層の 1 つのノードが単位時間当たり受け取ることができるデータの量は最下位層のあるノードが単位時間当たり受け取ることができるデータの量と等しいものとする. この制約のため, 上位層のノードは必ずしも全情報源からのデータ (データ全体) における頻出する項目を検出できるとは限らない. 本研究の目的は, このような制限下でも最上位層のノードができるだけデータ全体の頻出項目を検知できるようなストリームアルゴリズムを設計することにある. 本稿ではこれを**階層型ストリームアルゴリズム**と呼ぶ.

本研究では特に 2 階層のネットワーク上での階層型ストリームアルゴリズムについて考察を行う. 2 階層ネットワークは 1 つの親ノードと複数の子ノードにより表わされる. 今回, 親ノード数 1, 子ノード数 2, 各ノードのメモリを M として, 情報源から得られたデータの中で頻出する項目を近似的に検出する **2 階層型ストリームアルゴリズム**を設計し, 実験を行った.

2 準備

以下, 本稿で扱う記号用語を挙げる. 子ノード v_1, v_2, 入力アルファベット Σ, 項目 $a \in \Sigma$, 項目の種類数 $A = |\Sigma|$, 子ノード v が受け取るデータ列 x_i, 子ノードが受け取るデータ列の大きさ $\Sigma^{\frac{N}{2}}$, データ列全体 $x_{all} = \Sigma^N$, データ列全体の大きさ N, メモリ $B = \frac{1}{\theta} \lg N$, しきい値 $\theta (0 < \theta < 1)$, データ列 x 中の項目 a の数 $f_x(a)$, データ列 x における頻出項目 $f_x(a) = \{a \in \Sigma : f_x(a) > \theta |x|\}$. まず, 以下の定理が成立する.

定理 1. あるデータ列中の頻出項目は高々 $\frac{1}{\theta}$ 個である.

定理 2. 2 階層型ストリームアルゴリズムにおいて, ある項目 a がデータ列全体 x_{all} において頻出項目ならば少なくとも 1 つの子ノードのデータ列 x_i においても a は頻出項目である.

3 提案手法と計算実験

定理 1 より, 親ノードが漏れなく頻出項目を検出するためには少なくとも $\frac{1}{\theta}$ 個の項目を捉えている必要がある. よって, 各ノードのメモリを $M = \frac{1}{\theta} \lg N$ として考える.

3.1 提案手法

今回, 2 種類の手法を用いて実験を行った. 各子ノードのデータ列 x_i 中のデータを親ノードがランダムに受け取りストリームアルゴリズム [1] を実行する手法を**サンプリング法**と呼ぶ. サンプリング法を用いることでほぼ正確な頻出項目を得ることができるがデータ列の項目の並びによってまれに頻出項目を検出できないことがある. このような問題を解決するために定理 2 の子ノードの受け取るデータ列における頻出項目の中にデータ列全体における頻出項目が必ず現れることを利用して, 以下のような手法を提案する. 各子ノードに対して, ストリームアルゴリズム [1] を用いて頻出項目を逐次的に検出し, それらの項目に対して重みを持たせ確率的に親ノードへデータを送る. この手法を**重み付バッグ法**と呼ぶ.

3.2 計算実験

パラメータは以下のように設定した. $N = 100000, A = 1000, B = 500 \lg 10$, 子ノード数 2, 実行回数 100. 今回, 上記のパラメータを固定し, 頻出項目の個数を変化させることで 2 つの手法の検出精度の比較を行った.

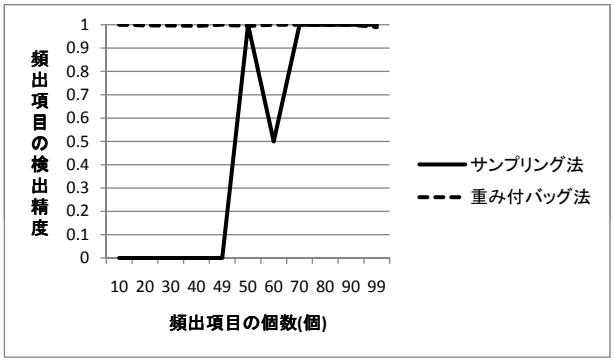


図 1: サンプリング法の結果が最悪になるような場合

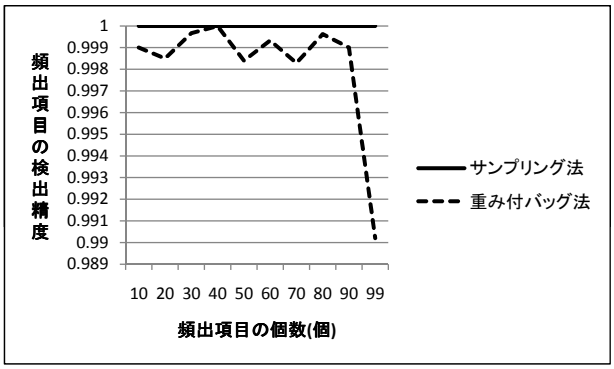


図 2: データ列中の項目の並びがランダムな場合

4 まとめ

本稿では, 2 階層のネットワーク上での頻出項目を検出するための 2 つの階層型ストリームアルゴリズムを提案した. 計算実験の結果, サンプリング法は最悪の場合以外はほぼ正確に頻出項目を検知することができた. しかし, 最悪の場合まったく頻出項目を検知できなかった. 重み付バッグ法は全体的にサンプリング法と比べ頻出項目の検出精度は落ちるものの極端に検出漏れをすることがなかった.

参考文献

[1] Richard M. Karp, Scott Shenker. A Simple Algorithm for Finding Frequent Elements in Streams and Bags, ACM Transactions on Database Systems, Vol. 28, No.1, pp. 51–55, March 2003.

[2] 徳山 豪.「オンラインアルゴリズムとストリームアルゴリズム」, August 2007

[3] 玉木 久夫.「乱択アルゴリズム」, August 2008

A Biased k-Random Walk to Find Useful Files in Unstructured Peer-to-Peer Networks

Hiroo Kitamura Satoshi Fujita
Hiroshima University, Japan

Abstract

In this paper, we consider a problem of finding "useful" files matching a given query in unstructured P2Ps. The proposed scheme is a variant of k-random walk, which combines a synchronization mechanism proposed by Lv *et al.* with a mechanism to evaluate the usefulness of discovered files. In addition, we apply a variant of popularity-biased k-random walk to accelerate the file search in normal k-random walk under uniform distribution. The goodness of the scheme is evaluated by simulation. The result of simulations indicates that the proposed biased k-random walk scheme certainly finds useful files in short time, without significantly increasing the number of message transmissions.

Keywords: Unstructured P2P, k-random walk, usefulness of file, file search.

1 Introduction

Recently, unstructured Peer-to-Peer (P2P) systems have attracted considerable attentions as a way of sharing information among several computers (i.e., peers) in distributed networks. In contrast to structured P2Ps [2, 10, 11, 13, 14, 15], which have a systematic way of forwarding a given query to a peer holding an index to a target file, a key issue in designing unstructured P2Ps is how to quickly identify the location of a target file with as small number of message transmissions as possible.

Most of conventional file search schemes for unstructured P2Ps are based on the notion of *uncontrolled query propagation* such as flooding and random walk. A flooding-based scheme could find a number of files matching a given query in a relatively short time, although it needs a large number of message transmissions compared with a simple random walk. The number of message transmissions of both schemes could be reduced by appropriately setting TTL (Time-To-Live) to each query, while it restricts the location of peers from which the requester can receive the search result to a small portion of the network centered at the requesting peer.

In general, a user of a P2P file sharing service is interested in finding a "useful" file matching a given query. There may exist several ways of defining usefulness of files in real applications, such as the generation time (e.g., a newer file is better), existence of certification, reputation on the contents, and reputation on the contents holder (e.g., a user does not want to download a file from suspicious users). Let us consider a search of files (e.g., free-software) in a P2P file sharing service; i.e., consider a collection of files each of which is attached a set of tags representing attributes of the file. A query given by a user designates a tag attached to the target file (e.g., audio_and_video or utility), and given such query, we can imaginarily consider a set of files matching the query as a candidate for the search result. The problem we want to consider in this paper is stated as the problem of selecting files with high usefulness from such candidates (e.g., matching files recommended by many users), with as small number of message transmissions as possible.

In this paper, we propose a scheme to solve such file search problem. The proposed scheme is based on a *parallelized* version of random walk, which is known as k-random walk in the literature [9]. Note that, although a simple extension of a flooding-based scheme could find

*Department of Information Engineering, Hiroshima University, Kagamiyama 1-4-1, Higashi-Hiroshima, Japan

several useful files in a small region of the network centered at the requesting peer, it is difficult to select an appropriate value of TTL without enough knowledge about the target domain. On the other hand, although a naive extension of random walk could also find useful files by continuing a file search until it finds a matching result with high usefulness, such scheme takes a long time since it is essentially sequential.

The key idea of our proposed scheme is to use a variant of k-random walk as the basic scheme, and to combine it with a synchronization mechanism proposed by Lv *et al.* [9], and a mechanism to evaluate the usefulness of discovered files. In addition, we apply a popularity-biased k-random walk [16] to our scheme in order to accelerate the speed of a file search process compared with normal k-random walk under uniform distribution.

The performance of our scheme is evaluated by simulation. The result of simulations indicates that the proposed scheme certainly finds useful files in short time, without significantly increasing the number of message transmissions. More concretely, the usefulness of discovered files is much higher than files discovered by conventional schemes, which could not be attained by the previous scheme even by increasing the number of walkers and/or TTL of the transmitted query.

The remainder of this paper is organized as follows. Section 2 overviews related work. Section 3 describes our proposed scheme, and the result of simulations is summarized in Section 4. Finally, Section 5 concludes the paper with future problems.

2 Related Work

2.1 Basic Schemes

Conventional file search schemes for unstructured P2Ps can be classified into two categories; i.e., flooding-based schemes [3, 12] and random walk [1, 7]. Let s be the originator of query q (q is referred to as a "walker" in random walk). A flooding of q to peers in a network proceeds as follows:

FLOODING_BASED_SCHEME

Initialization: At first, peer s initializes TTL of the query to an appropriate positive constant, and transmits it to all neighbors of s.

Matching: Upon receiving query q from a neighbor v, a peer u checks if it owns a file matching the query. If it owns, u sends a reply message to s and stops the propagation of the query. Otherwise, it executes the following forwarding step.

Forwarding: If the TTL of q is zero, peer u simply discards the received query. Otherwise, u forwards q to all neighbors except for v after decrementing its TTL by one.

It is well known that the performance of flooding-based schemes depends on the initial value of TTL; i.e., a large TTL attains a high hit rate while it significantly increases the network traffic, where a hit rate is the percentage of trials which find a file matching the given query. In general, we have to use relatively large TTL to keep a sufficiently high hit rate (e.g., 90%), which indicates that the traffic of flooding-based schemes is usually high.

On the other hand, a random walk circulates a "walker" in the network until it finds a matching file or it reaches a predetermined TTL. Concrete procedure is described as follows:

RANDOM_WALK

Initialization: At first, peer s initializes TTL of the walker to an appropriate positive constant, and transmits it to a random neighbor of s.

Matching: Upon receiving query q from a neighbor v, peer u checks if it owns a file matching it. If it owns, u sends a reply message to s and stops the propagation of the query. Otherwise, it executes the following forwarding step.

Forwarding: If the TTL of q is zero, peer u discards the received query, and if it is positive, u forwards q to one of randomly selected neighbors except for v after decrementing its TTL by one.

2.2 k-Random Walk

In general, the hit rate of random walk is not as high as the hit rate of flooding-based schemes, while it significantly decreases the number of message transmissions compared with flooding-based schemes. Such drawback of random walk can be overcome by increasing the number of walkers from one to k. Such scheme is referred to as k-random walk in the literature. In the k-random walk proposed in [9], k walkers are independently circulated among peers,

and in order to reduce the number of message transmissions, those walkers are controlled via a synchronization mechanism called checking.

More concretely, checking is a task of asking the originator of a random walk whether or not a target file matching the query has been found. If such file has been found, walkers immediately terminate their search process even if there remains TTL. Each walker periodically executes such checking. Let c denote the cycle of checking. The effect of checking to reduce the number of redundant message transmissions increases as decreasing c, while it also increases the cost for the checking.

A file search based on k-random walk with checking proceeds as follows:

k-RANDOM_WALK

Initialization: At first, peer s initializes TTL of each walker to an appropriate positive constant, and for each walker q, s transmits it to a random neighbor of s.

Matching: Upon receiving walker q from a neighbor v, peer u checks if it owns a file matching it. If it owns, u stops the circulation after sending a reply message to s, and otherwise, it executes the following forwarding step.

Forwarding: If the TTL of q is zero, peer u simply discards the received query, and if it is positive, s executes the following operation:

1. If the TTL of q is a multiple of c, u sends an inquiry message to s asking whether a target file for the query has been found. If it has been found, s stops the search process of u by sending a termination message. Otherwise, it asks u to continue the search process.

2. If it is not terminated by s, u forwards q to a random neighbor of u after decrementing the TTL by one.

It is shown that the number of peers visited by k walkers within t steps is almost equal to the number of peers visited by one walker within kt steps [9]. It implies that the search time of a k-random walk linearly decreases as increasing the number of walkers provided that the spatial distribution of target files is uniform; i.e., by appropriately tuning parameter k, we could obtain a scheme which finds a target file in short time, while keeping the number of message transmissions relatively small. Unfortunately however, such naive extension of random walk can not be directly applied to the problem of finding useful files, since it immediately stops k walkers after finding the first matching file regardless of the usefulness of the discovered file. Such a behavior of the scheme looses a chance of finding highly useful files, since the probability of finding the most useful file as the first matching one is not very high, in general.

2.3 Popularity-Biased Random Walk

Recently, Zhong and Shen proposed a variant of k-random walk, in which the probability of selecting the next peer is biased according to the popularity of files and the hit rate [5, 16]. Let d_i denote the number of neighbors of peer i, and p_i be the number of hits at peer i divided by the number of visits to peer i (i.e., p_i denotes a kind of success rate). Then, the probability that i selects its neighbor j as its next peer, denoted by $P_{i,j}$, is determined as follows:

$$P_{i,j} = \begin{cases} 1/2d_i & \text{if } \sqrt{p_i}/d_i \leq \sqrt{p_j}/d_j, \text{ and} \\ (1/2d_j)\sqrt{p_j/p_i} & \text{otherwise.} \end{cases}$$

Note that since $\sqrt{p_i}/d_i > \sqrt{p_j}/d_j$ implies $\sqrt{p_j/p_i} < d_j/d_i$, we have $P_{i,j} \leq 1/2d_i$ for any i; i.e., $\sum_j P_{i,j} \leq d_i \times (1/2d_i) = 1/2$ [4]. This implies that peer i selects itself as the next peer (i.e., self-loop) with probability at least 1/2, and such probability increases if it has many neighbors whose success rate[1] is lower than that of peer i. The remaining part of the scheme is similar to [9], i.e., the number of walkers is fixed to k and the circulation of walkers is synchronized by checking.

3 Proposed Method

3.1 Usefulness of Files

In this paper, we assume that the usefulness of a file is represented by a vector in a coordinate space, where each axis in the coordinate space corresponds to a metric representing the usability, reliability, and the availability of the file. More concretely, in this paper, we will consider the following five metrics:

[1] More precisely, it is success rate normalized by a square of its degree.

- Generation time of a file, e.g., a newer file is more useful if they are given the same set of tags.
- File size, i.e., a file is not useful if the size of the file is too large (e.g., exceeding 1GB) or too small (e.g., empty).
- Existence of authorized certification, where certification is graded by the time stamp; e.g., a file with old certification is less useful than a file with new certification.
- Reputation on the contents [6, 8], which will reflect the popularity of the file.
- Reputation on the contents holder, which is necessary to avoid download from suspicious users.

In order to compare the usefulness of two files, we introduce the following five classes of evaluation functions; i.e., 1) projection (i.e., ignore other metrics except for specific one), 2) count (i.e., the number of metrics whose value exceeds a predetermined threshold), 3) vector (i.e., element-wise comparison of vectors), 4) lexicographical order, and 5) weighted sum.

3.2 Basic Procedure

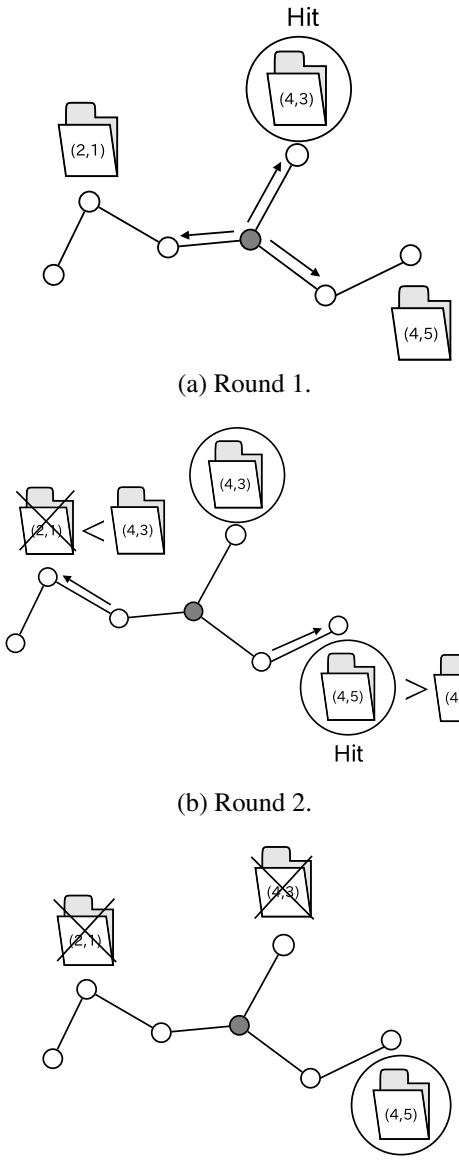
In this subsection, we propose a procedure to find a useful file matching the given query by circulating k walkers. Concrete procedure is described as follows:

PROPOSED_SCHEME

Initialization: At first, peer s initializes TTL of each walker to an appropriate positive constant, and for each walker, s transmits it to a neighbor which is selected according to a predetermined probability as in the popularity-biased random walk (details on the transition probability will be described in the next subsection).

Matching: Upon receiving query q from a neighbor, peer u checks if it owns a matching file which is *more useful* than the most useful file discovered so far (such information is acquired via checking). If it owns such file, u sends a reply message to s with the usefulness of the discovered file (the circulation process does not stop at this time unlike k-RANDOM_WALK).

Forwarding: If the TTL of q is zero, peer u discards the received query, and if it is positive, s executes the following operation:



(a) Round 1.

(b) Round 2.

(c) Round 3.

Figure 1: Running example of the proposed scheme.

1. If the TTL of q is a multiple of c, u sends an inquiry message to s to acquire the usefulness of the most useful file discovered so far. If a file with sufficiently high usefulness has been found, s stops the walker by sending a termination message to u. Otherwise, it returns the usefulness of the (currently) most useful one to u.

2. If it is not terminated by s, u forwards the walker to a neighbor after decrementing the TTL by one, where the next peer is selected according to a predetermined probability.

Example: Figure 1 illustrates a running example of the proposed scheme, where each circle represents a peer and a gray circle represents the originator of a k-random walk. The number of walkers is fixed to three, and the checking cycle is set to one (i.e., each walker communicates with the originator for every round to report the current status to the originator). The usefulness of files is represented by a vector of size two such as $(4, 3)$ and $(1, 2)$, where we assume that a larger value indicates a higher usefulness. Now, let us consider a case in which the search process terminates when it finds a file of usefulness of $(4, 4)$ or more. In this example, a walker finds a file of usefulness $(4, 5)$ in the third round. Such information is collected to the originator immediately after the round (since we are assuming that the checking cycle is one), and after receiving it, the originator sends a termination message to all walkers to stop the search process.

3.3 Transition Probability

As the transition probability of walkers, we propose the following three schemes. In the following, we use symbol $P^k_{i,j}$ to denote the probability that peer i selects its neighbor j as its next peer in the search of useful files concerned with the k^{th} metric of the usefulness.

- The first one is a simple extension of Zhong and Shen's popularity-biased k-random walk (we call it Simple hereafter). Let p^k_i denote the number of hits at peer i concerned with the k^{th} metric (i.e., the number of visits of the peer which find a matching file at the peer such that the usefulness of the file concerned with the k^{th} metric exceeds a predetermined threshold) divided by the number of visits to peer i (note that $p_i \leq \sum_k p^k_i$). Then, probability $P^k_{i,j}$ is determined as follows:

$$P^k_{i,j} = \begin{cases} 1/2d_i & \text{if } \sqrt{p^k_i}/d_i \leq \sqrt{p^k_j}/d_j \\ (1/2d_j)\sqrt{p^k_j/p^k_i} & \text{otherwise.} \end{cases}$$

  where d_i denotes the degree of peer i.

- The second idea is to limit the number of repetitive selection of a self-loop by at most two; i.e., in the first two steps, it follows the probability used in Simple, but if it selects itself as the next peer in both of the first two steps, in the third step, it conducts a (biased) coin toss until one of its neighbors is selected as the next peer. In the following, we refer to the resultant scheme Limit.

- The third idea is to replace the denominator of the probability p^k_i by the number of intentional visits to peer i. Note that it would exclude the effect of random visits from the evaluation of the hit rate. In the following, we refer to the resultant scheme Intention.

Finally, if a user is interested in two metrics k_1 and k_2, it simply takes an average of the probabilities concerned with those metrics in selecting the next peer in the biased random walk, i.e., the probability of selecting j as the next peer of i is determined as

$$P_{i,j} = \frac{P^{k_1}_{i,j} + P^{k_2}_{i,j}}{2},$$

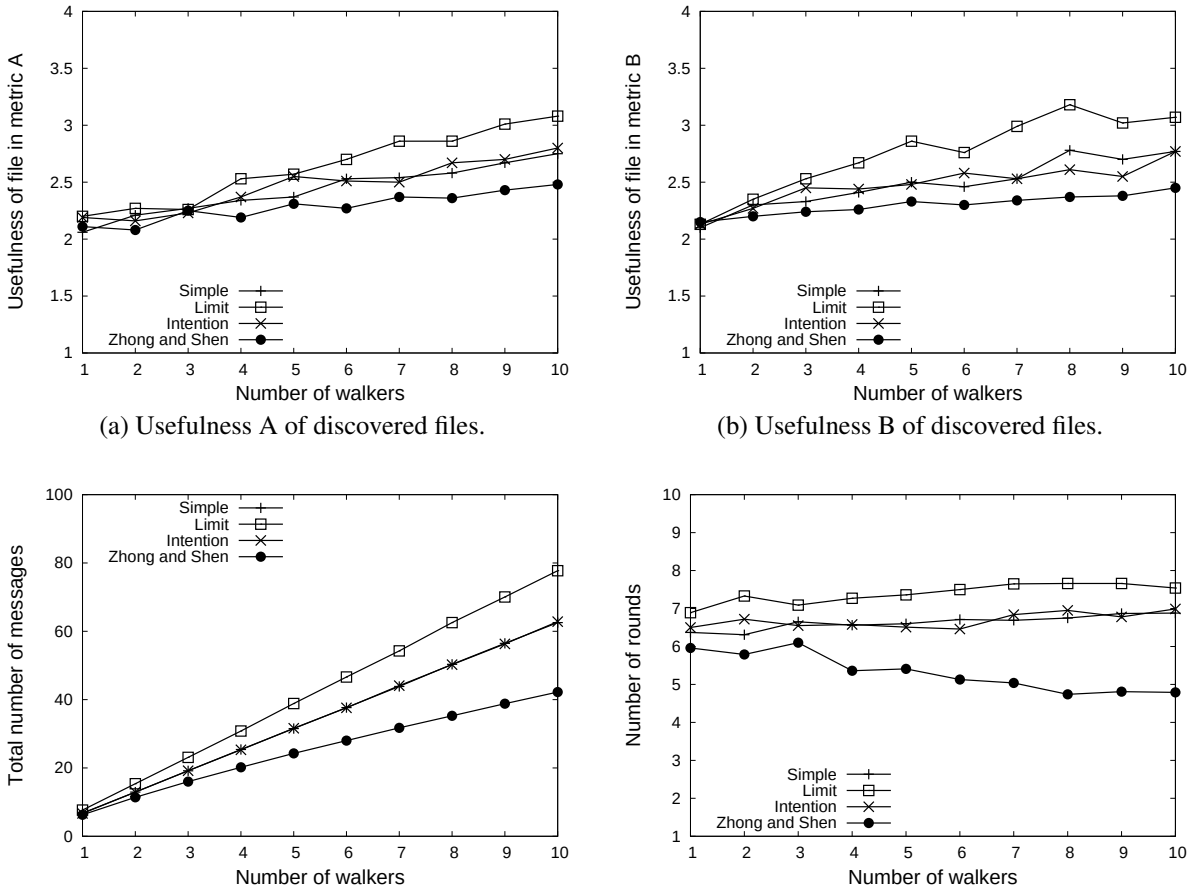
which is truncated in scheme Limit and the denominator of the hitting rate is refined in scheme Intention.

4 Evaluation

In this section, we evaluate the performance of the proposed scheme by simulation.

4.1 Simulation Model

In the following, we consider an unstructured P2P consisting of 1000 peers, in which edges are placed randomly in such a way that an average degree of each peer is 10. In



(a) Usefulness A of discovered files.

(b) Usefulness B of discovered files.

(c) Total number of transmitted messages.

(d) Number of rounds.

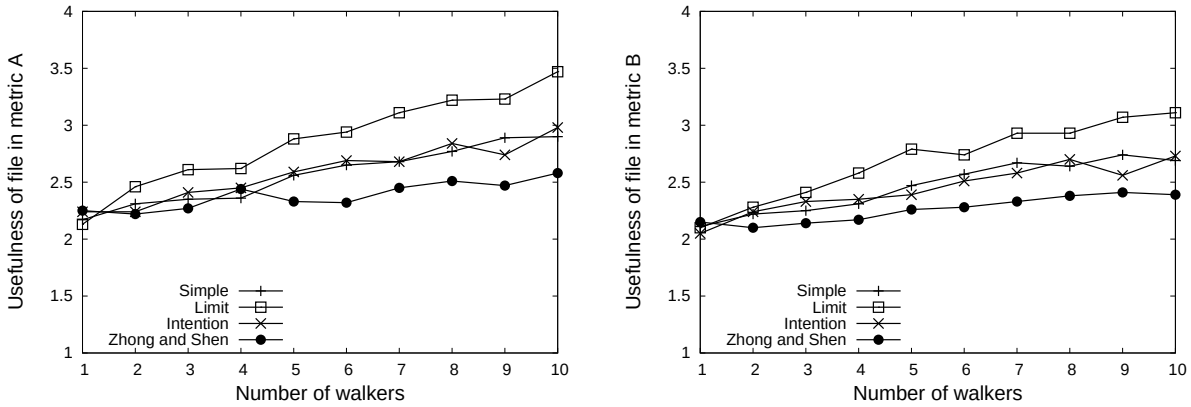
Figure 2: Result for single metric.

each run of the simulation, we randomly select an originator, and the originator issues a query requesting for a target file according to the Zipf's law; i.e., the probability of selecting the i^{th} popular file is proportional to $(1/i)^{\alpha}$, where α is a real called Zipf's parameter[2]. In the following, without loss of generality, we assume that a file with smaller sequence number has a higher popularity. TTL of a random walk will be varied from 8 to 28, and the cycle for checking is fixed to five (in the following figures, we fix the TTL to 14). The number of walkers is varied from $k = 1$ to 10.

We prepare 5000 files, each of which is assigned a sequence number from 1 to 5000, and is associated with two kinds of usefulness A and B. The grade of each metric ranges from 1 (least useful) to 5 (most useful), and we assume that grade 4 is the threshold to terminate a search process. In the following, we often represent the usefulness of a file as a vector of length two; i.e., (x, y) means that the usefulness of the file concerned with metrics A and B is x and y, respectively.

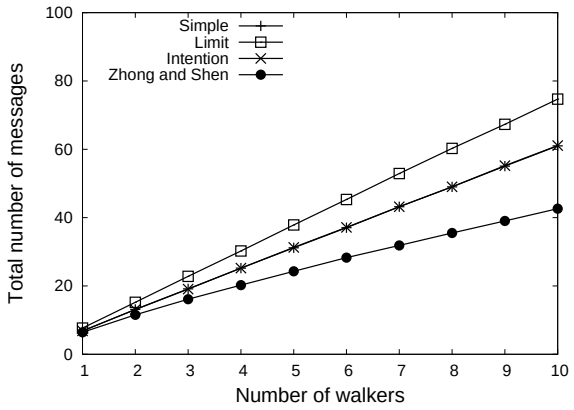
The performance of each scheme is evaluated in terms of the following three metrics (each value described below

[2]The result of experiments conducted by Sripanidkulchai indicates that the value of α should be selected from 0.6 to 1.2, and Zhong and Shen [16] indicates that the effect of popularity-biased random walk becomes significant for large α. Hence in the following, we fix α to 1.2.

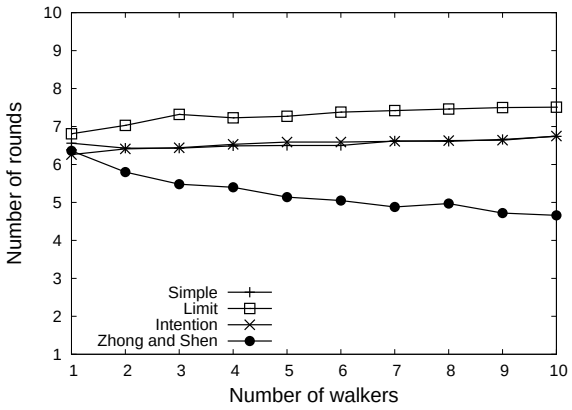


(a) Usefulness A of discovered files.

(b) Usefulness B of discovered files.



(c) Total number of transmitted messages.



(d) Number of rounds.

Figure 3: Result for weak correlation.

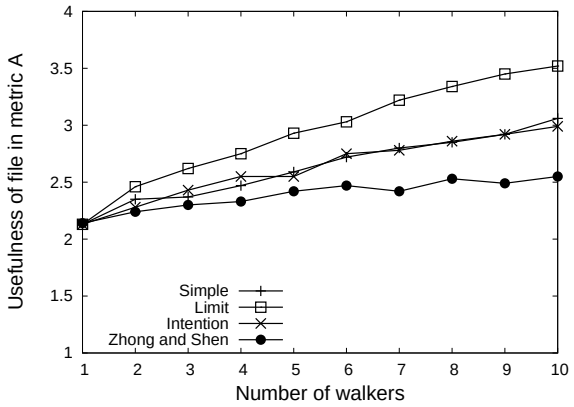
is an average over 10000 runs):

- The usefulness of most useful files discovered by a search process.
- The number of messages which are transmitted during a search process.
- The number of rounds required for finding a target file of sufficiently high usefulness.

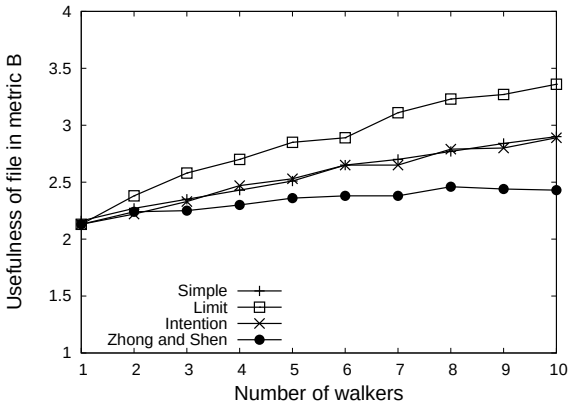
4.2 Single Metric of Usefulness

At first, we evaluate the performance of the proposed scheme for a single metric A. Usefulness of files is determined as follows: 1) 10% of given 1000 peers hold files matching the given query, 2) 15% of them holds matching files with high usefulness of 4 or more, and 3) the remaining peers hold matching files with usefulness of 3 or less.

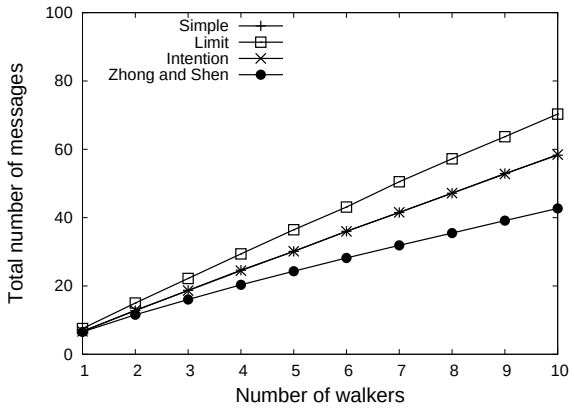
Figure 2 illustrates the result for TTL=14. The usefulness of discovered files is certainly improved by spending more time and more message transmissions similar to the previous scheme. Among three proposed schemes, the impact of Limit to the usefulness of the discovered files is more significant than the other two schemes; i.e., the usefulness of discovered files increases by 17%, while the number of transmitted messages increases by 59% and the number of rounds increases by 41% compared with



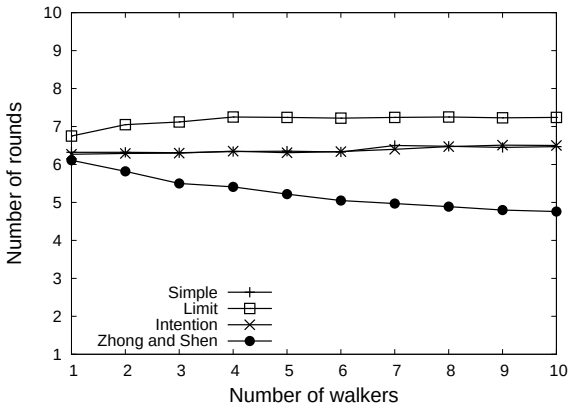
(a) Usefulness A of discovered files.



(b) Usefulness B of discovered files.



(c) Total number of transmitted messages.



(d) Number of rounds.

Figure 4: Result for strong correlation.

the previous scheme. On the other hand, the difference between Simple and Intention is not large, and in both schemes, the usefulness of files increases by 8%, while the number of transmitted messages increases by 30% and the number of rounds increases by 26% compared with the previous one.

4.3 Correlated Usefulness

Next we evaluate the performance of the schemes for two metrics A and B, by focusing on the *strength of correlation* between them. We separately consider two cases, i.e., the case of weak correlation and the case of strong correlation.

The setting for a weak correlation is determined as follows: 1) the grade of two metrics given to each file can differ by at most two, 2) 10% of given peers hold files matching the given query, 3) 15% of them holds matching files with high usefulness of 4 or more, and 4) the remaining peers hold matching files with usefulness of 3 or less. The coefficient of the resultant correlation between two metrics is 0.43.

Result of simulation is summarized in Figure 3, where we fix TTL of the schemes to 14, as before. The figure indicates that the impact of the proposed scheme is slightly *enhanced* by considering a weak correlation of two met-

rics. In Limit, the usefulness of files increases by 20%, while the number of transmitted messages increases by 54% and the number of rounds increases by 41% compared with the previous scheme. In Simple and Intention, the usefulness of files increases by 8%, while the number of transmitted messages increases by 28% and the number of rounds increases by 26% compared with the previous one.

Such effect of enhancement is increased by increasing the strength of correlation, as shown in the Figure 4. The setting for a strong correlation is determined as follows: 1) the grade of two metrics given to each file can differ by at most one, 2) 10% of given peers hold files matching the given query, 3) 15% of them holds matching files with high usefulness of 4 or more, and 4) the remaining peers hold matching files with usefulness of 3 or less. As a result, the coefficient of correlation between two metrics becomes 0.77.

As shown in the figure, such a strong correlation gains the effect of improving the performance of the proposed scheme. In Limit, the usefulness of files increases by 22%, while the number of transmitted messages increases by 48% and the number of rounds increases by 37% compared with the previous scheme. In Simple and Intention, the usefulness of files increases by 10%, while the number of transmitted messages increases by 24% and the number of rounds increases by 22% compared with the previous one.

5 Concluding Remarks

This paper proposed a distributed scheme to find useful files in unstructured P2Ps. The proposed scheme is a variant of k-random walk with cyclic synchronization, and it applies a popularity-biased k-random walk to accelerate the search process.

Our future work is as follows: 1) We should reduce the cost of the proposed scheme (i.e., the number of messages and the number of rounds) since it is greater than Zhong and Shen's original scheme which does not consider the usefulness of discovered files; 2) We should combine the proposed scheme with an appropriate reputation system since a part of usefulness should be imported from such reputation systems; 3) We need to examine the effect of network topology to the performance; and 4) We have to conduct extensive simulations including cases with several metrics combined with different evaluation functions.

References

[1] Adamic, L.A., Lukose, R.M., Puniyani, A.R. and Huberman, B.A. Search in Power-Law Networks. *PHYSICAL REVIEW-SERIES E*, Vol. 64, 046135, 2001.

[2] Aspnes, J. and Shah, G. Skip Graphs. *Proc. the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.384-393, 2003.

[3] Clip, DSS. Gnutella protocol specification v0. 4. http://dss.clip2.com/GnutellaProtocol04.pdf

[4] Cohen, E. and Shenker, S. Replication Strategies in Unstructured Peer-to-Peer Networks. *ACM SIGCOMM Computer Communication Review*, Vol. 32, Issue 4, pp.177-190, 2002.

[5] Cooper, B.F. Quickly Routing Searches without Having to Move Content. *Lecture Notes in Computer Science*, Vol. 3640, pp.163-172, 2005.

[6] Damiani, E., Paraboschi, S., Samarati, P. and Violante, F. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. *Proc. 9th ACM Conference on Computer and Communications Security*, pp.207-216, 2002.

[7] Gkantsidis, C., Mihail, M. and Saberi, A. Random Walks in Peer-to-Peer Networks. *Proc. INFOCOM 2004*, pp.241-263, 2004.

[8] Gupta, M., Judge, P. and Ammar, M. A Reputation System for Peer-to-Peer Networks. *Proc. 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp.144-152, 2003.

[9] Lv, Q., Cao, P., Cohen, E., Li, K. and Shenker, S. Search and Replication in Unstructured Peer-to-Peer Networks. *Proc. 16th International Conference on Supercomputing*, pp.84-95, 2002.

[10] Maymounkov, P. and Mazieres, D. Kademlia: A Peer-to-Peer Information System based on the XOR Metric. *Proc. International Workshop on Peer-to-Peer Systems (IPTPS)*, pp.53-65, 2002.

[11] Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Schenker, S. A Scalable Content-Addressable Network. *Proc. 2001 SIGCOMM*, pp.161-172, 2001.

[12] Ripeanu, M. and Foster, I. Peer-to-Peer Architecture Case Study: Gnutella Network. *Proc. International Conference on Peer-to-peer Computing*, p.99, 2001.

[13] Rowstron, A. and Druschel, P. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp.329-350, 2001.

[14] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *Proc. 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp.149-160, 2001.

[15] Zhao, B.Y., Kubiatowicz, J. and Joseph, A.D. Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing. *Technical Report UCB/CSD-01-1141*, University of California at Berkeley, Computer Science Department, 2001.

[16] Zhong, M. and Shen, K. Popularity-Biased Random Walks for Peer-to-Peer Search under the Square-Root Principle. *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, 2006.

不完全情報渋滞ゲームと近似ナッシュダイナミクス

山田陽介 * 小野廣隆 ** 山下雅史 **
(九州大学大学院 *システム情報科学府 **システム情報科学研究院)

1 はじめに

Chien と Sinclair は, 非協力対称渋滞ゲームにおける分散的なローカルダイナミクスによって, プレイヤー数 n, プレイヤーの遷移への動機を制限する係数 α, 辺の遅延関数の増加を制限する係数 ε の多項式回数以下の遷移ステップ数で近似的ナッシュ均衡状態を実現できることを示した [1]. 本論文は, その結果を拡張し, 分散的なコスト計算において個々のプレイヤーが他のプレイヤーの戦略についてある種の不完全な情報しか持たない場合に, 同様の遷移回数によって必ず遷移が収束するための十分条件について考察する.

2 渋滞ゲーム

2.1 渋滞ゲームとナッシュダイナミクス

プレイヤーの有限集合を $P = \{1, \ldots, n\}$, P によって共有されている資源の集合を $E = \{e_1, \ldots, e_m\}$ とする. プレイヤーの戦略は E のある部分集合であり, 各プレイヤー i が取る戦略 s_i の組 $s = (s_1, \ldots, s_n)$ を状態という. 状態 s において資源 $e \in E$ を利用する戦略を取るプレイヤーの人数を $f_s(e) = |\{i : e \in s_i, 1 \leq i \leq n\}|$ とする. 資源 e の利用コスト d_e は e を利用するプレイヤーの人数を引数とする, 非負の非単調関数である. したがって, 状態 s におけるプレイヤー i の戦略 s_i にかかるコスト $c_i(s)$ は $c_i(s) = \sum_{e \in s_i} d_e(f_s(e))$ である. このようにして定まるゲームを渋滞ゲームと呼ぶ. プレイヤー i が取り得る戦略の集合 $S_i \subset 2^E$ は必ずしも同一ではない. S_i がすべて同一ならばゲームは対称であるという. 状態の集合を $S = \Pi_{i=1}^n S_i$ で表す.

状態を $s = (s_1, s_2, \cdots, s_n)$ とする. s におけるプレイヤー i の戦略 s_i を $r \in S_i$ に置き換えて作られる状態を $s(i, r)$ と表す. 任意のプレイヤー i と戦略 $r \in S_i$ に対して $c_i(s) \leq c_i(s(i, r))$ が成立するとき, 状態 s は純粋ナッシュ均衡であるという. s が純粋ナッシュ均衡であるならば, 各プレイヤーは（その他のプレイヤーが戦略を変更しない限り）戦略を変更することでコストを減少させることはできないという意味で均衡している. 渋滞ゲームには純粋ナッシュ均衡が存在することが知られている.

任意の状態 s を考える. あるプレイヤー i が戦略 s_i を r に変更することでコスト c_i を減少できるならば戦略を変更することで生ずる状態遷移 $\rightarrow \in S^2$ を考える. すなわち, $s \rightarrow s'$ であるのは, ある $i \in P$ と $r \in S_i$ が存在して, $s' = s(i, r)$ かつ $c_i(s) > c_i(s')$ が成立するとき, かつそのときに限る. $\rightarrow$ をナッシュ遷移, 状態の遷移 $s^0 \rightarrow s^1 \rightarrow \cdots$ をナッシュダイナミクスと呼ぶ. 任意のナッシュダイナミクス $s^0 \rightarrow s^1 \rightarrow \cdots$ は有限である, すなわち, ある純粋ナッシュ均衡 s^k に到達することが知られているが, 一方, 純粋ナッシュ均衡を求める問題は, 渋滞ゲームが対称, すなわち $S_i = S_j (1 \leq i < j \leq n)$ の場合でも PLS-完全であることが知られている.

2.2 ϵ-ナッシュ均衡と ϵ-ナッシュダイナミクス

純粋ナッシュ均衡を求める問題の困難さを緩和するために以下の近似ナッシュ均衡を考える. ある定数を $\epsilon > 0$ とする. 任意のプレイヤー i と戦略 $r \in S_i$ に対して $(1 - \epsilon)c_i(s) \leq c_i(s(i, r))$ が成立するとき, 状態 s は ϵ-ナッシュ均衡であるという.

状態 $s \in S$, プレイヤー $i \in P$, 戦略 $r \in S_i$ に対して $s' = s(i, r)$ であるとき, 改善比を $\rho_s(i, r) = \frac{c_i(s) - c_i(s')}{c_i(s)}$ と定義する. あるプレイヤー i が戦略 s_i を r に変更することで改善比 $\rho_s(i, r) > \epsilon$ を達成できるときのみ状態遷移を許すナッシュダイナミクスを ϵ-ナッシュダイナミクスと呼ぶ. すなわち, $s \rightarrow_\epsilon s'$ であるのは, ある $i \in P$ と $r \in S_i$ が存在して, $s' = s(i, r)$ かつ $\rho_s(i, r) > \epsilon$ が成立するとき, かつそのときに限る.

資源コスト関数を d_e とする. ある定数 $\alpha \geq 1$ が存在し, すべての $t \geq 1$ に対して $d_e(t + 1) \leq \alpha d_e(t)$ が満たされるとき, d_e は α-跳躍であるという. 渋滞ゲームが対称的で, すべての資源コストがある α に対して α-跳躍であるならば, 任意の ϵ-ナッシュダイナミクスは有限であり, ϵ-ナッシュ均衡に到達するばかりでなく, ϵ-ナッシュ均衡に到達するまでの遷移回数は $\lceil n\alpha\epsilon^{-1} \log(nC) \rceil$ で上から抑えられることが知られている [1]. ここで, C はあるプレイヤーのコストの上限である.

3 不完全情報渋滞ゲームと近似ナッシュダイナミクス

3.1 不完全情報渋滞ゲーム

あるプレイヤー j が別のあるプレイヤー i の戦略を知ることができないような状況を考える. プレイヤー i の戦略をプレイヤー j が知ることができるとき, 有向辺 (i, j) を定義することによって構成される有向グラフ $G = (P, A)$ を可視グラフと呼ぶ. プレイヤー j が戦略を知ることができるプレイヤーの集合を $N^-[j] = \{i : (i, j) \in A\} \cup \{j\}$ とする. 通常の渋滞ゲームでは $N^-[j] = P$ が任意の $j \in P$ に対して成立していた. 不完全情報渋滞ゲームでは, プレイヤー j にとって, s_i は $i \in N^-[j]$ であるときに限り既知である. そこで, $i \notin N^-[j]$ であるようなすべてのプレイヤー i について s_i を $*$ に置き換えて s からできるベクトルを $v_j(s)$ と書く. ここで, $*$ は対応する状態が未知であることを示す記号である. 任意の $i \in P$ に対して $S_i^* = S_i \cup \{*\}$, $S^* = \Pi_{i=1}^n S_i^*$ とする. 関数 $\phi_i : S^* \rightarrow S$ を (プレイヤー i の) 仮説関数と呼ぶ. 仮説関数は s^* の $*$ である要素のそれぞれに対してある戦略を代入したベクトルを返す関数であり, $v_i(s)$ を元に何らかの仮説, すなわち $*$ を含まないある状態 $s' \in S$ を想定するために用いる. 具体的には, 仮説関数 ϕ に引数 $s^* \in S^*$ を与えたとき, $s_i = s_j = \cdots = s_m = *$ であるプレイヤー $i, j, \cdots, m$ の戦略が $s'_i, s'_j, \cdots, s'_m \in S_i$ に置き換えられた状態 $\phi(s^*)$ を得たとする. このとき, $\phi(s^*)$ を ϕ による仮説と定義し, $\phi(s^*) = s^*(i, s'_i; j, s'_j; \cdots; m, s'_m)$ と表記する. また, $*$ を含む状態 s^* から任意の仮説関数によって作られ得る仮説の集合を $\Delta(s^*)$ と定義する.

まず, 簡単のため, ゲームは対称で, 全プレイヤーは同一の仮説関数を用いるものとする.

3.2 不完全情報渋滞ゲームの近似ナッシュダイナミクス

可視グラフ G に加えて, 各プレイヤー i に対する仮説関数 $\phi = (\phi_1, \phi_2, \cdots, \phi_n)$ も与えられているとする. このと

き, プレイヤー i の s に関する既知の情報 $v_i(s)$ から仮説関数 ϕ_i を用いて復元した状態 $t = \phi_i(v_i(s))$ が正しいと仮定したときに, プレイヤー i の戦略を r に変更したときの改善比は $\rho_t(i,r)$ である.

ある定数を $\epsilon > 0$ とする. あるプレイヤー i が戦略 s_i を r に変更することで改善比 $\rho_t(i,r) > \epsilon$ を達成できるときのみ状態遷移を許す ϵ-ナッシュダイナミクスを不完全情報下での ϵ-ナッシュダイナミクスと呼ぶ.

以降の議論を具体的にするために, 特に断りがない限り, 遷移は $\mathrm{argmax}_i \max_{r \in S_i} \rho_t(i,r)$ を満たすプレイヤーが $\mathrm{argmax}_{r \in S_i} \rho_t(i,r)$ を満たす戦略に遷移することによってなされるものとする.

我々の目的は, 任意の ϵ-ナッシュダイナミクスが有限, すなわち ϵ-ナッシュ均衡に到達するための十分条件を検討することである.

4 悲観的な仮説関数

悲観的な仮説関数によって想定された状態とは, プレイヤーによって実現され得る最大の改善比が, 最小となる状態であるとする. 具体的には, 状態 s^* におけるプレイヤー i に対する悲観的な状態を次式で定義する.

$$\phi_{p,i}(s^*) = \mathrm{argmin}_{t \in \Delta(s^*)} \max_{r \in S_i} \rho_t(i,r)$$

4.1 多項式時間で収束できる不完全情報ゲームにおける ϵ-ナッシュダイナミクス

定理 1. α-跳躍の条件を満たす同一の遅延関数 d を持つ辺 e_1, e_2 が存在し, プレイヤーの取り得る戦略が $S_i = \{\{e_1\}, \{e_2\}\}$ であり, 可視グラフ G の補グラフが長さ n のサイクルとなっている不完全情報 n 人対称渋滞ゲームにおいて, プレイヤーが悲観的な仮説関数によって状態を推定する場合, ϵ-ナッシュダイナミクスは任意の初期状態から, $\lceil n\alpha\epsilon^{-1}\log(nC)\rceil$ ステップ以下で収束する. ただし, C は全てのプレイヤーのコストの上界である.

これを示すために, 次の補題を用いる.

補題 2. α-跳躍の条件を満たす同一の遅延関数 d を持つ辺 e_1, e_2 が存在し, プレイヤーの取り得る戦略が $S_i = \{\{e_1\}, \{e_2\}\}$ であり, 可視グラフ G の補グラフが長さ n のサイクルとなっている不完全情報 n 人対称渋滞ゲームで, プレイヤーが悲観的な仮説関数によって状態を推定するものとする. このとき, ϵ-ナッシュダイナミクスによってプレイヤー i による遷移が生じたならば, 任意のプレイヤー j について, $c_j(s) \leq \alpha c_i(s)$ が成立する.

補題 2 の証明. ϵ-ナッシュダイナミクスにより, プレイヤー i による遷移が生じたならば, 任意のプレイヤー j のコストの改善比は多くともプレイヤー i のコストの改善比と等しくなる. これは, 可視グラフ G の補グラフが長さ n のサイクルとなっているので, いかなる場合においても少なくとも 1 人は, 実際のコストの改善比と, 悲観的な仮説関数に基づいて想定したコストの改善比が共に最大となるプレイヤーが存在し, それらのプレイヤーが ϵ-ナッシュダイナミクスによって遷移するプレイヤーの候補となるので, 遷移したプレイヤー i のコストの改善比は他のプレイヤーのそれと等しいかあるいは大きくなるためである. また, 仮にプレイヤー i とは異なるプレイヤー j が, 状態 s' に遷移した後のプレイヤー i の戦略と同じ戦略を取ることにより, 状態が s から s'' に遷移していたと仮定する. このとき, 遷移後のプレイヤー i の戦略に含まれる辺について, プレイヤー i およびプレイヤー j が遷移後に負うコストを比較することにより, $c_j(s'') \leq \alpha c_i(s')$ が得られる. 以上より, $\frac{c_j(s)-\alpha c_i(s')}{c_j(s)} \leq \frac{c_i(s)-c_i(s')}{c_i(s)}$ が成立する. したがって, $c_j(s) \leq \alpha c_i(s)$ が成立する. □

定理 1 の証明. 補題 2 より, あるプレイヤー i による遷移が生じたなら, プレイヤー i のコストは他のプレイヤーのコストより $\frac{1}{\alpha}$ 倍以上大きいといえる. ところで, 渋滞ゲームは正確なポテンシャル関数を持つ. ポテンシャル関数を $\gamma(s) = \sum_{e \in E} \sum_{t=1}^{f_s(e)} d_e(t)$ と定義すると, 任意の状態 s において, $\gamma(s) \leq \sum_j c_j(s)$ が成立しているので, $c_i(s) \geq \frac{1}{\alpha n}\gamma(s)$ が成り立つ. プレイヤー i が戦略を変更し, 状態が s から s' に変更されたとき, ポテンシャル関数の減少量は $\gamma(s) - \gamma(s') = c_i(s) - c_i(s') > \epsilon c_i(s) \geq \frac{\epsilon}{\alpha n}\gamma(s)$ となる. これより, 遷移が生じるたびに, ポテンシャルには元の γ の $\frac{\epsilon}{\alpha n}$ 倍以上の減少が生じることになる. ここで, γ は非負の整数値を取るので, γ_{max} をポテンシャルの初期値とすれば, 可能な遷移回数は多くとも $\lceil n\alpha\epsilon^{-1}\log(\gamma_{max})\rceil$ 以下となる. さらに, $\gamma_{max} \leq nC$ が成立しているので, 定理 1 は成立する. □

5 まとめ

個々のプレイヤーが対象渋滞ゲームの状態について不完全な情報しか持たない場合においても, 各プレイヤーが悲観的な仮説関数を用いてゲームの状態を推定することによって, ある種のゲームの近似的ナッシュダイナミクスは常に多項式時間で収束するということを示した. より一般的な場合における近似ナッシュダイナミクスの収束性や, 不完全情報ゲームにおいて近似ナッシュダイナミクスが収束するための必要条件ならびに十分条件を明らかにすることは, 今後の課題である.

参考文献

[1] S. Chien and A. Sinclair: "Convergence to approximate Nash equilibria in congestion games", *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 169-178, 2007.

Publish/Subscribe システムにおけるオーバーレイネットワーク構成の最適化について

細田 淳† 泉 泰介†† 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

† jun@phaser.elcom.nitech.ac.jp †† {t-izumi,wada}@nitech.ac.jp

1. ま え が き

Publish/Subscribe(Pub/Sub) システム [5] では複数の送信者 (Publisher) から送信されるメッセージを受信することを希望する受信者 (Subscriber) に配送するシステムである. 本研究では Pub/sub システムの一種である Topic-based Pub/Sub システムを扱う. Topic-based Pub/Sub システムでは, トピックと呼ばれる固有の識別子をもつチャンネルを通して, ユーザはメッセージの送受信をする. 受信者は利用するトピックを選択することができ, 選択されたトピックを通して送信されるメッセージが配送される. Pub/Sub システムはインターネットアプリケーションで広く利用されている. 例えば, 株式市場監視エンジン [7], RSS フィード [4], オンラインゲームなどで利用されている.

既存研究として分散型の Topic-based Pub/Sub システムの通信をサポートするオーバーレイネットワークの設計問題の研究 [2], [3] がある. 送信されるメッセージのオーバヘッドや接続のコストをできる限り小さくするオーバーレイネットワークの構成方法について研究され, [2] では最適化問題 Minimum Topic-Connected Overlay(Min-TCO) 問題が提案されている. Min-TCO 問題は, 完全グラフ G と各ノードが利用するトピック集合の割り当てが与えら, 各トピックで, 同じトピックを利用するノードだけで誘導される部分グラフが多くとも 1 つの連結成分である辺数最小の G の部分グラフを求める問題である. Min-TCO 問題は NP 完全であることが示されている. 多項式時間近似アルゴリズムが存在し, この多項式時間アルゴリズムの近似率は大変タイトであることが示されている.

Min-TCO 問題では完全グラフを扱っている. そこで本研究では一般グラフの場合について考え一般グラフ Min-TCO 問題を提案した. 一般グラフの場合は, 同じトピックを利用するノードだけで必ずしも 1 つの連結成分を構成できるとは限らない. そこで構成されるオーバーレイネットワークの条件を同じトピックを利用する任意の 2 つのノードは連結であるオーバーレイネットワークに変更した. 一般グラフ Min-TCO 問題はトピックが 1 種類であるとき, シュタイナー木問題である. よってトピックが 2 種類以上であるときシュタイナー木問題の一般化である. また, 一般グラフ Min-TCO 問題は最小サブモジュラー集合被服 (SSC) 問題 [1] に帰着することで, 近似率 $O(log|V||T|)$(V,T はそれぞれノード集合とトピック集合) の多項式時間アルゴリズムが存在することを証明した [6]. さらに, 一般グラフ Min-TCO 問題はシュタイナー木問題の一般化であるが, シュタイナー木問題のアルゴリズムを単純には利用できないことを考察した.

2. 最小トピック連結オーバーレイ (Min-TCO) 問題

2.1 定 義

この章では TCO 問題を紹介する. ノード集合 V とトピック集合 T が与えられたとき、$V \times T$ を入力とするブール関数の興味関数 Int が定義される. $Int(v,t) = true$ のとき, ノード v がトピック t に興味を持つ. また逆も成り立つ.

ノード集合 V のオーバーレイネットワークは $W = V$,$E \subseteqq V \times V$ であるグラフ (W,E) である. V のオーバーレイネットワーク G, $V \times T$ を入力とする興味関数 Int, トピック t が与えられたとき、ノード集合 $\{v \in V | Int(v,t)\}$ によって誘導される G の部分グラフをトピック連結成分 (topic-connected components) と言う. 各トピック t で,G のトピック連結成分が最大で 1 つであるとき, G をトピック連結 (topic-connected) と言う.

[Min-TCO 問題]

[入力 1] ノード集合 V の完全グラフ G, トピック集合 T,$V \times T$ を入力とする興味関数 Int

[出力 1] トピック連結を満たす辺数最小の G の部分グラフ

[定理 1] Min-TCO 問題には近似率 $O(\sum_{v \in V}\{t \in T | Int(v,t)\})$ の多項式時間アルゴリズムが存在する.

3. 一般グラフ Min-TCO 問題

Min-TCO 問題では完全グラフを扱っている. そこで本研究では一般グラフの場合を考え, 新たな問題を提案する. 提案する問題では一般グラフを扱うのでオーバーレイネットワークがトピック連結を満たすことができるとは限らない. そこで, 構成されるオーバーレイネットワークの制限を変更した.

[一般グラフ Min-TCO 問題]

[入力 2] ノード集合 V の一般グラフ G, トピック集合 T,$V \times T$ を入力とする興味関数 Int

[出力 2] 各トピック $t \in T$ でノード集合 $\{v \in V | Int(v,t)\}$ の任意の 2 つのノードが連結である辺数最小の G の部分グラフ

$|T| = 1$ のとき, 一般グラフ Min-TCO 問題はシュタイナー木問題とみなすことができる. よって $|T| \geqq 2$ のとき, 一般グラフ Min-TCO 問題はシュタイナー木問題を一般的にした問題である.

証明に利用する最小サブモジュラー集合被覆 (SSC) 問題を導入する.

[定義 1] f は有限集合 N のすべての部分集合上で定義される実数値関数とする. このとき,$S \subseteqq T \subseteqq N$ ならば $f(S) \leqq f(T)$ であるとき,f は非減少であると言う. $S,T \subseteqq N$ で $f(S) + f(T) \geqq f(S \cap T) + f(S \cup T)$ であるとき, f はサブモジュラーであると言う.

[SSC 問題]

[入力 3] 有限集合 N, 要素 $j \in N$ の非負コスト c_j, 非減少サブモジュラー関数 $f : 2^N \rightarrow Z^+$

[出力 3] $min\{\sum_{j \in S} c_j : f(S) = f(T)\}$

$H(i)$ を i 番目の調和数とするとき,SSC 問題には近似率 $H(max_{j \in N} f(j))$ のアルゴリズムが存在することが示されている [1].

[定理 2] 一般グラフ Min-TCO 問題には近似率 $O(log|V||T|)$ の多項式時間アルゴリズムが存在する.

[証明] 一般グラフ Min-TCO 問題を SSC 問題に帰着することで証明する. 一般グラフ Min-TCO 問題で与えられた一般グラフ G の辺集合を N としたとき, 辺集合 $S \subseteqq N$ で構成されるオーバーレイネットワークを $G' = (V,S)$ とし, G' に含まれる 2 点が連結であるかどうかを表すグラフ $C(S)$ を以下のように定義する. G においてトピック t でノード集合 $\{v \in V | Int(v,t)\}$ のノード u とノード v が連結であるときかつそのときに限り, グラフ $C(S)$ でノード u とノード v は辺を持つ. トピック t で, ノード集合 $\{v \in V | Int(v,t)\}$ から誘導される $C(S)$ の部分グラフの連結成分数を $g_t(S)$ として, N の部分集合上の関数 $f(S)$

を $f(S) = \sum_{t \in T} (g_t(N) - g_t(S))$ と定義する.
［補題 1］ $f(S) = \sum_{t \in T} (g_t(N) - g_t(S))$ は非減少モジュラー関数である.
［証明］ 辺の追加によって, 各トピック t で, ノード集合 $\{v \in V | Int(v,t)\}$ から誘導される $C(S)$ の部分グラフの連結成分数が増加することはないので f は明らかに非減少関数である. 次に $f(S \cup \{e\}) + f(S \cup \{e'\}) \geqq f(S) + f(S \cup \{e, e'\})$ を示して f がモジュラー関数であることを示す. (集合 A,B,C が互いに disjoint であるとき, $f(A \cup B) + f(A \cup C) \geqq f(A \cup B \cup C) + f(A)$ は $f(A \cup \{b\}) + f(A \cup \{c\}) \geqq f(A \cup \{b, c\}) + f(A)$ から再帰的に求めることができる.)

ここで, 辺 e の追加によって各トピック t で減少する連結成分数の総和を x, 辺 e' の追加によって各トピック t で減少する連結成分数の総和を y とする.

各トピック t,$C(S)$ で,1 つの辺の追加によって減少する連結成分数は最大でも 1 である. グラフ G' で辺の追加によってノード $u \in V$ とノード $v \in V$ が接続されたとする. このとき,u と v は G' の 1 つの連結成分を構成するノードであるか, 2 つの異なる連結成分を構成するノードであるかのいずれかである. 前者の場合, 各トピック t,$C(S)$ で,u と v は連結であるので連結トピック数の総和は減少しない. 後者の場合, 各トピック t,$C(S)$ で,2 つの異なる連結成分のノードが接続するので連結性分数が最大で 1 減る. したがって, 各トピック t,$C(S)$ で,1 つの辺の追加によって減少する連結成分数は最大でも 1 である.

辺 e, e' の追加によって各トピック t で減少する連結成分数の総和は最大でも $x + y$ である. これは, 各トピック t,$C(S)$ で,1 つの辺の追加によって減少する連結性分数が最大でも 1 であることから明らかである. したがって $f(S) + f(S \cup \{e, e'\} \leqq 2\{\sum_{t \in T} (g_t(N) - g_t(S))\} + (x + y) = f(S \cup \{e\}) + f(S \cup \{e'\})$ となり f は非減少モジュラー関数である. □

一般グラフ Min-TCO 問題でノード集合 V の一般グラフ G, トピック集合 T, $V \times T$ を入力とする興味関数 Int が与えられたとき, SSC 問題の有限集合 N を G の辺集合, $j \in N$ のコストを 1, 非減少サブモジュラー関数を f と帰着する. $S \subseteqq N$ とすると,$f(N) = f(S)$ となるとき, 各トピック $t \in T$ でノード集合 $\{v \in V | Int(v,t)\}$ の任意の 2 つのノードは連結である. SSC 問題の S のコストは, オーバーレイネットワークの辺数となる. また,$max_{e \in E} f(e) \leqq |V||T|$ である.

以上より一般グラフ TCO には近似率 $O(log|V||T|)$ の多項式時間アルゴリズムが存在する. □

3.1 考　　察

一般グラフ Min-TCO 問題はシュタイナー木問題を一般化した問題であるので, シュタイナー木を構成するアルゴリズムを利用しようと考えるのは自然な流れである. そこで 2 つのゴリリズムについて考察する. どちらのアルゴリズムも一見うまくいくように思えるが常に最適解を導くことはできない.

1 つ目は, 与えられたノードにたいして最小木を生成してオーバーレイネットワークを構成するアルゴリズムである. 2 つ目は, 各トピックで t で, ノード集合 $\{v \in V | Int(v,t)\}$ をターミナルとする最小シュタイナー木を生成してそれらを結合してオーバレイネットワークを構成するアルゴリズムである.

同じ集合に含まれるノードは他ノードと少なくとも 1 つの共通したトピックに興味を持つように分類する. 2 つの集合に分類されたときを考える. このとき, それぞれの集合でシュタイナー木を生成して 2 つのシュタイナー木を 1 つの辺で接続したグラフが生成されたとする. このグラフは木であり,1 つ目のアルゴリズムで生成されるグラフの条件を満たしている. しかし,2 つのシュタイナー木を接続する辺は明らかに無駄であるので, このグラフは最適解ではない.

図 1 のグラフが与えられている場合を考える. ノード n_s, n_g はトピック t_1, t_2 に, ノード $n_1, \cdots, n_x$ はトピック t_1 に, ノード $n'_1, \cdots, n'_y$ はトピック t_1, t_2 以外のそれぞれ異なるトピックに興味を興味を持っているとする. また,$x > y$ であるとする. このとき,2 つ目のアルゴリズムではトピック t_1 でノード集合 $\{n_s, n'_1, \cdots n'_y, n_g\}$ から成るオーバーレイネットワークを, トピック t_2 でノード集合 $\{n_s, n_1, \cdots n_x, n_g\}$ から成るオーバーレイネットワークを構成する. これにより, 生成されるオーバーレイネットワークはノード集合 $\{n_s, n_1, \cdots, n_x, n_g, n'_1, \cdots, n'_y\}$ から成る. しかし, 最適解はノード集合 $\{n_s, n_1, \cdots, n_x, n_g\}$ から成るオーバーレイネットワークであり,2 つ目のアルゴリズムで生成されたオーバーレイネットワークは最適解ではない.

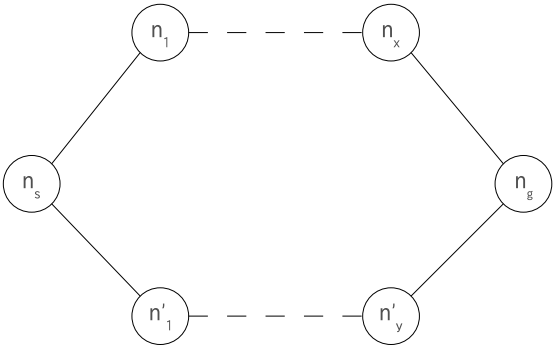


図 1　最適解を導けない例

4. ま と め

本稿では,Publish/Subscribe システムの最適なオーバーレイネットワーク構成のための一般グラフ Min-TCO 問題を提案した. 一般 Min-TCO 問題には近似率 $log(|V||T|)$ の多項式時間アルゴリズムが存在すること示した. また, 一般 Min-TCO 問題はシュタイナー木問題を一般化した問題であるが, シュタイナー木問題を解くアルゴリズムを単純には利用できないことを考察した.

今後の課題は, 一般 Min-TCO 問題を解くアルゴリズムの考案と問題の難しさを明かにすることである.

文　　献

[1] Toshihiro FUJITO. Approximation aogorithms for submodular set cover with applications. 2000.

[2] Yoav Tock Gregory Chockler, Roi Melamed and Roman Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. ACM, 2007.

[3] Yoav Tock Gregory Chockler, Roi Melamed and Roman Vitenberg. Spidercast:a scalable interest-aware overlay for topic-based pub/sub communicaion. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*. ACM, 2007.

[4] Venugopalan Ramasubramanian Hongzhou Liu and Emin Gun Sier. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *Internet Measurement Conference*.

[5] R.Guerraoui P.T.Eugaster, P.A.Felber and A.-M.Kermarrec. The many faces of publish/subscribe. In *ACM Computing Survey*, pp. 35(2):114–131. 2003.

[6] Hirotaka ONO Tomoko IZUMI, Taisuke IZUMI and Koichi WADA. Relationship betweeen approximability and request structures in the minimum certificate dispersal problem. 2009.

[7] A.Harpaz Y.Tock, N.Naaman and G.Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. 2005.

並列凸包計算アルゴリズムのマルチコアプロセッサ上での評価

中河 雅弥, 伊藤 靖朗, 中野 浩嗣
(広島大学大学院 工学研究科 情報工学専攻)

1 はじめに

共有メモリ型マルチコアプロセッサは各プロセッサがそれぞれローカルメモリを持つ分散メモリシステムと違い，各プロセッサコアが１つのメモリを共有しているため，並列分散処理の為にネットワークを経由するデータ通信を行う必要がない．そのため，より高速な並列処理を行うことが可能である．

本稿では幾何学の代表的な問題である凸包計算 [1] の並列アルゴリズムを共有メモリ型マルチコアプロセッサ上で実装し，評価する．凸包とは (x, y) 平面上に与えられた有限個の点集合に対して，それら全てを含む最小の凸多角形のことである．凸包の上辺のみを取り出したものを上部凸包，下辺のみを取り出したものを下部凸包と呼ぶ (図 1)．

なお本稿中では，n，p はそれぞれ点集合の要素数，使用するプロセッサコア数を示す．また，ここでは上部凸包を計算する場合についてのみ述べるが，同様の方法で下部凸包を求めて組み合わせることで容易に全体の凸包を構成することができる．

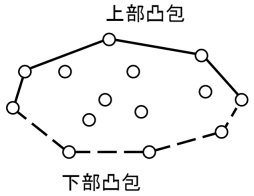


図 1　上部凸包，下部凸包

2 凸包の計算

x の昇順でソートされた点集合 $G = \{g_0, g_1, ..., g_n\}$ の上部凸包を求める線形なアルゴリズムについて述べる．

まず，スタック S を用意し，g_0，g_2 のインデックス 0，1 を push しておく．次に g_3 以降の各点を順に見ていき，スタックの２番目 s_2 にあるインデックスが示す点，スタックトップ s_1 にあるインデックスが示す点，注目している点 g_i が時計回りになる場合は i をスタックに push する (図 2)．３点が反時計回りになる場合は時計回りになるまで S から pop する (図 3)．これにより $O(n)$ 時間で凸包を求めることができる．

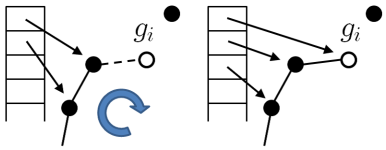


図 2　時計回りの場合 g_i を push する

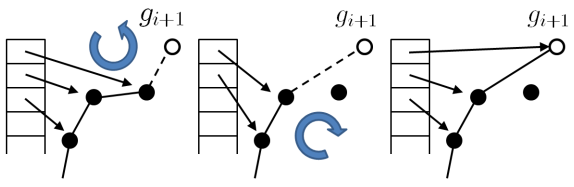


図 3　反時計回りの場合 pop する

3 共通接線の計算

２つの凸包の両方に接する線分を共通接線と呼ぶ (図 4)．上部凸包 L，R の共通接線 lr を求めるアルゴリズムを以下に示す．L，R の点の数はそれぞれ u，v 個とし，L が左側，R は右側の上部凸包とする．ここで，ある点が線分 lr より下にあるとは，l から r へ向かって伸びる直線の右側にその点が位置していることを示す．

(1) L の中間点 ($u/2$ 番目の点) を l の初期値として選択する．

(2) lr が l から R への接線となるような接点 r を以下のように二分探索によって求める．

(2-a) R の中間点 ($v/2$ 番目の点) を r の初期値として選択する．

(2-b) r の隣接点を調べ，右隣の点が lr より下になければ求める接点は右側にあるので r を右へ移動する．同様に，左隣の点が下になければ求める接点は左側にあるので r を左へ移動する．移動方法は二分探索に従う．

(2-c) r の両隣の点が lr より下になるまで (2-b) を繰り返す．

(3) (2-b) と同様に l の左右どちらかの隣接点が lr より下になければ二部探索に従って l を移動する．

(4) l の両隣の点が lr より下になるまで (2)，(3) を繰り返す．

以上の処理の終了後，l と r の隣接点は lr より下にあるので，明らかに lr は L，R の共通接線である．(2) の計算は二分探索であるため計算時間は $O(\log v)$ であり，(2) は最悪 $O(\log u)$ 回繰り返されるため，全体の計算時間は $O(\log u \cdot \log v)$ となる．

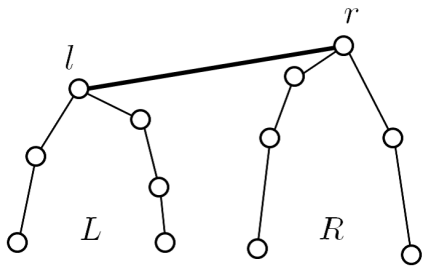


図 4　上部凸包 L,R の共通接線 lr

4 並列凸包アルゴリズム

使用する並列凸包アルゴリズムは分割統治を使用するもので，３つの Step で構成されている．

Step 1　ソートされた点集合 G を p 個に分割して部分集合 $S_i (0 \leq i \leq p-1)$ として，対応するプロセッサコア P_i に割り当てて並列に上部凸包 T_i を計算する．

Step 2　並列に T_i の左右それぞれ最も上向きの共通接線を求める．

Step 3　各 T_i をマージして全体の上部凸包 U とする．

Step 1 ではプロセッサ P_i は S_i に前節の上部凸包計算アルゴリズムを実行して T_i を求める．各 S_i に含ま

表 1　100,000,000 個の点の凸包の計算時間

分布	プロセッサ	計算時間				高速化	上部凸包
	コア数	Step 1 [s]	Step 2 [s]	Step 3 [s]	Total [s]	倍率	点数
正方形内	1	4.089559			4.089559		31
	2	1.981760	0.000019	0.000005	1.981784	2.064	
	4	0.956251	0.000048	0.000009	0.956308	4.276	
	8	0.550238	0.000050	0.000006	0.550294	7.431	
円内	1	4.094669			4.094669		789
	2	1.989088	0.000030	0.000007	4.094669	2.059	
	4	0.964389	0.000060	0.000007	1.989125	4.246	
	8	0.555851	0.000073	0.000007	0.555931	7.345	
円周上	1	3.244920			3.244920		49999712
	2	1.496761	0.000035	0.153402	1.650198	1.966	
	4	0.722253	0.000086	0.097231	0.819570	3.959	
	8	0.597337	0.000137	0.092665	0.690139	4.702	

れる点は n/p 個あるので，計算時間は $O(n/p)$ である．

Step 2 では P_i は前節の共通接線計算アルゴリズムを用いて，自身で計算した T_i と，他のプロセッサ P_j が計算した $T_j(0 \le j \le p-1, i \ne j)$ との間の共通接線を計算する．その中で，T_i から左側，右側に伸びる共通接線の中でそれぞれ最も上向きなものを選択し，それらの T_i に含まれる側の接点を l_i，r_i とおく (図 5)．各プロセッサが $p-1$ 本の共通接線を計算しているので，計算時間は $p-1$ と共通接線の計算時間に依存する．各上部凸包 T_i は，最大で n/p 個の点を持っているので，最悪計算時間は $O(p(\log(n/p))^2)$ となる．

Step 3 では r_i が l_i より右側にあるとき，T_i から l_i，r_i とそれらの間にある点を選んで一つの配列に格納する．この時も各 P_i は T_i の点を並列に配列に格納する．l_i の左側，r_i の右側にある点は共通接線より下にあることになるため，U には含まれない．もし，r_i が l_i より左側にあるなら，T_i の点は全て U には含まれない．この Step の後，配列には点集合全体の上部凸包 U が入っている．各 T_i は，最大で n/p 個の点を持っているので，最悪計算時間は $O(n/p)$ となる．

3 つの Step の計算時間を合計すると，$O(n/p) + O(p(\log(n/p))^2) + O(n/p) = O((n/p) + p(\log(n/p))^2)$ となる．

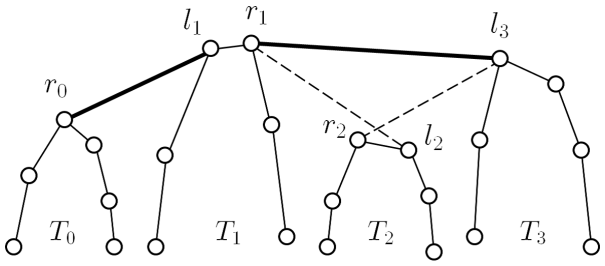


図 5　共通接線によって各 T_i をマージ

5　実験結果

実験ではクアッドコアプロセッサ Intel Xeon X5355 を 2 台搭載したコンピュータを使用した．つまり，合計 8 個のプロセッサコアを使用することができる．プログラムは C 言語で作成し，Linux CentOS 5.1 の OS 上で gcc 4.1.2 を用いてコンパイルを行った．また，共有メモリ型マルチプロセッサ環境における並列処理のために OpenMP[2] を使用した．この環境では複数のプロセッサがメモリ空間を共有しており，並列処理の際に各ローカルメモリ全てにデータをコピーする必要がないため，メモリ容量や通信時間を節約することができる．そのため，今回の凸包計算のような大容量のデータを扱う場合に適している．

$n = 100,000,000$ として，正方形内部，円内部，円周上の 3 通りの分布の点集合を用いた．それぞれの点の座標は各分布内でランダムに選んだ．正方形内部，円内部，円周上の順に凸包の点の数が多くなり，円周上では全ての点が凸包の点となる．使用するプロセッサコア数 p が 1，2，4，8 の場合でそれぞれ実験した．

計算時間は表 1 のようになった．コア数 1 の場合，Step 1 のみで凸包が計算できるので Step 2,3 は行われないため空欄になっている．

正方形，及び円形の分布についてはコア数 8 で約 7.4 倍と，ほぼプロセッサコア数に比例して高速化している．コア数 2，4 のときには高速化が約 2.4 倍，4.2 倍となり，スーパーリニアスピードアップが発生している．これは使用するコア数が増加したことで利用できるキャッシュメモリの領域が増加し，キャッシュ効率が良くなったことが原因だと考えられる．また，凸包の点の数が少ないため，Step 2,3 の計算時間は非常に小さくなっている．そのため，合計計算時間は Step 1 の計算時間とほぼ等しい．

円周上の分布を見ると，正方形，円形の場合と比べてコア数 1，2，4 では計算時間は短く，コア数 8 では長くなっており，並列化による高速化が小さい．コア数が少ないとき計算時間が短いのは，円周上の点は全てが凸包の点として選ばれるので，Step 1 でスタックから pop されることがないためだと考えられる．また，凸包の点が多いことにより，Step 2,3 の計算時間が正方形，円形の場合より大きくなっている．その結果，コア数 8 のときの合計計算時間は正方形，円形の場合よりもわずかに長くなっている．今回使用した環境ではコア数 8 が最大であるが，コア数が 9 以上になれば差はさらに大きくなっていくと考えられる．

本稿では並列凸包アルゴリズムの共有メモリ型マルチコアプロセッサ上での実装結果について述べた．今後はスーパーリニアスピードアップの原因についての仮説をキャッシュに関する実験を行って検証していく予定である．

参考文献

[1] S.G. Akl and K.A. Lyons. *Parallel Computational Geometry*. Prentice Hall, 1993.

[2] L. Dagum and R. Menon. *OpenMP: an industry standard API for shared-memory programming*. IEEE Computational Science and Engineering, Vol. 5, No. 1, pp. 46-55, 1998.

Parallel Sampling Sorting on the Multicore Processors

Duhu Man , Yasuaki Ito and Koji Nakano
Department of Information Engineering, Hiroshima University

1 Introduction

Sorting, a process of arranging elements of a list in a certain order, is one of the most important tasks in computer science. We can find its applications in many fields, such as database operations, image processing, statistical methodology and so on. Hence many sequential sorting algorithms have been studied in the past. Several parallel sorting algorithms such as parallel merge sort [2], bitonic sort [3] and parallel radix sort [4] have been devised. In this paper, we present a new sampling based parallel sorting algorithm for the multicore processors. The algorithm is based on a parallel k-merge algorithm on the PRAMs [1].

The idea behind our parallel sorting algorithm is simple. Samples are selected from an input sequence, and the range of the input sequence is determined by sorting samples. Threshold values are chosen from the sorted samples and threshold values are used to divide the input sequence into several buckets. We will obtain a sorted sequence by sorting each bucket. So, in order to achieve good loading balance for each bucket, we need to choose appropriate threshold values to divide the input sequence into equal-sized buckets.

2 Partition Keys using Samples

In this section, we will show that how to choose appropriate threshold values from the sorted samples.

Let $A = \langle a_0, a_1, ..., a_{n-1}\rangle$ be a sequence of keys stored in a memory to be sorted. We partition A into p blocks B_i $(0 \leq i \leq p-1)$ of the same size such that $B_i = \langle a_{i\cdot\frac{n}{p}}, a_{i\cdot\frac{n}{p}+1}, ..., a_{(i+1)\cdot\frac{n}{p}-1}\rangle$. Suppose that each block B_i $(0 \leq i \leq p-1)$ is sorted independently, and $B_i = \langle b_{i,0}, b_{i,1}, ..., b_{i,\frac{n}{p}-1}\rangle$ denotes the sorted sequence thus obtained. For the arbitrary integer $k > 0$, we further partition each sorted block B_i $(0 \leq i \leq p-1)$ into pk sub-blocks $B_{i,0}, B_{i,1}, ..., B_{i,pk-1}$ such that $B_{i,j} = \langle b_{i,j\cdot\frac{n}{p^2k}}, b_{i,j\cdot\frac{n}{p^2k}+1}, ..., b_{i,(j+1)\cdot\frac{n}{p^2k}-1}\rangle$. Clearly, each $B_{i,j}$ has $\frac{n}{p^2k}$ keys. Let C_i denotes the sequence of keys obtained by picking the minimum key from each of the sub blocks $B_{i,0}, B_{i,1}, ..., B_{i,pk-1}$. In other words, $C_i = \langle b_{i,0\cdot\frac{n}{p^2k}}, b_{i,1\cdot\frac{n}{p^2k}}, ..., b_{i,(pk-1)\cdot\frac{n}{p^2k}}\rangle$. Let C denotes the combined sequence of $C_0, C_1, ..., C_{p-1}$. Since each C_i has pk, C has p^2k keys. Let $\langle c_0, c_1, ..., c_{p^2k-1}\rangle$ denote the sorted sequence of C. In other words, $c_0 < c_1 < ... < c_{p^2k-1}$ holds. We pick every pk keys from sorted C. Let $D = \langle d_0, d_1, ..., d_{p-1}\rangle$ be the sequence thus obtained. In other words, $d_i = c_{i\cdot pk}$ $(0 \leq i \leq p-1)$ holds. We use keys in D as threshold values to partition elements in A. Let A_i $(0 \leq i \leq p-1)$ denotes a set of values such that $A_i = \{x \in A | d_i \leq x < d_{i+1}\}$, where $d_p = +\infty$. By sorting keys in each A_i independently, we can obtain the sorted sequence of A.

Now we can prove that the number of keys in A_i are well balanced as follows. Let $D_i = \{x \in C | d_i \leq x \leq d_{i+1}\} = \{c_{i\cdot pk}, c_{i\cdot pk+1}, ..., c_{(i+1)\cdot pk-1}\}$. Clearly, each D_i has pk keys. Further, let $D_{i,j} = B_i \cap D_j$ and $A_{i,j} = B_i \cap A_j$. In general, we have

$$| A_{i,j} | \quad \leq \quad (| D_{i,j} | +1)\frac{n}{p^2k} - 1 \qquad (1)$$

Thus, we can compute the upper bound of the number of keys in A_j as follows:

$$\begin{aligned} | A_j | &= \sum_{i=0}^{p-1} | A_{i,j} | \\ &\leq \sum_{i=0}^{p-1}((| D_{i,j} | +1)\frac{n}{p^2k} - 1) \\ &= (pk+p)\frac{n}{p^2k} - p \\ &= \frac{n}{p} + \frac{n}{pk} - p \end{aligned} \qquad (2)$$

Thus, we have the following important lemma.

Lemma 1 Each A_j $(0 \leq i \leq p-1)$ has no more than $\frac{n}{p} + \frac{n}{pk} - p$ keys.

Note that, if A is equally partitioned into p groups, each of them has $\frac{n}{p}$ keys. It follows that, A_j may have at most $\frac{n}{pk} - p$ additional keys, and the number of additional keys decreases as k increases.

3 Parallel Sampling Sorting

We assume that input n keys are stored in array A. The outline of our sorting algorithm for p processors is as follows:

Step 1 Partition A into p groups and let each processor sorts each corresponding group. Each processor selects pk (k is an arbitrary integer) samples from corresponding group.

Step 2 Select p threshold values $d_0, d_1, ..., d_{p-1}$ from sorted samples. Partition A into p new groups $A_0, A_1, ..., A_{p-1}$ using threshold values such that $A_i = \{x \in A | d_i \leq x < d_{i+1}\}$, where $d_p = +\infty$.

Step 3 Sort keys in each group A_i using one processor per group independently.

Let $P(i)(0 \leq i \leq p-1)$ denote a processor i. We assume that the parallel sorting algorithm stores the sorted n keys in array R. The details of the parallel algorithm are spelled out as follows:

Step 1.1 Partition A into p groups $B_0, B_1, ..., B_{p-1}$ and sort each group B_i $(0 \leq i \leq p-1)$ using $P(i)$. This can be done in expected $O(\frac{n}{p}\log\frac{n}{p})$ time using the quick sort.

n	p	Step 1 [s]	Step 2 [s]	Step 3 [s]	Total time [s]	Speed-up
10,000,000	1	–	–	–	2.229482	1.0
	2	1.07845	0.000026	0.074748	1.153226	1.93
	4	0.51858	0.000046	0.052057	0.570686	3.90
	8	0.28592	0.000122	0.038356	0.324406	6.87
100,000,000	1	–	–	–	25.56230	1.0
	2	12.3892	0.000059	0.750054	13.13940	1.94
	4	6.12791	0.000151	0.528196	6.656263	3.84
	8	3.43506	0.000513	0.357294	3.792863	6.73

Table 1: Performance of parallel sorting random 32-bit unsigned integers

Step 1.2 We use an array of size p^2k to store C. Each $P(i)$ picks every $\frac{n}{p^2k}$ keys in B_i and copy them to the array for C in obvious way. It takes $O(pk)$ time.

Step 2.1 Sort keys in C using parallel sampling sorting algorithm recursively. Since keys in each $C_0, C_1, ..., C_{p-1}$ are sorted , so we just select p local samples from each C_i and sort this p^2 samples using $P(0)$. After that, we recursively execute the parallel sampling sorting algorithm from Step 2.2 . This will takes $O(p\log pk + pk\log p + p^2\log p)$ time.

Step 2.2 Processor $P(0)$ picks every pk keys in C as threshold values $d_0, d_1, ..., d_{p-1}$. It takes $O(p)$ time.

Step 2.3 Let $s_{i,j}$ $(0 \leq i, j \leq p-1)$ be the minimum index of a key in B_i satisfying $b_{i,s_{i,j}} \geq d_j$. Clearly, $A_{i,j} = \{b_{i,s_{i,j}}, b_{i,s_{i,j}+1}, ..., b_{i,s_{i,j+1}-1},\}$ holds, where $s_{i,p} = \frac{n}{p}$. Each $P(i)$ $(0 \leq i \leq p-1)$ computes the values of $s_{i,0}, s_{i,1}, ..., s_{i,p-1}$ using the obvious binary search. It can be done in $O(p\log\frac{n}{p})$ time.

Step 2.4 Clearly, $A_{i,j}$ has $s_{i,j+1} - s_{i,j}$ keys. Each $P(j)$ $(0 \leq j \leq p-1)$ computes $|A_{0,j}| + |A_{1,j}| + ... + |A_{p-1,j}|$, which is equal to $|A_j|$. After that, $P(0)$ computes the prefix sums $\alpha_j = |A_0| + |A_1| + ... + |A_j|$ for each j $(0 \leq j \leq p-1)$. This can be done in $O(p)$ time.

Step 3 Let R_j be a subset of array R such that R_j consists of $|A_j|$ elements from $\alpha_j - th$ elements of R. Each $P(j)$ $(0 \leq j \leq p-1)$ sort sub-array R_j independently. Note that, R_j consists of $A_{0,j}, A_{1,j}, ..., A_{p-1,j}$. Also, each $A_{i,j}$ is sorted. Hence, the sorting of R_j can be done by merging $A_{0,j}, A_{1,j}, ..., A_{p-1,j}$. According to Lemma 1, each R_j $(0 \leq i \leq p-1)$ has no more than $\frac{n}{p} + \frac{n}{pk} - p$ keys. So, this step can be done in expected $O(\frac{n\log p}{p} + \frac{n\log p}{pk})$ time.

From the algorithm, we have:

Theorem 2 Sorting of n keys can be done in $O(\frac{n}{p}\log\frac{n}{p} + p\log pk + pk\log p + \frac{n\log p}{p} + p^2\log p + p\log\frac{n}{p})$ time using p processors.

Note that, if $p \ll n$ and $k \ll n$, then the computing time is $O(\frac{n\log p}{p})$. Since the sequential sorting takes $O(n\log n)$ time, our algorithm achieves the speed up of factor p using p processors. Therefore, our parallel sorting algorithm is optimal.

4 Experimental Results

We have implemented our parallel sorting algorithm in a Linux server with two Intel quaed-core processors (i.e. eight processors cores) and evaluate the performance. The software has implemented in C language with OpenMP.

Table 1 shows the performance of our implementation when random 32-bit unsigned integers are sorted. The performance evaluation has been carried out for $k = 100$ (when the number of input keys is 10,000,000) , $k = 1000$ (when the number of input keys is 100,000,000) and different values of n and p (n represents the number of the input data and p represents the number of using processing cores). Note that in Step 1, if $p = 1$, the implementation performs only sequential quick sorting for the whole input data.

The experimental results show that our parallel sorting algorithm is 6.87 times faster than sequential sorting. Since the speed up factor cannot be more than 8 if we use 8 cores, our algorithm is close to optimal.

References

[1] Tatsuya Hayashi, Koji Nakano, and Stephan Olariu, *Work-Time Optimal k-merge Algorithms on the PRAM*, IEEE Trans. on Parallel and Distributed Systems, Vol. 9, No.3, pp. 275–282, Mar, 1998.

[2] M.Jeon and D.Kim, *Parallel Merge Sort with Load Balancing*, International Journal of Parallel Programming, pp.21-33, Vol.31, No.1, February,2003.

[3] M.F.Ionescu and K.E. Schauser, *Optimizing Parallel Bitonic Sort*, in Proceedings of the 11th International Symposium on Parallel Processing, pp.303-309, Geneva, Switzerland, April, 1997.

[4] A.Sohn and Yuetsu Kodama, *Load Balanced Parallel Radix Sort* , in Proceedings of the 12th ACM International Conference on Supercomputing, July, 1998.

A Dynamic User Management in Networked Consumer Electronics via Authentication Proxies

Kazuma Kadowaki* Satoshi Fujita
Hiroshima University, Japan†
email:{kadowaki, fujita}@se.hiroshima-u.ac.jp

Abstract

In this paper, we propose a user management system for networked consumer electronics (nCE). The proposed system extends Tokunaga's hybrid P2P system in such a way that the inquiries concerned with client peers submitted by service-provider peers are handled by several authentication proxies instead of a centralized management peer, although client peers still have to register themselves to the centralized server. A prototype of the proposed system is implemented using JXTA framework, and is evaluated via experiments. The result of experiments indicates that the proposed system certainly reduces both the load of the centralized server and the time required for the authentication process.

Keywords: Networked consumer electronics, user authentication, P2P, JXTA, multicasting.

1 Introduction

According to the recent advancement of information technologies, networked consumer electronics (nCE) have attracted considerable attentions in recent years as convenient tools that support us to enrich our daily life [1]. In nCE applications, several consumer devices such as digital audio-visual equipment, air conditioners, and lighting equipment are integrated with various kind of security equipment such as intercoms and electronic locks through a communication network, which enables a remote access to foregoing equipment and devices.

Currently, various kinds of communication standards are used in nCE, including Bluetooth, IEEE 802.11a, HSPA (High Speed Packet Access), HD-PLC (High Definition Power Line Communication), and others. Such a diverseness of standards makes it difficult to directly connect those devices in organizing an integrated network. Hence, in order to promote the use of nCE, we need to develop a common framework that enables anybody to easily construct and maintain an nCE application. In [2], we proposed a network architecture to support such a task of system designers in Peer-to-Peer (P2P) environments. The proposed architecture is based on JXTA framework [3,4,5], and we implemented a prototype system to demonstrate the effectiveness of our architecture in actual nCE applications.

A key issue in realizing nCE applications over a P2P network is how to provide such network services to users in a secure and safe fashion, since the lack of centralized control in P2P networks generally increases the risk of vulnerability against attacks and the problem of free riders. The architecture proposed in our previous paper is a hybrid P2P, in which an authentication and access control over the network are conducted in a centralized manner, whereas actual services (*i.e.*, communication between service-providers and users) are conducted in a P2P manner. More concretely, it adopts a special peer called a *management peer* (MP) to manage information on all peers in a network to realize an efficient authentication of such peers. In addition, it prepares a caching mechanism in each peer to reduce the number of inquiries submitted by those peers to the MP.

Such a centralized approach certainly overcomes the security issue in a P2P network, and would work well if the number of participant peers is relatively small. However, it could not be scaled since such an authentication process heavily relies on the MP, although the load of the MP could be (slightly) reduced by adopting a caching mechanism. For example, in our prototype system reported in [2], a join of peer to a group of subscribers of a service (*i.e.*, a *peer group*), takes 4,894 ms, which is much longer than the time required for finding a demanded service in the network.

In this paper, we propose a new authentication scheme to reduce the load of the MP. The basic idea of the scheme is to partition the tasks of the MP into two sub-tasks, *i.e.*, response to clients and response to service-providers, and distribute the latter sub-task among several peers called *authentication proxies* (APs). We implemented the proposed scheme using JXTA framework, and conducted experiments to evaluate the impact of the load balancing to the overall performance. The result of experiments indicates that to adopt the notion of APs certainly reduces both of the load of the MP and the time required for an authentication process.

The remainder of this paper is organized as follows: Section 2 describes an overview of JXTA. Section 3 overviews related work. Section 4 describes the proposed scheme in detail. The result of experiments is given in Section 5. Finally, Section 6 concludes the paper with future problems.

2 JXTA

JXTA is a platform for P2P computing which provides a common set of open protocols and its reference implementations written in various languages. The protocols are realized in such a way that it is independent of both programming language and transport protocols, thus it suits the use in nCE applications.

*Corresponding author.
† Department of Information Engineering, Hiroshima University, Kagamiyama 1-4-1, Higashi-Hiroshima, Japan

A JXTA network is built on the top of existing network transport protocols such as TCP/IP, HTTP, Bluetooth, HSPA, etc., which may cross network boundaries such as firewalls and NATs. It allows peers to communicate with each other regardless of their locations by providing a unique address, called an ID, for all peers in the network.

A JXTA network consists of several basic components described below:

- A *peer* is any networked device that implements the core JXTA protocols. Each peer has a unique ID, and executes its operation in an independent and asynchronous manner.
- *Pipes* and *sockets* are communication channels that connect peers directly. They can be used to send any type of data including text, images, Java Objects, and others, in either point-to-point or multicast manner. They also support secure communications over TLS (Transport Layer Security).
- *Advertisements* are used to represent JXTA resources such as peers, pipes, and services. Each peer discovers resources by searching for their advertisements, and may cache any advertisements in its local storage.

3 Related Work

3.1 Overview

With respect to the security of P2P networks, several peer-group admission control architectures have been proposed in the literature [6,7,8]. In such architectures, a group of peers subscribing the same service, called a *peer group*, is organized. Each client peer willing to subscribe a service first joins the corresponding peer group to connect with a service-providing peer. A service is provided to a client peer if the identity and the authority of the client peer are confirmed either by using some kind of threshold cryptosystems [9] or by being confirmed by a special peer that stores the authority list.

Although such an authentication mechanism is provided in JXTA, it is based on a simple substitution cipher, which is highly insecure. Additionally, in the original JXTA, account information is shared among peers by "handing over" the management information, which causes a serious vulnerability of the overall application system.

3.2 Tran System

Tran *et al*. [10] proposed a hybrid P2P architecture to support a secure user authority management in JXTA networks. Figure 1 illustrates an overview of their system (in the following, we call it "Tran system"). In order to overcome the vulnerability of the original JXTA, Tran system introduces a centralized peer, called a *management peer* (MP), which plays various roles such as user authentication, management of peer information, prohibition of illegal access from the outside, and establishment of secure connections between participating peers.

Each of the other peers acts as a *client peer* (CP) or a *service-providing peer* (SPP). A CP can find an appropriate SPP that provides a service demanded by its user (*e.g*., pictures taken by a security camera and the control of lighting equipment) by

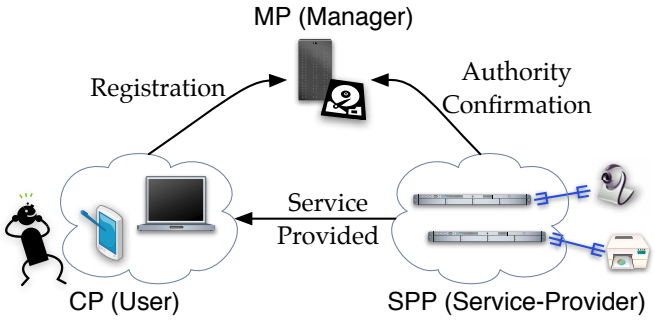


Figure 1: Tran system.

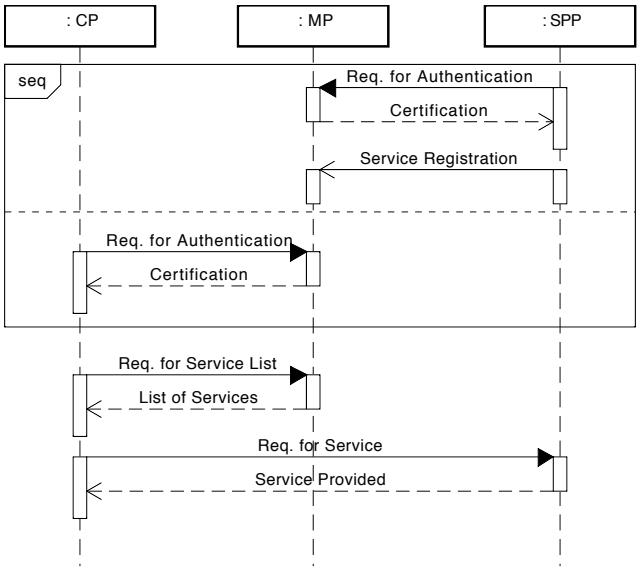


Figure 2: Basic flow in the Tran system.

asking the MP for a list of reliable SPPs. An SPP in the list provides its service after receiving a request from the CP, without confirming the CP's authority, since the advertisement of the service has been transferred only to those CPs certified by the MP.

More concretely, a service given by an SPP is provided to a CP in the following steps (see Figure 2 for illustration):

Step 0 (Authority List) The MP statically holds a user authority list, which assigns each SPP to a subset of CPs that are allowed to receive its service.

Step 1 (Peer Authentication) Each peer (*i.e*., CPs and SPPs) requests the MP to authenticate the peer itself, and receives a certification from the MP when it succeeds. Note that the security of the overall scheme is enhanced by providing these certifications to both of CPs and SPPs.

Step 2 (Service Registration) Each certified SPP registers its services to the MP. Note that in this system, the MP merely manages the registered service information (including advertisements and certifications), *i.e*., the actual services will be provided by SPPs in a P2P manner.

Step 3 (Service Discovery) A certified CP requests a list of available services to the MP. After receiving the request,

the MP replies the advertisements of service pipes connecting to SPPs that provide the services available to the CP.

Step 4 (Providing the Service) When one of the available services is chosen by a user via a CP, the CP starts communicating with an SPP through a service pipe that is associated with one of the received advertisements.

It should be noted that all of the above pipe communications could be encrypted after acquiring certifications of the communicating peers.

3.3 Tokunaga System

In Tran system, the assignments of CPs to SPPs are determined by the MP statically in a centralized manner. In other words, no SPP can start a new service unless it was registered to the MP in advance, and no CP can unsubscribe from a service safely once it received its service advertisement from the MP. However, in distributed systems such as nCEs, it should be useful if the memberships could be controlled dynamically by SPPs. For instance, let us consider a case in which an SPP provides a Video-on-Demand (VoD) service to a group of admitted audience peers. With a dynamic membership control, the SPP can refuse requests from kids after 8 p.m., without shutting down the service itself (*i.e.*, access from adults are still accepted). Another example is an exceptional acceptance of remote access to the main cock of gas line in case of emergency such as earthquake or fire, where (possibly any) CPs must be temporally admitted by SPPs before recovering the gas line.

In order to realize such a dynamic user authority management, we proposed the following functions in [2] (we refer to the resultant system "Tokunaga system" hereafter):

Confirming the Authority User authority is securely managed by the MP in the way similar to the Tran system. An SPP admits access to its resources by dynamically referring to the list of *communication keys* allowed for that service, which is managed by the MP. A communication key is used to ensure the identity of a certified CP.

Service List Updates An SPP can launch a new service and quit it on itself, whereas the information on each service is registered to the MP. A CP always can find such a new service by inquiring to the MP.

In addition, in this system, a *caching* mechanism at the SPP is introduced, to enable the service provision without repeatedly communicating with the MP. However, it should be noted that in order to keep the security of the scheme sufficiently high, the cached keys must be re-authenticated for every adequate time interval.

The overall configuration of the Tokunaga system is illustrated in Figure 3. The difference from Tran system is that SPP confirms the user authority to the MP before providing a service.

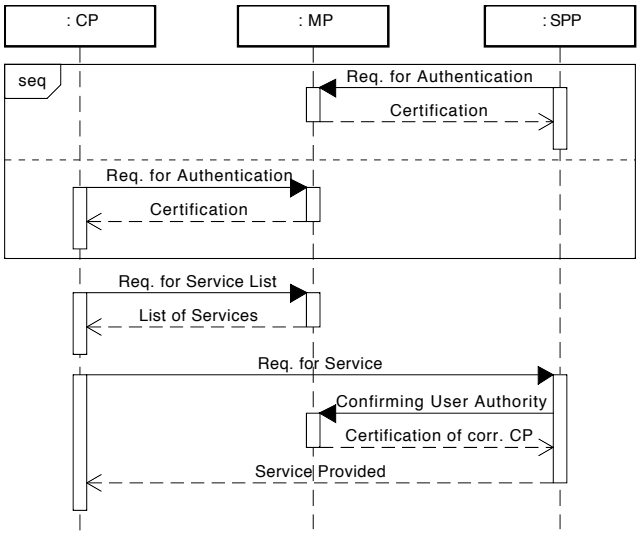


Figure 3: Basic flow in the Tokunaga system.

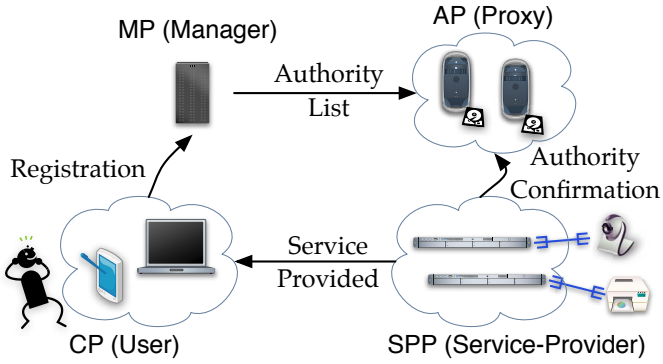


Figure 4: Proposed system.

4 Proposed Scheme

4.1 Scheme

This section proposes a new authentication scheme for P2P systems. Figure 4 illustrates an overview of the proposed scheme. In the proposed system shown in the figure, we introduce a new kind of peers called *authentication proxies* (APs) to our previous system. More concretely, in this system, the authority list generated by the MP is transferred to APs, which receive and process inquiries submitted by SPPs as a "proxy." Note that the number of APs is not restricted to one, which avoids the system to strongly rely on a small number of peers (the details will be explained later). Also, note that each networked device can simultaneously play several roles in the proposed system; *e.g.*, an AP may reside on a device providing a service as an SPP.

The behavior of each peer in the proposed system is described as follows (see Figure 5 for illustration):

Step 1 (Registration) Each CP generates and registers its certification to the MP. The MP determines the CP's authority when it receives a certification. Note that a determination may come up with a payment for the service, although it is out of scope of the current paper.

Step 2 (Broadcasting the Authority List) From a collection of received registration messages, the MP generates an au-

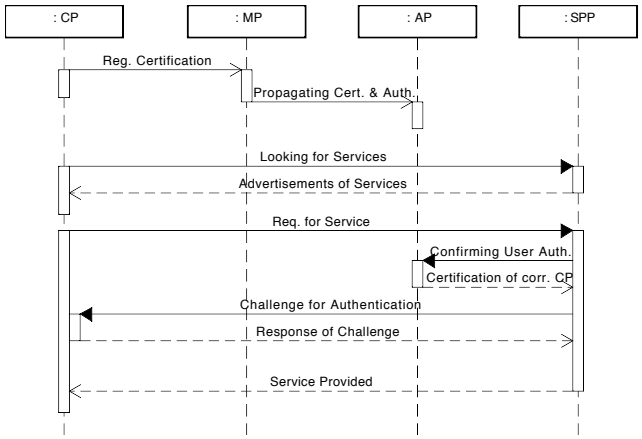


Figure 5: Basic flow in the proposed system.

thority list consisting of the following two tables, where each entry in the tables is attached a signature of the MP:

ACL: Service ID, Client ID, and one or more validity periods

Clients: Client ID and Client Certification

The MP broadcasts the generated list to all APs through a multicast tree, and whenever the authority list is updated, MP broadcasts the difference to the APs again (it sends the whole list if the receiver is a newly joined AP). Additionally, the public key of the MP is also transferred to those APs and SPPs in a secure way.

Step 3 (Storing the Authority List) After receiving (a part of) the authority list, each AP certifies its entries using the public key of the MP, and stores a copy of the received list to its local table.

Step 4 (Service Discovery) A user finds (an advertisement of) an SPP providing a desirable service using a function provided by JXTA. This operation could be realized in a P2P manner, without explicitly sending an inquiry message to the MP.

Step 5 (Service Request) After finding a desired service, the user sends a request for the service to the discovered SPP p through the CP q corresponding to him/her.

Step 6 (Confirming the Authority) After receiving the request, p first sends an inquiry about the authority of q to an AP r, with the service ID of p and the client ID of q. AP r looks for an entry concerned with q in its local table, and replies p a list of authorities and the certification of q, if any. The received list could be certified by p since it is attached the signature of the MP. Note that although p can cache the response from r as in our previous scheme, it is not critical in this scheme since the response time from r is sufficiently small as will be evaluated in the next section.

Step 7 (Confirming the Identity) SPP p confirms the identity of the CP q by a challenge-response authentication. After receiving a challenge from p, q generates a response to the challenge using its private key, and sends it back to p, which is then certified by p using the certification of q.

Step 8 (Providing the Service) If the challenge-response authentication succeeds and the response from AP r contains an entry corresponding to the service, p starts to provide its service to q. The authority expires according to the validity listed in the ACL. If SPP p detects such expiration, p informs the fact to q, in order to start the next authentication process.

Note that in the above scheme, MP plays the role of a certificate authority (CA) in addition to the management of the authority of users, *i.e.*, it maintains the validity of certifications of the CPs.

4.2 Advantages

The proposed system has the following advantages compared to our previous system:

1) At the SPP side: It significantly reduces the load of the MP, since the handling of inquiries received from SPPs is handed over to the APs, and the transmission of the authority list to APs could be realized using a multicast tree (*i.e.*, it is not necessary for the MP to directly communicate with all APs). This means that it significantly improves the scalability of the overall system.

2) At the SPP side: It tolerates the faults of APs since inquiries submitted by SPPs can be handled by "any" AP, provided that all APs have the same authority list redundantly.

3) At the CP side: The role of the MP is restricted to the management of the authority of CPs and the generation of the authority list. Thus, by broadcasting the public key of the MP to APs and SPPs beforehand, we can increase the number of MPs from one to any k. It would significantly improve the dependability and the scalability of the overall system, as well as the fault-tolerance.

4.3 Security and Privacy

In this subsection, we explain the reason why the proposed system provides as good security and privacy performances as our previous system does:

1) MP-AP security: The validity of the authority list received from MP by APs is guaranteed by the signature of MP attached to the list.

2) AP-SPP security: The validity of reply messages received from AP by SPP is certified by the signature of MP held by the SPP.

3) SPP-CP security: Spoofing of a CP to receive a service from an SPP is prohibited since it conducts a challenge and response certification for each service with a public key of the client. Additionally eavesdropping of a service provided by an SPP can be avoided since the message transmitted by the SPP is encrypted using a certification of the receiving client acquired through the AP.

4) Privacy of CPs: The list of services available to a CP is not disclosed to the other peers, since each CP can use different client IDs for each service in the authority list; *i.e.*, any peer cannot identify CPs from client IDs contained in the authority list. In other words, the proposed user management scheme could keep the privacy of each user in the system.

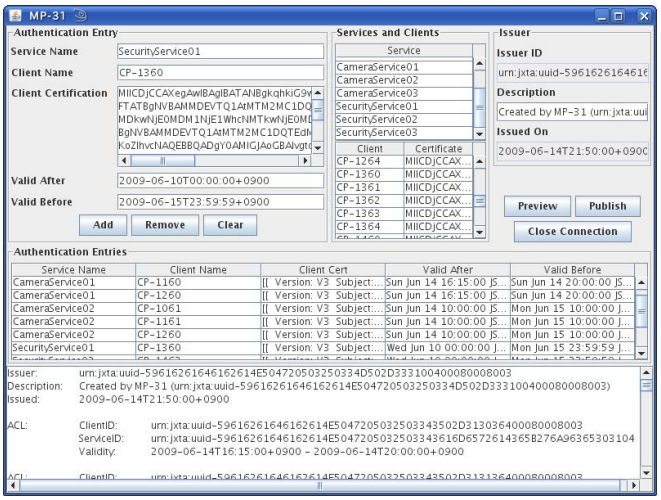


Figure 6: A screen shot of the prototype MP.

4.4 Implementation

A prototype of the proposed system is implemented for JXTA JXSE v2.5 running on Java SE Development Kit (JDK) 6 and openSUSE 11.0.

A screen shot of the prototype MP is shown in Figure 6. In this figure, the reader can observe that the authority is added and modified in the top of the window, and the resultant list is shown in the bottom of the window.

5 Evaluation

We conducted experiments to evaluate the performance of the proposed system in two aspects: the loads of peers and the elapsed time in an authentication process.

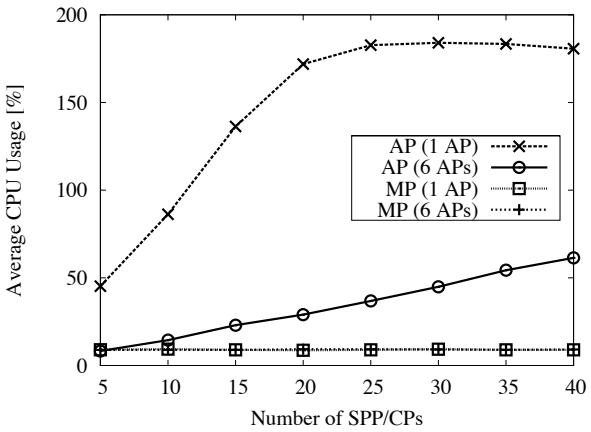
5.1 Environment

Parameters are determined as follows: At first, we fix the number of MPs to one. The MP generates authorities of CPs for every three seconds, where the validation time of each authority is set to six seconds. The number of APs is ranged from one to six. Note that the proposed system is equivalent to the Tokunaga system when the number of AP is one, except that it connects to the MP instead of the back-end database server. The number of SPPs is ranged from 5 to 40, and those SPPs are associated to at most eight PCs evenly so that each PC is associated with five SPPs. Each SPP connects to an AP, and transmits a set of 500 inquiries to the AP for every two seconds.

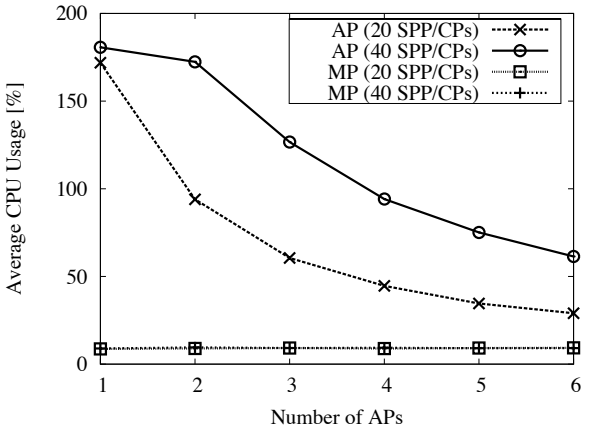
These peers are executed on 15 PCs each with Intel Core2 Duo E6600 2.4 GHz processor and 2 GB RAM equipped, which are connected over Gigabit Ethernet. In the experiment, we associate one CP for each SPP; *i.e.*, each PC is associated with five CPs. In addition, we consider a situation in which only half of the requesting services are accepted by the MP (*i.e.*, half of the service requests will be rejected).

5.2 Load of Peers

At first, we evaluate the load of peers in the proposed system. Figure 7 summarizes the result, where the horizontal axis of



(a) Impact of the number of SPP/CPs.



(b) Impact of the number of APs.

Figure 7: CPU usage of APs and MP.

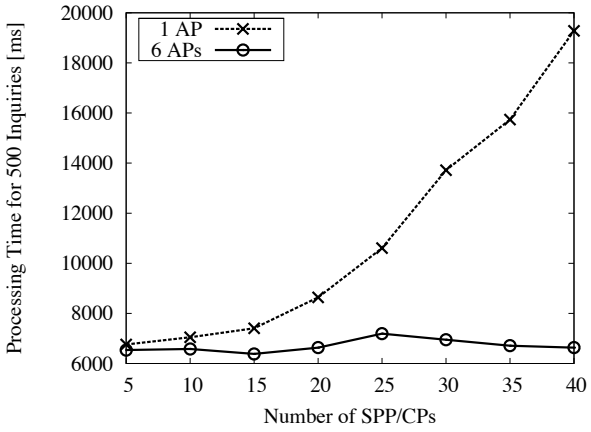
Figure 7a is the number of CPs and that of Figure 7b is the number of APs (in both figures, the vertical axis is the CPU usage of APs and MP). Note that the CPU usage in the figure ranges up to 200 % since we used a dual-core CPU in this evaluation.

As shown in Figure 7a, although the load of APs increases as increasing the number of CPs (*i.e.*, the number of inquiries), it realizes a graceful degradation by adopting multiple APs, which is apparently due to the effect of load balancing. In addition, we can also find from the figure that the load of the MP does not increase even when the number of APs increases. On the other hand, as shown in Figure 7b, the load of AP gracefully decreases when the number of APs increases. Again, this indicates that the use of APs certainly reduces the load of the communication to SPPs, and in fact, such collection of APs works as a kind of caching proxies in the overall system.

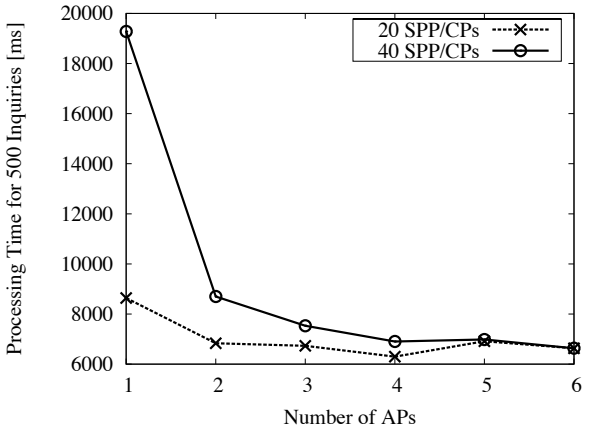
5.3 Authentication Processing Time

We next evaluate the time required for processing inquiries. Figure 8 summarizes the results, where the horizontal axis of Figure 8a is the number of CPs and that of Figure 8b is the number of APs (in both figures, the vertical axis is the processing time for 500 inquiries).

As shown in Figure 8a, the processing time of inquiries does



(a) Impact of the number of CPs.



(b) Impact of the number of APs.

Figure 8: Processing time of inquiries.

not increase even as increasing the number of CPs if there are several APs, although it rapidly grows if there is a single AP. In particular, the processing time exceeds 18 seconds if the number of CPs is 40 and the number of APs is one (recall that the CPU usage of the single AP "saturates" when the number of CPs exceeds twenty). On the other hand, as shown in Figure 8b, the processing time gradually decreases as increasing the number of APs, which saturates around three APs. A reason of such relatively rapid saturation is that the size of the network is too small so that three APs are enough to attain the best performance under the scheme (in fact, the processing time for 500 inquiries is 7,531 ms with 40 CPs under three APs, which is sufficiently small compared with the conventional model). An extensive experiment using a larger number of peers with a larger number of PCs is left as a future work.

6 Concluding Remarks

This paper proposed an improved user management system for network consumer electronics (nCE). The proposed system extends Tokunaga's system in such a way that the authority list is distributed and processed in the authentication proxies (APs), which resides between the centralized server and the service-providing peers. The proposed system was implemented as a JXTA network, and evaluated via experiments.

A future problem is to improve the efficiency of multicast among APs, possibly considering the overlap in users' interests in provided services. Although the current implementation could be applied to a network with a relatively small number of participating peers and a low frequency of updates of the authority list, we will have to reduce the traffic of multicasting the authority lists if we want to apply the scheme to a large network. Another interesting problem is how to control the service providing in a payment-based authority model; *e.g.*, we should solve many challenging problems such as how to realize a refund of payment when a requested service could not be provided to a client, and how to check the correctness of the declaration given by the users.

References

[1] Chris Loeser, Wolfgang Mueller, Frank Berger, and Heinz-Josef Eikerling. Peer-to-peer networks for virtual home environments. *Hawaii International Conference on System Sciences*, 9:282c, 2003.

[2] Tatsuya Tokunaga and Satoshi Fujita. Dynamic and secure user management in networked consumer electronics. In *Proc. the 2008 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2008)*, volume 2, pages 451–457, July 2008. Las Vegas.

[3] JXTA™ Community Projects. `https://jxta.dev.java.net/`.

[4] Jérôme Verstrynge. *Practical JXTA*. Lulu.com, August 2008.

[5] Joseph D. Gradecki. *Mastering JXTA: Building Java Peer-to-Peer Applications*. John Wiley & Sons, September 2002.

[6] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for mobile ad hoc networks. *Network Protocols, IEEE International Conference on*, page 251, 2001.

[7] Nitesh Saxena, Gene Tsudik, and Jeong Hyun Yi. Admission control in peer-to-peer: design and performance evaluation. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 104–113, New York, NY, USA, 2003. ACM.

[8] Yongdae Kim, Daniele Mazzocchi, and Gene Tsudik. Admission control in peer groups. In *NCA '03: Proceedings of the Second IEEE International Symposium on Network Computing and Applications*, page 131, Washington, DC, USA, 2003. IEEE Computer Society.

[9] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[10] Xuanhoa Tran, Kenji Sugihara, Tsutomu Yoshinaga, and Masahiro Sowa. Introduction of User Authentication and Access Control Mechanism into JXTA Network. *IPSJ SIG Notes, CSEC*, 126:65–70, 2003.

二連結性を保証する分散センサカバーアルゴリズム

中本 浩太　　藤原 暁宏

九州工業大学院 情報工学府 情報システム専攻

k-nakamoto@zodiac30.cse.kyutech.ac.jp　　fujiwara@cse.kyutech.ac.jp

概 要

近年，センサネットワークが，ユビキタス社会を実現するために必要な技術として注目を集めている．この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必要が生じる．この操作は一般に照会と呼ばれるが，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挙げられる．この通信コストの削減手法の一つとして，要求領域を検知領域で被覆するセンサの部分集合を求める手法が研究されている．

このような連結かつ要求領域を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．この連結センサカバーをセンサネットワークに対して求める手法はいくつか提案されているが，既存手法では求めた連結センサカバーの二連結性を保証しておらず，センサが故障などで使用できなくなった場合，照会が実行できなくなる．

そこで，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分散アルゴリズムを提案する．まず最初に，既存手法の連結性について解説し，次に，二連結性を保証する分散連結センサカバーアルゴリズムを 2 種類提案する．最後に，既存アルゴリズムと提案アルゴリズムをシミュレーション環境に実装し，その結果を検証する．実験結果より，提案アルゴリズムは既存手法と同程度のセンサ数と少ない通信コストで，二連結性を保証する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，小型自律センサにより構成される無線通信ネットワークである．各センサは，検知機能と通信機能を併せ持っており，各々のセンサが，自身の検知範囲内の情報を通信範囲内の他のセンサへメッセージ通信することにより，ネットワーク全体で広大な範囲の情報を得る事ができる．

一般に，センサネットワークを用いて監視を行う場合は，監視を行いたい領域にセンサを配置し，各センサの検知領域内の情報を収集し，処理を行う．このとき，ある特定の監視を行いたい領域 (要求領域) を各センサの検知領域で被覆し，さらに要求領域の検知を行うセンサは任意のセンサと通信可能 (連結) である必要がある．このように，要求領域をセンサの検知領域で被覆する連結なセンサネットワークを連結センサカバーとよぶ．

一方，近年の半導体技術や無線通信技術の発展から，より利用環境に適応するために，センサの小型化が進んでおり，それに比例して，内蔵のバッテリーの容量も減少している．また，センサは充電が困難である環境化での使用が一般的であるため，センサネットワーク全体での消費電力の削減が必要であり，この消費電力削減の手段として，通信コストの削減がある．したがって，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，様々な研究が行われている．

この通信コスト削減方法の 1 つとして，センサの情報を収集する際に，前述の連結センサカバーに属するセンサのみを使って照会を行うことにより，有効な省電力化が可能であることがわかっている．この連結センサカバーの構築は，大量のセンサを用いれば容易であるが，電力消費の観点から，連結センサカバーをできるだけ少数のセンサで構築することが望ましい．しかし，センサ数最小の連結センサカバーを求めることは，非常に困難 (NP 困難) であることが知られている [1].

また，この連結センサカバーを求める手法は既に多く研究されている [1, 2, 4]．例えば，Gupta ら [1] は，連結センサカバーを求める集中型アルゴリズムを提案し，さらにこの集中型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 [4] は，Gupta らの分散アルゴリズムを拡張し，連結センサカバーの構築を開始するセンサを複数にすることで，構築に要する時間や通信コストが削減されることを示している．しかし，Gupta らや森川らのアルゴリズムは，求めたセンサカバーが連結であることは保証しているが，二連結であることは保証していない．そのため，故障やバッテリー切れなどにより，連結センサバーを構成するあるセンサが使用不可能な状態に陥った場合，ネットワーク全体の連結性を保証する事ができない．

一方，白石 [2] は二連結性を保証しつつ連結センサカバーを構築するアルゴリズムを提案した．しかし，提案アルゴリズムは集中型のアルゴリズムであり，個々のセンサの動作を一意に決定してしまい，独立した動作を保証していない．

そこで本研究では，与えられるセンサ集合と要求領域に対して，二連結性を保証する分散型連結センサカバーアルゴリズムを提案する．まず最初に，二連結性を保証する分散型アルゴリズムを，構築の方向の違いにより 2 種類提案する．一つは，任意のセンサを中心に， 連結センサカバーに属するセンサ同士による検知領域交点を被覆しながら，要求領域の端へ向かって連結センサカバーを拡張する手法である．もう一つは，要求領域の端を環状な連結センサカバーで被覆し，要求領域の中心へ向かって連結センサカバーを拡張する手法である．

次に，既存アルゴリズムと提案アルゴリズムをシミュレーション環境に実装し，その結果を検証する．実験結

果より，提案アルゴリズムは既存手法と比べて，同程度のセンサ数と少ない通信コストで二連結性を保証する連結センサカバーが得られることを示す．

本論文の構成は以下の通りである．2 章でシステムモデルの説明と問題定義，及び，既存手法の説明を行う．3 章では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言及する．

2 準備

2.1 システムモデル

本研究で対象となるセンサネットワーク $G = (V, E)$ は，センサ集合 $V = \{I_1, I_2, \ldots, I_n\}$($n$ はセンサ数) と，通信可能なセンサ間の通信経路であるリンク集合 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持っている．

本研究で用いるセンサモデルを，図 1 に示す．各センサ I_i は，センサを中心とした半径 t の円形の通信領域 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．センサ I_i, I_j が互いに通信可能であるとき，G 上の I_i, I_j 間に双方向通信リンク $e_{ij} \in E$ が存在し，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，ネットワーク中のセンサはすべて起動しており，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，改変は起こらないとする．なお，センサ I_i, I_j 間はメッセージ交換によってのみ情報交換を行う．

センサ I_i の検知領域 S_i とは，センサ I_i が平面 R 上のセンサを中心とした半径 s の円形の監視を行う領域である．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 の最小センサを介した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 I_i, I_j 間の距離は，検知領域が重なり合う事から $2s$ 以内であり，このとき，センサ I_i とセンサ I_j を結ぶ直線上に十分なセンサが存在するならば，$\frac{2s}{t} + 1$ ホップ以内で I_i と I_j 間は通信可能である．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数をホップ数という．

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，および被覆すべき領域が与えられたとき，可能な限り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 と呼び，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 上の連続する平面領域である．

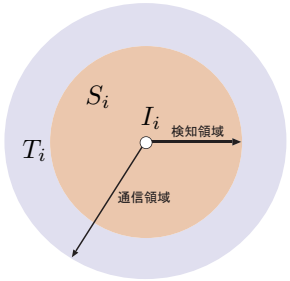


図 1: センサモデル

つまり，要求領域 R_Q が共通領域を持たない 2 つ以上の平面領域で与えられることはない．

これらを用いて，連結センサカバーを以下に定義する．

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 $M = \{I_{i_0}, I_{i_1}, \ldots, I_{i_{m-1}}\}(M \subseteq V)$ を連結センサカバーとよぶ．

1. $R_Q \subseteq (S_{i_0} \cup S_{i_1} \cdots \cup S_{i_{m-1}})$
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，この問題からの帰着により連結センサカバー問題もまた NP 困難として知られている [1]．

本研究では，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し，連結センサカバー M に属するセンサにより被覆することのできる検知領域を $S(M)$ とする．

図 2 に連結センサカバーの例を示す．この例では，センサ集合 $\{I_1, I_2, \ldots, I_9\}$ が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を保っている．

2.3 評価指標

本研究で提案する連結センサカバーアルゴリズムの評価指標は以下の通りである．

評価指標：

- 連結センサカバーに属するセンサ数
- 連結センサカバー構築に要したステップ数
- 連結センサカバー構築に要した通信コスト

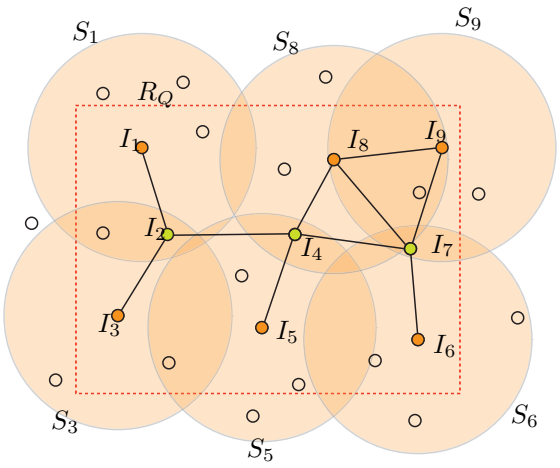


図 2: 連結センサカバーの例

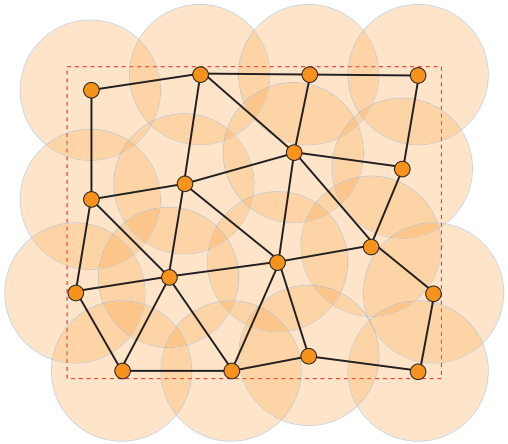


図 3: 二連結性を保証する連結センサカバーの例

なお，本研究での 1 ステップとは，メッセージを受信したすべてのセンサが，そのメッセージに対する処理を終える単位を指し，ステップ数はアルゴリズム終了時点までに必要なステップ数とする．通信コストとは，センサネットワーク上で行われたメッセージ送信の総和である．

2.4 グラフの二連結性と関節点

アルゴリズムが構築する連結センサカバーを無向グラフ $G = (V, E)$ と捉えると，センサを節点 $u, v \subseteq V$，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つ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．

この二連結性をもつ連結センサカバーを図 3 に示す．本研究では，図 3 に示すような，任意のセンサに連結となるセンサが存在し，かつそれらの連結度が 2 であるような連結センサカバーを求める．

2.5 Gupta らの連結センサカバーアルゴリズム

本研究では，連結センサカバーアルゴリズムでセンサを二連結性を保証しつつセンサを追加する手法を提案する．各センサが実行する連結センサカバーアルゴリズムには，Gupta らの手法を改良したものを用いている．そこで本節では，分散環境における Gupta らのアルゴリズムの概要を説明する．

まず，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は以下の値を保持する．

連結センサカバーの集合 ： k ステップ終了時点で，アルゴリズムにより連結センサカバーに選ばれたセンサの集合 M_k

候補パス ： k ステップにおいて，連結センサカバー M_k に含まれる候補となるパス (センサの連なり)P

候補パス集合 ： 各ステップにおける候補パスの集合 SP

コーディネータ ： ステップ k の時点で最後に連結センサカバー M_k に加えられたセンサ C_k．コーディネータは，次のステップ $k+1$ で連結センサカバー M_{k+1} に追加するセンサを探索する

以下では，アルゴリズムの概要を順番に説明する．Gupta らのアルゴリズムでは，要求領域内の任意の 1 センサから以下に示すアルゴリズムを開始する．まず最初に，M_k に含まれないセンサから M_k に含むことのできるセンサと，定数個のセンサを介して通信が可能な周辺センサを選出する．どの周辺センサを加えるかは，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，連結センサカバーにセンサが追加される毎に変わり，$k_1 \neq k_2$ ならば $C_{k_1} \neq C_{k_2}$ である．

連結センサカバー M_k において，コーディネータ C_k は検知領域 $S(M_k)$ の周辺のセンサから複数センサを選出し，連結センサカバー M_{k+1} に追加して連結センサカバーを構成するセンサを増やしていく．以下，その方法を述べる．

コーディネータ C_k は，周辺センサに探索メッセージを送信し，周辺にどのようなセンサが存在するのかを確かめる．得られた情報に基づき，コーディネータ C_k は効果的に要求領域を被覆できるセンサを一つ，または複数選出する．このとき，センサの検知領域と要求領域の共通領域を個々に評価してセンサカバーに加えるセンサを選出すると，多くのセンサの検知領域が同一領域を被覆してしまう可能性がある．

そこで，探索メッセージを受信したセンサ I_j に対し，I_j からコーディネータ C_k までの通信経路の 1 つをセンサ I_j の候補パス $P_{j,k}$ としてコーディネータへ返信する．コーディネータ C_k は被覆されていない要求領域の広い範囲を少数のセンサで被覆する候補パスを，以前のコーディネータが得た候補パスも含めた中から選出する．そして，選出された候補パス上に存在するすべてのセンサをカバー M_{k+1} に追加する．

連結センサカバー M_k におけるアルゴリズムを，“探索フェーズ”，“応答フェーズ”，“決定フェーズ”の 3 つのフェーズに分けて説明する．

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連結センサカバー M_k の検知領域 $S(M_k)$，およびコーディネータ C_k が送信者である情報を付加する．探索メッセージはコーディネータ C_k から $2r_G$ ホップ以内で通信可能なすべてのセンサへブロードキャストされる．すなわち，コーディネータ C_k の検知領域と自身の検知領域が共通領域をもつセンサ I_i と，さらにその S_i と検知領域に共通領域をもつセンサ I_j すべてに探索メッセージが送信される．各センサ I_i は，初めに受信した探索メッセージのみを受理し，その探索メッセージが経由した経路がセンサ I_i の候補パス $P_{i,k}$ となる．

応答フェーズ ： 応答フェーズでは，探索メッセージを受信したセンサ I_i がコーディネータ 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 と連結センサカバー 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識別子の列) と，パス上の各センサの検知領域の情報を付加する．

決定フェーズ ： 決定フェーズでは，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 が連結センサカバー M_k に追加すべき候補パスを選出し，選出された候補パス上のすべてのセンサを連結センサカバー M_k に追加する．コーディネータ C_k は，応答メッセージによって得られた各候補パス $P_{i,k}$ を候補パス集合 SP_k に追加し，さらに各候補パス $P_{i,k}$ の検知領域 $S(P_{i,k})$ を保持しておく，各候補パス $P_{i,k}$ の検知領域 $S(P_{i,k})$ は以下で求められる．

$$S(P_{i,k}) = \bigcup_{I_j \in P_{i,k}} (S_j \cap R_Q)$$

連結センサカバー問題では，少数のセンサでできる限り広い範囲の要求領域を被覆する事が求められる．そこでコーディネータ 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 (絶対値は面積を表す)．

候補パス $P_{i,k}$の評価値

$$= \frac{|\, S(P_{i,k}) \cap \overline{S(M_k)} \,|}{\begin{array}{l}P_{i,k}\text{上に存在し，かつ} \\ \quad \text{カバー } M_k\text{に含まれていないセンサ数}\end{array}}$$

評価値が最大となるパスを $P_{i,k}$ とする．コーディネータ C_k は決定メッセージを $P_{i,k}$ 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築するセンサになる．すなわち，連結センサカバー M_{k+1} は連結センサカバー M_k に含まれるセンサと，新たに追加した $P_{i,k}$ 上のセンサで構築される．決定メッセージには候補パス集合 SP_k，および SP_k に含まれる各候補パスの検知領域の情報を付加し，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．

以上のフェーズをカバー M_{k+1} の検知領域が要求領域 R_Q を被覆するまで繰り返す．図 4 と図 5 にアルゴリズムが連結センサカバーを構築する様子を示す．図 4 は，ステップ k におけるコーディネータ C_k が，探索により得た候補パス $P_{2,k}, P_{3,k}, P_{4,k}$ と，ステップ $k-1$ と $k-2$ から引き継がれた候補パス $P_{6,k-1}, P_{9,k-2}$ を示している．図 5 は，$k+1$ ステップ後を表しており，これらの候補パスの中から $P_{4,k}$ を選出し，そのパス上のセンサ (I_2, I_3, I_4) が連結センサカバー M_{k+1} に追加されている．選出されたパス $P_{4,k}$ 上の最後尾のセンサ I_4 が $k+1$ ステップのコーディネータとなり，探索により候補パス $P_{5,k+1}, P_{6,k+1}$ を得ている．

3 二連結を保証する連結センサカバーアルゴリズム

3.1 二連結性の保証

Gupta らのアルゴリズムは，最初のコーディネータを出発点とし，周囲に存在するセンサに探索メッセージを送信する．また，最後に連結センサカバーへ追加されるのは，自身の検知領域が連結センサカバーに含まれるセンサの検知領域と自身の検知領域で共通領域を持つセンサである．

つまり，応答メッセージを返信する最後に選択されるセンサを除いては，互いに 2 本の通信辺をもっていることがメッセージ通信により保証されている (ただし，連結センサカバーに属するセンサが一つである連結センサカバー構築開始直後では，パス上の先頭センサの二連結性は保証されない)．よって，最後に連結センサカバーへ追加するセンサと連結センサカバーに含まれるセンサ間に通信辺が存在すれば，この候補パス全体の二連結性が保証されることになる．ここで，図 6 に，二連結性を保証する候補パスの一例を示す．

上記のアイデアにより，Gupta らのアルゴリズムの応答フェーズに以下の条件を追加することにより，Gupta らのアルゴリズムでも，連結センサカバーの二連結性を保証することができる．

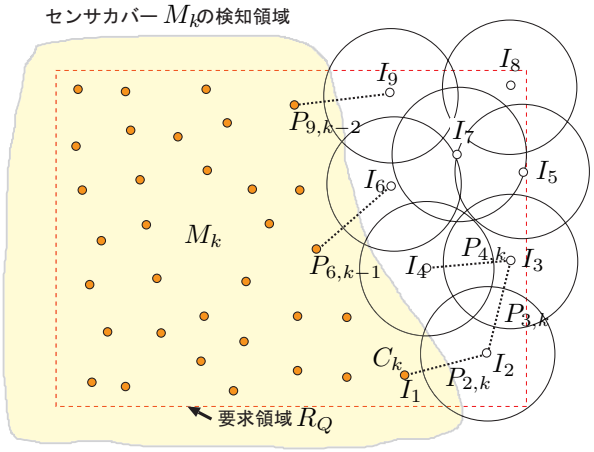


図 4: Gupta らの手法の動作例 1

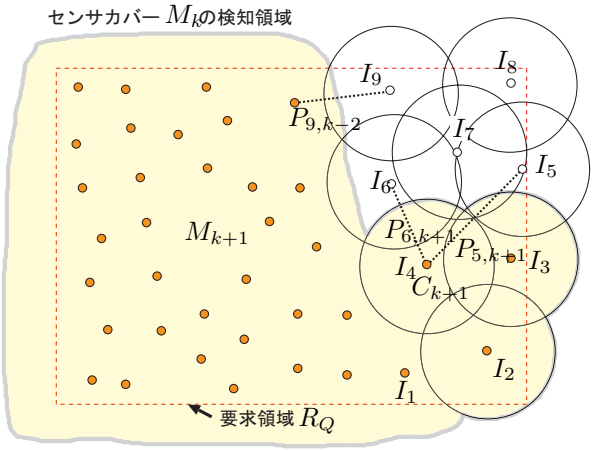


図 5: Gupta らの手法の動作例 2

- 探索メッセージを受信したセンサは，連結センサカバーに含まれるセンサと通信可能である場合にのみ応答メッセージを返信する

次節では，連結センサカバーのセンサ数を減らす為のアイデアとして，応答メッセージを返信する複数のセンサからどのセンサを選択するかについて，2 つの手法を提案し，それぞれについて説明する．

3.2 センサ数を減らすためのアイデア

各センサの二連結性を保証すると，通信のみを担うセンサが多く選出されるため，センサ数が増加してしまう．この問題を少しでも抑えるために，提案手法では連結センサカバーへ追加するセンサを，構築中の連結センサカバーに含まれるセンサ同士の検知領域の交点を被覆するセンサのみに限定する．

この節では，検知領域交点集合を被覆するアイデアを，検知領域交点集合の定義とともに示し，次に，任意のセンサから連結センサカバーへの通信辺の有無を探索するアイデアについて説明する．

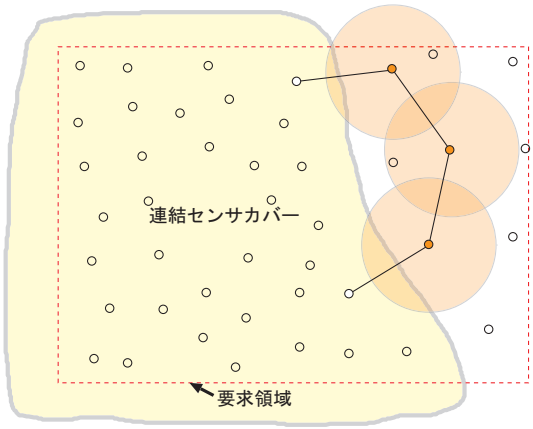


図 6: 二連結性を保証する候補パスの例

3.2.1 連結センサカバーに含まれるセンサ同士の検知領域交点を被覆する手法

Gupta らのアルゴリズムでは，コーディネータは隣接のセンサへ探索メッセージを送り，応答のあったセンサの中から，最大の評価値を得る候補パスを追加する操作を繰り返すことにより，連結センサカバーを構築している．この際，応答メッセージを返すセンサは，連結センサカバーに属するセンサの検知領域と自身の検知領域が重なりを持つものである．しかし，この条件のもとでセンサを追加すると，図 7 のようにコーディネータの周りに微小未被覆領域が生じる場合が考えられる．この結果，この微小未被覆領域を被覆するためのセンサを別に選出する必要が出てくるため，連結センサカバーのセンサ数が増大してしまう．

この問題を回避するために，提案手法では，コーディネータと近隣のセンサの検知領域を表す 2 つの境界円により生じる交点 (検知領域交点) を被覆することを必要条件としセンサを探索する．

ここで，検知領域交点とは，要求領域内に存在するセンサが持つ検知領域同士の境界円が交わる点のことで，2 個以上のセンサの検知領域が交差すれば必ず存在し，連結センサカバーの構築が進むほど検知領域交点の集合は増加する．しかし，追加するセンサはできるだけ大きい評価値を得たいので，連結センサカバーに属するセンサによって既に被覆されている検知領域交点は探索には用いたくない．よって，検知領域交点集合を以下のように定義する．

定義 2 (連結センサカバー上の検知領域交点集合) 連結センサカバー M_k 上の検知領域交点集合 IP_k は，M_k に含まれるセンサの検知領域交点から M_k に属する他のセンサによって被覆可能な検知領域交点を除いた集合である．センサ I_i と I_j の検知領域交点を $ip_{i,j}$ と表す

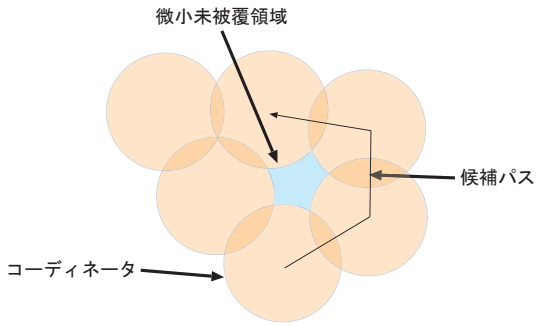


図 7: Gupta らのアルゴリズムで発生する微小未被覆領域の例

と，検知領域交点集合 IP_k は以下の式で表される．

$$IP_k = \{ip_{i,j} | I_i, I_j \in M_k, ip_{i,j} \notin S(M_k) - \{S_i, S_j\}\} \quad \square$$

図 8 に，上記の定義にそった検知領域交点の集合を示す．

センサカバー構築開始直後では，連結センサカバーに属するセンサは一つしかないため，センサ同士の検知領域交点はない．よって，このステップのみ Gupta らのアルゴリズムに則り隣接センサを追加する．次のステップ以降では，選出済みセンサは二つ以上存在するため，それらのセンサが持つ検知領域交点集合を更新する．ここで，検知領域交点集合の更新とは，連結センサカバー M_k のセンサによる検知領域交点集合 IP_k を，上記の定義に従い再選出することである．

次に，コーディネータは更新された検知領域交点を自身に記憶し，この検知領域交点を被覆するセンサの探索フェーズに移る．コーディネータが送信する探索メッセージを受信したセンサは，コーディネータが持つ検知領域交点を一つでも自身の検知領域で被覆できるかを判断する．被覆可能な場合は応答メッセージを返信し，メッセージ上のパスの長さがリンク半径内であれば探索メッセージをブロードキャストする．

応答メッセージを受信したコーディネータは，メッセージ上の候補パスの評価値を計算し，候補パス集合に保存する．コーディネータが連結センサカバーに追加する候補パスを選出するステップになると，候補パス集合から最大の評価値となる候補パスを一つ選び，そのパスへ決定メッセージを送信する．このときコーディネータの検知領域交点を新しいコーディネータに引き継がせるため，選出した候補パス上のセンサへ検知領域交点集合も渡す．

Gupta らのアルゴリズムでは，決定メッセージを受信したセンサは，メッセージの宛先でない場合，ただセンサネットワークに追加されるだけであったが，提案手法ではコーディネータが持つ検知領域交点集合から，自身の検知領域にて被覆可能な検知領域交点を削除する．一方，メッセージを受信したセンサがメッセージの宛先である場合は，パス上の他のセンサと同様に検知領域交点集合から自身で被覆可能な交点を削除し，新しいコーディネータとなる．

ここで，IP_{C_k} をコーディネータ C_k が保持する検知領域交点集合，ip_i をメッセージのパス上のセンサ I_i の検知領域で被覆可能な検知領域交点，SP_k をコーディネータ C_k が持つ候補パス集合とすると，$k+1$ ステップの交点集合 IP_{k+1} は以下の式で表される．

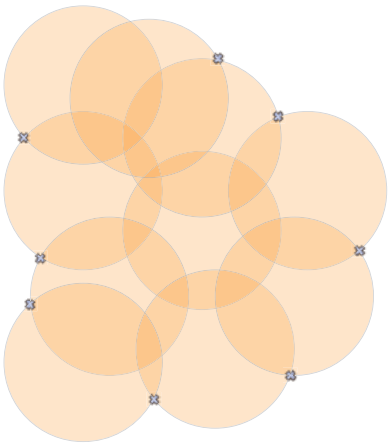


図 8: センサネットワークの検知領域交点

$$IP_{k+1} = IP_{C_k} - \{ip_i | I_i \in SP_k\}$$

このとき，検知領域交点集合がまだ残っている場合は，新しいコーディネータはその検知領域交点集合を記憶しておき，上記と同様のアルゴリズムを開始する．これとは逆に，候補パス上のセンサに検知領域交点が全て被覆されている場合は，このステップにおいて連結センサカバーに属するセンサを用いて，検知領域交点集合を更新させ，上記のアルゴリズムを開始する．それぞれの場合の，新しいコーディネータ C_{k+1} が探索に用いる検知領域交点集合は以下の式で表される．なお，式中の $IP_{i,j}$ は，センサ I_i と I_j による検知領域交点集合とする．

$$IP_{C_{k+1}} = \begin{cases} IP_{C_{k+1}} & (IP_{k+1} \neq \phi) \\ \{IP_{i,j} | I_i, I_j \in M_{k+1}\} & \\ -\{IP_{i,j} | IP_{i,j} \subseteq I_k, k \neq i\} & \\ & (\text{それ以外}) \end{cases}$$

次に，任意のセンサが連結センサカバーと連結となるか否かを判断するアルゴリズムについて以下に示す．

3.2.2 通信辺探索の手法

この節では，任意のセンサが連結センサカバーに属するセンサと連結であるか否かを探索する手法を説明する．例えば，図 9 のような，候補パス上の最後尾のセンサが連結センサカバーに属するセンサと連結であるか判断することに用いることができる．実際の探索は以下に示す 3 つのステップからなる．

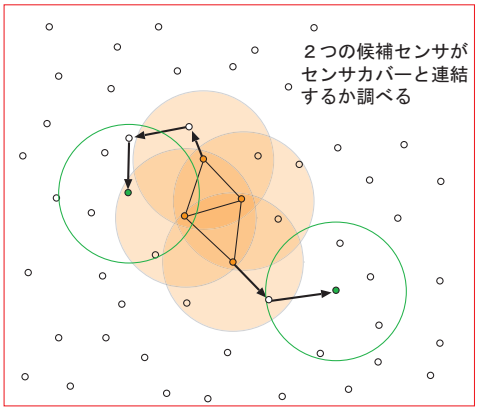


図 9: 探索メッセージによる選出済み候補センサ

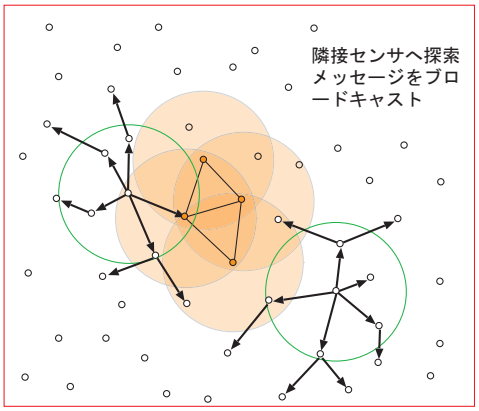


図 10: 候補センサによる探索メッセージのブロードキャスト

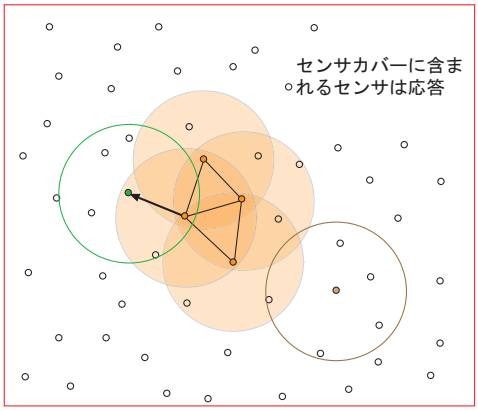


図 11: センサカバーに属するセンサからの応答

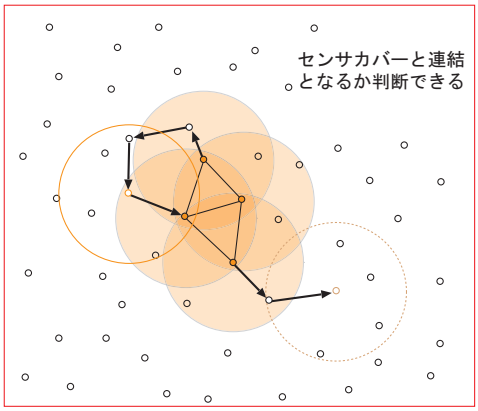


図 12: 通信辺探索アルゴリズムによる結果

Step 1: 探索 まず，連結センサカバーと連結であるかを知りたいセンサは，自身の隣接センサへ探索メッセージをブロードキャストする．この探索メッセージ上のパスは，自身を含んだ長さを 3 で制限し，それ以上の探索を行わないものとする．この探索メッセージを受信したセンサは，自身が選択済みセンサであり，かつ候補パス上のセンサでない，つまりコーディネータ以外の連結センサカバーに属するセンサである場合は，応答メッセージを返信する．

選択済みセンサでない場合は，メッセージのパスへ自身を追加しパスの長さが 3 を超えない場合は，隣接センサへブロードキャストする．

Step 2: 応答 Step 1 で探索メッセージをブロードキャストしたセンサは，受信した応答メッセージから，パスの長さが 2 であるものがあるか確認する．パスの長さが 2 であるものは，自身の隣接センサへ選択済みセンサが存在することを意味し，同時に連結センサカバーと連結であることがわかる．

一方，応答メッセージ中のパスの長さがすべて 3 である場合は，この探索を行うセンサは連結センサカバー中のセンサと連結でないことがわかる．

Step 3: 決定 応答メッセージの条件により，自身がセンサカバーと連結であるかがわかる．得られた結果により各手法の処理に従う．

図 9〜図 12 に，上記の手法により構築済みセンサカバーと，通信編が存在するか否かを判断する様子を示す．

図 9 は，緑で示す候補センサがセンサカバーと連結であるかを探索する様子を示している．図 10 は，任意の探索を行いたいセンサが，隣接センサへ探索メッセージをブロードキャストする様子を示している．この場合，緑で示した候補センサが探索を行うセンサである．探索メッセージの長さは 3 であるので，最大で 2 つ先のセンサまでメッセージが伝送されている．

図 11 は，条件を満たすセンサから応答メッセージが返信される様子を示す．この場合，緑で示したセンサのみ応答メッセージを受信できており，このセンサはセンサカバーと連結であるといえる．図 12 は，このアルゴリズムにより，連結が保証できるセンサと保証できないセンサを表している．この場合，左側の候補パスは全体の二連結性を保証できているが，右側の候補パスは保証できていない．

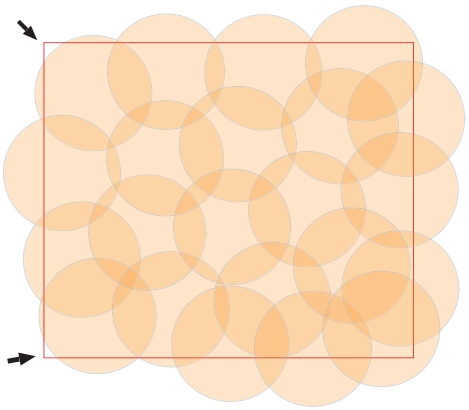


図 13: センサ同士の検知領域交点が無い状態

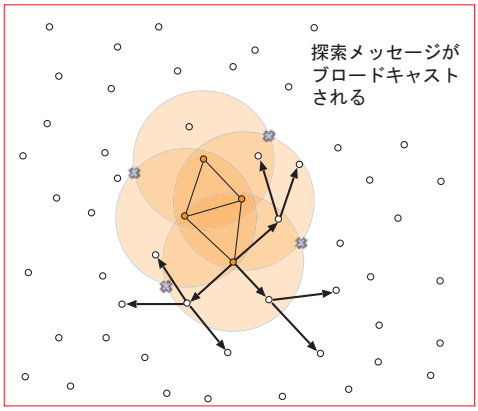


図 14: 検知領域交点集合の探索

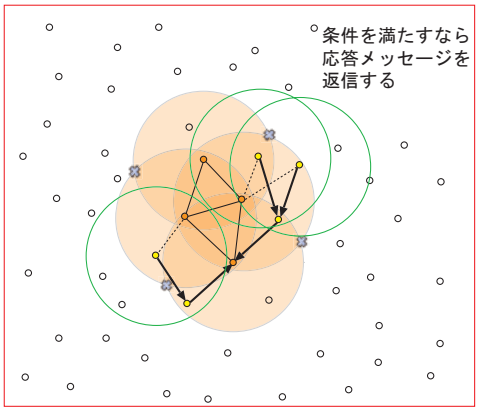


図 15: 条件を満たすセンサからの応答

3.3 任意のセンサを中心に構築するアルゴリズム

本節では，具体的な分散連結センサカバーアルゴリズムとして，任意のセンサを中心に連結センサカバーを構築するアルゴリズムを提案する．

このアルゴリズムは，Gupta らのアルゴリズムに前述の連結センサカバーに属するセンサ同士の検知領域交点を被覆する手法を追加したものである．このアルゴリズムにより，任意のセンサ (最初にコーディネータとなるセンサ) を中心にほぼ同心円上のセンサを追加していく．これにより，図 7 に示す微小未被覆領域の発生を限りなく抑えることができる．よって，連結センサカバーに二連結性を保証しつつ少量のセンサ数により構築することが予想できる．

ただし，このアルゴリズムで連結センサカバーの構築を行うと，図 13 に示すように，連結センサカバーに含まれるセンサ同士で検知領域交点集合ができない場合が発生することがある．この場合は，要求領域の角を検知領域交点集合として更新する．

以下に，本手法の手順を示す．

探索フェーズ： コーディネータが持つ検知領域交点集合をもとに，これらの検知領域交点を一つでも被覆するセンサを探す．コーディネータは自身の隣接センサへ探索メッセージをブロードキャストする．探索メッセージを受信したセンサは，自身の検知領域内に検知領域交点を含むか否かにより，以下の操作を行う．

- 検知領域交点を含む場合： 前述の通信辺探索アルゴリズムを実行し，その結果により連結である場合は応答メッセージを返信する．
- 検知領域交点を含まない場合： 探索メッセージが定められたホップ数以内であれば，その探索メッセージを自信の通信近隣センサへブロードキャストする．ホップ数を超えている場合は，何もしない．

図 14 に探索メッセージをブロードキャストする様子を示す．

応答フェーズ： 応答メッセージを受信したセンサは，自身がコーディネータの場合，候補パスの評価値を算出し，候補パス集合へ追加する．コーディネータでない場合は，候補パス上の自分の一つ前のセンサへ，候補パスの情報を含むメッセージを送信する．

図 15 に応答フェーズの例を示す．図中の緑色で示すセンサは，検知領域交点集合を含み，連結センサカバーと通信辺を持つので，応答メッセージをコーディネータへ返信している．

決定フェーズ： コーディネータは，自身が持つ候補パス集合の中から，パス上のセンサが検知領域交点を被覆する候補パスのみを選出する．その中から最大評価値となる候補パスを選び，そのパスへ決定メッセージを送信する．このとき，コーディネータは自身の持つ検知領域交点集合を，決定メッセージを送信するパス上のセンサへ送信し，最終的にパスの最後尾に存在する新たなコーディネータへ検知領域交点を引き継ぐ．

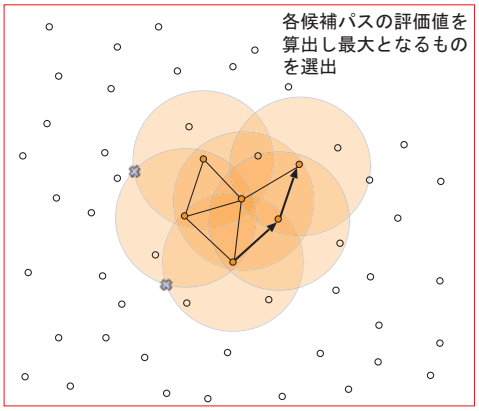


図 16: 本手法の条件を満たすセンサの選出

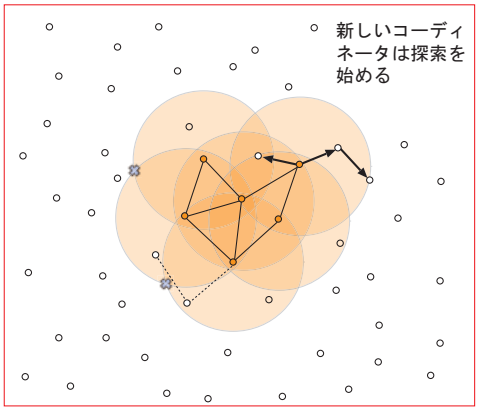


図 17: 新しいコーディネータによる探索

決定メッセージを受信したセンサは，コーディネータが持つ検知領域交点集合と自身の検知領域内に存在する検知領域交点を比較し，一致する交点を削除し，パス上の次のセンサへ新しい交点集合が付加された決定メッセージを送信する．パス上の最後のセンサが決定メッセージを受信すると，他のセンサと同様に自身の検知領域で被覆する検知領域交点を削除し，残りの検知領域交点集合 IP_k を確認する．空の場合 ($IP_k = \phi$) は，連結センサカバーに含まれるセンサを用いて検知領域交点集合の更新を行い，まだ検知領域交点が残っている場合 ($IP_k \neq \phi$) は，残りの検知領域交点集合を次ステップで被覆する検知領域交点集合 $IP_{k+1} = IP_k$ としてコーディネータとなる．

図 16 に決定フェーズの様子を示す．選出されたパス上のセンサがすべて連結センサカバーに追加され，検知領域交点集合中のパス上のセンサによって被覆される交点が削除されている．この図では，矢印が決定メッセージの経路を示し，矢印の宛先にあたるセンサが，図中の交点集合を持つ新しいコーディネータとなる．この図が示すように，連結センサカバーは常に二連結性を保証しながら構築される．

図 17 では，上記の探索により連結センサカバーに追加された新しいコーディネータが探索を開始している様子を示す．また，図中の破線は前のコーディネータから引き継いだ候補パスを示す．

以上の 3 つのフェーズを連結センサカバーが要求領域をすべて被覆するまで繰り返すことにより，二連結性を保証する連結センサカバーを構築する．

3.4 要求領域の端から中心へ構築するアルゴリズム

前述の提案手法では，任意のセンサを中心に要求領域の端へ向かって連結センサカバーを構築している．しかし，この手法では要求領域の端付近で検知領域交点ができる場合や，要求領域の角に追いやられたセンサがコーディネータとなった場合，次回の候補パスを探すために多くの処理が必要となる．

この問題点を解決するために，まず，要求領域の端を環状な連結センサカバーで被覆する手法を提案する．この手法により，要求領域の角の部分を最初に被覆でき，かつ要求領域を環状に覆う連結センサカバーに二連結性が保証されることが明らかである．

本手法は，大きく分けて以下の 3 つのパートにより構成される．

任意のセンサから要求領域の角を探索する

まず，前述の提案手法で最後に残り易い要求領域の角を最初に被覆する．このため，最初に与えられる任意のセンサから最も近い要求領域の角を被覆するセンサを探索する．この探索は，開始センサからブロードキャストメッセージを送ることにより実現し，自身の検知領域で要求領域の角を被覆するセンサがメッセージを受信すると，そのセンサがコーディネータとなる．

要求領域の端を被覆する

前述の探索によって得られたコーディネータはアルゴリズムを開始し，自身の検知領域の境界円と要求領域の辺との交点集合を保持する．コーディネータは，自身の持つ交点集合を被覆するセンサを探索メッセージを用い探索する．探索から得られたセンサを追加していくが，要求領域の端を全て被覆し終わったときに，一番最初に連結センサカバーに追加した要求領域の角を被覆するセンサと最後に追加したセンサとの間に通信辺が存在する必要がある．これを満たすため，通信辺探索アルゴリズム (3.2.2 節) により，最後に追加したセンサと最初のセンサ間が連結であるか探索し，連結でない場合は，以下のパートに示すアルゴリズムにより連結性を保証する．

要求領域の中心へ向けてセンサを追加する

要求領域の端が全て被覆され，前のパートにより二連結を保証する環状な連結センサカバーが構築されるので，前述の提案手法と同様に連結センサカバー

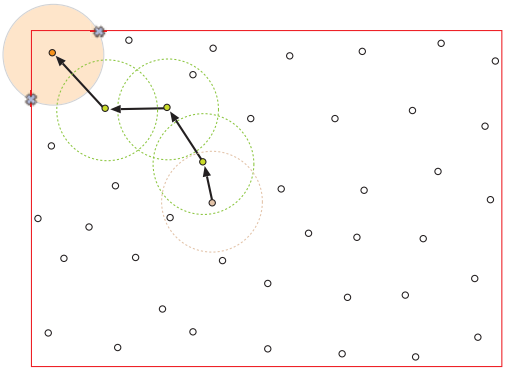


図 18: 要求領域の角を被覆するセンサの探索

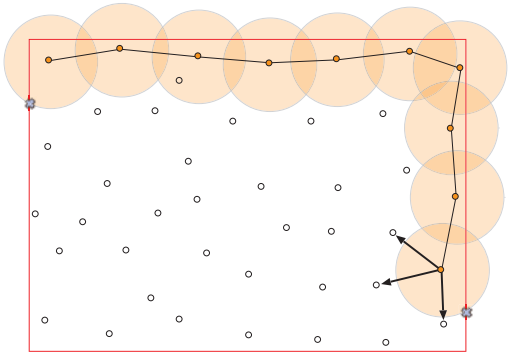


図 19: 検知領域と要求領域の端との交点を被覆するセンサの追加

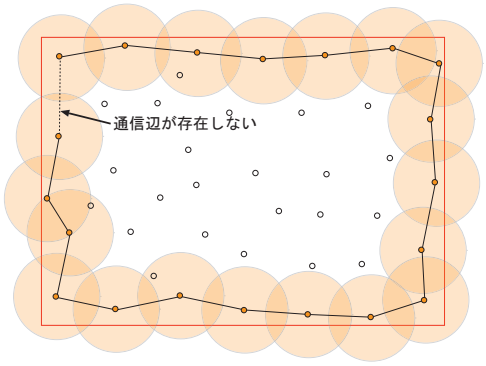


図 20: センサカバーの最初と最後のセンサ間に通信辺が存在しない状態

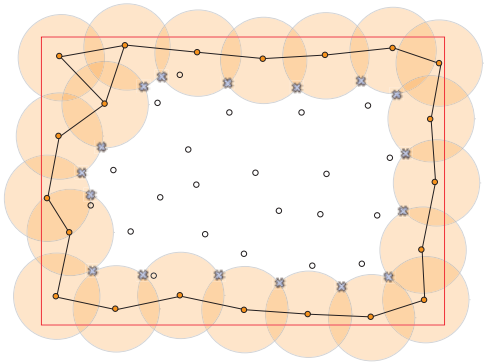


図 21: 環状二連結連結センサカバーの構築

に属するセンサの検知領域同士でできる検知領域交点を更新し，この検知領域交点を被覆するセンサを連結センサカバーに追加していくことで，連結センサカバーを拡張する．

この 3 つのパートを要求領域がすべて被覆されるまで繰り返す．

次に，実際のアルゴリズムの動作を，以下の 3 つのパートにわけて詳しく説明する，

任意のセンサから要求領域の角を探索するパート

まず，与えられる任意のセンサがコーディネータとなる．コーディネータは，探索メッセージを作成し隣接センサへブロードキャストする．メッセージを受信したセンサは，自身の検知領域が要求領域の角を一つでも被覆すればそのセンサがコーディネータとなる．被覆しない場合は，メッセージのパスへ自身を追加し，隣接センサへブロードキャストする．

図 18 に，任意のセンサから角を被覆するセンサを，メッセージ通信により探索する様子を示す．なお，この図では左上のセンサが選出されている．

要求領域の端を環状に連結センサカバーを構築するパート

上記パートにて選出されたコーディネータは，まず自身の検知領域と要求領域の端により生じる交点を更新する．このとき，センサの左右により交点集合は 2 つ存在するが，以下の探索で用いる交点集合はどちらか片方とし，もう片方の交点集合は以後使わないものとする．コーディネータは，以下のフェーズを繰り返すことにより，要求領域の端に環状な連結センサカバーを構築する．

図 19 に，以下のフェーズに従い環状な連結センサカバーを構築する様子を示す．

探索フェーズ コーディネータは探索メッセージを隣接センサへブロードキャストする．探索メッセージを受信したセンサは，自身の検知領域がコーディネータの持つ交点を被覆するなら応答メッセージを返信する．被覆する交点集合が存在しない場合は，探索メッセージを自身の隣接センサへブロードキャストする．

応答フェーズ コーディネータは，受信した候補パスから評価値を算出し，候補パス集合へ追加する．コーディネータ以外のセンサがメッセージを受信した場合は，パス上の自分より一つ前のセンサへメッセージを転送する．

決定フェーズ コーディネータは自身の持つ候補パス集

合から，最大の評価値を持つ候補パスを選び，その候補パスへ決定メッセージを送信する．決定メッセージを受信したセンサは，パス上の先頭でない場合は連結センサカバーに追加されるのみであるが，先頭の場合は連結センサカバーに追加されるとともに自身が新しいコーディネータとなる．

新しいコーディネータが選出された場合は，連結センサカバーに含まれるセンサと要求領域の端とでできる交点から，連結センサカバーに含まれるセンサで被覆される交点を除外した点で，次期コーディネータが持つ交点を更新する．このときも，前パートの要求領域の角を被覆するセンサで外した交点は除外し，コーディネータが持つ交点は１つだけとする．

以上の操作は，新しいコーディネータの交点を更新しても交点が無くなる状態，つまり要求領域の端を環状の連結センサカバーで被覆する状態まで繰り返す．

ただし，この繰り返しにより，要求領域の端は環状な連結センサカバーにて被覆されたが，この連結センサカバーが二連結性を必ずしも保証しているとは限らない．これは，要求領域の角を被覆する最初のセンサと，最後に連結センサカバーに追加されたセンサとの間に通信辺が存在していない場合が存在するからである．

図 20 に，それらのセンサ間に通信辺が存在しない様子を示す．この場合は，第 3.2.2 節の通信辺探索アルゴリズムを実行し，最後のセンサと最初のセンサ間で連結が保証できない場合は，このアルゴリズム中の長さが 3 のパスから一つセンサを追加することで，連結性を保証する．

このパートにより得られる環状な連結センサカバーを図 21 に示す．

要求領域が完全に被覆されるまで連結センサカバーを拡張するパート

前のパートにより得られたセンサ集合を連結センサカバー M_k とし，前節の要求領域交点を被覆するセンサを追加する手法を用いて，残りの連結センサカバーを構築する．

4 実験結果

本章では，Gupta らの分散型アルゴリズムと，提案手法の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モデルは，一辺の長さが 100 の正方領域を要求領域 R_Q とする．また，入力のセンサ集合はセンサ数を 1000 とし，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 $s = 10$，通信領域 T_i は通信半径 t の単一円とし，通信半径 t を 10，12，15，20 と変化させて，計測を行う．

シミュレーションはステップ方式で動作され，1 回のステップでコーディネータは自身の処理を行い，メッセージを受信したすべてのセンサは，そのメッセージに対する適切な処理を行う．

また，任意のセンサを中心に外側へ構築する手法を提案手法 1，要求領域の端から中心へ構築する手法を提案手法 2 とする．

4.2 シミュレーション結果

それぞれの提案手法をシミュレーション環境で実行した結果を，それぞれの評価指標毎に示す．

4.2.1 連結センサカバーのサイズ

図 22 に，それぞれの手法による構築センサネットワークのサイズを示す．まず，既存手法と提案手法を比較すると，提案手法は二連結性を保証しているため，既存手法に比べて全体的にセンサ数が増加している．しかし，通信半径を大きくすると，既存手法との間にそれほど大きな差は生じていない．これは，通信半径を大きくすると手法に関係なく通信可能なセンサが増え，多くのセンサを用いること無く，二連結性が保証されるからだと考えられる．

次に，提案手法の 2 種類を比べる．全体的なセンサ数を見ると，提案手法 1 より，提案手法 2 の方がセンサ数が抑えられている．これは，提案手法 1 では要求領域の端や角に微小未被覆領域が生じ易く，これを被覆するセンサが必要となるためだと思われる．

4.2.2 ステップ数

図 23 に，それぞれの手法が連結センサカバーの構築に要したステップ数を示す．まず，既存手法と提案手法を比べると，提案手法の方が既存手法より少ないステップ数で連結センサカバーを構築している．これは，提案手法では追加するセンサが一方向に延びることなく，周囲のセンサを無駄なく連結センサカバーに追加するからであると考えられる．

次に，提案手法の 2 種類を比較すると，提案手法 1 より提案手法 2 の方が，ステップ数が抑えられている．これは，連結センサカバーのサイズと同様に，微小未被覆領域が生じることがなく，それを被覆するセンサを探索するステップが省かれるからだと考えられる．

4.2.3 通信コスト

図 24 に，それぞれの手法により連結センサカバーの構築に要した通信コスト示す．通信コストは，構築のためにネットワーク上のセンサが通信を行った総計を表している．

まず，既存手法と提案手法を比べると，提案手法の方が大幅な通信コストの削減が図られている．それぞれの手法で，通信半径を大きくすると通信コストが増加して

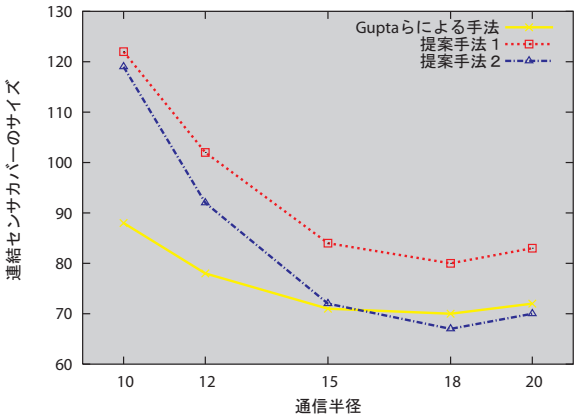


図 22: 構築センサカバーのサイズ

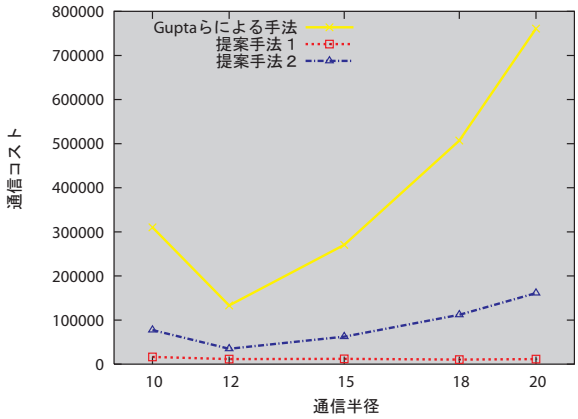


図 24: 通信コスト

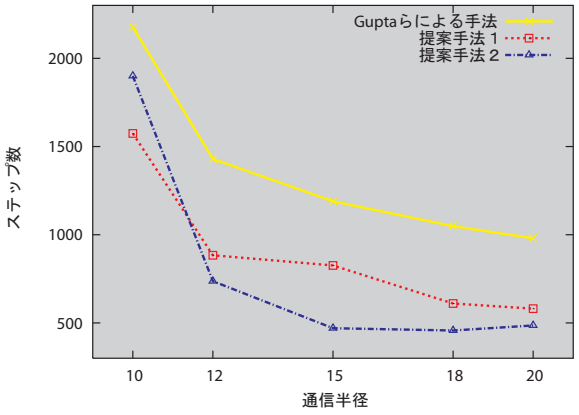


図 23: ステップ数

いるのは，探索メッセージを送信するセンサが増加するためであるが，既存手法ではそれが顕著な反面，提案手法ではそれほど差が生じていないことがわかる．これは，探索メッセージをブロードキャストする幅が，連結センサカバーに属するセンサ同士の交点を被覆する条件や，二連結性を保証するため連結センサカバーに通信辺を持つ，といった条件のために制限されたことが考えられる．

提案手法の２種類を比較すと，提案手法１の方が，コストが大幅に抑えられている．これは，提案手法２は，任意のセンサから要求領域の角を探索するステップにより，通信コストが加算され，提案手法１の方は任意のセンサから即座に連結センサカバーの構築へ移行できるためであると考えられる．

5　まとめ

本研究では，二連結性を保証する分散型連結センサカバーアルゴリズムを２種類提案した．また，提案アルゴリズムをシミュレーション環境に実装し，既存のアルゴリズムと比較した．

提案アルゴリズムは，センサカバーに追加するセンサの方向の違いから以下の２種類を提案した．

- 任意のセンサを中心に，その周りのセンサを追加することで要求領域の外側へ向けてセンサカバーを構築する手法
- 要求領域の端を被覆した後，要求領域の中心へ向けセンサを追加する手法

これらの手法によって求められる連結センサカバーは，既存手法によって求められる連結センサカバーと比べて，センサ数，構築に要するステップ数，構築に要する通信コスト，全てにおいて同等または減少している．これより既存手法と比べて，同等のセンサ数，より少ないステップ数，より少ないメッセージ通信により二連結を保証する連結センサカバーを構築できることがわかった．

今後の課題としては，構築を終えた連結センサカバー上のセンサが故障した場合を想定し，故障した付近のセンサが自動的に周囲のセンサを用い修復する機能が挙げられる．また，この機能を時間軸を取り付けた環境上で，故障，発見，探索，再構築までをシミュレーションすることなども挙げられる．

参考文献

[1] H. Gupta, Z. Zhou, S. Das, and Q. Gu. Connected sensor cover:self-organization of sensor networks for efficient query execution. *ACM/IEEE Transactions on Networking*, 14(1), 2006.

[2] 白石，藤原．”二連結性を保証する連結センサカバーアルゴリズム”．第 4 回情報科学ワークショップ, pp. 113–122, 2008.

[3] 高田，藤原．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．情報処理学会研究会報告, Vol. 2008, No. 24 (AL-117), pp. 17–24, 2008.

[4] 森川，鈴木，大下，角川，増澤．領域被覆のためのセンサネットワークアルゴリズム．情報処理学会研究会報告, Vol. 2007, No. 16 (20070301), pp. 357–362, 2007.

耐攻撃性を考慮したセンサ攻撃アルゴリズムとセンサ配置アルゴリズム

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

1　はじめに

近年，センサネットワークに関する研究が多く行われている．センサネットワークとは，特定の情報を観測する検知機能と無線による短距離通信を行う通信機能を持った小型のセンサによって構成されるネットワークである．このセンサネットワークが構築される環境の 1 つとして，戦場などのようにセンサを破壊する敵対者が存在する環境が考えられる．このような環境では，いくつかのセンサが破壊されても，センサネットワーク全体としては可能な限り与えられた領域の観測を続けるようなセンサの配置が必要となる．

このような攻撃が行われる環境においてのセンサ配置の評価指標として，耐攻撃性 [3] という概念が提案されている．この耐攻撃性は，センサ配置と破壊されるセンサの集合により定義される評価指標であり，各センサに対して被覆による利得と破壊コストを定義し，配置センサの利得の和から破壊されるセンサの破壊コストの和を減算した値として定義される．この耐攻撃性を用いると，どのようにセンサを破壊すれば最も効果的かという問題や，どのようにセンサ配置を行えば耐攻撃性が最大になるかという問題等を考えることができるようになり，センサネットワークへの攻撃に関する様々な問題を定式化することができる．

この耐攻撃性を用いた先行研究として，Kannan ら [3] は，入力として 2 次元平面上に利得点の集合を与え，その利得点の被覆を配置センサの利得として定義することにより，センサ配置の耐攻撃性を定式化している．また，この定式化において，攻撃するセンサを選択する問題は，ネットワークの最大流問題に帰着できることを示すとともに，耐攻撃性が最大となるセンサ配置を求める問題は NP 完全であることを示している．

そこで，本研究では，2 次平面上の連続領域を入力とし，その領域の被覆を配置センサの利得と考えた場合のセンサ配置の耐攻撃性について，問題の考察とアルゴリズムの提案を行う．まず最初に，領域の被覆を利得とする場合の問題について，耐攻撃性を用いた場合の定式化を行う．次に，この問題において，攻撃するセンサを選択する問題は，利得点の被覆を配置センサの利得として定義する問題への帰着により解くことができることを示す．最後に，この問題において耐攻撃性が最大となるセンサ配置を求めるアルゴリズム関する考察を行う．

2　準備

本節では，本稿で用いるセンサモデルやセンサネットワークにおける耐攻撃性の定義，及び，耐攻撃性を用いた既存の問題について説明する．

2.1　センサモデル

センサとは，通信機能，及び，検知機能を持ち，小型のバッテリーを用いて様々な環境下において動作可能な機器である．本研究では，n 個のセンサからなるセンサ集合 $S = \{s_1, s_2, \ldots, s_n\}$ を定義する．各センサ s_i は，センサを中心とした半径 r_i の円形の検知領域 m_i を持ち，その中に存在する物体，及び，現象を検知可能とする．また，各センサ s_i は二次元平面 R 上に配置されており，自身の地理的位置情報を座標 (sx_i, sy_i) として保持している．更に，各センサごとに破壊に必要なコスト $C(s_i)$，及び，センサ s_i が存在することによって得られる利得 $B(s_i)$ が定義される．これらの定義の詳細については後述する．

2.2　検知領域

二次元平面 R において，センサ s_i が被覆する領域を m_i とし，その集合を $M = \{m_1, m_2, \ldots, m_n\}$ とする．このとき，センサ集合 S が被覆する領域 $M(S)$ は以下の式で与えられる．

$$M(S) = \bigcup_{s_i \in S} m_i \tag{1}$$

次に，複数のセンサによる検知領域の重なりを表す概念として，部分検知領域の定義を行う．

定義 1. 部分検知領域
複数のセンサが存在する場合，各センサの検知領域が重なる領域が存在する．このとき，複数の検知領域の重なりにより定義される最小の領域を部分検知領域と呼ぶ．言い換えれば，同じセンサの集合により被覆される点の集合を部分検知領域として定義する．

図 1 に部分検知領域の例を示す．この例では，2 つのセンサ s_1, s_2 の検知領域 m_1, m_2 によって，検知領域が重なる領域が 1 つ存在する．このとき，センサ集合 $\{s_1\}$, $\{s_2\}$, $\{s_1, s_2\}$ により被覆される検知領域をそれぞれ部分検知領域 e_1, e_2, e_3 として定義する．

2.3　耐攻撃性

耐攻撃性 [3] とは，敵からの攻撃を想定した場合のセンサネットワーク強度を示す指標であり，センサを破壊するとしたときに，センサ配置側が失う利得から，センサ破壊に必要なコストの合計を減算したものである．これはセンサ配置側の観点から提唱された指標で，耐攻撃

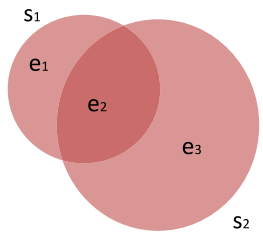


図 1: 部分検知領域

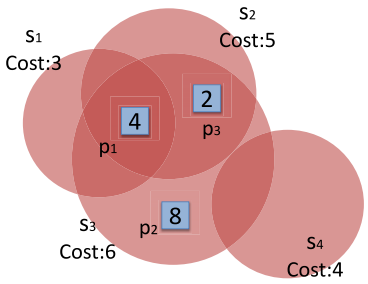


図 2: Kannan らのセンサモデル

性が大きい程，センサ破壊側が効率良くセンサを破壊できないことを表す．

以下に耐攻撃性の定義を示す．

定義 2. 耐攻撃性
センサ集合 S と S の部分集合 $L(L \subseteq S)$ が与えられたとき，L に含まれるセンサを破壊するのに必要なコストは以下の式で与えられる．

$$\sum_{s_i \in L} C(s_i) \tag{2}$$

また，センサの破壊により失われる利得は，以下の式で与えられる．

$$B(S) - B(S - L) \tag{3}$$

このとき，センサの破壊による耐攻撃性 $INT(S, L)$ は，上記の式の差として以下の式で定義される．

$$INT(S, L) = (B(S) - B(S - L)) - \sum_{s_i \in L} C(s_i) \tag{4}$$

2.4 Kannan らのセンサモデル

本節では，Kannan ら [3] によって提案されているセンサの利得モデルについて説明する．2 次元領域において，被覆により利得が発生する点の集合 $P = \{p_1, p_2, \cdots, p_m\}$ を定義し，各点 p_j を利得点と呼ぶ．各利得点 p_j は，位置情報として二次元座標 (px_j, py_j)，及び，利得 b_j を持つ．また，センサ s_i の検知領域 m_i に利得点 p_i が含まれる場合，その利得点はセンサ s_i に被覆されているとする．このとき，利得点の集合 P に対して，センサ集合 S による利得 $B(S)$ は以下の式で定義される．

$$B(S) = \sum_{p_j \in M(S)} b_j \tag{5}$$

図 2 に Kannan らのセンサモデルの例を示す．この例において，センサ集合を $S = \{s_1, s_2, s_3, s_4\}$ とすると，その利得は $B(S) = 2 + 4 + 8 = 14$ となる．一方，センサの部分集合 $S_1 = \{s_1, s_2, s_3\}$ について利得を考える場合も，$B(S_1) = 2 + 4 + 8 = 14$ となる．これはセンサ s_4 の検知領域に利得点が含まれていないためである．また，センサの部分集合 $S_2 = \{s_1, s_2\}$ について利得を考えると，$B(S_2) = 6$ となる．

2.5 最大効率センサ破壊問題

最大効率センサ破壊問題とは，耐攻撃性が最小となるように破壊するセンサ集合を求める問題であり，任意のセンサ集合 S を与えたときに，以下の式を満たすセンサ集合 L_{max} を求める問題である．

$$INT(S, L_{max}) = \min\{INT(S, L) \mid L \subseteq S\} \tag{6}$$

センサ破壊側にとって，本問題の解となるセンサ集合を破壊することが，破壊コストとセンサ配置側が失う利得の差が最大となる結果を得ることができる．

この最大効率センサ破壊問題は，Kannan らのセンサモデルを用いる場合は，以下の手順により，2 部グラフの最大流問題に $O(min(nm'^2, mm'^2))$ 時間で帰着できることが示されている [3]．ここで，n はセンサ数，m は利得点の総数，及び，m' はセンサに被覆されている利得点の数を表す．

Step 1：センサ，及び，利得点をグラフの頂点とし，各頂点間に，実際のセンサと利得点の被覆関係に基づき利得点からセンサ方向へ辺を作成する．また，その辺の重みを無限大とする．ただし，どのセンサにも被覆されていない利得点，及び，利得点を被覆していないセンサは無視するものとする．

Step 2：最大流問題における始点，終点として頂点 X，Y をそれぞれ作成する．始点 X から全ての利得点を表す頂点に向けて有向辺を作成し，その重みは利得点を表す端点の利得とする．同様に，全てのセンサを表す頂点から頂点 Y に向けて有向辺を作成し，その重みをセンサを表す端点の破壊コストとする．

Step 3：作成した重みつき有向グラフに対して，最大流問題を解くアルゴリズムを用い，最大流と最小カットを求める．

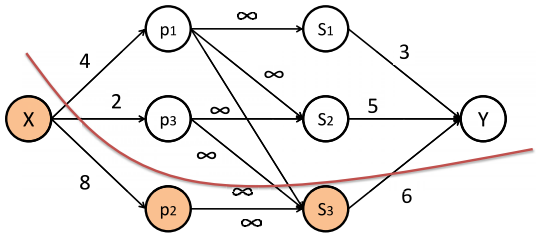


図 3: 最大流問題への帰着

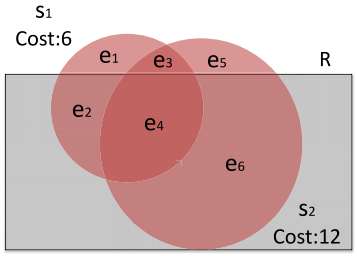


図 4: センサモデル

図 3 に図 2 のセンサ集合を入力として帰着を行った例を示す．この例ではセンサ集合が $\{s_1, s_2, s_3\}$，利得点の集合が $\{p_1, p_2, p_3\}$，及び，各辺にある値が辺の重みとなる．上記手順の Step 1 より，センサ s_4 は利得点を被覆していないためにグラフ上には存在しない．

この図 3 のグラフに対して最大流を考えると，最小カットは $\{X, p_3, s_3\}$ と $\{p_1, p_2, s_1, s_2, Y\}$ となる．最大効率センサ破壊問題では，図中の赤線を跨ぐ辺を持つセンサ，言い替えると，頂点 X が含まれるカットに所属するセンサを破壊することが最も効率の良いセンサの破壊戦略となり，本例における最大効率センサ破壊問題の解は $\{s_3\}$ である．

このときセンサ破壊に必要なコストは 6，センサを破壊することでセンサ配置側が失う利得は 8 であり，耐攻撃性は -2 となる．

2.6 最大耐攻撃性問題

最大耐攻撃性問題とは，センサ配置側の問題として，耐攻撃性を最大とするセンサ配置を求める問題であり，センサ集合 S を与えたときに，以下の式を満たすセンサの配置 S_{max} を求める問題である．

$$INT(S_{max}, L) = \max\{INT(S, L) | \subseteq S\} \tag{7}$$

この最大耐攻撃性問題は Kannan らのモデルにおいて NP 完全であると証明されている [2]．そこで，本研究では，後述する提案モデルにおいて最大耐攻撃性問題を解くアルゴリズムについて考える．

3 提案モデル

本節では，本研究にて提案するセンサモデル，及び，提案モデルにおけるセンサ問題の説明を行う．

3.1 提案センサモデル

本モデルは，広大な土地全体の観測等の状況を考慮して作成したモデルである．まず，入力として被覆すべき領域である要求領域 R を定義する．この要求領域 R に対して，センサ集合 S の利得を，S に含まれるセンサの検知領域により被覆される要求領域 R の面積として定義する．

以下に，要求領域 R に対して，センサ集合 S による利得 $B(S)$ を定義する式を示す．

$$B(S) = \left|M(S) \cap R\right| \tag{8}$$

また，センサ s_i の破壊コスト $C(s_i)$ は検知領域の面積の定数倍として以下の式で定義される．

$$C(s_i) = \alpha\ r_i^2 \quad (0 < \alpha) \tag{9}$$

例として，図 4 に提案センサモデルの例を示す．領域全体は各センサの検知領域と要求領域 R によって部分検知領域化されており，センサ s_1 が被覆しているの部分検知領域は $\{e_1, e_2, e_3, e_4\}$ となる．しかし，上記の式 (8) で示されるように，センサ s_1 の利得計算に用いられる部分検知領域は要求領域 R 内に存在する $\{e_1, e_2\}$ となるため，$B(s_1) = |e_2| + |e_4|$ となる．

なお，重複して被覆された領域の利得は，一度しか利得として加算されない．そのため，センサ集合を $S = \{s_1, s_2\}$ とすると，その利得は $B(S) = |e_2| + |e_4| + |e_6|$ となる．

3.2 提案モデルにおける最大効率センサ破壊問題

提案モデルにおける最大効率センサ破壊問題を考える．提案モデルにおいても，Kannan らのモデル同様に，式 (6) を満たすセンサ集合が本問題の解となる．

以下では，提案モデルにおける最大効率センサ破壊問題が，Kannan らのモデルにおける最大効率センサ破壊問題に多項式時間で帰着可能であることを示す．

（Kannan らのモデルへの帰着）

まず最初に，センサ集合 S からすべての部分検知領域を求める．なお要求領域 R も部分検知領域を考える際の要素をとして加える．例として，図 4 にセンサ集合と

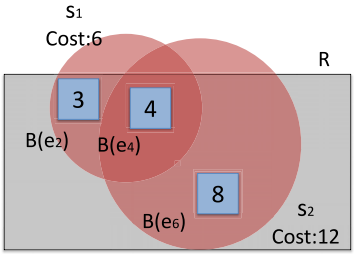


図 5: 部分検知領域へ内への利得点配置

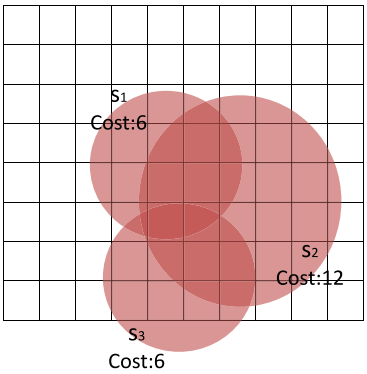


図 6: 正方領域を用いた近似

要求領域から構成される部分検知領域を示す．この場合は，6 個の部分検知領域で構成されている．この部分検知領域は，センサ数が n の場合，個数が $O(n^2)$ となるので，最大 $O(n^2)$ 時間で求めることができる．

次に，各部分検知領域に対して面積を求める．各部分検知領域は，最大 n 個の円の重なりで構成されているので，$O(n^2)$ 時間でその面積を計算することができる．したがって，すべての部分検知領域の面積の計算には $O(n^3)$ 時間で可能である．

最後に，要求領域 R に含まれる各部分検知領域内に，その部分検知領域の面積と同じ値の利得点を配置する．図 5 に図 4 の入力から面積を利得点に変換した例を示す．これにより，問題が Kannan らのモデルにおける最大効率センサ破壊問題となり，この問題を解くことにより，提案モデルにおける問題の解を得ることができる．

3.3 提案モデルにおける最大耐攻撃性問題

3.2 節の帰着で示したように，提案モデルにおける最大耐攻撃性問題は Kannan らのモデルにおける最大耐攻撃性問題に多項式時間で帰着できるので，提案モデルによる最大耐攻撃性問題は，Kannan らのモデルによる場合よりも容易であることがわかる．しかしながら，本モデルによる問題が NP 完全か否かはわかっていないため，まずその部分に関する考察が必要である．また，NP 完全か否かに関わらず，センサ配置を求めるアルゴリズムの提案は非常に重要であると考えられるので，近似解を求めるセンサ配置アルゴリズムを現在考察中である．

4 シミュレーション環境の構築

本節では，耐攻撃性に関するアルゴリズムのシミュレーションを行うための環境について説明する．

4.1 被覆面積の近似

各部分検知領域の面積の計算を簡略化するため，領域を面積が 1 の正方領域で近似する．このとき，提案モデルにおいて各センサの被覆する面積の大きさは，被覆する正方領域の数で表現できる．また，本実験においては，センサが正方領域を被覆しているかどうかの判断に格子点を用いる．そのため，図 6 における格子点に利得 1 の利得点を配置し，被覆する利得の合計を被覆面積として計算を行う．

4.2 シミュレーション環境

LEDA[1] と呼ばれるライブラリを種々の演算に用いる．また，同ライブラリによってグラフィカルに実行結果を表示することで，センサの選択過程や，センサ配置等を視覚的にも理解しやすいものとなるようにする．

4.3 シミュレーション結果

現在までのところ，Kannan らのモデルと提案モデルの両方について，最大流問題への帰着により最大効率センサ破壊問題を解くアルゴリズムの実装を行っている．

図 7，及び，図 8 にシミュレーションの様子を図示する．図中の赤い円はセンサの検知領域，青い小円はグリッド近似における利得 1 の利得点を示している．図 7 の入力に対して最大効率センサ破壊問題の解のセンサを破壊したものが図 8 となっている．

5 まとめと今後の課題

本研究では，センサネットワークにおける耐攻撃性に関する問題について，モデルの提案とアルゴリズムに関する考察を行った．まず，耐攻撃性に関する新しいモデルの提案を行い，提案モデルの最大効率センサ破壊問題が，既存のモデルの同じ問題に帰着できることを示した．また，最大効率センサ破壊問題について，既存モデルと提案モデルの両方について，最大流問題への帰着により解を求めるアルゴリズムの実装を行った．

今後の課題として，現在提案しているモデルにおいて，最大耐攻撃性問題を解くアルゴリズムの提案を行っている．また，現在提案しているモデルの他にも，センサを

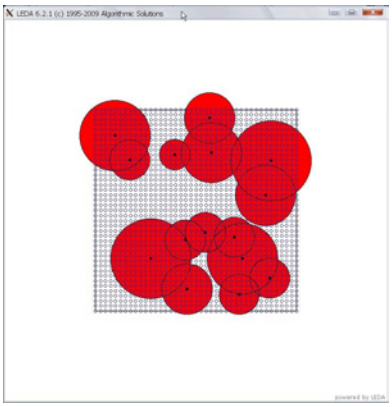


図 7: ショミレーションにおける入力

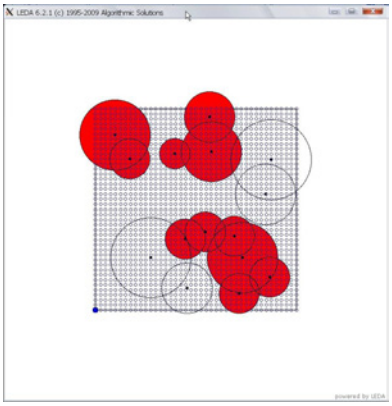


図 8: シュミレーションにおける最大効率センサ破壊問題の解

初期配置から選択するのではなく，使用できるコストの上限を決定して，配置していく場合等，様々な条件が考えられる．そのため，現在のモデルにおいて最大耐攻撃性問題を解くアルゴリズムの提案後に，更なるモデルを提案することも今後の課題である．

参考文献

[1] LEDA:http://www.algorithmi−csolutions.com/leda/.

[2] K. Chakrabarty, S. S. Iyengra, H. Qi, and E. C. Cho. Integrity and its applications. *Proceedings of the 45th annual southeast regional conference*, 2007.

[3] R. Kannan, S. Sarangi, S. Ray, and S. S. Iyengra. Minimal sensor integrity in sensor grids. *Proceedings of the 2002 International Conference on Parallel Processing*, 2002.

An Experimental Evaluation of Clustering Algorithm using Attractor Selection

Naoko Uemura†, Gen Nishikawa †, Fukuhito Ooshita †, Hirotsugu Kakugawa †
and Toshimitsu Masuzawa †

†Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka Suita, Osaka, 565-0871 Japan
Email: { n-uemura | g-nisikw | f-oosita | kakugawa | masuzawa } @ist.osaka-u.ac.jp

Abstract—Clustering is a problem to partition a network into small groups of nodes called clusters. Clustering is useful for maintaining large-scale networks because it gives hierarchical structure to the networks. Nishikawa et al. proposed a clustering algorithm based on attractor selection, which is one of the biologically inspired approaches to attain adaptability to dynamics of environments, and they showed the effectiveness of the algorithm by simulations. However, since the simulation settings are based on a theoretical model, the effectiveness in real environments is not evaluated. In this paper, we evaluate Nishikawa's algorithm in realistic environments. First, we implement Nishikawa's algorithm in a real sensor network composed of 36 sensor devices SunSPOTs and show that the algorithm is efficient in the environment. Second, we determine realistic simulation settings by analyzing the behavior of the above-mentioned real sensor network. Using the simulation settings, we show that Nishikawa's algorithm is efficient in large-scale sensor networks.

1. Introduction

With the spread of wireless technologies, mobile ad hoc networks (MANETs) are getting increased attention in recent years. Since MANETs can be established without any fixed infrastructure, they are often used to construct temporary networks in the situation where it is difficult to use any fixed infrastructure. These applications often consist of a lot of nodes, and thus it is important to manage large MANETs efficiently.

Clustering is a fundamental technique to manage large MANETs. Clustering divides a network into several groups of nodes called *clusters*. Each cluster consists of one cluster head (representative node) and several ordinary nodes. Clustering facilitates managing networks because networks can utilize hierarchical structures composed of inter-cluster networks and intra-cluster networks. For this use, clusters should satisfy some conditions. First, the number of clusters should be minimized. This is because the size of the inter-cluster network depends on the number of clusters. Consequently a smaller number of clusters make it easier to manage networks. Second, each ordinary node should be able to directly communicate with its cluster head. This is because each cluster head maintains information of ordinary nodes in its cluster. This requires a lot of communications between a cluster head and ordinary nodes.

On the other hand, in MANETs, network topologies change frequently since each node moves freely. Since desirable clusters depend on network topologies, clusters are required to be reconstructed when topology changes occur. Consequently, frequent topology changes cause a lot of overhead to reconstruct clusters. Thus, it is necessary to reduce the reconstruction overhead.

Johnen et al. proposed a clustering algorithm for weighted networks where each node has a weight[2]. A weight of a node represents suitability to become a cluster head (e.g. computing capacity, amount of the battery, bandwidth of wireless communication etc). Johnen et al. aim to maximize the average weight of cluster heads as well as satisfy the conditions described above. Johnen's algorithm uses a fixed threshold to reduce the reconstruction overhead. This causes useless reconstructions to Johnen's algorithm in some situations.

Nishikawa et al. adopt *attractor selection* scheme instead of fixed threshold[5]. Attractor selection is one of the biologically inspired approaches to attain adaptability to dynamics of environments[3]. The underlying mechanism of attractor selection is as follows: The system changes its state randomly while the system state is bad. Then, if the system state becomes good, the system keeps the good state with high probability. The appealing feature of this mechanism is that it is highly noise-tolerant and can even be stimulated by noises[4]. Nishikawa et al. regarded frequent topology changes as noises and applied attractor selection to MANETs. They show by simulations that their algorithm reduces the reconstruction overhead compared to Johnen's algorithm. However, since the simulation settings are based on a theoretical model, the effectiveness in real environments is not evaluated.

In this paper, we evaluate Nishikawa's algorithm in realistic environments. First, we implement Nishikawa's algorithm in a real sensor network composed of 36 sensor devices SunSPOTs and show that the algorithm is efficient in the environment. Second, we determine realistic simulation settings by analyzing the behavior of the above-mentioned real sensor network. Using the simulation settings, we show that Nishikawa's algorithm is efficient in large-scale sensor networks.

2. Model

A network is represented by an undirected graph $G = (V, E)$, where V is a set of nodes and E is a set of bidirectional communication links. Each node v has a unique identifier $id(v)$. For simplicity, we use v and $id(v)$ interchangeably. We say nodes v and u are neighbors with each other if and only if link $(v, u) \in E$. We denote a set of neighbors of v as N_v. We assume that each node communicates by a *locally shared memory model*. That is, each node v has a finite set of local variables that can be read by nodes $u \in N_v \cup \{v\}$ and can only be updated by v. These assumptions are the ones used in Nishikawa's algorithm [5] and Johnen's algorithm [2].

Clustering is a problem to divide a network into several *clusters* such that each node belongs to exactly one cluster. In this paper, we consider clustering in mobile ad hoc networks where nodes can move freely. Thus, network topology changes arbitrarily and might be disconnected. Valid clustering in mobile ad hoc networks is defined as follows.

DEFINITION 1 *Valid clustering*

Clustering is represented by $C = (\overline{V}, H)$, *where* $\overline{V} = \{\overline{V}_1, \overline{V}_2, \ldots, \overline{V}_{|H|}\}$ $(\overline{V}_i \subseteq V)$ *is a set of clusters and* $H = \{h_1, h_2, \ldots, h_{|H|}\}$ $(h_i \in \overline{V})$ *is a set of nodes. If clustering* $C = (\overline{V}, H)$ *satisfies the following properties, we say C is a valid clustering.*

1. $\bigcup_i \overline{V}_i = V$.
2. *for every* $i, j,\ i \neq j : \overline{V}_i \cap \overline{V}_j = \emptyset$.
3. *for every* $h_i, h_i \in \overline{V}_i$.
4. *for every* $i, \forall v \in \overline{V}_i - \{h_i\} : h_i \in N_v$.

Then, we say that h_i *is a cluster head of* $\overline{V}_i$ *and* $v \in \overline{V}_i$ *belongs to cluster head* h_i.

We consider the weighted network where each node has a weight value which represents suitability to become a cluster head. A weight of a node is determined appropriately from its battery capacity, computing capacity, or bandwidth, etc. We denote the weight of v as w_v. The node with a larger weight is more suitable as cluster head. It is also preferred to minimize the number of clusters and the number of cluster changes since a large number of clusters and frequent changes of clusters cause enormous overhead. Note that, if one wants to maximize the average weight of cluster heads and minimize the number of clusters, nodes have to reconstruct clusters more frequently and suffer from reconstruction overheads. There is a trade-off between quality of clustering (weights of cluster heads and the number of clusters) and reconstruction overheads [5].

3. Existing algorithms

In this section, we explain Johnen's algorithm and Nishikawa's algorithm.

3.1. Johnen's algorithm

Johnen's algorithm aims to reduce reconstruction overheads by updating clusters loosely. That is, even when quality of clusters is degraded by topology changes, each node does not change clusters if the degradation is below a threshold. In more details, Johnen's algorithm defines strongly-valid clustering as follows and constructs them.

DEFINITION 2 *Strongly valid clustering*

If valid clustering C satisfies the following property, C is a strongly valid clustering.

1. *Every ordinary node v belongs to one cluster head whose weight is larger than or equal to v.*
2. *For a given real number h, every ordinary node v whose cluster head z has weight* w_z, *and every cluster head* $u \in N_v$, *the weight of u is not larger than* $w_z + h$.
3. *For a given integer k (*$0 \leq k < n$*), every cluster head has at most k neighboring cluster heads.*

By using thresholds k and h, Johnen's algorithm can reduce the number of cluster changes. However, Johnen's algorithm can not avoid frequent changes of clusters around the threshold.

3.2. Nishikawa's algorithm

Nishikawa et al. proposed a clustering algorithm using attractor selection[5]. Attractor selection is a biological model first introduced by Kashiwagi el al.[3]. Attractor selection works according to the following mechanism.

1. Each node evaluates its current state.
2. If the current state is good, the node keeps its state with high probability.
3. If the current state is bad, the node changes its state with high probability.

The main feature of attractor selection is using random noise which corresponds to an inherent noise term found in the original gene expression model. This random noise term causes the system to be constantly in motion. However, once it has converged to an attractor (good state), it remains there as long as the attractor is stable.

The selection of the appropriate attractor is controlled by an *activity* α $(0 \leq \alpha \leq 1)$. When the current state is good, α approaches to 1. When the current state is bad, α approaches to 0. If α approaches to 1, the influence of random noise becomes smaller, and a node keeps its state with high probability. If α approaches to 0, the influence of random noise becomes bigger, and a node changes its state with high probability.

In Nishikawa's algorithm, each node independently operates its attractor selection. Activity α of node v reflects the weight of v and $u \in N_v$ and the number of cluster heads in N_v. When there exists $u \in N_v$ which is more suitable for cluster head than v's current cluster head, the activity

α becomes small. For example when the weight of cluster head $u \in N_v$ is much larger than that of v's cluster head, or v is a cluster head but has a large number of cluster heads in N_v, activity α becomes small and v changes its state with high probability. By using attractor selection, Nishikawa's algorithm constructs more stable clustering.

4. Experiments and Simulations

In this section, we explain experiments and simulations. We compare Nishikawa's algorithm with Johnen's algorithm.

4.1. Settings of experiments and simulations

In our experiments, we construct an actual sensor network with 36 SunSPOT devices, which are wireless sensor devices developed by Sun Microsystems[6]. Since SunSPOT devices do not have shared memories, we apply *message passing model* instead of shared memory model. It is shown in [1] that algorithms designed for shared memory model can be easily transformed into algorithms for the message passing model by CST(Cached Sensornet Transformation). In the transformed algorithms, each node v repeats a round. In each round, node v broadcasts its updated state. When v receives a state from u, v preserves u's state in its cache. Node v takes an action based on its own cache instead of neighbor's current state. We set a round time as 500 msec.

In our experiments, we put all nodes statically within the communication range, however we introduce virtual locations and simulate topology changes caused by node's mobility. We give a virtual location to each node and move the virtual location with time. On the basis of virtual locations, each node judges whether it can receive a packet or not. That is, even when a node receives a packet from another node, it discards the packet if their virtual locations are not within the (virtual) communication range.

In all of our experiments, each node's communication radius is 25. Each node moves around a given square field. Then we set the field size and the number of nodes at each experiment. In addition, we randomly set weight of each node within [50..100]. We adopt RWP (random way point) model, which is a commonly-used mobility model. We execute 200 rounds in each experiment, and perform 100 experiments with different initial deployments.

We also run computer simulations to evaluate algorithms in large-scale networks. The settings of computer simulations are similar to those of experiments with SunSPOT devices.

We define *the number of clusters* and *the number of changes* as evaluation criteria. The number of clusters means the average number of cluster heads per a round over all the experiments. Then, the number of clusters reflects quality of clustering. If an algorithm constructs a small number of clusters, the network is divided efficiently. The number of changes means the average number of nodes which change its state per a round over all the simulations. Then, the number of changes reflects stability of clustering. We also evaluated the average weight of cluster heads, however the values of both algorithms are almost the same and thus we omit the results.

4.2. Results of experiments and simulations

Result 1 We implement Nishikawa's algorithm and Johnen's algorithm on the actual environment with 36 SunSPOT devices. We set field size as 100×100. We show the result in Figure 1. Each algorithm can take some parameters, and each point in the figure shows the behavior for the parameter. When numbers of clusters of the two algorithms are the same, numbers of changes of Nishikawa's algorithm are smaller than those of Johnen's algorithm. Thus, we can say Nishikawa's algorithm is effective in an actual sensor network.

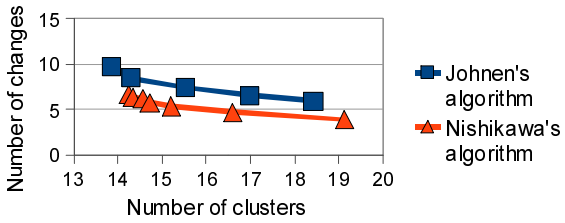


Figure 1: Experimental result using SunSPOT devices

Result 2 In actual sensor network, a lot of factors such as interference affect the behavior of algorithms. We have shown that Nishikawa's algorithm works better in such networks. The number of our SunSPOT devices is only 36, however, general sensor networks consist of a larger number of sensors. Thus, to construct realistic simulation settings for large networks, we analyze some important parameters feature of the experiment environment.

First, we hypothesize that probability of communication loss is an important factor. Actually, in the experiments with SunSPOT devices, each packet is delivered with probability 75% on average. To investigate the effect of the communication loss, we conduct simulations where each packet is delivered with possibility 75%. We show the results in Figure 2. As the result shows, we still find a considerable difference between the experimental environments and simulation environments.

Next, we focus on the communication success ratio of each node pair. By checking the communication success ratio in the previous experiments, we find that the communication success ratio depends on node pairs (See Figure 3). Remind that the success ratio is 75% on average. However, as the figure shows, there are many node pairs that have much lower success ratio or higher success ratio. We consider the effect of this fact as follows. When the success ratio depends on node pairs, packets are often lost for some node pairs and seldom lost for some node pairs. The former node pairs keep the link between them unestablished for a

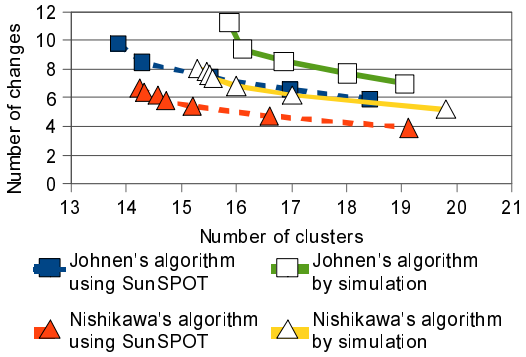


Figure 2: Comparison result under the same communication probability

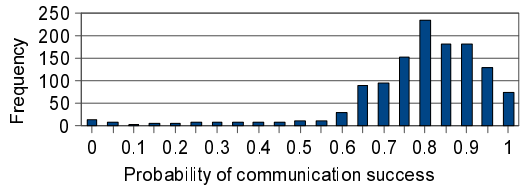


Figure 3: Distribution of communication success probability

long time, and the latter node pairs keep the link between them established for a long time. These behaviors make the network topology more stable, and thus number of changes become smaller in the experimental environments. Then we simulate both algorithms under distribution of Figure 3. We show this result in Figure 4. The simulation results with the new settings becomes closer to those by experiments with SunSPOT devices.

Result 3 Lastly, we simulate two algorithms in large-scale networks where the distribution of the success ratio is the same as that of Figure 3. We set the number of nodes as 1000 and field size as 500×500. We show the results in Figure 5. According to Figure 5, we find that Nishikawa's algorithm attains smaller number of changes than Johnen's algorithm in a large-scale network. Thus, Nishikawa's algorithm is also efficient in large-scale networks.

5. Conclusion

In this paper, we have evaluated Nishikawa's algorithm in realistic environment, and determined realistic simulation settings. By using 36 SunSPOT devices, we have compared Nishikawa's algorithm with Johnen's algorithm under influences of actual environment. The result shows Nishikawa's algorithm is useful in actual sensor networks. Then, we have analyzed the feature of experimental environment and determined realistic simulation settings. By this simulation, we have shown Nishikawa's algorithm is also useful in large-scale sensor networks.

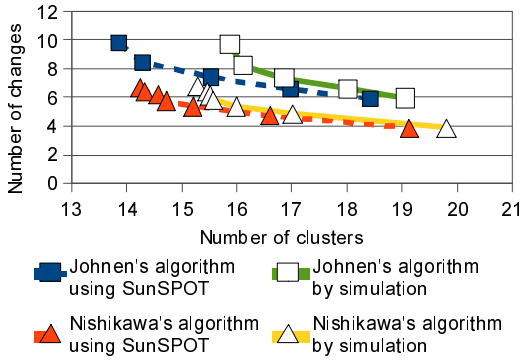


Figure 4: Comparison result under the same distribution

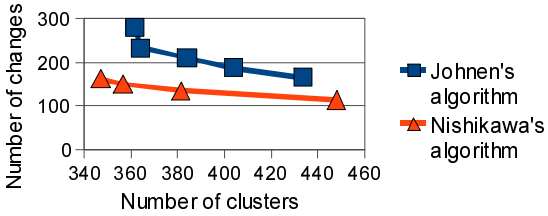


Figure 5: Simulation result in large-scale networks

Acknowledgments

This work is supported in part by Global COE Program of MEXT, Grant-in-Aid for Scientific Research((B)19300017, (B)20300012) of JSPS, and Kayamori Foundation of Information Science Advancement.

References

[1] T. Herman. Models of Self-Stabilization and Sensor Networks. In *Proc. of the International Workshop on Distributed Computing(IWDC)*, pp. 205–214, 2003.

[2] C. Johnen and L. H. Nguyen. Robust self-stabilizing clustering algorithm. In *Proc. of the 10th Intern Conference On principles of Distributed Systems*, 2006.

[3] A. Kashiwagi, I. Urabe, K. Kaneko, and T. Yomo. Adaptive response of a gene network to environmental changes by fitness-induced attractor selection. *PLoS ONE*, 1(1):e49, 12 2006.

[4] K. Leibnitz, N. Wakamiya, and M. Murata. Biologically inspired self-adaptive multi-path routing in overlay networks. *Commun. ACM*, 49(3):62–67, 2006.

[5] G. Nishikawa, F. Ooshita, H. Kakugawa, and T. Masuzawa. A stable clustering algorithm for mobile ad hoc networks based on attractor selection. In *Proc. of the 1st Intern Workshop on Technologies for Ambient Information Society*, November 2008.

[6] Sun Microsystems. SunSpotWorld website'. `http://www.sunspotworld.com/`.

Multiple Mobile Agents Rendezvous in Trees

Daisuke Baba[†] Tomoko Izumi[‡] Fukuhito Ooshita[†] Hirotsugu Kakugawa[†]
Toshimitsu Masuzawa[†]

[†]Graduate School of Information Science and Technology, Osaka University, Osaka, Japan.
{d-baba, f-oosita, kakugawa, masuzawa}@ist.osaka-u.ac.jp, Fax: +81-6-6879-4119
[‡]College of Information Science and Engineering, Ritsumeikan University, Shiga, Japan.
izumi-t@fc.ritsumei.ac.jp, Fax: +81-77-561-3445

Abstract

This paper reveals the relation between the time complexity and the space complexity for the rendezvous problem with k agents in asynchronous tree networks. The rendezvous problem requires $k \geq 2$ agents to meet at a single node. We propose three algorithms for the rendezvous problem. All the algorithms guarantee that all the agents meet at a single node if the tree is not symmetric, otherwise gather at two neighboring nodes. In this paper, we firstly present two time priority algorithms for the rendezvous problem. One has time complexity $O(n)$ and space complexity $O(n \log n)$ per agent. The other has time complexity $O(n \log n)$ and space complexity $O(n)$. Lastly, we present the asymptotically space-optimal one. This algorithm has space complexity $O(\log n)$ and time complexity $O(\Delta n^8)$ where Δ is the maximum degree of the tree.

Keywords: mobile agent, rendezvous, tree, time complexity, space complexity

1 Introduction

1.1 Background and Motivation

In this paper, we are interested in the relation between the time and the memory size for each mobile agent to solve the rendezvous problem. In the problem, each agent, which is initially distributed in a network, has to meet on a single node. The rendezvous problem is one of the fundamental problems that are required for a lot of agent systems. For example, an application may require rendezvous to share the information of all the agents. In another case, rendezvous may be needed to synchronize the process of each agent.

To solve the rendezvous problem is easy if each node in a network has a unique identifier or ID: Each agent explores the network and terminates at the node with the smallest ID. However, such unique ID may not be available for the agents in some reasons. It is prohibited to publish the unique ID to agents for security reasons, or the agents cannot perceive it because of its small memory. Hence, it is important to design algorithms which work in anonymous networks.

In anonymous networks, agents using the same deterministic algorithm cannot meet at a single node if there are cycles in the network. The problem can be feasible with some additional assumptions such that agents can leave marks on a node or the network topology

is restricted. We are interested in the rendezvous problem where agents with the same deterministic algorithm meet without leaving marks. To solve the problem in this model, we restrict the network to a tree. However, if the tree is symmetric, agents cannot rendezvous at a single node. In some cases, each agent is required to terminate without meeting if the tree is symmetric. This stronger task is also called *rendezvous with termination* problem. We deal with rendezvous with termination problem in this paper: All the agents terminate at a single node if the tree is not symmetric, otherwise the agents terminate at two neighboring nodes. For the purpose of convenience, we simply describe rendezvous with termination problem by *rendezvous* problem. We present three algorithms for the rendezvous problem with arbitrary number of agents on arbitrary tree networks in asynchronous systems.

1.2 Related Work

Since the rendezvous problem was introduced in [15], this problem has been studied by a number of researchers (read [3] for details). The hardness of the rendezvous problem almost lies in how to break symmetry in an anonymous network. Typically, randomized algorithms or different deterministic algorithm for each agent are used to break these symmetry [2, 4]. Baston et al. considered the case that agents can mark their starting nodes, however they still rely on the randomization or different deterministic algorithms [5].

In the area of anonymous networks and anonymous agents, identical tokens are often used to solve the rendezvous problem since Kranakis et al. showed that two agents in a ring network can meet by the same deterministic algorithm using a token [14]. These tokens are indistinguishable with others. Once these tokens are put on a node, they cannot move by themselves. According to the model, agents can take up these tokens to put on the other nodes. If one token is available for each agent, the rendezvous problem in a synchronous ring is solvable [9, 11]. The lower bound of the space complexity in this model is $\Omega(\log k + \log\log n)$ where k is the number of agents and n is the number of nodes, and the asymptotically space-optimal algorithm is proposed for uni-directional ring networks in [11]. The effect of token failure is also considered in [6, 7, 8].

If the underlying network is a tree, agents do not have to use even a single token. In [10], Fraigniaud et al. proved that two anonymous agents with no token can rendezvous in any synchronous tree network unless the tree is symmetric. The memory size of this algorithm is $O(\log n)$ and this is asymptotically optimal. However, if the number of agents is larger than two or the agents move asynchronously, this algorithm does not work.

1.3 Our Results

In this paper, we present three algorithms for the rendezvous problem with k agents in asynchronous tree networks. Table 1 shows the results of this paper (The mark * is used to indicate that the value is asymptotically optimal). Each of these performances is asymptotically time-optimal, asymptotically space-optimal, or balanced in terms of time and space. All these algorithms are based on an idea: every agent meets at the center node of the tree. If the agents have $O(n \log n)$ memory space, rendezvous is done in linear time, which is asymptotically time-optimal. The performance of second algorithm is balanced in terms of time and space. That is, the time complexity is $O(n \log n)$ and the space complexity is $O(n)$. Finally, we present a space-efficient rendezvous algorithm. In this algorithm, each agent needs only $O(\log n)$ memory space to solve the problem. This memory complexity is asymptotically optimal since the lower bound for the rendezvous problem on trees is $\Omega(\log n)$ [10].

Table 1: our results

Algorithm	**Time**	**Space**
Asymptotically time-optimal	$O(n)$	$O(n \log n)$
Balanced	$O(n \log n)$	$O(n)$
Asymptotically space-optimal	$O(\Delta n^8)$	$O(\log n)$

2 Terminology and Preliminaries

2.1 The Network Model

We consider an anonymous tree network $T = (V, E)$ where V is the set of nodes and E is the set of undirected edges. Let $n = |V|$ be the number of nodes and $m = |E|$ be the number of edges. A tree network is an arbitrary connected network with no cycle and thus $m = n - 1$. The network is anonymous in the sense that nodes and edges have no distinct identifier. Every node v has some ports, each of which connects to an edge. The number of ports node v has is denoted by (v) or *degree* of v. Every port at node v has a *port number*, and it is locally distinguishable. Each edge $e = \{u, v\}$ has two *labels* $_u(e)$ and $_v(e)$, which denote the port number at u and v respectively. We use *label* or *port number* interchangeably. These labels or port numbers are assigned locally at each node and thus there is no coherence between $_u(e)$ and $_v(e)$. Without loss of generality, we assume that every port number at a node is selected from the set $\Lambda = \{1, \ldots, O(n)\}$. Each port number has two means of expression, one is the *number* itself and the other is the *order* at the node. In the latter case, we call the port with i-th smallest number as *i-th port* and the edge adjacent to the i-th port at node u as *i-th edge* of u.

There are $k \geq 2$ anonymous agents in the tree. Each agent has no identifier and bounded amount of memory. Each node can host at most k agents, but it does not provide agents with any whiteboard. Each agent initially stays at a node called its *home node* and starts the same deterministic algorithm at any time. The agents have no priori knowledge about the network and other agents, that is, they do not know n, k, the shape of the tree, or where other agents are. After the algorithm is started, the agent can move in the network by the following three operations such that: 1) When the agent walks across an edge e into node v (resp. immediately after the agent initiates the algorithm at node v), it remembers $_v(e)$ (resp. 0), the order of $_v(e)$ at node v (resp. 0) and the degree of v. 2) The agent computes internally at v, and determines the port number it leaves next or notices that it should terminate. 3) If the agent decides to move to the neighboring node, it leaves node v through the port which is determined by the previous operation. Their action, such as computing or moving, is progressed asynchronously in the sense that the processing period is finite but there is no assumption of the upper bound on the length of the period.

2.2 Definition of Terms and Problem

We give definitions of terms used in this paper. The *path* $P(v_0, v_k) = (v_0, v_1, \ldots, v_k)$ with length k is a sequence of nodes from v_0 to v_k such that $\{v_i, v_{i+1}\} \in E$ $(0 \leq i < k)$ and $v_i \neq v_j$ if $i \neq j$. Note that the path from u to v is identical in a tree. The *distance* from u to v, denoted by $dist(u, v)$, is the length of the path from u to v. The *eccentricity* $r(u)$ of node u is the maximum distance from u to an arbitrary node, i.e., $r(u) = \max_{v \in V} dist(u, v)$. The

diameter D is the maximum eccentricity, i.e., $D = \max_{u,v \in V} dist(u, v)$. The *radius* R is the minimum eccentricity. A node with eccentricity R is called a *center*.

A tree T is *symmetric* iff there exists a bijection function[1] $g : V \to V$ such that all the following conditions hold:

1. For any $v \in V$, the sets of port numbers on v and $g(v)$ are the same.
2. For any $u, v \in V$, u is adjacent to v iff $g(u)$ is adjacent to $g(v)$.
3. For any $\{u, v\}, \{g(u), g(v)\} \in E$, $_u(\{u, v\})$ is equal to $_{g(u)}(\{g(u), g(v)\})$.

In the *rendezvous problem*, $k \geq 2$ agents have to meet on a single node. However, if tree T is symmetric, the agent with the same deterministic algorithm cannot meet at a single node [10]. For this reason, we modify the requirement of the rendezvous problem as follows: All the agents meet and terminate at a single node if the tree is not symmetric, otherwise all the agents gather and terminate at two neighboring nodes. We say an algorithm $\mathcal{A}$ *solves* the rendezvous problem if agents executing $\mathcal{A}$ satisfy the above conditions for any tree, any location of home nodes, any starting time of agents, and any execution of agents.

To measure the efficiency of the algorithm, we evaluate *time complexity* and *space complexity*. However, measuring the real time is meaningless because there is no assumption about the period of each action by an agent. Thus, we define time complexity as the maximum number of moves for each agent. We define space complexity as the maximum number of bits each agent requires to store all local variables.

2.3 Basic properties

In the followings, we show basic properties of tree networks.

Theorem 1 *There exist one or two center nodes in a tree. If there exist two center nodes, they are neighbors [1].*

Theorem 2 *Let v' be the farthest node from node v in a tree (i.e., $dist(v, v') = r(v)$). The eccentricity of node v' is equal to the diameter of the tree, that is, $r(v') = D$.*

Proof. We assume $r(v') < D$ to prove the theorem by contradiction. Let u ($u \neq v'$) be a node satisfying $r(u) = D$. Let u' be the farthest node from u. Clearly, we have $dist(u, u') = D$. We also have $v' \neq u'$ since otherwise $r(v') = D$ holds. Note that v', u, and u' are leaf nodes. We define x as the last node in the longest common prefix of $P(u, u')$ and $P(u, v')$. Then, we have $dist(x, v') < dist(x, u')$ since otherwise $r(v') \geq D$ holds.

Next, we consider the location of v. We can find that $P(x, v)$ and $P(x, u')$ have a common prefix including other than x. This is because otherwise u' is further than v' from v. The location of v implies that $dist(x, v') \geq dist(x, u)$ since v' is the furthest node from v.

From the above discussion, we have $dist(u', v') = dist(u', x) + dist(x, v') \geq dist(u', x) + dist(x, u) = D$. This implies $r(v') \geq D$, which contradicts our assumption. □

Theorem 3 *Let the distance from node u to node v be D. The node c is a center if and only if c is included in the path $P(u, v)$ and $r(c) = \lceil \frac{D}{2} \rceil$ holds [13].*

[1]The function g is not an identity function. Thus, there exist a node u such that $g(u) \neq u$.

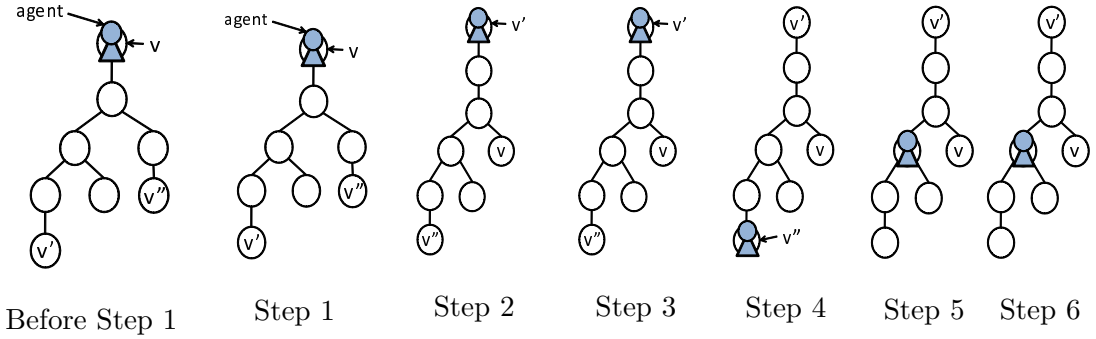


Figure 1: Location of the agent at the end of each step

3 Algorithms for the Rendezvous Problem in Trees

3.1 Outline of the Algorithms

In this section, we present the outline of the proposed algorithm Rendezvous. We use Rendezvous for the common framework of the proposed algorithms in this paper. In Rendezvous, each agent traverses the tree, finds the center, and terminates at the center. Note that, since each agent does not have any device to influence other agents, each agent does such works independently. The center at which all the agents terminate is called *rendezvous point*.

We assume that each agent starts Rendezvous at a leaf node of the tree. If the home node of an agent is not a leaf, the agent moves to a leaf. This work is easily done without memory by using basic steps, which we explain later. A leaf node where the agent starts the algorithm is called its *start node*. In Rendezvous, each agent performs the following six steps to find a rendezvous point. Figure 1 shows the location of the agent at the end of each step.

Step1: The agent computes the eccentricity $r(v)$ of the start node v.

Step2: The agent moves to the farthest node v' from v (we call node v' *second node*).

Step3: The agent computes $r(v')$ of the second node v' (i.e., $r(v') = D$).

Step4: The agent moves to the farthest node v'' from v' (we call node v'' *third node*),

Step5: and the agent moves back to the first node c which satisfies $dist(v', c) = \lceil \frac{D}{2} \rceil$.

Step6: If there exist two centers, the agent chooses the rendezvous point from two centers and terminates there unless the tree is symmetric.

First, each agent computes the diameter D in Step 1 to 3. After that, the agent moves to the center. Note that there may be one or more nodes whose distance from the second node v' is $\lceil \frac{D}{2} \rceil$. To detect the center among them correctly, the agent once moves to the third node v'' whose distance from v' is D because the center node is on the path from v' to v''. That is, the center node is the first node whose distance from v' is $\lceil \frac{D}{2} \rceil$ and that the agent visits after leaving from v''. These works are done in Step 4 to 5. In Step 6, the agent terminates at the rendezvous point, which is one of the centers. Note that, the agent can also understand whether the tree is symmetric or not in Step 6.

To realize Rendezvous, we introduce two functions MoveAndCompute and Choose. By calling function MoveAndCompute(h_1, h_2) at node v (called *initial node*), the agent starts

DFS-traversal of the tree. The argument h_1 implies that the agent continues the DFS-traversal until it visits node s_1 satisfying $dist(v, s_1) = h_1$. The argument h_2 implies that, after the visit to s_1, the agent continues the DFS-traversal until it visits node s_2 satisfying $dist(v, s_2) = h_2$. The agent stops the execution of MoveAndCompute at s_2. The agent keeps the maximum distance from the initial node v to a visited node during the execution of MoveAndCompute, and the maximum distance is returned as the output of function MoveAndCompute. Function MoveAndCompute is used in Rendezvous as follows: In Step 1, by calling MoveAndCompute$(1, 0)$ at a leaf v, the agent visits all nodes, computes $r(v)$, and comes back to v. In Step 2, by calling MoveAndCompute$(0, r(v))$ at v, the agent moves to v'. In Step 3, by calling MoveAndCompute$(1, 0)$ at the second node v', the agent visits all nodes, computes $r(v') = D$, and comes back to v'. In Step 4 and 5, by calling MoveAndCompute$(D, \lceil \frac{D}{2} \rceil)$ at the second node v', the agent moves to the third node and moves back to center c. Function Choose chooses a rendezvous point from two centers, and it is used to achieve Step 6 when there exist two centers.

In the followings, we focus on the implementation of MoveAndCompute and Choose. We give different implementations according to the memory space of each agent. In the rest of this section, we explain the common procedures of MoveAndCompute.

First, we introduce the *basic step*, which is the way of movements for agents. The basic step of an agent starts by leaving the 1-st port of its staying node v. Arriving at node u through the i-th port on u, the agent leaves u through the $\{i \mod (u) + 1\}$-th port in the next step. By continuing the basic step, each agent realizes DFS-traversal. We also define *reverse step* as the backward step of the basic step. When an agent starts the reverse step at a node, it leaves the node through the port passed in the previous step. The agent starts reverse steps only after its basic steps.

In the function MoveAndCompute, the agent has to compute the distance from its initial node v to a current node. To compute the distance in the anonymous networks, the agent uses the port numbers for the following strategy: Whenever the agent leaves a node u and moves to the adjacent node, it checks whether the step leads it close to or away from its initial node v. The following lemma implies that the agent can decide it by checking the order of the port the agent will move to. From the property of trees, the lemma clearly holds.

Lemma 1 *Let v be an initial node of* MoveAndCompute. *Assume that an agent has arrived at u for the first time through the i-th port on u. When the agent leaves u through the i'-th port, the agent is close to v if $i = i'$, and it is away from v otherwise.*

This chapter continues as follows. We explain time priority two algorithms using $O(n \log n)$ memory space or $O(n)$ memory space in Section 3.2. In section 3.3, we present the memory size priority algorithm using $O(\log n)$ memory space.

3.2 Time priority two algorithms

We explain two algorithms for the rendezvous problem in this section. The implementation of MoveAndCompute is the same for these algorithms, however the implementation of Choose is different.

Implementation of MoveAndCompute By Lemma 1, the agent can calculate the distance from the initial node v to the current node u by computing whether the following condition holds or not: The port through which the agent will pass to leave u is the same

as the one through which it first visited u. To compute it correctly, the agent keeps the sequence $H = h_1 h_2 \dots$ called *history*.The i-th element h_i of the history indicates the i-th basic step. Each step is kept by the fact whether the agent gets close to the initial node v or gets away from v. In more details, $h_i =$'+' if the agent gets away from v in the i-th step, and $h_i =$'−' otherwise. Note that, since each step is kept with one bit and the agent moves at most $2(n-1)$ times in each MoveAndCompute, the agent requires only $O(n)$ memory space to keep the history. By using the history, when the agent leaves a node, it calculates whether it gets close to the initial node v or away from v. The way of calculation is derived from the following lemma.

Lemma 2 *We assume that an agent visits a node u through the i'-th port on u after l basic steps in* MoveAndCompute*, and its history is $H_0 = h_1, h_2, \dots, h_l$. Let the i-th port be the one through which the agent first visits u in the* MoveAndCompute*. Then, the followings hold.*

Case1: *If $h_l =$'+' holds, $i' = i$ holds.*

Case2: *If $h_l =$'−' holds, we define $H_1, H_2, \dots$ as follows: Let S_0 be the minimum suffix of H_0 in which the number of '+' is equal to the number of '−'. Then, we define H_1 as the prefix of H_0 such that $H_0 = H_1 S_0$. If the last element of H_1 is '−', we can define H_2 in similar way. We continue the definitions, and assume the last element of H_t is '+'. Then, $i = (i' - t - 1) \mod (u) + 1$ holds.*

Proof. We consider the tree network as the rooted tree whose root is the initial node v. Remind that, in MoveAndCompute, each agent makes DFS-traversal from v by basic stepsf. In the case of $h_l =$'+', the lemma clearly holds from the property of DFS-traversal.

In the rest of proof, we consider the case of $h_l =$'−'. Then, the agent returns to u from its child w. From the property of DFS-traversal, once the agent visits w, it finishes the DFS-traversal of the subtree with root w in its subsequent steps. Consequently, the suffix S_0 corresponds to the DFS-traversal of the subtree with root w. Thus, H_1 is the history that the agent has on its last visit of u before the l-th step. From the behavior of basic steps, the agent visits u through the $((i'-2) \mod (u)+1)$-th port at that time. By continuing the above discussion, we can show that H_t is the history on the first visit of u and the agent first visits u through the $((i'-t-1) \mod (u)+1)$-th port. □

From Lemma 2, when the agent visits u, it can locally compute the port order through which it first visits u. Thus, from Lemma 1, the agent can determine whether it gets away from v or close to v in the next step.

We explain the implementation of MoveAndCompute(h_1, h_2) as follows. Let d be the distance from the initial node v to a current node and d_{max} be the maximum number of d the agent has computed. The agent prepares an empty history and set two variables d and d_{max} to be 0 at v. In MoveAndCompute(h_1, h_2), it performs following three operations when the agent leaves u, : 1) By using the history, the agent determines whether it gets close to the initial node v or away from v in the next step. 2) Next, it moves to the neighboring node by the basic step. 3) After that, it updates the history and two values d and d_{max}. The agent can calculate the distance from v to each node it visits by performing above three operations repeatedly. If d becomes h_2 after d becomes h_1 once, the agent stops the execution of MoveAndCompute.

Implementation of Choose Here, we explain two implementations of Choose. We assume that D is the odd number, i.e., there are two centers in the tree. Before the agent starts the

execution of Choose, it moves to a center node c by executing MoveAndCompute. Let c' be another center and e be the edge that connects two centers c and c'. Here, we assume the agent recognizes e and n. We define $t_v[1..j]$ as the sequence of the port numbers the agent has left during j basic steps from a node v. More precisely, $t_v[1..j] = p_1, \ldots, p_j$ where $p_k (1 \leq k \leq j)$ is the port number the agent leaves by k-th basic step.

First, we present the algorithm when each agent has $O(n \log n)$ memory space. In this case, the agent can keep all the port numbers in the network. Thus, the agent firstly performs $f = 2(n-1)$ basic steps started at c and gets the sequence $t_c[1..f]$. Next, the agent moves to another center c' through edge e. Then, it performs f basic steps started at c' and gets the sequence $t_{c'}[1..f]$. Finally, the agent compares $t_c[1..f]$ with $t_{c'}[1..f]$ lexicographically. If $t_c[1..f]$ (resp. $t_{c'}[1..f]$) is lexicographically smaller, the agent terminates at c (resp. c'). If the sequence of $t_c[1..f]$ and $t_{c'}[1..f]$ are the same, the tree is symmetric and terminates at c.

Second, we present the other algorithm when each agent has only $O(n)$ memory space. In this case, the agent cannot keep $t_c[1..f]$ or $t_{c'}[1..f]$ at a time because $O(n \log n)$ memory space is required to keep each of them. Thus, the agent keeps the subsequences of $t_c[1..f]$ and $t_{c'}[1..f]$ at a time, and compares these subsequences repeatedly to recognize which sequence is lexicographically smaller. The strategy is as follows. We describe $t_c[1..f] = p_1 p_2 \ldots p_f$ and $t_{c'}[1..f] = q_1 q_2 \ldots q_f$. The agent starts Choose at a center c. Since every port number is described in $\log n$ length, the agent with $O(n)$ memory space can keep $r = \lceil \frac{n}{\log n} \rceil$ port numbers. In the k-th round of Choose ($k = 1, 2, \ldots, \lceil \frac{2(n-1)}{r} \rceil$), the agent compares $p_{(k-1)r+1}, p_{(k-1)r+2}, \ldots, p_{kr}$ with $q_{(k-1)r+1}, q_{(k-1)r+2}, \ldots, q_{kr}$. In each round, the agent obtains the subsequences as follows: 1) The agent makes kr basic steps from c and keeps the last r port numbers, 2) returns to c by kr reverse steps and moves to c'. 3) Next, the agent makes kr basic steps from c' and keeps the last r port numbers, 4) returns to c' by kr reverse steps and moves to c. If the subsequences are different, the agent moves to an appropriate center and terminates there, otherwise the agent takes the next round. If the subsequences of the final round, i.e., $k = \lceil \frac{2(n-1)}{r} \rceil$, are the same, the agent terminates at c since the tree is symmetric.

The remaining issue is the way to recognize e and n, however it is easily solved. To recognize e, the agent continues MoveAndCompute of Step 5. Then, c' is the first node s which satisfies $dist(v', s) = \lceil \frac{D}{2} \rceil - 1$. In addition, e is the edge through which the agent moves to c'. Thus, by storing the port numbers of the edge, the agent can recognize e. To recognize n, the agent counts the number of steps during the execution of MoveAndCompute in Step 1. The agent can compute n after Step 1 since the number of steps of DFS-traversal is $2(n-1)$ in trees.

Efficiency of the algorithm The agent with $O(n)$ memory space can execute MoveAndCompute in $O(n)$ time for the following reasons: The agent moves at most $2(n-1)$ basic steps in each execution of MoveAndCompute. The maximum length of the history is $2(n-1)$, and the distance from initial node to visited node is at most D. In Choose, we first consider the case that $O(n \log n)$ memory space is available for each agent. Since the agent can keep all the port numbers in the tree at a time, it moves at most $4(n-1)+2$ steps. Next, we consider the case that each agent has $O(n)$ memory space. The agent moves $4kr + 2$ steps in k-th round (where $k = 1, 2, \ldots, \lceil \frac{2(n-1)}{r} \rceil$ and $r = \lceil \frac{n}{\log n} \rceil$), and thus, it takes $O(n \log n)$ time to execute Choose. Consequently, we have the following theorems.

Theorem 4 *The rendezvous problem in a tree network with n nodes is solved in $O(n)$ time*

with $O(n \log n)$ memory space on each agent.

Theorem 5 *The rendezvous problem in a tree network with n nodes is solved in $O(n \log n)$ time with $O(n)$ memory space on each agent.*

3.3 Memory size priority algorithm

In this section, we present an asymptotically space-optimal rendezvous algorithm which uses $O(\log n)$ memory space.

Implementation of MoveAndCompute The idea of the algorithm is the same as that of previous algorithms in Section 3.2: Whenever it moves to the adjacent node, it computes the distance from the initial node v to current node u. This computation is executed by determining whether that step makes the agent close to or away from v. The maximum value of the distance from v to the visited node is also kept by the agent.

However, we cannot use the same approach in Section 3.2 since the agent requires $O(n)$ memory space to keep the history. For this reason, we use other approaches. Consider the step such that the agent leaves node u through the i-th port on u. Let e be the edge connecting to the i-th port on u. In the asymptotically space-optimal algorithm, the agent computes whether it passes through e for the first time or not. Since the initial node is a leaf and the agent makes DFS-traversal, the agent gets away from v when it passes through an edge for the first time. Otherwise, the agent gets close to v. To realize the computation, we use two existing functions LogExploration and MatchingEdge [12]. These functions are proposed in the *symmetric-label* tree, where all the edges have the same label in both sides. Thus, in the followings, we consider only symmetric-label trees. Note that, however, this restriction is removed in Remarks of this section.

Function LogExploration is proposed to solve the exploration problem for trees [12]: The agent traverses all the edges and all the nodes in a tree started at its *home node* s. After the execution of LogExploration, the agent returns back to s. If and only if the tree is symmetric, there exists exactly one edge called *orphan edge* defined only by the topology of the tree. Therefore, the agent can check whether the tree is symmetric or not by checking if there is an orphan edge. The orphan edge corresponds to an edge connecting two center nodes in a symmetric tree. This work is also performed by LogExploration. The space complexity of LogExploration is $O(\log n)$, and the time complexity is $O(\Delta n^7)$, where Δ is the maximum degree of nodes in the network.

Function MatchingEdge is used as a subroutine in LogExploration [12]. To explain MatchingEdge, we consider the basic steps started at a home node s. Let $S = e_1 e_2 \dots e_{2(n\ -\ 1)}$ be the sequence of edges that the agent passes through in the basic steps. Since basic steps imply DFS-traversal, each edge is included in S exactly twice. By calling MatchingEdge(i) at s, the agent can compute $j(\neq i)$ satisfying $e_i = e_j$ in S if the tree is not symmetric. The space complexity of MatchingEdge is also $O(\log n)$, and the time complexity is $O(\Delta n^7)$.

By using these two functions, the agent can compute whether it passes through e for the first time or not. We assume the agent stays at u and it is about to make the l-th step. Let $S = e_1 e_2 \dots e_{2(n\ -\ 1)}$ be the sequence of edges that the agent passes through in basic steps started at initial node v. Here, we give the behavior of the agent in the case that the tree is not symmetric. First, the agent returns to initial node v by using $(l-1)$ reverse steps. By calling MatchingEdge(l) at v, it computes j satisfying $e_j = e_l$ in S. If $l < j$ holds, the agent passes e for the first time at the l-th step. Otherwise, the agent has passed e before the l-th

step. Note that, only when the agent executes MatchingEdge(l) at v, it can compute whether it passes through e for the first time or not by this way. After the computation, the agent can get back to u by $(l-1)$ basic steps. In the case that the tree is symmetric, the agent cannot use the above approach. Thus, the agent first executes LogExploration in algorithm Rendezvous to determine whether the tree is symmetric or not. If the tree is symmetric, the agent executes other procedures explained later.

We explain the implementation of MoveAndCompute. Before the agent starts MoveAndCompute (i.e., before Step 1 in Section 3.1), it executes LogExploration to check whether the tree is symmetric or not. If the tree is symmetric, the agent moves to a node adjacent to the orphan edge and terminates there. Thus, the agent executes the function MoveAndCompute only if the tree is not symmetric. In MoveAndCompute, the agent performs following three operations when it is about to make the l-th step and leave a node v: 1) By executing MatchingEdge(l) in the above way, the agent determines whether it gets close to the initial node v or away from v in the next step. 2) Next, moves to the neighboring node by the basic step. 3) Finally, it updates two values d and d_{max} (these variables have the same meaning as ones in Section 3.2). The agent can compute the distance from v to each node u by performing these three operations repeatedly.

Implementation of Choose We implement Choose in a similar way to one in Section 3.2. Note that, in this section, Choose is executed only if the tree is not symmetric. When the agent starts Choose, it stays at a center c with recognizing $f = 2(n-1)$, another center c' and the edge connecting c and c' in the same way explained in the previous section. In Choose, the agent compares $t_c[1..f]$ with $t_{c'}[1..f]$ lexicographically and terminates at a center with the smaller one. In this section, the agent compares these two strings one by one element to accomplish this work with $O(\log n)$ memory space.

The implementation of Choose is realized as follows. Initially, the agent sets the variable i to be 1. The agent makes i basic steps from c and gets the value $t_c[i]$. Next, it returns to c by i reverse steps and moves to c'. Similarly, it gets the value of $t_{c'}[i]$. If $t_c[i]$ is different from $t_{c'}[i]$, then it terminates at the node with the smaller one. If $t_c[i]$ is the same as $t_{c'}[i]$, the variable i is incremented by one and compares $t_c[i]$ with $t_{c'}[i]$ repeatedly. Note that, since the tree is not symmetric, $t_c[i]$ must be different from $t_{c'}[i]$ for some integer i.

Efficiency of the algorithm We consider the space complexity and time complexity of the algorithm showed in this section. The agent keeps five pieces of information: the distance from initial node v to current node, the maximum distance it has computed, the number of moves of basic step, at most two port numbers, and the variables to execute LogExploration and MatchingEdge. The agent makes at most $2(n-1)$ basic steps in each function, and the variables to execute LogExploration or MatchingEdge are also kept in $O(\log n)$ memory space. Therefore, it can store these values with $O(\log n)$ memory size. Next, we analyze the time complexity. It takes $O(\Delta n^7)$ to execute LogExploration or MatchingEdge. In MoveAndCompute, the agent executes MatchingEdge at each basic step and it makes at most $2(n-1)$ steps. In Choose, the agent can terminate in $O(n^2)$ time. The reason is that the agent can get $t_c[i]$ and $t_{c'}[i]$ for each i $(1 \leq i \leq 2(n-1))$ in $O(n)$ time. Finally, we can state the following theorem.

Theorem 6 *The rendezvous problem in a tree network with n nodes is solved in $O(\Delta n^8)$ time with $O(\log n)$ memory space on each agent.*

Remarks In the above discussion, we consider only symmetric-label trees. However, we show that our algorithm is available for general trees. To explain it, we use the method introduced in [12]. In this method, general trees are reduced to (*virtual*) symmetric-label trees by putting *virtual nodes* on some edges. Let T be a tree which is not symmetric-label. We call the original tree T *real tree* and nodes in T *real nodes*. We construct a *virtual tree* T' by putting *virtual nodes* in edges which have different labels in T: More precisely, for any edge $e = \{u, w\}$ in T such that $\ _u(e) \neq \ _w(e)$, we put a virtual node x on the edge e and assign $\ _u(e)$ and $\ _w(e)$ to $\ _x(\{x, u\})$ and $\ _x(\{x, w\})$, respectively. Since virtual tree T' is a symmetric-label tree, the agent can solve the rendezvous problem by executing the algorithm in T'.

However, virtual trees may cause two troubles: 1) The rendezvous point is a virtual node, and 2) The real tree is not symmetric but the virtual tree is symmetric. For the first case, after the agent moves to the center c (i.e., after Step5 in Section 3.1), it executes the following additional procedures. If c is a virtual node and the only center, it leaves c through the first port on c and terminates at the arrival node. If there are two centers and one of them is a virtual node, the agent terminates at the real one. For the second case, after the agent recognizes the symmetry of T' by LogExploration, it determines whether the real tree T is symmetric or not: The agent moves to one of nodes w and w' that is adjacent to the orphan edge. Let $v_w(i)$ be *true* (resp. *false*) if the agent visits a real (resp. virtual) node after i basic steps from w. If $v_w(i) = v_{w'}(i)$ holds for all the integer i ($1 \leq i \leq 2(n' - 1)$, where n' is the number of nodes in T' computed in LogExploration), T is also symmetric. If $v_w(i) \neq v_{w'}(i)$ holds for some i, T is not symmetric. In this case, agents can terminate and meet at w (resp. w') when $v_w(i) =$*true* (resp. $v_{w'}(i) =$*true*) holds. Notice that, the number of virtual node is at most n. Thus, the complexities remain unchanged asymptotically.

4 Conclusion

In this paper, we coped with the rendezvous problem in trees with k agents. We have shown that arbitrary number of agents can rendezvous at a single node in trees even if their actions are progressed asynchronously. We have presented three algorithms, each of which is asymptotically time-optimal, balanced in terms of time and space, and asymptotically space-optimal. Thus, we have shown the tradeoffs between the time complexity and the space complexity for the rendezvous problem. As for the time complexity of the asymptotically space-optimal algorithm, it is as efficient as the one presented in [10]. However, the time complexity is $O(\Delta n^8)$. Therefore, it is an interesting problem to improve the time complexity of the asymptotically space-optimal algorithm.

References

[1] G. Agnarsson and R. Greenlaw. *Graph Theory : Modeling, Applications, and Algorithms*, chapter 3.5, pages 77–80. Person Education, 2007.

[2] S. Alpern. The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33:673–683, 1995.

[3] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous*. Kluwer Academic Publishers, Norwell, Massachusetts, 2003.

[4] V. Baston and S. Gal. Rendezvous on the line when the players' initial distance is given by an unknown probability distribution. *SIAM Journal on Control and Optimization*, 36:1880–1889, 1998.

[5] V. Baston and S. Gal. Rendezvous search when marks are left at the starting points. *Naval Res. Log*, 48:722–731, 2001.

[6] S. Das. Mobile agent rendezvous in a ring using faulty tokens. In *Proc. 9th International Conference in Distributed Computing and Networking(ICDCN 2008)*, pages 292–297, 2008.

[7] S. Das, M. Mihalak, R. Sramek, E. Vicari, and P. Widmayer. Rendezvous of mobile agents when tokens fail anytime. In *Proc. 12th International Conference on Principles of Distributed Systems*, pages 463–480, 2008.

[8] P. Flocchini, E. Kranakis, D. Krizanc, F. L. Luccio, N. Santoro, and C. Sawchuk. Mobile agents rendezvous when tokens fail. In *Proc. 11th Colloquium on Structural Information and Communication Complexity(SIROCCO 2004)*, pages 599–608, 2004.

[9] P. Flocchini, E. Kranakis, D. Krizanc, C. Sawchuk, and N. Santoro. Multiple mobile agents rendezvous in a ring. In *Proc. 6th Latin American Theoretical Informatics(LATIN 2004)*, pages 599–608, 2004.

[10] P. Fraigniaud and A. Pelc. Deterministic rendezvous in trees with little memory. In *Proc. 22nd International Symposium on Distributed Computing(DISC 2008)*, pages 242–256, 2008.

[11] L. Gasieniec, E. Kranakis, D. Krizanc, and X. Zhang. Optimal memory rendezvous of anonymous mobile agents in a uni-directional ring. In *Proc. 32nd International Conference on Current Trends in Theory and Practice of Computer Science(SOFSEM 2006)*, pages 282–292, 2006.

[12] L. Gasieniec, A. Pelc, T. Radzik, and X. Zhang. Tree exploration with logarithmic memory. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms(SODA 2007)*, pages 585–594, 2007.

[13] E. Korach, D. Rotem, and N. Santoro. Distributed algorithms for finding centers and medians in networks. *ACM Transactions on Programming Langages and Systems*, 6(3):380–401, 1984.

[14] E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk. Mobile agent rendezvous in a ring. In *Proc. 23rd International Conference on Distributed Computing Systems(ICDCS 2003)*, pages 592–599, 2003.

[15] T. Schelling. *The strategy of conflict.* Oxford University Press, Oxford, 1960.

トポロジ変化に対して出力の変化数を極小化する全域木更新アルゴリズム

高田 篤史 † 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

†† 奈良先端科学技術大学院大学 情報科学研究科

概要 分散システムとは，ネットワークで相互に接続された複数の計算機 (ノード) からなるシステムである．近年，モバイルアドホックネットワークのようなトポロジが時々刻々と変化する動的な分散システムが注目を集めている．分散システムの出力は，各ノードが管理する出力変数で構成されており，外部のアプリケーションで利用される．動的な分散システムでは，ネットワークのトポロジ変化が生じたときに出力の再計算が必要となる．同時に，システムの可用性を向上させるため，出力の変化数を抑制した再計算が要求されている．分散システムにおける重要な問題の一つに，トポロジ変化に対してネットワークの全域木を再計算する問題 (全域木更新問題) がある．これまでに，全域木更新問題を解く分散アルゴリズムは提案されているが，出力の変化数を抑制することを考慮したものはない．本稿では，再計算途中における各ノードの出力の変化回数を制限しながら，出力変化ノード数の最小化を実現する全域木更新問題として，出力変化極小全域木更新問題を新たに提案する．そして，この問題を解く分散アルゴリズムを提案し，その性能を理論的に評価する．

1 はじめに

分散システムとは，ネットワークで相互に接続された複数の計算機 (以降，ノードと呼ぶ) からなるシステムである．分散システムは，ユーザに対して出力を供給し，ユーザや外部システムはこの値を用いて他のアプリケーションを実行する．出力は，各ノードが管理する出力変数から構成される．分散アルゴリズムは，与えられた問題に対する各ノードの出力変数の値を計算し，各ノードが目的の出力変数の値を保持するネットワークの状況を解状況として求める．近年，移動無線端末の急速な普及に伴い，アドホックネットワークのような動的な分散システムが注目を集めている．動的な分散システムは，ノードの参加，離脱もしくはノードの移動による通信リンクの出現，消滅により，ネットワークのトポロジが時々刻々と変化する．そのため，トポロジ変化後の解状況を再計算する分散アルゴリズムが必要となる．同時に，システムの可用性を向上させるため，再計算途中における各ノードの出力の変化回数や，出力を変化させるノードの数を抑制することが要求されている．

本稿では，トポロジ変化に対して全域木を再計算する問題である**全域木更新問題**に取り組む．全域木は，ネットワーク上でメッセージの放送を行う場合などに利用されており，全域木を構成するリンクのみを使ってメッセージを配送することで効率のよい放送が行える．更新問題とは，解状況のネットワークにトポロジ変化が生じたとき，トポロジ変化後の解状況を求める問題である．本稿では，全域木の再計算途中における各ノードの出力の変化回数や出力を変化させるノード数といった，出力変化数の抑制について考察する．

更新問題を解く分散アルゴリズムを**更新アルゴリズム**と呼ぶ．全域木に対する更新アルゴリズムは，これまでにいくつか提案されている [9, 10]．しかし，再計算途中の出力変化数については一切考慮しておらず，各ノードでの頻繁な出力変化や，大域的な再計算が発生する．

トポロジ変化やノードの一時故障に対して自律的な適応性を有する分散アルゴリズムとして，**自己安定アルゴリズム**の研究が行われている [3, 4]．自己安定アルゴリズムは，任意のネットワーク状況から実行を開始しても，いずれ与えられた問題に対する解状況を計算することを保証する．例えば，全域木構成自己安定アルゴリズム [1] は，トポロジ変化やノードの一時故障が生じたネットワークを初期状況とみなし，いずれ解状況として全域木を求める．しかし，再計算途中における出力変化数の抑制については一切考慮していない．

強安定アルゴリズム[2, 8] は，自己安定の性質とともに，特定のトポロジ変化に対して，再計算途中に変更が必要なノードの出力のみを変化させ，その他のノードの出力を変化させないことを保証する．文献 [8] では全域木の構成について，文献 [2] ではリンクに重み付けされたネットワーク上での Steiner 木の構成について取り組んでいる．しかし，これらの強安定アルゴリズムは，任意のトポロジ変化に対しては，再計算途中の出力変化数の抑制を一切保証していない．

文献 [7] では，再計算途中における各ノードの出力の変化回数と出力を変化させるノード数が，解状況から局所変数の値を変化させたノード数のみに依存する自己安定アルゴリズムとして，**出力安定な自己安定アルゴリズム**を定義している．そして，自己安定アルゴリズムを出力安定な自己安定アルゴリズムに変換する出力安定機構 (output stabilizer) を提案している．しかし，出力安定機構を適用できる問題のクラスは，各ノードの局所的な性質に関する問題 (マッチング問題，支配集合問題など) に限定し，全域木のようにネットワークの大域

的な性質に関する問題に対しては適用できないことを示している．

本稿では，再計算途中における各ノードの出力変化回数を制限しながら，ネットワークの出力変化ノード数を最小化する全域木更新問題として，出力変化極小全域木更新問題を新たに提案する．そして，この問題を解く分散アルゴリズムを提案し，その性能を理論的に評価する．

2 諸定義

2.1 ネットワークモデル

本稿では，ノードとそれらを接続する通信リンク (以後，リンクと呼ぶ) からなる連結なネットワーク $G = (V, E)$ を対象とする．ここで，V は**ノード集合**，E は**リンク集合**を表す．本稿では，任意の形状 (以後，**トポロジ**と呼ぶ) のネットワークを扱う．ネットワークはノードを頂点，リンクを辺としたグラフとみなせるため，グラフに対する用語をネットワークに対しても用いる．また，G には特別なノード $v_0 \in V$ が存在することを仮定する．リンク E は，V の異なる 2 要素の非順序対の集合であり，$(v_i, v_j) \in E$ のとき，ノード v_i，v_j 間に双方向のリンクが存在し，v_i と v_j は**隣接する**という．各 v_i に対し，$neig_i$ を隣接ノード集合とする．各 v_i，v_j に対し，$v_i \in neig_j$ ならば $v_j \in neig_i$ である．

ネットワーク G において，ノードの参加・離脱もしくはノードの移動によるリンクの出現・消滅が生じる事象を**トポロジ変化**と呼ぶ．つまり，トポロジ変化によって，G のノード集合 V と G のリンク集合 E に変化が生じ，あるノード $v_i \in V$ の隣接ノード集合 $neig_i$ が変化する．ただし，v_0 は離脱することなく，常に G 中に存在する．

各ノード v_i は，自身の**識別子**，**局所アルゴリズム**，および様々な**局所変数**をもつ．各ノードの識別子は固有のものとし，簡単のため各ノード v_i と v_i の識別子は区別しない．ノードの局所変数は，ユーザが観測する**出力変数**と，計算の途中に利用する**内部変数**に分類する．

2.2 通信モデル

本稿では，通信モデルとして**非同期メッセージパッシングモデル**を仮定する．メッセージパッシングモデルでは，各リンクの各方向にメッセージチャネル (以後，**チャネル**と呼ぶ) が存在し，チャネル上でのメッセージの送受信によってノード同士が通信する．各ノード v_i は，以下の通信命令を用いて通信を行う．

$send(m)_{i,j}$
v_i が隣接ノード $v_j \in neig_i$ にメッセージ m を送信するために，チャネル (v_i, v_j) に m を挿入する命令．

$receive(m)_{j,i}$
v_i が隣接ノード $v_j \in neig_i$ からメッセージ m を受信するために，チャネル (v_j, v_i) から m を取り出す命令．

本稿では，*FIFO (First In First Out)* のチャネルを仮定する．また，リンクの消滅が生じない限り，チャネル中ではメッセージの損失が起こらず，任意の送信されたメッセージは有限時間内に受信されるものと仮定する．

2.3 アルゴリズムの実行

アルゴリズムの実行は，**始動ノード**が自発的に自身の局所アルゴリズムの実行を始めることで開始する．始動ノード以外のノードは，他のノードからのメッセージを受信することによって自身の局所アルゴリズムの実行を開始する．つまり，始動ノード以外のノードは他のノードからのメッセージを受信するまで，メッセージの受信を待ち続ける．

ノードの**状態**は，そのすべての局所変数の値の組で定義する．ネットワークの**状況**は，すべてのノードの状態とすべてのチャネル上のメッセージの組で定義する．各ノード v_i の 1 **原子動作**は，次のように定義する．

1. あるチャネル (v_j, v_i) に対して，$receive(m)_{j,i}$ を実行する．
2. 受信したメッセージ m に基づく動作を実行する．
3. 必要であれば，チャネル (v_i, v_k) に対して，$send(m')_{i,k}$ を実行する．

ここで，2 つ以上のノードが同時に 1 原子動作を行っても，互いにその 1 原子動作による影響は受けない．そのため，本稿では任意の時点でただ 1 つのノードが 1 つの 1 原子動作を行うものと仮定する．アルゴリズムの**計算**は，状況の極大な系列 $E = c_0, c_1, c_2, \ldots, c_t$ で定義する．ここで，c_{i+1} は，c_i においてある 1 つのノードが 1 つの 1 原子動作を行うことよって得られる状況である．c_t は，すべてのノードの局所アルゴリズムが停止した状況であり，**停止状況**と呼ぶ．

2.4 出力変化極小全域木更新問題

定義 1 全域木

与えられた無向グラフ $G = (V, E)$ に対して，$V' = V$ かつ $E' \subseteq E$ かつ $|E'| = |E| \quad 1$ を満たす連結グラフ $T = (V', E')$ を G の全域木という．

本稿で扱うネットワークは，辺に重みがない無向グラフで表わす．ただし，アルゴリズムによって辺に向きと重みが与えられることを仮定するため，以降で有向全域木と重み最小有向全域木について定義する．有向グラフにおける全域木は，各辺に向きがないことを仮定したとき，無向グラフの全域木の定義を満たす．あるノード u から別のノード v への有向辺を (u, v)，ある辺 e の重みを $w(e)$ と書く．

定義 2 有向全域木

有向グラフ $G = (V, E)$ について，ある 1 つの頂点 $v_a \in V$ を除いた各頂点 $v_i \in V$ から v_a への有向路が存在するとする．G の全域木 $T = (V', E')$ が次の全条件を満たすとき，T を頂点 $v_a \in V'$ を根とする G の有向全域木という．

根 v_a の出次数が 0 である．
根 v_a を除いた各頂点 $v_i \in V'$ の出次数が 1 である．

定義 3 重み最小有向全域木

辺重み付き有向グラフ $G = (V, E)$ について，ある 1 つの頂点 $v_a \in V$ を除いた各頂点 $v_i \in V$ から v_a への有向路が存在するとする．ノード $v_a \in V'$ を根とする G の有向全域木 $T = (V', E')$ が次の条件を満たすとき，T を頂点 v_a を根とする G の重み最小有向全域木という．

E' に含まれる辺の重みの総和 $\sum_{e' \in E'} w(e')$ が G の有向全域木の中で最小である．

本稿では，v_0 を根とする全域木の更新問題を考える．全域木における各ノードの接続関係を表すため，G 中の各ノード v_i は親を表す出力変数 $P(v_i)$ を管理し，$P(v_i)$ の値としてある 1 つの隣接ノード $v_j \in neig_i$，または自身 v_i を設定する．あるノード v_j から親ノード $P(v_j)$ を参照し，さらに $P(v_j)$ の親ノード $P(P(v_j))$ を参照するというように，親ノードの参照を順に繰り返すことを v_j から**親を辿る**という．

定義 4 親子関係

任意の 2 ノード v_i と v_j について，次の論理式 $Par_{i,j}$ を満たすとき，v_i を v_j の親ノード，v_j を v_i の子ノードと呼び，v_i と v_j の間に親子関係が成立しているという．

$$Par_{i,j} \equiv (v_i \in neig_j) \wedge (v_j \in neig_i) \wedge (P(v_j) = v_i)$$

本稿では，全域木が計算されている状況でのトポロジ変化を想定する．そのため，問題の入力にはトポロジ変化直後のネットワーク G と，G の初期状況 c_0 が与えられる．つまり，c_0 において，G 中の各ノードはトポロジ変化前の局所変数の値を保持する．また，G 中にはトポロジ変化前の全域木が分断された木 (**フラグメント**) が存在する．フラグメントは以下のように定義する．

定義 5 フラグメント

与えられたグラフ $G = (V, E)$ について，$V' = V$ とし，E' を親子関係が成立している 2 ノードを接続するリンクの集合とする．G の部分グラフ $G' = (V', E')$ について，$V_i \subseteq V'$，$E_i \subseteq E'$ なる G' の極大な各連結成分 $F_i = (V_i, E_i)$ を G の全域木のフラグメントという．

G の初期状況 c_0 は，全域木の構成された状況からトポロジ変化のみで到達可能な状況である．そのため，すべてのフラグメント F_i は無閉路である．つまり，任意のノード $v_j \in V_i$ から親を辿るとき，再び v_j を参照しない．

本稿では，与えられたネットワーク G の初期状況 c_0 で，各ノード v_i はトポロジ変化前における自身の親子関係を知っているものとする．v_i は木上での隣接ノード集合を表す出力変数として $N^F(v_i)$ を管理し，v_i に親ノード v_j(または子ノード v_k) が存在すれば，v_j(または v_k) が $N^F(v_i)$ の値に含まれる．

トポロジ変化後のネットワークと状況をそれぞれ $G = (V, E)$，c_0 とし，停止状況を c_t とする．計算 E のある状況 c_i において，各ノード v_j が所属するフラグメントを $F(v_j, c_i)$ とし，c_i における v_j の出力変数 $N^F(v_j)$ の値を $neig_j^{F(v_j,c_i)}$ とする．計算 E の各状況 $c_i(0 \quad i \quad t \quad 1)$ におけるすべてのノード $v_j \in V$ について，$neig_j^{F(v_j,c_i)} \subseteq neig_i^{F(v_j,c_{i+1})}$ が成り立つとき，c_0 における木上の隣接関係が維持されたという．

定義 6 解状況

与えられたネットワーク $G = (V, E)$ について，すべてのノード $v_i \in V$ が次の全条件を満たす状況を**解状況**と呼ぶ．解状況は，G の全域木が構成された状況を表す．

$(P(v_i) = v_j) \wedge (v_j \in neig_i) \wedge (v_j \in N^F(v_i))$
$(\forall v_k)\{(P(v_k) = v_i) \wedge (v_k \in neig_i)$
$\wedge (v_k \in N^F(v_i))\}$.
v_iから親を辿るとき，v_0に到達する．

定義 7 出力変化極小全域木更新問題

与えられた連結なネットワーク $G = (V, E)$ と G の初期状況 c_0 に対して，以下の全条件を満たしながら G の全域木を求める問題を出力変化極小全域木更新問題と定義する．

停止状況が解状況である．
c_0 における木上の隣接関係が維持されるもとで，親を変化するノード数が最小化される．
計算途中における各ノード $v_i \in V$ の出力変数 $P(v_i)$ の値の変化回数が最小化される．

3 アルゴリズム

本章では，出力変化極小全域木更新問題を解く分散アルゴリズムを説明する．提案アルゴリズムは，木上の隣接関係を維持しながら，親を変化させるノード数を最小化する．さらに，各ノード v_i の出力 $P(v_i)$ は，求める全域木における親が決まるときにのみ変化させる．出力 $N^F(v_i)$ は，求める全域木における親が決まるとき，また子が決まるときにのみ値を変化させる．以上の意味で，提案アルゴリズムは各ノードの出力変化回数を制限しながら，ネットワークの出力変化ノード数を最小化する．

3.1 アルゴリズムの概要

3.1.1 アイデア

提案アルゴリズムでは，次の 2 フェーズによって出力変化極小全域木更新問題の解を得る．

フェーズ 1　重み付き有向仮想グラフを想定
フェーズ 2　仮想グラフ上の重み最小有向全域木を構成

以降で，各フェーズの内容について説明する．

フェーズ 1:　フェーズ 1 では，与えられたネットワーク (以降，実グラフと呼ぶ) に対する仮想グラフを考える．この仮想グラフ上で重み最小有向全域木を求めれば，実グラフ上で問題の解を求められるように，辺の向きと重みを設定する．ただし，仮想グラフはアルゴリズムの説明上用いるものであり，仮想グラフを作成する処理を実際に行うわけではないことに注意する．

以降では，想定する仮想グラフについて示す．ここでは，実グラフにおけるノードを v_i，仮想グラフにおける頂点を u_i で記述する．仮想グラフの各頂点は，実グラフの 1 つのフラグメントに対応する．ここで，実グラフの異なる 2 つのフラグメント間にリンクが存在するとき，かつそのときに限り，対応する仮想グラフの 2 つの頂点 u_i, u_j の間に有向辺 (u_i, u_j) と (u_j, u_i) が存在する．次に，仮想グラフの各有向辺に重みを設定する．各有向辺の重みは，有向辺の始点が対応する実グラフのフラグメント内で，親を変化させるノード数を意味する．つまり，実グラフ上の異なるフラグメント間のリンク (v_i, v_j) によってフラグメントを接続するとき，v_i(または v_j) のフラグメント内で，根から v_i(または v_j) までの直系のノードが親子関係を逆転させ，v_i(または v_j) が v_j(または v_i) を新たな親として設定する．各有向辺の重みの定義を次に示す．仮想グラフの各有向辺 (u_1, u_2) について，端点 u_1，u_2 に対応する実グラフのフラグメントをそれぞれ $F_1 = (V_1, E_1)$，$F_2 = (V_2, E_2)$ とする．このとき，F_1 と F_2 の間にリンク (v_1, v_2) が存在する $(v_1 \in V_1, v_2 \in V_2)$．ここで，$F_1$ において根からの距離が最小となる v_1 について，根から v_1 までの距離を $dist(v_1)$ とするとき，仮想グラフの有向辺 (u_1, u_2) の重みの値を $dist(v_1) + 1$ と定義する．

仮想グラフの辺の重み割当ての例として，図 1(a) の実グラフを考える．図 1 で示す矢印は始点が子ノード，終点が親ノードを表す．図 1(a) では，フラグメント F_1 と F_2 の間にリンク (v_1, v_2) が存在する．ここで，ノード v_1 がノード v_2 を親とした場合，v_1 が属す F_1 内で親を変更するのは 3 ノードである (図 1(b) の灰色のノードが親を変更)．一方，v_2 が v_1 を親とした場合，v_2 が属す F_2 内で親を変更するのは 2 ノードである ((図 1(c) の灰色のノードが親を変更))．したがって，F_1，F_2 をそれぞれ頂点 u_1，u_2 とした仮想グラフにおいて，有向辺 (u_1, u_2) の重みは 3，(u_2, u_1) の重みは 2 に設定する (図 1(d))．

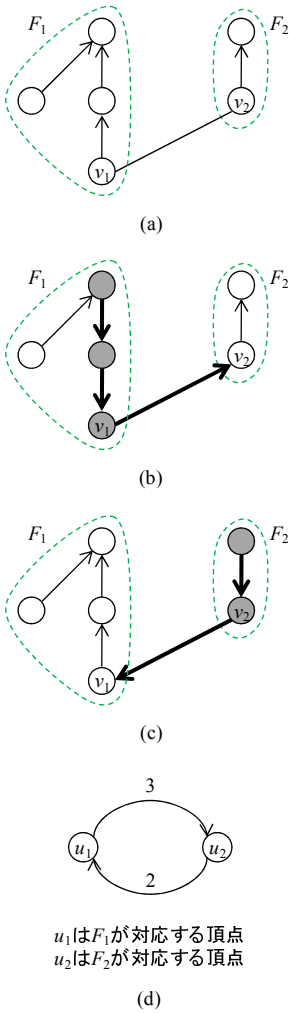


図 1　重み割当ての例

フェーズ 2:　フェーズ 2 では，フェーズ 1 で想定した仮想グラフ上で頂点 u_0 を根とする重み最小有向全域木を求め，その構成に従って実グラフの各ノードが局所変数の値を設定することで問題の条件を満たす全域木を求める．仮想グラフの頂点 u_0 には，実グラフにおける特別なノード v_0 を根とするフラグメントが対応する．

仮想グラフ上での重み最小有向全域木の構成法については次節で説明し，ここでは重み最小有向全域木の各有向辺が決まったものとして全域木の構成法を説明する．仮想グラフ上の重み最小有向全域木に含まれる有向辺 (u_i, u_j) を決定したとき，(u_i, u_j) に対応する実グラフ上のフラグメント F_i, F_j 間のリンク (v_i, v_j) を全域木の辺として選択する．つまり，ノード v_i がノード v_j を親に設定し，v_i の所属するフラグメント F_i 内で，F_i の根から v_i までのノードが親子関係を逆転させる．このように，選択した有向辺で導かれる木の構造に従い，必要に応じて各ノードが自身の局所変数の値を設定する．以上の流れで，木上の隣接関係を維持するもとで親を変化させるノード数を最小化した全域木を求める．

3.1.2 重み最小有向全域木の構成法

本節では，仮想グラフ上での重み最小有向全域木を実グラフを用いて実際に求める処理を説明する．提案アルゴリズムでは，実グラフの各フラグメントを仮想グラフにおける 1 つの頂点に対応させるため，実グラフの各ノードに対して，所属するフラグメントの識別子 (以降，FID と書く) の割当てを行う．FID には，そのフラグメントの根ノードの識別子を用いる．さらに，仮想グラフにおける辺の重みを定義するため，実グラフの各ノード v_i に対して重み $w(v_i)$ を定義する．所属するフラグメントにおける根から v_i までの距離を $dist(v_i)$ とするとき，$w(v_i)$ の値を $dist(v_i)+1$ とする．そして，仮想グラフの頂点 u_i に対応するフラグメントのノード v_i と，別の頂点 u_j に対応するフラグメントのノード v_j を接続する辺が存在するとき，外向辺 (u_i, u_j) の重みを $w(v_i)$ とする．ここで，有向辺 (u_i, u_j) を頂点 u_i の**外向辺**と呼ぶ．

提案アルゴリズムは，仮想グラフ上で重み最小有向全域木を求めるため，最初に各フラグメントに対して 1 つのクラスタを形成する．その後，クラスタの合併を繰り返し，すべてのフラグメントが含まれる 1 つのクラスタを形成したとき，重み最小有向全域木が求まる．以降では，v_0 が含まれるクラスタを**根クラスタ**と呼び，クラスタ内に含まれるクラスタを**内部クラスタ**と呼ぶ．

まず，根クラスタを除いた各クラスタ $clst_i$ 内で，重み最小の外向辺を**クラスタ外向辺**として選択する．そして，$clst_i$ 内の各ノードの重みからクラスタ外向辺の重みを減じる．次に，各フラグメントの辺と選択した各クラスタ外向辺を辿って，各 $clst_i$ が根クラスタに到達可能か否かを判定する．

> 根クラスタに到達不可能なクラスタ $clst_j$ について，各 $clst_j$ のクラスタ外向辺からなる有向閉路を検出する．そして，有向閉路に含まれるすべての $clst_j$ を合併し，1 つの新しいクラスタ $clst_{new}$ に置換する．次に，$clst_{new}$ のクラスタ外向辺として，$clst_{new}$ 内の重み最小の外向辺を選択する．そして，$clst_{new}$ 内の各ノードの重みから $clst_{new}$ のクラスタ外向辺の重みを減じる．
>
> 根クラスタに到達可能なクラスタ $clst_k$ について，$clst_k$ を根クラスタに合併する．つまり，根クラスタと $clst_k$ からなる新しい根クラスタに置換する．$clst_k$ 内に内部クラスタのクラスタ外向辺による有向閉路が存在するならば，$clst_k$ のクラスタ外向辺の始点ノードの属す最初のクラスタ (つまり，フラグメント)$clst_s$ を根とする有向木を $clst_k$ 内で構成する．これは，$clst_k$ 内において $clst_s$ が含まれる各内部クラスタのクラスタ外向辺をすべて除去することで構成できる．

以降，実グラフにおけるすべてのフラグメントが根クラスタに含まれるまで，さらに新しいクラスタへの置換かまたは根クラスタへの合併を繰返す．

3.2 実行例

本節では，入力として図 2 の実グラフを考えたときの提案アルゴリズムの実行例を示す．図 2 で示す矢印は始点が子ノード，終点が親ノードを表す．また，塗りつぶしたノードが特別な根 v_0 を，点線の辺がフラグメント間を接続するリンクを表す．

図 2 のグラフに対して仮想グラフを求めると図 3(a) のグラフが得られる．図 3 で示す辺の数字はその辺の重みの値を表す．提案アルゴリズムでは，図 3(a) の仮想グラフに対して重み最小有向全域木を求める．根クラスタを除く各クラスタが，クラスタ外向辺を選択し終えたときの仮想グラフを図 3(b) に示す．図 3(b) では，根クラスタ F_0 に到達可能なクラスタ F_1 が存在するため，F_1 が F_0 に合併される．次に，クラスタ外向辺からなる有向閉路を探索すれば，クラスタ F_2 とクラスタ F_3 が有向閉路に含まれることがわかる．そのため，F_2 と F_3 を 1 つの新しいクラスタ $clst_6$ として置換し，$clst_6$ のクラスタ外向辺を選択する (図 3(c))．図 3(c) で，クラスタ外向辺による有向閉路を再び探索すれば，クラスタ $clst_6$ とクラスタ F_4，クラスタ F_5 からなる有向閉路が検出できる．そのため，$clst_6$，F_4，F_5 を 1 つの新しいクラスタ $clst_7$ として削減し，$clst_7$ のクラスタ外向辺を選択する (図 3(d))．図 3(d) では，$clst_7$ が根クラスタに到達可能なため，$clst_7$ を根クラスタに合併し，すべてのフラグメントを含む根クラスタ $clst_8$ を形成する．このとき，$clst_7$ のクラスタ外向辺の始点 F_3 が含まれる各内部クラスタ (つまり，F_3 と $clst_6$) のクラスタ外向辺を取り除く (図 3(e))．以上により，仮想グラフ上で重み最小有向全域木が求まる (図 3(f))．そして，仮想グラフ上の重み最小有向全域木の構成に従い，各ノードは親と木上の隣接ノード集合の出力を変化させる．以上により，問題の条件を満たす全域木が求まる (図 4)．

3.3 アルゴリズムの詳細

本節では，提案する分散アルゴリズムの詳細について説明する．アルゴリズムは，根ノード v_0 とその他のノードで異なる．

提案アルゴリズムでは，各クラスタ内での動作を統括するため，クラスタ内のある 1 つのノードを**クラスタヘッド**として選出する．最初，各フラグメントについて要素数 1 のクラスタを形成するため，各フラグメントの根がクラスタヘッドである．クラスタヘッドは，自身のクラスタのリーダーとしてふるまい，クラスタヘッド以外のノードは，クラスタヘッドからのメッセージに従った動作を行う．これを実現するため，各クラスタヘッド

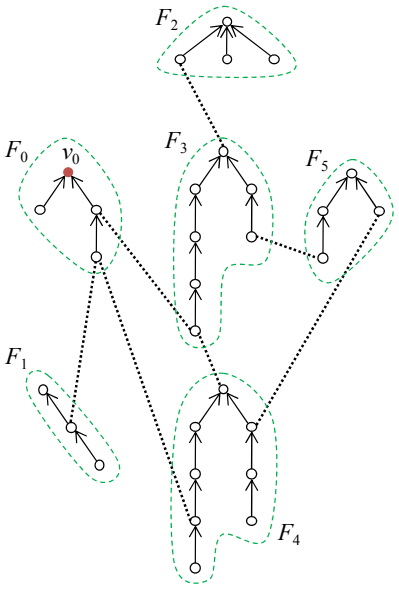


図 2　入力グラフ

は必要に応じて自身のクラスタ上で *PIF(Propagation Information with Feedback)*[5] を実行する．クラスタヘッドが PIF を実行すれば，クラスタ内に形成できる木に従ってメッセージが放送され，いずれクラスタヘッドがそのメッセージの放送を終えたことを知る．PIF は**放送フェーズ**と**帰還フェーズ**の 2 フェーズからなる．放送フェーズでは，根ノードから葉ノードに向かって目的のメッセージが伝搬する．帰還フェーズでは，葉ノードから根ノードに向かって目的のメッセージの応答が伝搬する．

以降では，クラスタヘッドが PIF を実行するためのクラスタの構造として，**クラスタ木**を説明する．そして，v_0 以外のノードのアルゴリズム，v_0 のアルゴリズムについて説明する．

3.3.1　クラスタ木

クラスタ木とは，クラスタ内の構造から仮想的に導出できる木である．要素数 1 のクラスタ内には，すでにフラグメントの根を根とする木が構成されている．要素数 2 以上のクラスタ $clst_i$ 内には，内部クラスタのクラスタ外向辺による有向閉路が存在する．そのため，一部のクラスタ外向辺を取り除くことでクラスタ内に木を構成し，メッセージの放送を効率化する．

構成するクラスタ木の根ノードを v_h とする．v_h は，アルゴリズムの処理において決定される．v_h を根とする $clst_i$ のクラスタ木は，$clst_i$ 内のすべての内部クラスタのクラスタ外向辺から，v_h が含まれる各内部クラスタのクラスタ外向辺 e_{not} を取り除くことで構成できる．

v_h を除いた各ノード v_i は，メッセージの送信元ノードをクラスタ木の親とみなし，クラスタ木の親を除いたフラグメント上の全隣接ノードをクラスタ木の子とみなす．また，v_i に入射するクラスタ外向辺が存在するならば，その端点ノードもまたクラスタ木の子とみなす．

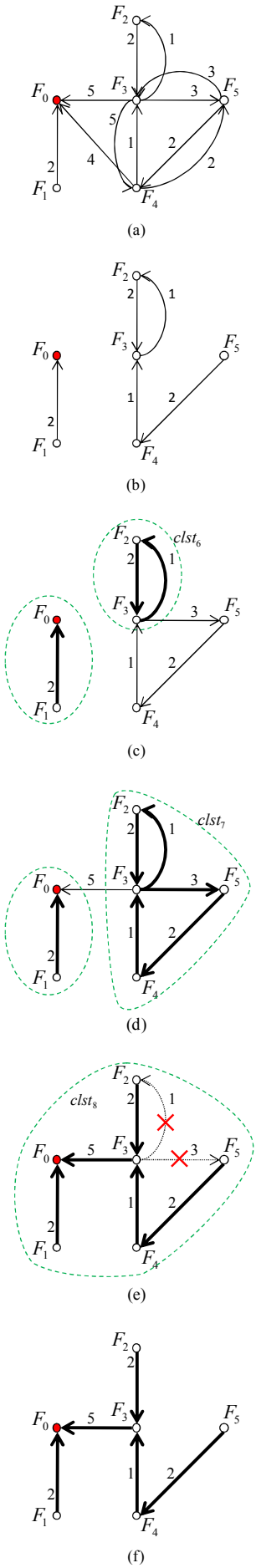


図 3　仮想グラフでの処理

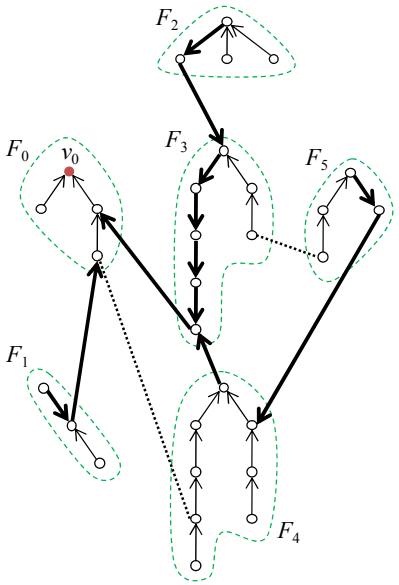


図 4　求めた全域木

ただし，入射するクラスタ外向辺が e_{not} ならば，その端点ノードは子とみなさない．

3.3.2　根ノード v_0 以外のノードのアルゴリズム

v_0 以外の任意の始動ノード v は，初めに親に**起動メッセージ**を送信する．v の親が v の隣接に存在しなければ，v はフラグメントの根であることがわかる．起動メッセージを受信したノード v' は，親に起動メッセージを送信する．ただし，v' がすでに親に起動メッセージを送信している場合は，起動メッセージを送信せず，v' の所属するフラグメントの根からのメッセージを待つ．あるフラグメント F' のすべてのノードが始動ノードでない場合，いずれ F' のあるノードが，他のフラグメントのノードから試験メッセージを受信し，親に起動メッセージを送信する．いずれ，すべてのノードが局所アルゴリズムの実行を開始する．

局所アルゴリズムの実行を開始した各フラグメントの根 v_r は，はじめに所属するフラグメント内に**初期化メッセージ**を放送する．v_r による初期化メッセージを受信した各ノード v_i は，FID の設定，重みの設定，**試験メッセージ**の送信を行う．試験メッセージは，v_i に接続する各辺が外向辺かどうかを判定するために送信する．試験メッセージを受信したノード v_j は，自身の FID を返信する．これを受信した v_i は，自身の属すクラスタに v_j のフラグメントが含まれないとき，返信を受信した辺を外向辺と判定する．

v_i は試験メッセージの返信を受信した各辺について，端点ノードの FID を記憶する．これは，自身のクラスタが新しいクラスタに置換されるとき，もしくは根クラスタに含まれるとき，外向辺として認識した辺が**内部辺**になるためである．ここで，内部辺とは両端点ノードのフラグメントが互いに同じクラスタに属す辺をいう．

初期化メッセージの帰還フェーズでは，各ノードが自分とその子孫の中で外向辺をもつ最も重みの小さいノードを選択し，その重みとノードの識別子を親に報告する．いずれ，v_r は自身のクラスタ内で，外向辺をもつ最も重みの小さいノード v_s の重みとその識別子を知り，v_s の外向辺をクラスタ外向辺に決定する．

クラスタ外向辺を決定した v_r は，v_s の重みと識別子を**外向辺決定メッセージ**に含めて放送する．外向辺決定メッセージを受信した各ノード v_k は，v_k の重みから v_s の重みを減じる．さらに，$v_k = v_s$ であれば自身がクラスタ外向辺の始点ノードであることを知る．

外向辺決定メッセージの放送を終えたことを知った v_r は，クラスタ外向辺が根クラスタに接続するかどうかを判定するため，自身のクラスタに含まれるすべてのフラグメントの FID を含んだ**識別子巡回メッセージ**を送信する．v_r が送信する識別子巡回メッセージは，v_r から v_s までの経路上のノードを伝搬し，v_s は自身のクラスタ外向辺を介して他クラスタのノードに送信する．他のクラスタから識別子巡回メッセージ m を受信したノードは，クラスタ木の親にメッセージを転送する．m を受信した v_r は，m に含まれる FID を初めて受信したならば，自身のクラスタのクラスタ外向辺から m を転送する．いずれ，v_r は識別子巡回メッセージが v_0 に伝搬するならば吸収メッセージを受信し，そうでなければ自身の FID の識別子巡回メッセージを受信する．

まず，v_r が自身の FID の識別子巡回メッセージを受信するときの処理を説明する．自身の FID の識別子巡回メッセージを受信した v_r は，自身のフラグメントが有向閉路に属すことを知る．このとき，v_r はその有向閉路内の他のクラスタに属すフラグメントの FID の収集を行うため，自身の FID を含めた**閉路検出メッセージ**を送信する．閉路検出メッセージは，識別子巡回メッセージと同様に v_r から v_s までの経路上のノードを伝搬し，v_s が自身のクラスタ外向辺を介して他クラスタのノードに送信する．他のクラスタから閉路検出メッセージ m' を受信した v_r は，自身のクラスタのクラスタ外向辺から m' を転送する．いずれ，v_r は自身の FID の閉路検出メッセージを受信する．このとき，v_r は有向閉路内のクラスタに属すすべてのフラグメントの FID を知る．ここで，識別子巡回メッセージと閉路検出メッセージの違いは，閉路検出メッセージでは有向閉路内のクラスタに属すフラグメントの FID のみを有向閉路内のノードが受信するのに対し，識別子巡回メッセージでは有向閉路外のクラスタに属すフラグメントの FID も有向閉路内のノードが受信しうることである．

有向閉路内のクラスタに属すすべてのフラグメントの FID を知った v_r は，有向閉路内のクラスタヘッドの中で FID の値が最も小さいクラスタヘッドを新しいクラスタ $clst_{new}$ のクラスタヘッドに選出する．v_r が新しいクラスタヘッドに選出されなければ，新しいクラ

スタヘッドからのメッセージに従った処理を行う．以降では，v_r が新しいクラスタヘッドに選出されたときの処理を示す．新しいクラスタヘッドに選出された v_r は，$clst_{new}$ に含まれるすべてのフラグメントの FID を含めた**置換メッセージ**を $clst_{new}$ 内で放送する．置換メッセージを受信した各ノードは，$clst_{new}$ に含まれる FID のノードを端点に持つ外向辺を内部辺として再認識する．置換メッセージの帰還フェーズでは，$clst_{new}$ のクラスタ外向辺を決定するため，各ノードが外向辺を探索する．そして，$clst_{new}$ クラスタ外向辺を決定した v_r は，外向辺決定メッセージ，識別子巡回メッセージの処理を行う．以降，v_r の属すクラスタのクラスタ外向辺が根クラスタに接続するまで，新しいクラスタへの置換を繰返す．

次に，v_r が v_0 からの**吸収メッセージ**を受信するときの処理を説明する．吸収メッセージは，あるクラスタ $clst_i$ から識別子巡回メッセージを受信した v_0 によって送信される．吸収メッセージには，$clst_i$ とすでに根クラスタに合併されたすべてのフラグメントの FID が含められる．吸収メッセージを受信した根クラスタに属す各ノード v_a は，すべての子に吸収メッセージを転送する．また，v_a が $clst_i$ のクラスタ外向辺 e と接続するとき，v_a は e の始点ノード v_s を v_a の子として設定し，木上の隣接ノード集合を表す出力 $N^F(v_a)$ に v_s を追加する．v_s は $clst_i$ 内で最初に吸収メッセージを受信する．吸収メッセージを受信した v_s は，$clst_i$ 内で自身を根とするクラスタ木に従って吸収メッセージを放送する．ここで，v_s を根とする $clst_i$ のクラスタ木は，$clst_i$ のクラスタヘッドを根とするクラスタ木と異なる場合があることに注意する．吸収メッセージを受信した $clst_i$ に属す各ノード v_b は，メッセージの送信元ノード v_p を親を表す出力 $P(v_b)$ に設定し，出力 $N^F(v_b)$ に v_p を追加する．さらに，v_s を根とするクラスタ木における v_b の各子ノード v_c を，$N^F(v_b)$ に追加する．

根クラスタに属す各ノード v_t は，吸収メッセージを受信するたびに，自身に接続する各辺が内部辺になるかどうかを判定する．いずれ，すべてのクラスタが根クラスタに合併され，v_t はすべての FID を受信する．自身に接続するすべての辺が内部辺になったノードは，局所アルゴリズムの停止処理を開始する．局所アルゴリズムの停止は全域木の葉から根 v_0 に向かって進み，停止処理を開始したノードがすべての子から**停止メッセージ**を受信したとき，親に停止メッセージを送信して自身の局所アルゴリズムの実行を停止する．

3.3.3 根ノード v_0 のアルゴリズム

v_0 は始動ノードのとき，または他のノードからメッセージを受信したとき，直ちに v_0 のアルゴリズムを実行する．

局所アルゴリズムの実行を開始した v_0 は，所属するフラグメント内に**初期化メッセージ**を放送する．同時に，v_0 は自身に接続する各辺の他方の端点ノードの FID を知るために，**試験メッセージ**の送信を開始する．その後，v_0 は他のクラスタからの**識別子巡回メッセージ**の伝搬を待つ．識別子巡回メッセージを受信した v_0 は，**吸収メッセージ**を $clst'$ を含めた根クラスタ全体に放送する．また，識別子巡回メッセージを受信してクラスタを合併するたびに，自身に接続する各辺が外向辺であるか内部辺であるかを判定し，すべての辺が内部辺になったとき，局所アルゴリズムの停止処理を開始する．いずれ，v_0 はすべての子から停止メッセージを受信して自身の局所アルゴリズムを停止し，その結果ネットワーク全体のアルゴリズムが停止する．

4 アルゴリズムの正当性と性能

本章では，提案する分散アルゴリズムの正当性と，アルゴリズムの停止までに必要なメッセージ複雑度について説明する．

補題 1 提案アルゴリズムはいずれ停止する．

証明 各クラスタヘッドは，自身の属すクラスタが根クラスタに合併されるまで，クラスタの置換を繰り返す．よって，いずれすべてのフラグメントが根クラスタに合併される．根クラスタに属す各ノードは，自身に接続するすべての辺を内部辺として認識するまで，根クラスタに合併されたフラグメントの FID の受信を待つ．すべてのフラグメントはいずれ根クラスタに合併されるため，根クラスタに属す各ノードはすべてのフラグメント識別子を受信する．したがって，各ノードに接続するすべての辺がいずれ内部辺として認識される．自身に接続する全辺が内部辺であることを検出したノードは，すべての子が停止したときに親に停止することを報告して停止する．いずれ，根ノード v_0 がすべての子が停止したことを検出して停止する．したがって，提案アルゴリズムはいずれ停止する．

補題 2 提案アルゴリズムの停止状況は，解状況である．

証明 アルゴリズムが停止するとき，各ノード v_i が属すフラグメントは根クラスタに含まれる．v_i は自身のクラスタ $clst_i$ が根クラスタに合併されるとき，v_0 からの吸収メッセージを受信する．$clst_i$ 内では，はじめにクラスタ外向辺の始点ノード v_s が吸収メッセージを受信し，その後 $clst_i$ 内で v_s を根とするクラスタ木に従って吸収メッセージが伝搬する．v_i は吸収メッセージの送信元のノード v_p を親を表す出力 $P(v_i)$ に設定し，v_p とクラスタ木の各子ノード v_c を木上の隣接ノード集合を表す出力 $N^F(v_i)$ に追加する．また，v_i が v_c に吸収メッセージを送信するため，いずれ v_c は v_i を親に設定

する．根クラスタに含まれる各ノード v_j は，他のクラスタのノード v_k から識別子巡回メッセージを受信したとき，v_k を $N^F(v_j)$ に追加する．上記の場合以外に，v_j が出力を変化させることはない．

よって，v_0 を除いた全ノードに親が存在し，かつ親を辿ったときに根ノード v_0 に到達する．したがって，アルゴリズムの停止状況では，解状況の条件が満たされる．

補題 3 提案アルゴリズムは仮想グラフ上で重み最小有向全域木を求める．

証明 まず，仮想グラフが強連結であることを証明する．本稿で扱う実グラフは連結であるため，任意の異なる 2 つのフラグメントの間に経路が存在する．仮想グラフでは，実グラフの異なる 2 つのフラグメント間の各辺を，双方向の有向辺に対応させる．そして，実グラフの異なる 2 つのフラグメント間に辺が存在するとき，それらの中の 1 つが仮想グラフの辺として選択される．よって，仮想グラフは強連結である．

次に，提案アルゴリズムが仮想グラフ上で重み最小有向全域木を求めることを，背理法を用いて証明する．提案アルゴリズムが求める有向全域木 T' が，重み最小でないと仮定する．このとき，仮想グラフ上での重み最小有向全域木 T が存在する．そして，T には含まれるが，T' には含まれない有向辺 $e = (u_i, u_j)$ が存在する．また, T' には含まれるが, T には含まれない $w(e) < w(e')$ なる有向辺 $e' = (u_i, u_k)$ が存在する．ここで，e を T' に加えると，u_i の出次数は 2 になる．このとき，T' から e' を取り除くことで，新しい有向全域木 T'' が構成でき，$W(T'') < W(T')$ が成り立つ．ここで，$W(T')$, $W(T'')$ はそれぞれ T'，T'' に含まれる辺の重み総和を表わす．しかし，アルゴリズムは各クラスタから出射する重み最小の辺を選択するため，e' を選択しない．つまり，アルゴリズムは重み最小でない有向全域木を求められず，仮定に矛盾する．よって，提案アルゴリズムは，重み最小有向全域木を求める．

補題 4 提案アルゴリズムが停止したとき，初期状況 c_0 における木上での隣接関係が維持されるもとで，親を変化させるノード数が最小化される．

証明 各ノード v_i が木上での隣接ノード集合を表わす出力 $N^F(v_i)$ を変更するのは，

> v_i の属すクラスタが根クラスタに合併される際の吸収メッセージを受信したとき，
> v_i が根クラスタに属していて識別子巡回メッセージを受信したとき，

に限る．しかし，$N^F(v_i)$ に含まれる要素数が減少することはない．つまり，計算途中で各 v_i の $N^F(v_i)$ の要素数は単調増加するため，c_0 における木上での隣接関係は維持される．

吸収メッセージが伝搬するクラスタ外向辺は，仮想グラフにおいて v_0 を含むフラグメントを根とする重み最小有向全域木の辺である．仮想グラフでは，木上の隣接関係を維持するもとで，親を変化させるノード数を，フラグメント間を結ぶ辺の重みとする．そのため，木上の隣接関係を維持するもとで，親を変化させるノード数はネットワーク全体で最小になる．

以上より，提案アルゴリズムが停止したとき，c_0 における木上での隣接関係が維持されながら，親を変更したノード数が最小化される．

定理 1 提案アルゴリズムは，出力変化極小全域木更新問題を正しく解く．

証明 補題 1，2，4 より明らか.

定理 2 入力ネットワーク G について，k を G のフラグメント数，n を G のノード数，m を G のリンク (辺) 数とする．提案アルゴリズムは $O(kn + m)$ のメッセージ複雑度で全域木を求めて停止する．

証明 提案アルゴリズムは，各クラスタヘッドによるメッセージが，クラスタ内に伝搬することで処理が進む．最初の各クラスタヘッド (つまり，フラグメントの根) が，自身のクラスタ内でメッセージを放送するために必要なメッセージ数は，ネットワーク全体で $O(n)$ である．よって，初期化メッセージと最初の外向辺決定メッセージで必要なメッセージ数はネットワーク全体で $O(n)$ である．試験メッセージは，フラグメントの辺でない各辺について両端点からのメッセージが往復するため，必要なメッセージ数はネットワーク全体で高々 $4m$ である．

クラスタの合併は，有向閉路内のクラスタの合併，根クラスタへの合併をあわせて，高々 k　1 回行われる．以降，1 回のクラスタの合併で行われる処理に必要なメッセージ数を示す．有向閉路内のクラスタの合併で行われる処理は，識別子巡回メッセージ，閉路検出メッセージ, 置換メッセージ, 外向辺決定メッセージであり，必要なメッセージ数はネットワーク全体で $O(n)$ である．根クラスタへの合併で行われる処理は，新しい根クラスタ全体に対する吸収メッセージの放送である．これに必要なメッセージ数はネットワーク全体で $O(n)$ である．したがって，クラスタの合併の処理全体で $O(kn)$ のメッセージ数がネットワーク全体で必要である．

また，起動メッセージ，停止メッセージに必要なメッセージ数はネットワーク全体で $O(n)$ である．

以上より，提案アルゴリズムは $O(kn + m)$ のメッセージ複雑度で全域木を求めて停止する．

5　まとめ

本稿では，ノードの出力の変化数を最小化する全域木更新問題として，出力変化極小全域木更新問題を提案した．そして，この問題を解くメッセージ複雑度 $O(kn + m)$ の分散アルゴリズムを提案した．今後は，提案した更新アルゴリズムの自己安定化を図る．

参考文献

[1] Gheorghe Antonoiu and Pradip K. Srimani. A self-stabilizing distributed algorithm to construct an arbitrary spanning tree of a connected graph. *Computers and Mathematics with Applications, 30: 1-7*, 1995.

[2] Lelia Blin, Maria Gradinariu Potop-Butucaru, and Stephane Rovedakis. A superstabilizing $\log(n)$-approximation algorithm for dynamic steiner trees. *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, to appear.

[3] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, Vol. 17, pp. 643–644, 1974.

[4] Shlomi Dolev. *Self-Stabilization*. The MIT Press, 2000.

[5] Sukumar Ghosh. *Distributed systems: an algorithmic approach*. Chapman & Hall/CRC, 2007.

[6] Pierre A. Humblet. A distributed algorithm for minimum weighted directed spaning trees. *IEEE Transactions on Communications*, Vol. COM-31, No. 6, pp. 756–762, 1983.

[7] Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Output stability of self-stabilizing protocols against topology changes and transient faults. *International Conference on Application and Principles of Information Science (APIS)*, pp. 306–310, 2009.

[8] 片山喜章, 長谷川敏之, 高橋直久. 任意の単一リンク故障を考慮した生成木構成強安定プロトコル. 電子情報通信学会論文誌 (D-I), Vol. J88-D-I, No. 11, pp. 1669–1678, 2005.

[9] 三浦康史, 増澤利光, 都倉信樹. 最短経路木更新問題を解く分散アルゴリズム. 電子情報通信学会論文誌 (D-I), Vol. J77-D-I, No. 11, pp. 33–44, 1994.

[10] 朴政鎬, 増澤利光, 萩原兼一, 都倉信樹. 重み最小生成木更新問題を解く分散アルゴリズム．電子情報通信学会論文誌 (D-I), Vol. J73-D-I, No. 10, pp. 802–810, 1990.

自己安定性を有した耐ビザンチン故障レプリケーションの検討

中村 純哉[1]　櫟 粛之[2]　大下 福仁[1]　角川 裕次[1]　増澤 利光[1]
[1] 大阪大学 情報科学研究科　[2]NTT CS 基礎研究所

概要

不正アクセスや乗っ取りなどの悪意のある攻撃から，電子政府や電子商取引などの重要なサービスを守ることは，近年ますます重要性を増している．このようなビザンチン故障に耐えられる信頼性の高いサービスを実現する一つの方法として，状態機械レプリケーションによる方法が広く知られている．この方法では，1 つのサービスを n 台のレプリカに複製し，全てのレプリカが同じリクエストを同じ順序で処理するように協調させる．クライアントは $f+1$ 台のレプリカから同内容の結果を受け取ったときに，はじめてその結果を受理する．このようにすることで，f 台までのレプリカのビザンチン故障に耐えることのできるサービスを実現することができる．故障には，プロセスの揮発性メモリの内容を任意に書き換えてしまうような故障も存在する．このような故障は一時故障としてモデル化され，広く研究されている．自己安定プロトコルは， 任意数の一時故障から自律的にシステムが正当な状況に復旧することを保証する，優れた耐故障性を提供する．

本発表では，自己安定性と耐ビザンチン故障性を兼ね備えた状態機械レプリケーションプロトコルの実現を検討する．状態機械レプリケーションは優れた耐故障手法であるが，故障レプリカの数が閾値 f を超えてしまうと，正常なサービスを提供することができなくなってしまうという問題がある．理論的解析から，f 台のレプリカに対する耐ビザンチン故障性を実現するには，$n \quad 3f+1$ 台のレプリカが必要なことが証明されている．そのため，閾値 f を大きくすることは高いコストを必要とし，好ましくない．提案プロトコルは，自己安定性を実現することによって，一時故障についてこの問題の解決を行う．

自律分散ロボットのモデルの統一的分類とその考察

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

概要 本研究は，自律分散ロボット群の既存のモデルを統一的に分類し，各モデル間の関係を明確にすることを目的とする．既存のモデルとして，完全同期，SYM，CORDA などが挙げられる．本稿では，実行のアトミック性，同期性の２つの性質によりモデルを分類し，各モデル間の関係について考察する．

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 [1] がある．この研究によって，初めて自律分散ロボット群に関する理論的モデルが構築された．この文献での実行モデルは準同期モデル（SYM）と呼ばれ，ロボットの動作に制限がある．文献 [2] では [1] 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．また，Souissi らによる研究 [3] によって，同期性にある値によって制限をかけたモデルが提案されている．その他にもロボットのモデルは，匿名性，無記憶性，ロボット同士の通信の有無など，様々な要素から構成されている．

これまで，上記のように様々な視点からモデルが提案されてきたが，そのモデル間の関係は完全には分かっていない．つまり，各モデルが解ける問題のクラスの差が明確に分かっていない．そこで本稿ではロボットの実行モデルに注目し，上記のようなこれまでに存在した実行モデルを，実行のアトミック性，同期性の 2 つを用いて分類した．実行のアトミック性とは，ロボットの不可分な実行のことで，何を不可分とするかで場合分けを行った．同期性は, [3] で導入された k-bounded-scheduler を基に，ある値 k で非同期な実行を制限する．これらを基にモデルを分類し，各モデルで解ける問題のクラスの差や，何がそのクラスの差に影響を与えているかを明らかにする際の指針とする．

2. 既存モデル

既存の実行モデルを説明する．ロボットは観測，計算，移動，待機の 4 つをそれぞれ 1 動作として，観測から待機で 1 サイクルとする．これまで提案されてきたロボットの実行モデルとして，代表的なものに，完全同期，SYM，CORDA の 3 種類がある．完全同期は，全ロボットの動作のタイミングが完全に同期しているモデルである．しかしロボットは，観測情報のみから移動先を決定するため，本質的には，全ロボットの観測時刻が完全に一致していれば，それは完全同期モデルとみなすことが出来る．これは，観測時刻さえ一致していれば，ロボットがいつ計算，移動しようが，ロボットの観測情報が変わらず，そのため実行も変わらないためである．SYM は，ロボットの観測時刻は一致しているが，各サイクルで，サイクルを実行しないロボットが存在する．完全同期との違いとしては，サイクルの実行回数に差が出てくることが挙げられる．CORDA は，動作のタイミングに仮定を何も置かない．そのため観測時刻は一致しておらず，また移動中のロボットを観測する場合がある．また，完全同期，SYM と違い，サイクルの長さがロボット間で一致していないため，他のロボットよりも古い観測情報を基に移動する場合がある．

3. モデルの分類

これまでのロボット群の実行モデルを新たな方法で分類する．まず，ロボットが実行する動作を抽象化する．これまでは観測，計算，移動，待機の４つが考えられていた．しかし，計算，待機はそれぞれのロボットが個別に行うものであり，他のロボットが計算もしくは待機中のロボットを観測しても，ただ停止しているだけにしか見えない．したがって今回は，ロボットの動作を「観測」,「移動」の 2 つに抽象化し，観測，移動で 1 サイクルとする．また，離散時間を導入する．離散時間には非負整数 0,1,2,... の連続した番号を付け，各々には，少なくとも 1 台のロボットが観測を行うとする．これは，今回考えるモデルでは，ロボットは観測情報のみから次の移動先を決定するため，ロボットが観測する時刻とその時の観測情報を表現出来れば，ロボットの実行が表現できるためである．

このようなロボットの実行に対して，分類を行う際の指針として，各モデル間の本質的な違いが表現出来ているかどうかに注目する．各モデル間の本質的は違いとしては，観測時刻の一致性，サイクル長の一致性，サイクルの実行回数に差があるかどうか，移動中に観測されるかどうかが挙げられる．今回は，これらの差を表現するために，実行のアトミック性，同期性の２つの性質を用いて分類する．

3.1 実行のアトミック性

アトミック性とは，ロボットの実行の不可分性である．あるロボットのアトミックな実行中には，他のロボットは任意の動作の開始，もしくは停止をすることが出来ない．ここではロボットの実行において，何をアトミックな動作とするかで，３種類：（１）1 サイクルがアトミックな場合,（２）1 動作（観測，移動）がアトミックな場合,（３）アトミックな動作が無い

場合に場合分けする．離散時間を考えるため，1 単位時間にアトミックな実行を当てはめる．（1）1 サイクルがアトミックな場合：1 単位時間に 1 サイクルを実行する．この場合，サイクルを実行するロボットについては観測時刻が一致し，またサイクル長が全ロボットで一致することになる．つまり，完全同期と SYM を表現出来る．（2）1 動作がアトミックな場合：1 単位時間に，観測，もしくは移動を実行する．あるロボット r_i の観測時刻と，他のロボット r_j の移動時刻が等しい時は，アトミック性の性質を保つために，r_i の観測が r_j の移動の開始前に終了すると仮定する．この場合，観測時刻はロボット間で必ずしも一致せず，サイクル長も一致しない．しかし，移動がアトミックなため，移動中に観測されることは無い．したがってこの移動を，ロボットが瞬間移動していると考える．（3）アトミックな動作が無い場合：観測，移動ともにアトミックでは無いが，観測はその動作の性質上，瞬間的に行うと仮定する．移動はアトミックでゃないが，離散時間で表すため，1 単位時間には，細分化された移動が行われるとする．この場合，移動途中の他のロボットを観測する場合がある．

3.2 同 期 性

同期性とは，ロボットのサイクルの実行回数にどれだけ差があるかを表す性質である．今回は，非負整数 k によって実行回数の差を制限する，k-bounded-asynchronous を定義し，この性質を表現する．k-bounded-asynchronous の定義のために，Full-Activation-Cycle を用いるが，これは, [3] で定義されたものを用いる．

- Full-Activation-Cycle

任意のロボットの Full-Activation-Cycle（以下 FAC）とは，ある観測時刻を t_1，その次の観測時刻を t_2 としたとき，時間 $[t_1, t_2)$ を言う．

- k-bounded-asynchronous

あるロボットが $k+1$ 回 FAC を実行する間に，他の全てのロボットは少なくとも 1 回 FAC を実行する．

$k = 0$ の場合は，全ロボットのサイクルの開始時刻，終了時刻が完全に一致し，完全同期となる．

3.3 モデルの対応関係

上で定義した 2 つの性質を用いて，既存のモデルを分類する．まず，同期性について，0-bounded-asynchronous の場合は，アトミック性に関わらず完全同期になる．これは，全ロボットのサイクルの開始時刻，終了時刻が一致する，つまり，観測時刻が一致するためである．$k > 0$ の場合は，アトミック性によって 3 つに分類出来る．アトミック性が 1 サイクルの場合は，SYM となる．1 単位時間に 1 サイクルを実行するため，観測時刻，サイクル長ともに一致している．$k > 0$ なので，サイクルの実行回数に差が現れてくる．これらの特徴より，SYM を表現出来ていると言える．アトミック性が 1 動作の場合は，瞬間移動 CORDA となる．1 単位時間に実行するのが観測もしくは移動のため，観測時刻はロボット間で完全に一致はしない．またサイクル長も決まらない．移動中に観測されることがないため，純粋な CORDA とは言えず，この場合を瞬間移動 CORDA と呼ぶ．アトミック性が無い場合は，CORDA となる．移動中に観測されうるのが，1 動作がアトミックな場合との差である．

表 1　モデルの分類

	1 サイクル	1 動作	無
0-bounded	完全同期		
1-bounded	SYM	瞬間移動 CORDA	CORDA
2-bounded			
⋮			

上記に従い分類表を作ると表 1 のようになる．この表により，完全同期，SYM，CORDA などの既存のモデルを一通り分類できた．

4. 考　　察

モデル間の関係に対して，表 1 に基づき考察する．まずは同期性について，$k = 0$ の場合は，アトミック性に依らず完全同期となり，$k > 0$ の場合はアトミック性により，SYM，瞬間移動 CORDA，CORDA と分類できる．従って，$k = 1$ を境界としてはモデル間に明らかに差が存在するのは自明である．今後はまず $k > 0$ の部分に対して，k の値でどれほど解ける問題に差があるのか，もしくは無いのか関心が持てる．現在の予想では，SYM については差がないのではないかと考えている．また，実行のアトミック性に関して，アトミック性によって解ける問題のクラスがどれほど違うのか，が考えられる．現在，表 1 において，より左にあるモデルが解ける問題クラスの方が，より右にあるモデルが解ける問題クラスよりも小さくないことは分かっているが真に大きいかどうかは定かではないため，この部分も明かにしたい．これらを通して，今回モデルの分類に用いた，実行のアトミック性，同期性において，何がどれほど各モデルによる問題の可解性の差に影響を与えているかを明かにしていきたい．

5. ま と め

本稿では，既存のモデルに対して，実行のアトミック性，同期性の 2 つの観点から統一的な分類を行った．また，それを基に，各モデル間の関係について考察を行った．今後さらにモデル間の関係を明らかにしていきたい．

文　　献

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

[2] 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).

[3] S. Souissi, Y. Yang and X. Defago: “Fault-tolerant flocking in a k-bounded asynchronous system”, Lecture Notes in Computer Science, **5401**, pp. 145–163 (2008).

自己安定アルゴリズムの多段構成を用いた 重み最小全域木 (MST) 構築アルゴリズムについて

三浦哲平　泉泰介　片山喜章　和田幸一　高橋直久

平成 21 年 9 月 6 日

1　はじめに

通信リンクによってプロセスが相互に接続されたネットワーク (分散システム) 上で，プロセスが互いに協調して問題を解くアルゴリズムが分散アルゴリズムである．通常の分散アルゴリズムでは，システム開始時のネットワーク状況があらかじめ決められた初期状況であると仮定される．一方，自己安定アルゴリズム (*self-stabilizing algorithm*) とは，任意のネットワーク状況から実行を始めても問題を解くことのできる分散アルゴリズムである．この性質から，自己安定アルゴリズムはプロセスの故障に対して耐性のある分散アルゴリズムであり，長期にわたってネットワーク状況を安定に保ち，故障に柔軟に対応することを求められる分散システムを動作させる場合に適している．

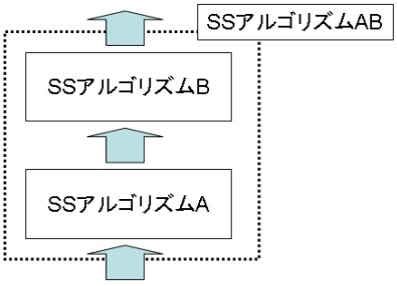


図 1: 自己安定アルゴリズムの公平な合成

S.Dolev ら [1] では，自己安定アルゴリズムの公平な合成が提案されている．公平な合成とは，例えば 2 つの自己安定アルゴリズム A,B を考える．A は外部から与えられる値を入力として実行を行う．一方，B は A の出力を入力として実行を行う．そうすると，A の出力が安定 (正当な状況に到達) することで，B の入力が安定し，いずれ B の出力も安定する．つまり，自己安定アルゴリズム A,B の公平な合成によってひとつの自己安定アルゴリズム AB が設計できる (図 1)．

また西村ら [2] では，極大クリーク分割問題を解く自己安定アルゴリズム $SSMM$ が提案されている．$SSMM$ はマッチングとマージという 2 つの自己安定アルゴリズムを重ね合わせるように合成する事よって設計されている (図 2)．この様に自己安定アルゴリズムを重ね合わせるように合成する設計手法を多段構成と呼ぶ．

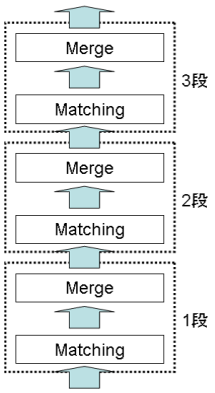


図 2: 自己安定アルゴリズムの多段構成

多段構成には以下の特徴がある．

- 複数の単純な問題を解くアルゴリズムを多段構成することによって，複雑な問題を解くアルゴリズムを設計することができる．そのため，アルゴリズムの設計や証明が容易になり，理解しやすくなる．
- アルゴリズムが解をだすのに必要なくり返し回数の分だけ段数を重ねる必要がある．
- 下段のアルゴリズムが安定するまで，上段は正しい入力が得られないので，スケジューリングによっては解状況に到達するまでに長時間必要になる場合がある．

このように多段構成による自己安定アルゴリズムの設計には利点があり，利用する価値がある．その一方で，多段構成によって解くことのできる問題のクラスを明確にすることは非常に重要である．そこで，西村ら [2] や，多段構成の性質から「グリーディ戦略で解くことのできる問題は自己安定アルゴリズムの多段構成を用いて解くことができる」という仮説をたてた．

グリーディ戦略とは，各時点で全体的なことは考えずに局所的に最良の選択をくり返すことで全体として最良の解を得るというものである．

本稿では仮説を証明するための手始めとして，グリーディ戦略で解くことのできる重み最小全域木 (MST) 構築問題を多段構成によって設計，その正しさを証明し，仮説を証明するための足掛かりとした．

以降の本稿の構成は以下の通りである．本稿の 2 章では，システムモデルと自己安定アルゴリズム，多段構成について述べる．また，MST 構築問題を定義する．3 章では，MST 構築問題を解く手法の概略を示す．4 章では，3 章で示した手法に基づく自己安定アルゴリズムを示す．5 章では，4 章で示したアルゴリズムの正当性の証明と時間複雑度の解析を行う．

*名古屋工業大学大学院 工学研究科 情報工学専攻

2 システムモデルと問題の定義

2.1 分散システム

一般に分散システムは端末をノード，通信リンクを辺とみなし，グラフで表現可能である．本稿では，分散システムを重み付きグラフとして表現する．

重み付きグラフ $G = (V, E, c)$ は以下のように与えられる．$V = \{v_1, v_2, ..., v_n\}$ は相異なる識別子を持つノードの集合，n はノード数とする．E　$V \times V$ はノード間の無向辺の集合とする．c はコスト関数である．つまり，ノード v_i とノード v_j の間に辺が存在する時，v_i と v_j は隣接しているといい，$e_{ij} = (v_i, v_j) \in E$ の重みは $w_{ij} = w_{ji} = c(e_{ij})$ となる．また，全ての辺の重みは相異なる自然数とする．(同じ重さの辺が存在しても，その両端ノードの ID が固有であり，全順序をつけることができるので，相異なる自然数に規定しても一般性を失わない)

各ノード v_i は固有の識別子 ID_i，隣接ノードの集合 VN_i，接続辺の集合 VE_i を持つ．

2.2 システムの状況

各ノード v_i の変数の集合を，そのノードの状態と呼び，q_i で表す．各ノードの状態の集合をシステム全体の状況と呼び，$conf = (q_0, q_1, ..., q_n)$ で表す．v_i の取りうる全ての状態を Q_i，システム全体が取りうる状況を　とすると，　$= Q_0 \times Q_1 \times ... \times Q_n$ となる．

2.3 通信モデル

本稿では，通信モデルとして状態通信モデル (*state communication model* または，*state reading model*) を仮定する．これは隣接ノードの状態を遅延なしで直接読むことができるモデルである．書き込みは各ノードが自身の持つ変数に対してのみ行うことができる．

2.4 実行

ノードの部分集合を S　V とする．ある状況 $conf_i \in$　において，S に属するノードが同時にアルゴリズム A の 1 原子動作を実行することによって $conf_{i+1}$ になったとき，$conf_{i+1} = C(conf_i, S, A)$ と表す．ここで 1 原子動作とは，以下の 3 つの動作を全て行うものとする．

1. 隣接ノードの変数を読み込む

2. 隣接ノードの変数と自身の変数をもとに内部計算を行う

3. 内部計算の結果を自身の変数に書き込む

空でないノードの集合の無限系列 $T = S(0), S(1), ...,$ をスケジュールと呼ぶ．状況の無限系列 $EXEC = conf_0, conf_1, ...,$ が $conf_{i+1} = C(conf_i, S(i), A)$ を満たすとき，$EXEC$ を「初期状況 $conf_0$，スケジュール T に対するアルゴリズム A の実行」と呼び，$EXEC(A, T, conf_0)$ で表す．また，T に全ノードが無限回現れるとき，公平なスケジュールという．本稿では任意の $t \geq 0$ に対して，$|S(t)| > 0$ かつ公平であるようなスケジュール (*D-daemon*) を扱う．

2.5 自己安定アルゴリズム

自己安定アルゴリズム A は，任意の初期状況から開始される実行がやがて到達し，その後現れる状況もそれに含まれる正当な状況 $\Lambda \in$　に対して次の 1，2 を満たす．

1. 到達可能性: 任意の状況 $conf_0 \in$　と任意のスケジュール T に対し，$EXEC(A, T, conf_0)$ に Λ に含まれる状況が現れる．

2. 閉包性: 任意の状況 $conf \in \Lambda$，ノードの任意の集合 S に対し，$conf' = C(conf, S, A)$ とすると $conf' \in \Lambda$ となる．

つまり，任意の初期状況から開始しても，公平なスケジューリングによるアルゴリズム A の実行は，有限時間内に正当な状況に到達し，一度正当な状況に到達すると，それ以降の状況は正当な状況となる．

2.6 自己安定アルゴリズムの多段構成

自己安定アルゴリズム $A_0, A_1, ..., A_k$ を，各アルゴリズム $A_j (0 \leq j \leq i)$ の出力が A_{i+1} の入力となるように組み合わせたアルゴリズム A を $A_0, A_1, ..., A_k$ の多段構成という．また，A は各 A_i のステップを無限にしばしば実行する．ことのき，$A_0, A_1, ..., A_i$ がそれぞれ正当な状況に到達すると，A_{i+1} の入力が安定し，いずれ A_{i+1} は正当な状況に到達する．これをくり返すことで，A_k が正当な状況に到達する．つまり，A_0〜A_k の全てが正当な状況となり，この時アルゴリズム A が正当な状況に到達したという．

2.7 アルゴリズム記述形式

本稿ではアルゴリズムを以下の形式で記述する．

```
入力変数
出力変数

R1: Condition ⟶ Action
R2: Condition ⟶ Action
...
...
```

各ノードは，全ての Condition を上から順に評価し，真となる全ての Condition に対する Action を実行する．

2.8 非同期ラウンド

各ノードの計算速度や通信速度が異なるようなシステムでの時間複雑度の尺度として非同期ラウンドがある．直感的には，全てのノードが少なくとも 1 回，1 原子動作を行ったときを 1 ラウンドと呼ぶ．1 ラウンドの間に 1 原子動作を何回も行うノードがあってもよい．正確にいうと，$S(0) \cup S(1) \cup ... \cup S(t_1) = V$ となるような最小の t_1 に対して，$R_1 = S(0), S(1), ..., S(t_1)$ を第 1 ラウンドと呼ぶ．同様に，$S(t_i + 1) \cup S(t_1 + 2) \cup ... \cup S(t_2) = V$ となる最小の t_2 に対して，$R_2 = S(t_1 + 1), S(t_1 + 2), ..., S(t_2)$ を第 2 ラウンドとする．本稿ではこのラウンドを用いて，アルゴリズムの時間複雑度をラウンド数で評価する．

2.9 *MST* 構築問題の定義

本稿では各ノード v_i に ID_i, VN_i, VE_i が与えられたとき，以下に定義するような *MST* を G 上に構築する自己安定アルゴリズムを提案する．

定義 1 *(MST の定義)*
グラフ $G = (V, E, c)$ に以下の条件を満たす部分グラフが構築されている時，*MST* が構築されている．

- G 上の連結全域木の中で，重みの総和が最も小さいもの

定義 2 （*MST* 構築問題を解く自己安定アルゴリズムの定義）
与えられる任意の初期状況から実行を開始し，有限時間内に定義 *1* を満たす辺の集合を出力するアルゴリズム

3 *MST* 構築手法の概略

3.1 提案手法で用いる基本戦略

本稿では，グラフ上で根つき木構造を構成している部分グラフをフラグメントと呼び，各フラグメントは相異なるフラグメント ID(*FID*) を持つ．フラグメントを構成する辺をフラグメント構成辺 (*FE*) と呼ぶ．また，自身の属するフラグメント以外のフラグメントに属する (*FID* が異なる) ノードと接続している辺を外向辺 (*OE*) と呼び，その中で重みが最小の辺を重み最小外向辺 (*MOE*) と呼ぶ．

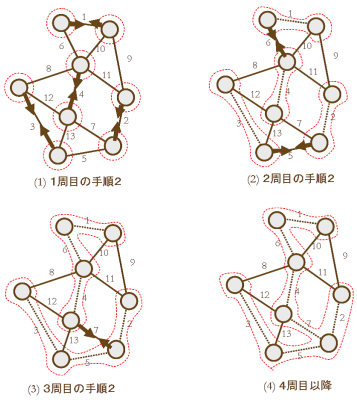


図 3: *MST* 構築の実行例

提案手法では，Gallager ら [3] より,「グラフ上の任意のフラグメントの *MOE* は *MST* を構成する辺に属する.」ことを利用する．まず G 上の各ノードを自ノードのみからなるフラグメントとする．次に，フラグメントの *MOE* を選択する．そして，*MOE* を共有するフラグメント対を *MOE* を構成辺とするひとつのフラグメントにする．その結果を用いて再び *MOE* を選択し，*MOE* を共有するフラグメント対をひとつのフラグメントにする．以上の動作を G 上のフラグメントがひとつになる (全てのノードが同じ *FID* を持つ) までくり返すことによって *MST* を構築する．例を図 3 に示す．図 3 ではノード群を囲む点線がフラグメントを表す．矢印は各フラグメントの *MOE* を表す．ノード間に結ぶ点線はフラグメント構成辺であることを表す．このアイデアを具現化した以下の手順を，自己安定アルゴリズムの多段構成によって実現する．

手順 1. 各ノードを自ノードのみからなるフラグメントにする

手順 2. 各フラグメントの *MOE* を求める

手順 3. *MOE* を共有するフラグメント対をひとつのフラグメントにする

手順 4. G 上のフラグメント数が 1 つになるまで手順 2,3 をくり返す

これらの手順を実現するために，*Init*(手順 1)，*Share*(手順 2)，*Merge*(手順 3) の 3 つの自己安定アルゴリズムを多段構成にする．以下，各アルゴリズムの概略を説明する．

Init の概略

G 上の各ノードを自ノードのみからなるフラグメントとする自己安定アルゴリズム．各ノードのフラグメントを構成するための変数を初期化する．

Share の概略

フラグメントの *MOE* をフラグメントを構成するノード間で共有する自己安定アルゴリズム．各フラグメントの葉ノードから根ノードに向かって順に局所的な (自身に接続する辺と子ノードから得た情報の中から)*MOE* を伝える (*gathering*)．やがて根ノードはそのフラグメントの *MOE* を知ることができ，それをフラグメント全体に (木構造を用いて) 放送する (*broadcasting*)．*gathering*，*broadcasting* の動作により，フラグメントを構成する全てのノードが *MOE* を共有する．

Merge の概略

MOE を共有するフラグメント対をひとつのフラグメントに結合する自己安定アルゴリズム．各フラグメントは結合回数を示すレベルを持つ．

同レベルのフラグメントの結合は，両フラグメントを *MOE* によって連結にした後に，*MOE* の両端ノードのうち *ID* の小さい方を結合後のフラグメントの根ノードとし，根からフラグメント全体に情報を放送するようにしてフラグメントを再編すると同時に，レベルを 1 増やす．例を図 4 に示す．例 4 では，ノード群を囲む点線がフラグメントを表す．矢印は親子関係を表す．ノード内の数字はレベルを表す．3 重線は根ノードであることを表す．

レベルが異なるフラグメント結合は，まず MOE を構成辺に加えることでフラグメント対をひとつの連結グラフとする．さらに，自身よりレベルの高いノードと FE によって隣接するノードから順にレベルの高いノードを親とし，レベルを親ノードと同じにする．そうすることで，レベルの高いフラグメントがレベルの低いフラグメントを吸収するように結合する．例を図 5 に示す．

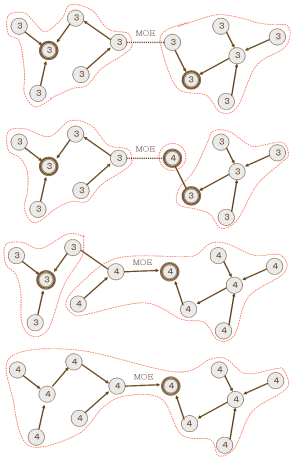


図 4: 同レベルのフラグメント結合の実行例

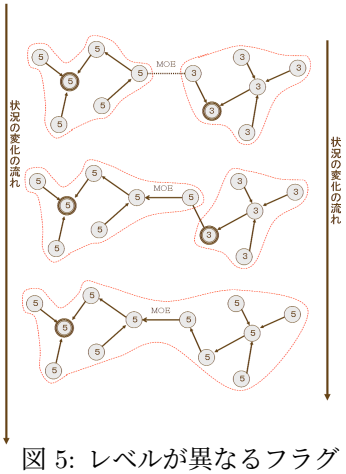


図 5: レベルが異なるフラグメント結合の実行例

3.2 提案アルゴリズムの構成

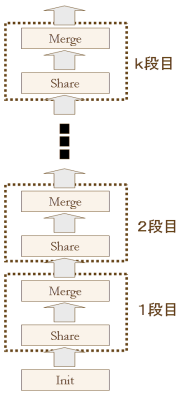


図 6: 提案アルゴリズムの多段構成のイメージ

提案手法では，自己安定アルゴリズム $Init$，$Share$，$Merge$ を必要な数だけ並行に動作させる．つまり，$Share$ によって MOE を求め (手順 2)，その結果を使って $Merge$ によって MOE を共有するフラグメント対をひとつにする (手順 3)．このアルゴリズムの組み合わせを 1 段とし，これを必要なだけ積み上げる (多段構成する) ことで，手順 4 のくり返しを実現する．提案手法のイメージを図 6 に示す．

4 提案アルゴリズム $SSSM$

$SSSM$ は各ノード v_i が固有の識別子 ID_i，隣接ノードの集合 VN_i，接続辺の集合 VE_i を入力として与えられ，定義 1 を満たす部分グラフを構成する隣接ノードの集合 FN_i と接続辺の集合 FE_i を出力する．

x 段目の状況を $C^x \in$　で表し，実行しているアルゴリズムを $Share^x$, $Merge^x$ と表す．$Init$ の出力は C^0 となる．$Share^x$ と $Merge^x$ を合成したアルゴリズムは $C^x (x \geq 0)$ を入力として，C^{x+1} を出力する．$Init$ の後に $Share$ と $Merge$ を n-1 段合成したアルゴリズム $SSSM$ を各ノードで実行することによって得られる状況 C^{n-1} から FN_i, FE_i が得られる．(図 7) 各ノードはグラフ G のノード数 n を知っていると仮定する．

4.1 定数

各ノード v_i は以下の値を持つ．

- ID_i：v_i の固有識別子
- VN_i：v_i の隣接ノードの集合
- VE_i：v_i に接続する辺の集合

ID_iは故障にあっても値は変化しないものとし，VN_i，VE_i については，故障によって誤った値に変化してもすぐに正常な値が得られるものとする．

4.2 フラグメントの定義

- フラグメント ID が FID であるフラグメント F について以下のように定義する．

$$F = (Fv, Fe) \begin{cases} \cdot\, Fv = \{v_i | FID_i = FID\} \\ \cdot\, Fe = \{(v_i, v_j) | v_i, v_j \in Fv \wedge (v_i, v_j) \in E\} \\ \cdot\, \exists v_i \in Fv \text{ を根とする木構造をもつ} \\ \quad (v_i\text{の } ID \text{ とフラグメント } ID \text{ は等しい}) \end{cases}$$

4.3 変数

各ノード v_i は x 段目のアルゴリズムに対して以下の変数を持つ．

- FID_i^x：v_i の属するフラグメントの ID
- FN_i^x：同じフラグメントに属する隣接ノード
  つまり，$FN_i^x = \{v_j | v_j \in Fv \wedge (v_i, v_j) \in Fe\}$
- FE_i^x：自身に接続しているフラグメント構成辺
  つまり，$FE_i^x = \{(v_i, v_j) | v_j \in FN_i^x\}$
- PAR_i^x：親ノード
- $tMOE_i^x$：自身と子ノードの中で重み最小の外向辺
  $tMOE_i^x = (v_j, v_k)(v_j \in (FN_i^x \cup \{v_i\} - \{PAR_i^x\}) \wedge v_k \in VN_j^x \wedge v_k \notin FN_j^x)$
- MOE_i^x：自身の属するフラグメントの最小外向辺
- FL_i^x：自然数 (v_i の属するフラグメントのレベル)

4.4 各アルゴリズムの入出力関係

各自己安定アルゴリズムの入出力関係を述べる．イメージを図 7 に示す．

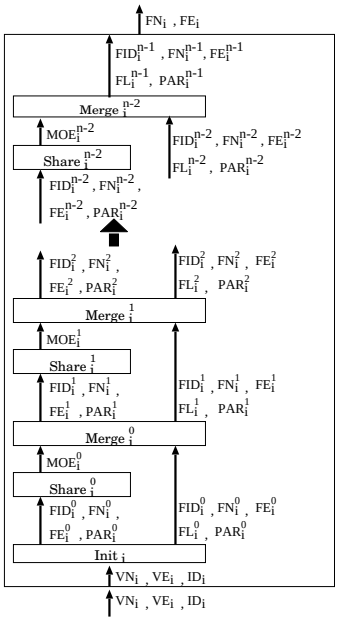


図 7: *SSSM* の各段の入出力関係

4.5 アルゴリズムで使用する述語と変数

- $isRoot_i^x$ $//v_i$ が x 段目のルートノードであることを表す**述語**

  $isRoot_i^x$: $FID_i^x = ID_i$

- $isLeaf_i^x$ $//v_i$ が x 段目の葉ノードであることを表す**述語**

  $isLeaf_i^x$: $\neg\ isRoot_i^x \wedge |FE_i^x| = 1$

- $outEdge_i^x$ $//v_i$ の x 段目の外向辺の集合を表す**変数**

  $outEdge_i^x = (v_i, v_j)(FID_j^x \neq FID_i^x \wedge (v_i, v_j) \in VE_i)$

- $Child_i^x$ $//v_i$ の x 段目の子ノードの集合を表す**変数**

  $Child_i^x = FN_i^x - \{PAR_i^x\}$

- $cMoe_i^x$ $//v_i$ が x 段目の MOE_i^x を接続辺として持っていて，その反対端ノード v_jが MOEが共有しているかを表す**変数**

$$cMoe_i^x = \begin{cases} (v_i, v_j) & if \quad (MOE_i^x \in VE_i \wedge \\ & \quad (MOE_i^x = (v_i, v_j) = MOE_j^x)) \\ \perp \quad if & other \end{cases}$$

- $minEdge_i^x$ $//x$ 段目の v_i に接続する重みが最小の外向辺を表す**変数**

  $minEdge_i^x = (v_i, v_j)$
  $((v_i, v_j) \in outEdge_i^x \wedge w_{ij} = min(c(outEdge_i^x))$

4.6 初期化アルゴリズム $Init_i$

$Init_i$ は G 上の各ノードを自ノードのみからなるフラグメントにする自己安定アルゴリズムである (図 8)．アルゴリズムの動作は次のとおり．

- 任意の状態で，ノード v_i は自ノードのみからなるフラグメントとなるように変数を初期化する．

入力変数:ID_i, VN_i, VE_i;
出力変数:$FID_i^0, FN_i^0, FE_i^0, FL_i^0, PAR_i^0$;

$//Init_i$
IR1：$ture \rightarrow FID_i^0 := ID_i$;
$FN_i^0 := null$;
$FE_i^0 := null$;
$FL_i^0 := 0$;
$PAR_i^0 := ID_i$;

図 8: $Init_i$

4.7 フラグメント MOE 共有アルゴリズム $Share_i^x$

$Share_i^x$ はフラグメントを構成するノード間で MOE を共有する自己安定アルゴリズムである (図 9)．アルゴリズムの動作は次のとおり．

- 葉ノードのとき，ノード v_i は接続する外向辺の中から重み最小の辺を $tMOE_i^x$ とする．
- 葉ノードでないとき，ノード v_i は接続する外向辺と $tMOE_{Child_i^x}^x$ の中から重み最小の辺を $tMOE_i^x$ とする．
- 根ノードのとき，ノード v_i は MOE_i^x を $tMOE_i^x$ とする．
- 根ノードでないとき，ノード v_i は MOE_i^x を $MOE_{PAR_i^x}^x$ とする．

以上の動作を行うことで，いずれフラグメントを構成するノード間で MOE を共有する．

入力変数:$FID_i^x, FN_i^x, FE_i^x, PAR_i^x$;
出力変数:MOE_i^x;

$//Share_i^x$
$//gathering$
SR1：$isLeaf_i^x \rightarrow tMOE_i^x := minEdge_i^x$
SR2：$\neg\ isLeaf_i^x \rightarrow tMOE_i^x :=$
$min(minEdge_i^x, tMOE_{Child_i^x}^x)$

$//broadcsting$
SR3：$isRoot_i^x \rightarrow MOE_i^x := tMOE_i^x$
SR4：$\neg\ isRoot_i^x \rightarrow MOE_i^x := MOE_{PAR_i^x}^x$

図 9: $Share_i^x$

4.8 フラグメント結合アルゴリズム $Merge_i^x$

$Merge_i^x$ で用いる述語を以下に示す.

- *MC1* //接続辺に *MOE* を持っていて他フラグメントと共有している，かつ両フラグメントのレベルが等しい，かつ自身の *ID* の方が小さいことを表す述語

  MC1：
  $cMoe_i^x \neq \perp \wedge FL_i^x = FL_j^x \wedge ID_i < ID_j$
  $(MOE_i^x = (v_i, v_j)$ である$)$

- *MC2* // 接続辺に *MOE* を持っていて他フラグメントと共有している，かつ両フラグメントのレベルが等しい，かつ自身の *ID* の方が大きいことを表す述語

  MC2：
  $cMoe_i^x \neq \perp \wedge FL_i^x = FL_j^x \wedge ID_i > ID_j$
  $(MOE_i^x = (v_i, v_j)$ である$)$

- *MC3* //接続辺に *MOE* を持っていて他フラグメントと共有している，かつ自身のフラグメントのレベルが結合相手のフラグメントのレベルより小さいことを表す述語

  MC3：
  $cMoe_i^x \neq \perp \wedge FL_i^x < FL_j^x$
  $(MOE_i^x = (v_i, v_j)$ である$)$

- *MC4* //接続辺に *MOE* を持っていて他フラグメント共有している，かつ自身のフラグメントのレベルが結合相手のフラグメントのレベルより大きいことを表す述語

  MC4：
  $cMoe_i^x \neq \perp \wedge FL_i^x > FL_j^x$
  $(MOE_i^x = (v_i, v_j)$ である$)$

- *MC5* //*MC1*，*MC2*，*MC3*，*MC4* でなく，かつ下段で子ノードだった隣接ノードが親を変更していることを表す述語

  MC5：
  $\neg\ MC1 \wedge \neg\ MC2 \wedge \neg\ MC3 \wedge \neg\ MC4 \wedge$
  $PAR_j^{x+1} \neq v_i (\exists v_j \in Child_i^x)$

- *MC6* //*MC1*，*MC2*，*MC3*，*MC4*，*MC5* でないことを表す述語

  MC6：
  $\neg\ MC1 \wedge \neg\ MC2 \wedge \neg\ MC3 \wedge \neg\ MC4 \wedge$
  $PAR_j^{x+1} = v_i (\forall v_j \in Child_i^x)$

$Merge_i^x$ は *MOE* を共有するフラグメント対をひとつのフラグメントに結合する自己安定アルゴリズムである (図 10). アルゴリズムの動作は次のとおり.

- *MC1* のとき，ノード v_i は *MOE* をフラグメント構成辺に追加し，フラグメント対をひとつの連結グラフにする. その後，自ノードを根とし，さらにレベルを 1 増やす.
- *MC2* または *MC3* のとき，ノード v_i は *MOE* をフラグメント構成辺に追加し，フラグメント対をひとつの連結グラフにする. その後，*MOE* によって隣接していたノードを親ノードとし，同時にフラグメントレベルとフラグメント *ID* を親の値と同じにする.
- *MC4* のとき，ノード v_i は *MOE* をフラグメント構成辺に追加し，フラグメント対をひとつの連結グラフにする. その後，入力の PAR_i^x が示すノードを親とし，同時にフラグメントレベルとフラグメント *ID* を親の値と同じにする.
- *MC5* のとき，ノード v_i は $Child_i^x$ が示すノードの中で親を変更しているノードを親とし，同時にフラグメントレベルとフラグメント *ID* を親の値と同じにする.
- *MC6* のとき，ノード v_i は入力の PAR_i^x が示すノードを親とし，同時にフラグメントレベルとフラグメント *ID* を親の値と同じにする.

以上の動作によって，いずれフラグメント対はひとつの連結グラフとなり，さらに根つきの木構造を持つ.

入力変数：
$FID_i^x, FN_i^x, FE_i^x, FL_i^x, PAR_i^x, MOE_i^x$;
出力変数：
$FID_i^{x+1}, FN_i^{x+1}, FE_i^{x+1}, FL_i^{x+1}, PAR_i^{x+1}$;

$//Merge_i^x$
MR1:$MC1 \rightarrow$
$FN_i^{x+1} := FN_i^x \cup \{v_j\}(MOE_i^x = (v_i, v_j))$;
$FE_i^{x+1} := FE_i^x \cup \{MOE_i^x\}$;
$PAR_i^{x+1} := v_i$;
$FL_i^{x+1} := FL_i^x + 1$;
$FID_i^{x+1} := ID_i$;

MR2:$MC2 \vee MC3 \rightarrow$
$FN_i^{x+1} := FN_i^x \cup \{v_j\}(MOE_i^x = (v_i, v_j))$;
$FE_i^{x+1} := FE_i^x \cup \{MOE_i^x\}$;
$PAR_i^{x+1} := v_j$;
$FL_i^{x+1} := FL_{PAR_i^{x+1}}^{x+1}$;
$FID_i^{x+1} := FID_{PAR_i^{x+1}}^{x+1}$;

MR3:$MC4 \rightarrow$
$FN_i^{x+1} := FN_i^x \cup \{v_j\}(MOE_i^x = (v_i, v_j))$;
$FE_i^{x+1} := FE_i^x \cup \{MOE_i^x\}$;
$PAR_i^{x+1} := PAR_i^x$;
$FL_i^{x+1} := FL_{PAR_i^{x+1}}^{x+1}$;
$FID_i^{x+1} := FID_{PAR_i^{x+1}}^{x+1}$;

MR4:$MC5 \rightarrow$
$FN_i^{x+1} := FN_i^x$;
$FE_i^{x+1} := FE_i^x$;
$PAR_i^{x+1} := v_j(v_j \in Child_i^x \wedge PAR_j^{x+1} \neq v_i)$;
$FL_i^{x+1} := FL_{PAR_i^{x+1}}^{x+1}$;
$FID_i^{x+1} := FID_{PAR_i^{x+1}}^{x+1}$;

MR5:$MC6 \rightarrow$
$FN_i^{x+1} := FN_i^x$;
$FE_i^{x+1} := FE_i^x$;
$PAR_i^{x+1} := PAR_i^x$;
$FL_i^{x+1} := FL_{PAR_i^{x+1}}^{x+1}$;
$FID_i^{x+1} := FID_{PAR_i^{x+1}}^{x+1}$;

図 10: $Merge_i^x$

5 正当性の証明と時間複雑度の解析

5.1 正当性の証明

まず，正当な状況 C^λ を定義し，提案アルゴリズム *SSSM* が C^λ に対して自己安定であることを示すことによって，*SSSM* が *MST* を構築する自己安定アルゴリズムであることを示す．

5.1.1 各段の正当な状況の定義

定義 3 各段の状況 C^x の正当な状況は以下のようになる．
C^0 : フラグメント数がn個
C^1 : フラグメント数が高々 *n-1* 個
C^2 : フラグメント数が高々 *n-2* 個
…
…
C^{n-1} : フラグメント数が 1 個

5.1.2 正当な状況 C^λ の定義

定義 4 *(*正当な状況は以下の条件を満たす*)*

1. C^λ 上の $\forall v_i, v_j$ において $FID_i^\lambda = FID_j^\lambda$ である．
2. C^λ 上のフラグメントは *MST* である．

5.1.3 正当性の証明

まず，補題 1〜3 により，$Init_i$，$Share_i^x$，$Merge_i^x$ がそれぞれの問題を解く自己安定アルゴリズムであることを示す．

補題 1 $Init_i$ は C^0 を構成する自己安定アルゴリズムである．

証明

全ノードが $Init_i$ を少なくとも *1* 回実行すれば C^0 が構成される．その後，値は変化しない． □

$C^0, ..., C^x$ が正しく構成されているとき，次の補題が成り立つ．

補題 2 $Share_i^x$ はいつかは MOE_i^x を v_i の属するフラグメントの最小外向辺とする自己安定アルゴリズムである．

証明

$Share_i^x$ の動作は *gathering* と *broadcasting* によって構成されている．*gathering* について *SR1* より，葉ノードでは *1* 度実行されると自ノードに接続する最小外向辺を *tMOE* として，その後は変化しない．*SR2* より，ノード v_i の $Child_i^x$ が自ノードに接続する外向辺と子ノードの *tMOE* の中で重み最小の辺を *tMOE* として，その後は変化しないとする．するとノード v_i は，いずれ自ノードに接続する外向辺と子ノードの *tMOE* の中で重み最小の辺を *tMOE* として，その後は変化しない．数学的帰納法より，根ノードはいずれ *tMOE* を全てのフラグメント構成ノードの外向辺の中で重み最小の辺として，その後は変化しない．

broadcasting について *SR3* より，根ノードでは一度実行されると *MOE* を *tMOE* として，その後は変化しない．*SR4* より，v_i は *MOE* を PAR_i^x の *MOE* として，その後は変化しないとする．すると，$Child_i^x$ は，いずれ *MOE* を v_i の *MOE* として，その後は変化しない．数学的帰納法より，全てのフ

ラグメント構成ノードは MOE を根ノードの MOE として，その後は変化しない．

$gathering$ と $broadcasting$ の動作より， $Share_i^x$ は MOE_i^x を v_i の属するフラグメントの最小外向辺とする自己安定アルゴリズムである． □

G 上の辺の重みは全順序なので C^x 上に複数のフラグメントが存在するとき，MOE を共有するフラグメント対が少なくとも 1 組は存在する．

$C^0, ..., C^x$ が正しく構成されていて $Share_i^x$ の出力が正しいとき，次の補題が成り立つ．

補題 3 $Merge_i^x$ は MOE を共有するフラグメント F_i と F_j をひとつのフラグメント F_{ij} に結合する自己安定アルゴリズムである．

証明

フラグメント F_i, F_j のレベルをそれぞれ FL_i, FL_j と呼ぶ．

MOE の両端ノード v_i, v_j は $MC1$〜$MC4$ のいずれかに該当し，MOE をフラグメント構成辺に加える．フラグメント F_i, F_j は木を構成しているので，それらに MOE を構成辺として追加したフラグメント F_{ij} も明らかに木を構成している．

$FL_i = FL_j$, $ID_i < ID_j$ のとき．

F_i では，v_i が $MC1$ に該当し，$MR1$ を実行することで MOE をフラグメント構成辺に追加して，レベルを 1 増加して，自ノードを根とする．すると，PAR_i^x が示すノードが $MC5$ に該当し，$MR4$ を実行することで v_i を親とし，同時にレベルとフラグメント ID を v_i の値と同じにする．その後，祖先ノードが順に $MR4$ を実行し，いずれ v_i から F_i の根までの経路で v_i を根とする親子関係が成立する．

F_j では，v_j が $MC2$ に該当し，$MR2$ を実行することで MOE をフラグメント構成辺に追加して，親ノードを v_i とし，同時にレベルとフラグメント ID を v_i の値と同じにする．すると，PAR_j^x が示すノードが $MC5$ に該当し，$MR4$ を実行することで v_j を親とし，同時にレベルとフラグメント ID を v_j の値と同じにする．その後，祖先ノードが順に $MR4$ を実行し，いずれ v_i から F_j の根までの経路で v_i を根とする親子関係が成立する．

$FL_i < FL_j$ のとき．

F_i では，v_i が $MC3$ に該当し，$MR2$ を実行することで MOE をフラグメント構成辺に追加して，v_j を親とし，同時にレベルとフラグメント ID を v_j の値と同じにする．すると，PAR_i^x が示すノードが $MC5$ に該当し，$MR4$ を実行することで v_i を親とし，同時にレベルとフラグメント ID を v_i の値と同じにする．その後，祖先ノードが順に $MR4$ を実行する．いずれ v_j から F_i の根までの経路で v_j を祖先とする親子関係が成立する，

F_j では，v_j が $MC4$ に該当し，$MR3$ を実行することで MOE をフラグメント構成辺に追加して，親ノードを PAR_j^x が示すノードとし，同時にレベルとフラグメント ID を親の値と同じにする．この実行によって PAR_j^x が示すノードが $MC5$ に該当することはない．

$MC1$〜$MC5$ に該当しないノードは，$MR5$ を実行することで入力の親ノードをそのまま親とし，同時にレベルとフラグメント ID を親の値と同じにする．

以上より，$Merge_i^x$ は F_i と F_j のレベルとフラグメント ID の値を同じにして，かつ MOE によって連結となる根つき木構造を造る．つまり，F_i と F_j をひとつのフラグメント F_{ij} に結合する自己安定アルゴリズムである． □

以下の補題にて，アルゴリズム $SSSM$ は正当な状況 C^λ の 1 に対して，到達可能性と閉包性を満たすことを示す．

補題 4 アルゴリズム $SSSM$ は正当な状況 C^λ の 1 に対して，到達可能性を満たす．

証明

補題 1 より，$Init$ によって C^0 上に n 個のフラグメントができる．補題 2，補題 3 より C^0〜C^x が正しくてかつフラグメントが複数存在する時，$Share^x$, $Merge^x$ によって少なくとも 1 組のフラグメント対が結合するので，C^{x+1} 上のフラグメント数は C^x より少なくとも 1 つ減る．よって，$Init$ の後 $Share$，$Merge$ を高々 $n-1$ 回繰り返せばフラグメント数が 1 つになる． □

補題 5 アルゴリズム $SSSM$ は正当な状況 C^λ の 1 に対して，閉包性を満たす．

証明

正当な状況 C^λ の 1 ではフラグメントが 1 つなので外向辺は存在しない．よって，$Share^\lambda$ で MOE が $\perp$ となる．なので，$Merge^\lambda$ においてフラグメントの結合は起こらず状況は C^λ のまま変化しない． □

さらに，以下の補題にてアルゴリズム $SSSM$ に従って正当な状況 C^λ の 1 に到達した時，C^λ の 2 も満たすことを示す．

補題 6 C^x 上の各フラグメントの最小外向辺は G の MST を構成する辺に属する．

証明

背理法で示す．$F = (Fv, Fe)$ を G の MST とし，C^x のフラグメント F の最小外向辺 MOE について $MOE \notin Fe$ と仮定する．すると，$OE \in Fe$ となるような F の外向辺 OE が存在する．しかし，$MOE < OE$ であり，かつ $F' = (Fv, Fe - \{OE\} \cup MOE)$ も木となり，$w(F') < w(F)$ となるので矛盾． □

補題 4，5，6 により次の定理が成り立つ．

定理 1 アルゴリズム $SSSM$ は正当な状況 C^λ に対して自己安定である．

よって，提案アルゴリズム *SSSM* は *MST* を構築する自己安定アルゴリズムである．

5.2 時間複雑度の解析

2.8 で述べた非同期ラウンドを用いて *SSSM* が正当な状況に到達するまでの時間複雑度を評価する．各アルゴリズムの Condition が真となるものが複数存在するときは，以下の優先順位で Action を実行するスケジュールを仮定し，その下で評価する．

1. $Init_i$.
2. 各段の $Share_i^x$ と $Merge_i^x$ に対して，最小となる x.
3. 同一段の $Share_i^x$ と $Merge_i^x$ とでは，$Share_i^x$.

つまり，多段構成されたアルゴリズムの下段から順に評価・実行されるものとする．

定理 2 *SSSM* は任意の初期状況から $O(n^2)$ ラウンドで正当な状況に到達する．

証明

1 ラウンドで各ノードは少なくとも *1* 原子動作をする．よって，$Init_i$ は *1* ラウンドで正当な状況になる．

Share と *Merge* が実行されると，フラグメントが少なくとも *1* つ減少する．従って，*Share* と *Merge* の組が高々 $n-1$ 段必要となる．一方，k 段目における各フラグメントの木構造の高さは，最悪 k となる．このとき，*Share* の *gathering* では，葉ノードから根ノードまで順に *tMOE* が決定されていくので k ラウンド必要となり，*broadcasting* では，根ノードから葉ノードまで順に *MOE* が決定されていくので k ラウンド必要となる．つまり，*Share* は $2k$ ラウンドで正当な状況になる．また，*Merge* では，結合後フラグメントの根ノードのレベルとフラグメント *ID* を根ノードから葉ノードまで順に伝わっていくので k ラウンドで正当な状況になる．

Init の上に *Share* と *Merge* の組を $n-1$ 段重ね合わせるアルゴリズム全体では $1+3+5+9+...+3(n-2)+3(n-1) = O(n^2)$ ラウンド必要となる．

よって，*SSSM* が正当な状況に到達するまでの時間複雑度は $O(n^2)$ ラウンドである． □

6 おわりに

本稿では *MST* を構築するアルゴリズムを *Share*，*Merge* という 2 つの自己安定アルゴリズムの多段構成を用いて設計し，その正当性と安定するまでの時間複雑度を評価した．

本稿の 1. で「グリーディ戦略で解ける問題は自己安定アルゴリズムの多段構成を用いて解くことができる」という仮説を示した．グリーディ戦略で解くことのできる問題のうちの *MST* 構築問題については自己安定アルゴリズムの多段構成を用いて解けることが確認でき，仮説に対する信頼性がより高まった．

グリーディ戦略で解くことのできる問題はマトロイドという組み合わせ論的な構造をもつことが分かっている．そこで，多段構成とマトロイドの等価性を証明することが今後の課題である．

参考文献

[1] S. Dolev, A. Israeli and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. Distributed Computing, 7:pp.3-16. (1993)

[2] 西村弘志.“ 極大クリーク分割に基づく自己安定クラスタリングアルゴリズム ”，名古屋工業大学　和田・犬塚研究室　修士論文，2007

[3] Gallager，R.G.，Humblet，P.A.，and Spira，P.M.,“ A distributed algorithm for minimum-weight spanning tree，”ACM Transactions on Programming Lan-guages and Systems 5,1，pp. 66-77，1983

[4] 片山喜章，増澤利光.“ 重み最小生成木を構成する故障封じ込めを考慮した強安定プロトコルについて ”，電子情報通信学会技術研究報告，Vol.97，No.375 pp. 25-32，1997

直感的な操作が可能な分散アルゴリズム学習用シミュレータの提案

長瀧 寛之[†]　　山内 由紀子[‡]　　角川 裕次[§]

1　背景

情報科学の教育において，分散アルゴリズムは近年重要な学習項目として位置づけられている [1]．分散アルゴリズムは，ネットワークでつながった複数のプロセスが非同期かつ並列に実行し，また各プロセスの実行のタイミングは非決定的要素の影響を受けるなど，その動作は非常に複雑である．そのため理論的な説明だけでは，学習者にアルゴリズムの肝要を理解させることが難しい．

そこで，シミュレータを用いた可視化によって学習者に分散アルゴリズムの具体的な動作を示そうとする試みが行われている [2]．ただし提示される可視化情報をただ眺めるだけでは学習効果は低いため，学習者が自らシミュレータを操作して，理解したいアルゴリズムについて様々な動作例を対話的に観察できることが重要である [3, 4]．しかし従来の学習用分散アルゴリズムシミュレータは，シミュレーション実行に伴う操作が煩雑でわかりにくく，結果として様々な実行事例を学習者自身で観察できるという学習環境を実現できていない [5]．

ところでインタフェースの研究分野では，タンジブル (tangible) インタフェースが注目を集めている [6]．tangible インタフェースは，コンピュータ上の仮想オブジェクトをそれに対応する実物体の物理的な動きによって直感的に操作するというものである．著者らはこの特性が，仮想的なモデルを扱う分散アルゴリズムのシミュレーション制御においても有効ではないかと考えた．

以上より著者らは，tangible インタフェースによる直感的な操作を指向した分散アルゴリズム学習用シミュレータの構築を目指すという着想に至った．本稿では，既存研究と比較検討することで本研究の方向性を示し，現在検討しているシミュレータの設計アイデアについて述べる．

2　関連研究

分散アルゴリズム学習支援の既存研究のほとんどは，シミュレータを用いたアルゴリズムの可視化を行う事例である [2, 5, 7]．いずれの研究事例も，分散アルゴリズムの基本概念を学習者に理解させることが難しいという問題認識で共通しており，効果的な可視化やそのためのシステム設計の工夫に焦点を当てることで，分散アルゴリズムの動作を分かりやすく学習者に示すことを目標としている．

一方ユーザによるシミュレータ操作という点では，主にマウスクリックによるコマンド選択 [5, 7] や独自の記述言語でのプログラミング [2, 5] によってシミュレーション実行の制御を行い，またユーザが能動的に必要な情報を選択表示してシミュレーション結果を観察する [2, 5] というアプローチが多い．これらの設計はいずれも，性能評価を目的とした従来の分散システムシミュレータがベースであると考えられる．しかしアルゴリズムを十分に理解していることが前提のこれらの操作体系は，まだ十分にアルゴリズムを理解していない学習者が様々な実行例を観察する環境として適しているとは言いがたい．

本研究では，問題認識は既存研究と共通しているが，その解決方法として，操作体系の改善に焦点を当てている点が既存研究と異なる．具体的には，学習者がより直感的にシミュレーション制御が行えるインタフェースを適用することで，試行錯誤的なシミュレーションによる学習スタイルを実現しようと試みるものである．

3　シミュレータ概要

本シミュレータのシステム概念図を図 1 に示す．本シミュレータのユーザは主に分散アルゴリズムの学習者であり，演習課題や自主学習などの目的で利用することを想定している．本システムは分散アルゴリズムの授業等における利用を想定し，アルゴリズム定義情報やデバイス操作の対応情報については，事前に教師が登録するものとする．

ユーザは tangible デバイスを操作することで，アルゴリズム実行中の分散システムの動作を制御し，シミュレーションに変化を起こしながら，その状況や結果を PC ディスプレイ上で確認する．ここでインタフェースとして用いる tangible デバイスは，**分散システムモデル上のプロセス 1 つに対して 1 つずつ用意する**．具体的には，あるデバイスに対する物理的な操作を，対応するプロセスの状態変化と対応付ける．これにより，プロセスに直接触る感覚でシミュレーションを制御する操作方法を実現する．また複数のデバイスを同時に操作できることから，動作の非同期性や並列実行という分散アルゴリズムの性質を体感しやすいという効果も期待できる．

実際の入力デバイスとして，振る，傾ける，裏返す，光を当てるなど様々な物理的操作を入力できるよう，**加速度**

[†]岡山大学教育開発センター
[‡]奈良先端科学技術大学院大学情報科学研究科
[§]大阪大学大学院情報科学研究科

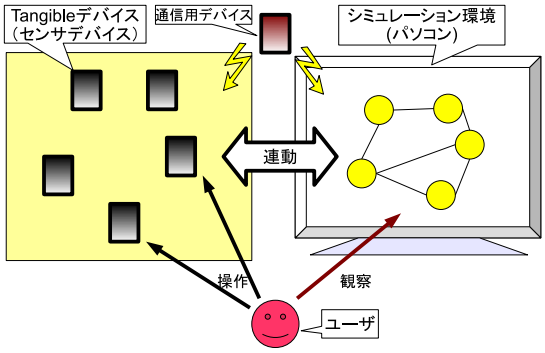


図 1: システム概念図

や照度などを検出可能なセンサデバイスを採用する．これにより，物理的にプロセスに影響を与える感覚での操作を実現すると同時に，マウスやキーボードでは限界がある様々な入力パターンを 1 デバイスで実現できる．

ユーザの入力デバイス操作は，シミュレーション実行前のネットワーク構築，またシミュレーション実行中のプロセス制御に影響を与える．例えばネットワーク構築時には「2 つのデバイスを同時に振るとリンク生成 or 切断」「振り方の速さでリンクの重み付けを変化」など，シミュレーション実行中は「デバイスに強い光を当てると，対応プロセスが initiator として動作」「デバイスを裏返すとプロセス故障発生」など，デバイス物理的な操作がプロセス内外の状況変化のトリガとして対応付けられる．アルゴリズムごとに制御したいトリガや対応付けたいデバイス操作は異なると考えられるため，シミュレーション実行においては，プロセスが実行するアルゴリズムの定義情報とともに，デバイス操作とトリガ制御操作の対応付け情報を別途外部から定義することとする．

また分散アルゴリズムはネットワーク上の様々な箇所で状況が変化していくことから，それらを同時にディスプレイ上に表示しては，ユーザが情報過多による混乱を起こす可能性がある．既存システムはユーザが能動的に提示情報を選択する必要があるが，本システムでは**デバイス操作に連動した提示情報の自動制御**を行う方法を考える．一例として，あるデバイスを操作している間，そのデバイスに対応するプロセスは内部データを細かく表示し，近傍プロセスの情報は現在の状態のみ簡略表示，それ以外のプロセスの情報は非表示とする方法を検討している．これにより，情報表示のためのユーザ操作を極力なくし，情報過多による理解の阻害を防ぐ．さらに提示情報の自動制御によって，注目すべき情報へ学習者の意識を誘導し，アルゴリズムの基本概念への気づきを促す効果も狙う．

4　進捗状況と今後の課題

本研究のアイデアを検証するため，現在本シミュレータのプロトタイプの構築を急いでいる．センサデバイスには SunSPOT[8] を用いることにし，PC 側のプログラムは Java によるスタンドアロンアプリケーションとして開発する．シミュレータのコア部分の開発は既に進んでおり，現在 SunSPOT とシミュレータ部の連携モジュールと，GUI での表示情報制御の実装が残っている．

プロトタイプの完成次第，予備的な評価実験を行い，本研究のアイデアの検証と必要機能の再検討を行う．特に，アルゴリズムごとに，どのようなデバイス操作とトリガの関連付けが考えられるかについて，またデバイスの操作と連動した情報提示の手法について，実験を通して最適な手法を検討していく予定である．

参考文献

[1] 情報処理学会情報処理教育委員会. 情報専門学科におけるカリキュラム標準 J07. (online), available from http://www.ipsj.or.jp/12kyoiku/J07/J0720090407.html.

[2] Steve Carr et al. ConcurrentMentor: A visualization system for distributed programming education. In *Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 1676–1682. CSREA Press, 2003.

[3] Markus Krebs et al. Student-built algorithm visualizations for assessment: exible generation, feedback and grading. In *ITiCSE '05*, pp. 281–285. ACM, 2005.

[4] 都倉信樹. プログラム設計環境のツール, 2. 分散アルゴリズム設計支援ツール. 情報処理, Vol. 30, No. 4, pp. 380–386, Apr 1989.

[5] Boris Koldehofe et al. LYDIAN: An Extensible Educational Animation Environment for Distributed Algorithm. *ACM Journal on Educational Resource in Computing*, Vol. 6, No. 2, pp. 1–21, Jun 2006.

[6] Hiroshi Ishii. Tangible bits: Beyond pixels. In *Proceedings of the Second International Conference on Tangible and Embedded Interaction (TEI'08)*, pp. xv–xxv. ACM, Feb 2008.

[7] Mordechai Ben-Ari. Interactive execution of distributed algorithms. *ACM Journal of Educational Resources in Computing*, Vol. 1, No. 2, pp. 2–8, Jun 2001.

[8] Sun Microsystems, Inc. SunSPOTWorld. (online), available from http://www.sunspotworld.com/. (accessed 2009-08-24).

アルゴリズム学習向け誤り発見型演習のための カスタマイズ可能な問題自動生成システム

藤原啓 †，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

‡ 岡山大学教育開発センター

概要　本研究では，ソースコードに含まれるアルゴリズム上の誤りを発見し修正する “誤り発見型演習” の問題を自動生成することを目標に，任意のアルゴリズムに対して問題の自動生成を行う手法を提案する．本手法は，ソースコードへの誤り挿入位置決定に関する処理を外部定義可能にすることで，ソースコードで実装されているアルゴリズムに応じて，誤り挿入位置をカスタマイズ出来るようにすることを可能とすることを狙う．

本稿では，誤り挿入位置候補の構文を外部定義するための仕様について紹介し，本仕様によってアルゴリズムに応じた誤り挿入位置の決定が可能となったことを確認する．

1　はじめに

アルゴリズムの学習は，情報科学の学習において重要とされている分野の一つである [1]．アルゴリズム学習における学習者の理解度向上を狙った 1 つの演習形式として，アルゴリズムを実際にプログラミングさせる形式の演習も行われている．正しいプログラムを作成するためにはアルゴリズムを十分に理解している必要があるので，プログラミングはアルゴリズムの理解度向上には有効であると思われる．しかしプログラミングは，構文エラーなどの言語仕様に関する事柄に時間と意識を取られやすく，本来の目的であるアルゴリズムの理解に学習者を集中させにくいという問題がある．

この問題に対して，**間違い探し演習** [2] が提案されている．間違い探し演習の手順は以下の通りである．(1) 教師は実装したアルゴリズムの名称とソースコードを学習者に提示する．ただし，提示するソースコードには，何らかのアルゴリズ上の誤りが含まれている．(2) 学習者はソースコードに含まれる誤りを発見し，該当部分を正しく動作するように修正する．(3) 学習者は，正しいアルゴリズムとどのように矛盾しているかを考察する．

間違い探し演習では，学習者がソースコード全体を自分で実装する必要はない．よって，構文エラーなどのアルゴリズムの理解と関係のない事柄に注意を払う必要が減るので，教師はアルゴリズムの理解に焦点を絞った演習を行うことができる．また，学習者の解答は，教師が提示したソースコードの一部を書き換えただけのものであるため，ソースコードの書き換えられた箇所を見るだけで採点が可能となる．したがって，教師にとっては採点の時間的負担を軽減することも可能となる．

著者らは以前より，間違い探し演習問題を自動生成する手法を提案している [2]．しかし既存の手法では対応可能なアルゴリズムが限定されるため，より多くのアルゴリズム学習に対応できるようにする必要があった．そこで著者らは，誤り挿入のプロセスをシステム外部で容易に定義できるようにすることで，ソースコードで実装されているアルゴリズムに応じて，誤り挿入位置をカスタマイズ出来るようにすれば良いのではないかと考えた．

本稿の構成は以下の通りである．まず，2 章で既存手法の概要を説明し，3 章で提案手法を具体的に説明する．次に，4 章で提案システムにより実際に任意のアルゴリズムを実装したソースコードに対して適切に誤り挿入位置を決定できるかの考察を行い，最後に 5 章で本研究についてまとめる．

2　既存手法

本研究での提案手法に先立って，著者らが文献 [2] で提案した間違い探し演習の自動生成手順について説明する．. 自動生成の手順は以下の通りである．(1) アルゴリズムの設計手法であるアルゴリズム設計パラダイムに基づいてアルゴリズム学習上有用な誤り挿入位置の候補を決定する．(2) 誤り挿入位置の候補から実際に誤りを挿入する箇所を選択する．(3) 誤り挿入位置の構文構造に対応する誤りパターンを適用する．既存研究における間違い探し演習問題自動生成システムを図 1 に示す．システムでは、まず C 風言語で記述された入力ソースコードに対して字句構文解析を行い，C 風言語のソースコードを構文木を表すリストのデータ構造に変換した中間コードを出力する．誤り挿入位置決定部では，中間コード中の誤りを挿入する位置を抽出する．最後に，誤りパターン適用部で，誤り挿入位置決定部で抽出した誤り挿入位置に対して，適切な誤りパターンを適用する．教師は基となるソースコードとパラダイムを指定することで，誤りを含むソースコードを自動的に作成することが出来る．

既存研究で構築された演習問題自動生成システムは，アルゴリズム学習で扱われることの多い分割統治法，再帰，動的計画法，貪欲法の 4 つのパラダイムを対象とし

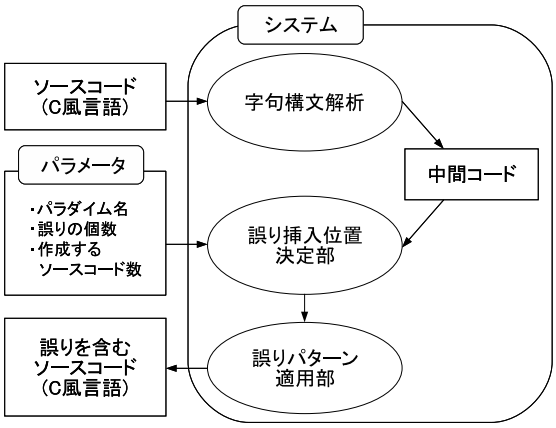


図 1　既存研究における間違い探し演習問題自動生成システム

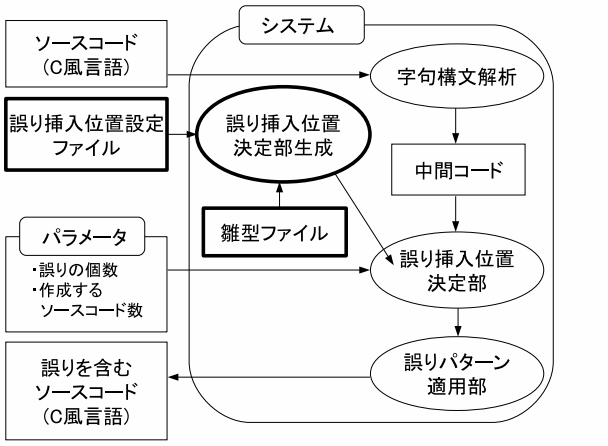


図 2　本研究における間違い探し演習問題自動生成システム

ている．しかし，全てのアルゴリズムが，4 種類のパラダイムに分類されるわけではなく，既存手法では対応パラダイムを拡張する度に，システムのプログラムコードを改修する必要がある．そこで著者らは，誤り挿入位置を特定するパターンをシステム外部でデータファイルにより定義できるようにすることで，自動生成可能なアルゴリズムの範囲を容易に拡張できるのではないかと考えた．次章で本研究の提案手法である，誤り挿入位置をパターンにて記述する手法について議論する．

3　提案手法

本章では，提案手法の概要を述べる．提案システムに対する要求事項は以下の通りである．

- 新しいパラダイムに対応する誤り挿入位置の定義を追加できるようにすること．
- 既存のパラダイムに対応する誤り挿入位置の定義を修正・拡張できるようにすること．
- 誤り挿入位置の定義の追加・修正は，外部のデータファイルで追加できるようにすること．つまり，システム本体を直接再構築する必要が無いような仕組みにすること．

以上の要求仕様を満たすため，本研究では，図 1 中の誤り挿入位置決定部に着目した．誤り挿入位置の定義を追加・修正する際には，誤り挿入位置決定部で利用するプログラムを，外部データファイルの情報に応じて変更出来る仕組みが必要がある．この仕組みを実装した本研究における間違い探し演習問題自動生成システムを図 2 に示す．提案システムでは，誤り挿入位置の定義を記述した外部データファイルである**誤り挿入位置設定ファイル**の情報を利用して，誤り挿入位置決定部で利用するプログラムを本システムが内部で動的に生成することで，利用者がシステム本体を再構築することなく，誤り挿入位置の定義の追加，修正を行うことが可能である．次節以降，本研究において誤り挿入位置を決定する方法，及び誤り挿入位置設定ファイルの概要について述べる．

3.1　誤り挿入位置の特定

誤り挿入位置決定部では，ソースコード中のどの位置に誤りを挿入するかを決定する作業を行う．誤り挿入位置決定部で利用するプログラム (以下誤り挿入位置決定プログラム) は，誤り挿入位置設定ファイルの内容を元に，雛形ファイルを用いて自動で生成する．雛形ファイルとは，誤り挿入位置決定プログラムの土台となる部分を記述したファイルである．具体的には，中間コードを走査する際に利用する構文，抽出した誤り挿入位置から実際に誤りを挿入する位置を選択する際に利用する構文の 2 種類である．この雛形ファイルに対して，誤り挿入位置設定ファイルに記述されている内容に従い，必要となるプログラム断片を追加することによって，誤り挿入位置決定プログラムを生成する．

3.2　誤り挿入位置設定ファイル

そこで，誤り挿入位置として抽出する構文の構造を指定する際に必要な要素について検討し，それぞれの要素に対応する誤り挿入位置設定ファイルの記述法を検討した．以降では，構文の構造を指定する際に必要な 3 つの要素について説明する．

- **構文の種類**：構文の種類とは，代入文，if 文などのプログラム中で使用される文の種類を表す．誤り挿入位置には，ソースコード中の文，式，さらに制御構文など複数の式の集合が該当する．よって，誤り挿入位置を指定する際には，誤り挿入位置で使用されている構文の種類を指定することが必要である．

- **構文の構成要素**：構文の構成要素とは，構文の種類で指定した文のうち，誤り挿入位置として抽出すべき箇所を表す．例えば，構文の種類として関数呼出文を指定した場合，構文の構成要素としては，関数呼出文そのものと関数呼出文の引数の 2 種類考えられる．
- **構文の条件**：構文の条件とは，指定した種類の構文のうち，どの文を適切な誤り位置とするかを指定するための条件である．構文の種類を指定しただけでは誤り挿入位置を決定することが出来ない場合がある．例えば，構文の種類として代入文を指定した場合，全ての代入文が誤り挿入位置となるため，誤り挿入位置として不適切な代入文に誤りが挿入されてしまう可能性がある．そのため，構文の条件も同時に指定する必要がある．

以上で述べた，構文の種類，構文の構成要素，及び構文の条件の 3 つの要素を用いて誤り挿入位置を指定する．例として，再帰呼出文を誤り挿入位置として指定する場合，構文の種類は「関数呼出文」，構文の構成要素は「関数呼出文」及び「関数呼出文の引数」，構文の条件は「関数呼出文で呼び出す関数名と，関数呼出文が所属している関数名が一致していること」となる．

誤り挿入位置の構文構造を誤り挿入位置設定ファイルで記述するために，誤り挿入位置設定ファイル記述用の構文を定義する．必要となる記述法を決定するために，既存研究で対応済みのパラダイムで誤り挿入位置を特定する際に利用されていた，構文の種類，構文の構成要素，構文の条件を解析した．既存研究で対応済みのパラダイムは，アルゴリズムの講義において扱われることの多いものである [3] ため，解析結果をもとに記述法を決定することで，誤り挿入位置を記述する際に必要な構文は網羅できると考えた．決定した記述法の要素の詳細は付録 A に記載した．

定義した記述法を用いて，再帰呼出文用の誤り挿入位置設定ファイルを記述した例を図**??**に示す．この例では，中間コードを走査し (search)，関数呼出文を見つけた場合 (found-call) に，found-call 以降の部分を適用する．if 文に対しては，呼び出す関数名 (call-fname) と，関数呼出文が所属している関数名 (fname) が一致 (==) しているかどうかを判定し，結果が真であれば関数呼出文と，関数呼出文の引数を誤り挿入位置として抽出する (enbug-call 及び enbug-call-arg)．

```
(define config
 '(search
   (found-call
    ((if (== (call-fname) (fname))
         ((enbug-call) (enbug-call-arg)))))))
```

図 3　再帰呼出文の誤り挿入位置設定ファイル記述

```
#define N 20

int data[N] = {RANDOM 10};
int h,i,j,t;

for (h=N/2; h>0; h=h/2){
  for (i=h; i<N; i++){
    t = data[i];
    for (j=i-h; j>=0 && data[j]>t; j=j-h){
      data[j+h] = data[j];
    }
    data[j+h] = t;
  }
}
```

図 4　シェルソートのソースコード

1. 既存研究で適切な誤り挿入が出来るパラダイムについて，誤り挿入位置設定ファイルを記述することが可能かどうかを検討
2. 既存研究では適切な誤り挿入位置を特定することが出来なかったソースコードに対して，ユーザが指定したい位置を誤り挿入位置として抽出可能かどうかを評価

評価項目 1 について，誤り挿入位置設定ファイル用に定義した構文で，既存研究で適切に誤りを挿入することが可能なパラダイムについて，等価な誤り挿入を行う誤り挿入位置設定ファイルを記述した結果，すべて記述できることが確認できた．これによって，本手法により既存手法を置換することが可能であることが評価できた．

評価項目 2 については，もし抽出が可能であれば，提案手法は既存手法よりも誤り挿入位置の特定という点において優れていると考えられる．

既存手法で誤り挿入位置を特定できなかったものの 1 つとして，シェルソートのソースコードを利用して評価を行った．

シェルソートとは，ソートアルゴリズムの一つで、挿入ソートを応用したアルゴリズムである。本研究での評価において，シェルソートを実装したソースコードの誤り挿入位置を，「ソーティングを行う間隔を定義するための変数を更新する文」とした．評価に使用したシェルソートのソースコードを図 4 に，誤り挿入位置設定ファイルの記述例を図 5 に示す

誤り挿入位置を抽出するための手順は，以下の通りである．

手順 1　間隔を定義するための変数を抽出する
手順 2　手順 1 で抽出した変数を更新する文を誤り挿入

4　提案システムの評価

4.1　概要

本研究の提案手法について評価を行う．評価の項目は以下の通りである．

```
(define config
  '((search
     (found-for
      ((if (not (in-loop?))
           ((enbug-for-cond3))))))))
```

図 5　誤り挿入位置設定ファイル記述例 (for 文を利用したシェルソート)

位置として指定する

シェルソートにおける間隔を定義するための変数は，ソースコード中のループ文のうち，一番外側にあるループ文の条件式に現れる．これは，内側にあるループ文は，挿入ソートを行うためのループ文となっているためである．そこで，手順 1 において間隔を定義するための変数を抽出する際には，一番外側のループ文の条件式で使用されている変数名を抽出することにした．また手順 2 では，代入文左辺の変数名，又はインクリメント文で使用される変数名が，手順 1 で抽出した変数名と一致した場合に，変数更新文を誤り挿入位置として抽出する．ただし，for 文を使用したソースコードの場合，一番外側の for 文における第 3 引数が，間隔を定義するための変数を更新する文となるので，誤り挿入位置として抽出する．図 5 においては，「中間コードを走査し (search)，for 文を見つけた場合に (found-for)，for 文がループ文の内部に含まれていなければ (not (in-loop?))，for 文の第 3 引数を誤り挿入位置とする (enbug-for-cond3)」という記述となる．

本研究の提案手法によって誤り挿入位置を特定することが出来るかどうかを検討した結果，誤り挿入位置を抽出することができた．また，同じ構造で記述されたソースコード，即ち，変数名が異なっていたり，改行位置が異なっていても，誤り挿入位置を抽出することが可能である．

5　終わりに

本研究では任意のアルゴリズムを実装したソースコードに対して間違い探し演習問題を自動生成する手法を提案した．特に自動生成において，誤り挿入位置の決定法をカスタマイズする機能を実現した．その実現方法として，誤り挿入位置の構文構造を定義した誤り挿入位置設定ファイルをパラダイムごとに用意することで，誤り挿入位置設定ファイルの情報から，誤り挿入位置決定部で動作するプログラムのソースコードを自動的に生成する手法を紹介した．また，誤り挿入位置設定ファイルを記述するために利用する記述用構文の文法を定義した．

評価結果から，今回提案した手法により，対応パラダイムの拡大とともに，生成できる演習問題の種類を拡張できることを確認した．

今後の課題としては，誤り挿入位置設定ファイルにおいて，記述仕様のさらなる改良とともに，教師がより容易にファイルを構築できるような GUI の検討があげられる．また，対応言語を任意の言語に拡張可能な仕組みの構築も今後検討すべき事項である．

参考文献

[1] SIGCSE, Computing Curriculam 2001, http://www.sigcse.org/cc2001/

[2] 長瀧寛之，伊藤亮太，大下福仁，角川裕次，増澤利光, “ アルゴリズム学習における間違い探し形式の演習問題を自動生成する手法の提案と評価 ”, 情報処理学会論文誌，Vol.49，No.10，pp3366-3376，2008.

[3] 浅野哲夫，和田幸一，増澤利光, “ アルゴリズム論 ”, オーム社，Jul. 2003.

付録 A　パラダイム設定ファイル記述用構文の要素

A.1　構文の種類

1. 関数呼出文
2. 変数更新文
3. 代入文
4. if 文
5. return 文
6. while 文
7. do-while 文
8. for 文

A.2　構文の構成要素

1. 関数呼出文 $f(x)$
   (a) 全体 ‘ $f(x)$ ’
   (b) 引数 ‘ x ’
2. 変数更新文
3. 代入文
4. if 文 if(*condition*)
   (a) 全体 ‘ if(*condition*) ’
   (b) 条件式 ‘ *condition* ’
5. return 文の引数
6. while 文の引数
7. do-while 文の引数
8. for 文 for(*exp1*;*exp2*;*exp3*)
   (a) 第 1 引数 ‘ *exp1* ’
   (b) 第 2 引数 ‘ *exp2* ’
   (c) 第 3 引数 ‘ *exp3* ’

A.3 構文の条件

1. 特定の関数の名前を取得する構文
   (a) 関数呼出文で呼び出す関数
   (b) 構文が所属している関数
2. 特定の変数の名前を取得する構文
   (a) 変数更新文で更新される変数
   (b) 代入文の左辺で使用されている変数
   (c) 関数呼出文の引数で使用されている変数
   (d) 代入文の右辺で使用されている変数
   (e) for 文の繰り返しのインデックスとなる変数
3. 特定の変数の名前と，その変数を含む文や式が所属している関数の名前を取得する構文
   (a) 変数更新文で更新される変数
   (b) 代入文の左辺で使用されている変数
   (c) 関数呼出文の引数で使用されている変数
   (d) 代入文の右辺で使用されている変数
   (e) for 文の繰り返しのインデックスとなる変数
4. 特定の変数が純変数と配列変数のどちらであるかを取得する構文
   (a) 変数更新文で更新される変数
   (b) 代入文の左辺で使用されている変数
5. 特定の構文の実行過程に含まれているかを判定する構文
   (a) if 文
   (b) while 文
   (c) do-while 文
   (d) for 文
   (e) ループ文の実行過程に含まれる if 文
6. 特定の箇所を探索するための構文
   (a) if 文の実行過程
   (b) while 文の実行過程
   (c) do-while 文の実行過程
   (d) for 文の実行過程

An Efficient Web Page Recommendation Based on Preference Footprint to Browsed Pages

Shuhei Hayashi, Yuuki Inoshita and Satoshi Fujita
Graduate School of Engineering, Hiroshima University
1-4-1 Kagamiyama, Higashi-hiroshima, Hiroshima, Japan
email:{sh, ino, fujita}@se.hiroshima-u.ac.jp

概要

This paper proposes a new scheme for web page recommendation which re ects the preference of each user to the recommended pages in an e cient and e ective manner. The basic idea of the scheme is to combine the notion of preference footprint to browsed pages with the collaborative filtering. More concretely, we introduce the notion of "tags" similar to conventional SBS (Social Bookmark Service), and attach all tags associated with a user to a page when it is browsed by him. We implemented a prototype of the proposed scheme, and conducted preliminary experiments to evaluate the performance of the scheme. The result of experiments indicates that it takes less than 0.5 sec to reorder a list of 500 URLs received from a search engine according to the preference of users.

1 Introduction

According to the rapid popularization of ICT (Information and Communication Technology) and the WWW, the number of web pages in the Internet explosively increases in recent years. In order to find a demanded page from such enormous number of pages, we usually utilize *search engines* such as Google and Yahoo!, in which the contents of web pages are periodically collected to a central server, and after receiving a query from a user, the server retrieves collected information to return a list of URLs matching the query. An increase of the number of web pages also increases the number of *hits* for a given query, which significantly increases the ratio of *unnecessary* pages in the search result. Thus, in order to exclude such unnecessary pages from the search result, most of conventional search engines try to give an appropriate "rank" to each page, and recommend each user "higher ranked pages" which seem to be relevant to the interest of many users.

However, such a popularity-based ranking scheme does not help to find highly specific pages, such as the pages interesting for primary school children and the pages that attracts attention in a leading-edge field. In order to resolve such problem of conventional search engines, several ideas have been proposed in the literature to re ect the preference of users to the page ranking. A representative of such approaches is SBS (Social Bookmark Service) [6, 7], in which each user is allowed to conduct an explicit annotation of tags to each page. Another approach is to focus on the browsing history of users, as in Google personalized search. Although such history-based approach would be e ective to automatically acquire the preference of users, many internet users do not want to register his private information to the system, and to disclose his browsing history to the other users.

In this paper, we propose a new web page recommendation system which can re ect the preference of each user to the recommended result in an e cient and e ective manner. The basic idea of the proposed scheme is to combine the notion of *preference footprint* to browsed pages with the collaborative filtering. To this end, we introduce the notion of "tags" similar to conventional SBSs, but in contrast to SBSs, we associate those tags to each user, and attach *all* tags associated with a user to a page when it is browsed by the user. Tags attached to pages are shared by all users. As a result, we could acquire the information on the distribution of interest of users relevant to the page, without forcing each user to explicitly designate "what kind of page it is" as in SBSs. It significantly reduces the load of users compared with conventional SBSs. In addition, it reduces the psychological resistance of users, since it needs no registration, and the preference of users is processed merely in a stochastic manner. We expect that such favorable properties of the proposed system motivate many internet users to use our system, which increases the chance of collecting a large amount of tags compared with conventional tag-based page recommendation systems.

The remainder of this paper is organized as follows. Section 2 outlines related work including an overview of collaborative filtering. Section 3 describes our proposed

system, and the details of our prototype system are described in Section 4. Finally, Section 5 concludes the paper with future problems.

2 Related Work

Conventional page ranking schemes can be classified into two categories; i.e., page-centered schemes and user-centered schemes. The former schemes calculate the ranking of pages merely by referring to the information attached to each page, and the latter schemes try to refine a (page-centered) ranking by taking into account the information on each user; i.e., it tries to *personalize* the resultant ranking.

2.1 Page-Centered Schemes

A page-centered ranking scheme returns the same list of URLs to all users without considering the personal information of each user. An advantage of such approach is high reproducibility of the search result. However, it is generally di cult to find an unpopular page from such universal list, and in addition, the outcome of page-centered schemes is easily a ected by an adversarial behavior of selfish users such as SEO (Search Engine Optimization) [11, 12].

2.2 User-Centered Schemes

In contrast to such schemes, a user-centered scheme takes into account the personal information of each user to obtain a personal ranking which is not (significantly) a ected by the popularity of pages. There are two types of user-centered schemes; i.e., schemes which merely rely on the personal information of a single user, and schemes based on the sharing of personal information among several users.

An example of the first type is Rerank.jp developed by Yamamoto *et al.* [4]. This scheme reorders the search result obtained by Yahoo! JAPAN Web API[*1] using editorial operations manually conducted by each user. More concretely, each user executes either "emphasis" or "delete" of keywords shown in a tag cloud, to change the importance of keywords in the search result (i.e., a page containing an emphasized word in its title or snippet will be given a high rank, and a page containing a deleted word will be given a low rank). Another example is Google personalized search, which allows each user to have a ranking according to his preference which is not disclosed to the other users.

A representative of the second type is the collaborative filtering. In this method, several users sharing similar interests are classified into a cluster, and the rank of a page is refined by referring to the preference (or reputation) of the other users in the same cluster. In the next subsection, we overview related work concerned with the collaborative filtering.

2.3 Collaborative Filtering

Recommendation systems based on the collaborative filtering can be classified into two categories by the type of information used in the clustering; i.e., recommendation based on explicit information and recommendation based on implicit information.

Explicit Information: The term "explicit" means that it is explicitly designated by the users as in annotation, or provided via appropriate feedback tools. In general SBSs, annotation is regarded as a private comment attached to the bookmark; i.e., it generally increases the load of users. Although such load of users could be reduced by using memorandum instead of annotation, it causes another problem since memorandum may contain ad hoc representation specific to the users. Sasaki proposed a method [5] to overcome such problem, by focusing on the collection of pages which are attached common tags by di erent users, rather than focusing on the ad hoc representation of those tags.

User profile is another source of explicit information used in many existing recommendation systems. However, it is hard to collect such private information in our case, since general internet users do not want to disclose there preferences, although the load of collection could be reduced by using a simple inquiry form [3] and semantic web technologies [9, 10].

Implicit Information: There are a lot of information recommendation systems based on an implicit information collected from the users, e.g., Amazon.com, YouTube, and RSS feed provided by goo. In those systems, a variety of *vita information* are used as the source of information, such as browsing history, purchase history, operation record, and retrieval words and phrases history. Intuitively speaking, those systems try to trace the "footprints" implicitly given by the users.

A drawback of such approaches is that the contents relevant to a user must be simultaneously contained in the history of (another) user, to realize an e cient recommendation. Another drawback is that we can not distinguish between intentional and random visits merely via a sequence of visits. Such a drawback could be partially overcome by measuring the browsing time (e.g., one can identify a browsing as an intentional one if the browsing time exceeds a predetermined threshold), but we can not

[*1] http://developer.yahoo.co.jp/start/

measure such time at the server side, and a measuring at the client side causes an additional cost.

3 Proposed Method

In this section, we propose a new page recommendation system. This system is designed to collect user's preference without forcing any cost and stress to the users.

3.1 Source of User Information

The basic technique used in the proposed system is user tag and preference footprint.

User tag (U-tag, for short) is a keyword (or a keyphrase) representing the preference of each user, e.g., baseball, major league, and red sox. Each user can register any U-tag representing his preference to the system, if it is not registered. When a user browses a web page by clicking a hyperlink provided by the system, all U-tags associated with the user are attached to the URL and are recorded to the system.

Preference footprint (PF, for short) is a collection of U-tags attached to the URL of a page, which represents the characteristics of the page and is used to recommend a page to an interested user. Note that PF is di erent from annotation of tags in SBSs, in a sense that: PF indicates "what type of users browsed that page," while the meaning of conventional tags is "what is the characteristics of that page." In addition, in our system, every user is also associated with a set of U-tags (similar to web pages), which enables a direct evaluation of the similarity of pages and users via calculating the similarity of the corresponding sets of U-tags (see Section 3.3 for the details).

3.2 Basic Flow of Recommendation

The basic ow of the recommendation process is described as follows:

1. At first, each user registers a set of U-tags representing his preference, to the system. Note that each user can register several U-tags. Let $T(u)$ denote the set of U-tags associated to user u.
2. User u sends a query to a search engine through our system. See Figure 3.1 for illustration. After receiving a list of URLs from the search engine, the system reorders the list according to the similarity between $T(u)$ and U-tags associated to each URL, and forwards the reordered list to the requesting user u.
3. The list of URLs shown to u is augmented with a list of corresponding PFs for each URL (if any). User u selects a URL in the list if he is interested in the contents of the web page. (At this point, u can refer to U-tags associated to URLs shown in the list, and he can use such tags to navigate the keyword search through our system. See Section 3.4 for the details.)
4. After receiving a URL selected by user u, the system adds U-tags contained in $T(u)$ to the set of U-tags associated with the selected URL.

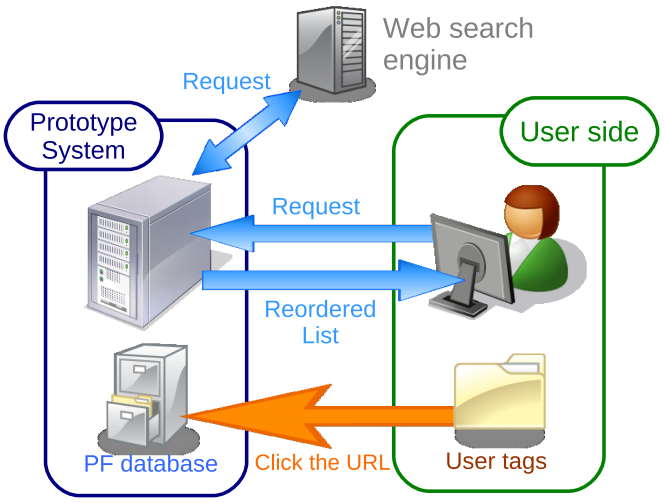


図 3.1 Basic flow of recommendation.

In the following, we describe the way of calculating similarity between two sets of U-tags (Section 3.3) and the way of clustering URLs according to the similarity between URLs (Section 3.4).

3.3 Calculation of Similarity of Tags

We adopt the cosine similarity as the measure of similarity between two sets of U-tags. More concretely, we consider a vector space model (VSM) in which a set of U-tags is represented by a vector, and the similarity between two sets is evaluated by calculating the similarity between two corresponding vectors. Each coordinate in the VSM corresponds to a U-tag; i.e., a vector has a non-zero entry at the i^{th} coordinate i the corresponding set contains the i^{th} U-tag. If it has a non-zero value, the value of the i^{th} coordinate is determined by applying a function known as tf-idf [2], in the following manner.

3.3.1 tf-idf

Let S be a multiset of U-tags associated with a URL or a user, and t be a U-tag contained in S. At first, function tf is defined as the number of occurrences of t in S (if S is a set of U-tags associated to a user, the value of function $\mathsf{tf}(t, S)$ is either zero or one). Next, function idf is defined such that a U-tag specific to the set takes a large value, and U-tags commonly contained in many sets take a small value. More concretely, function idf is

defined as follows:

$$\mathsf{idf}(t) = \log(N/n_t) \tag{3.1}$$

where N is the total number of multisets, and n_t is the number of multisets containing t. By definition, a popular tag which is contained in many sets takes a small value close to zero, and conversely, a rare tag which is contained in few sets takes a large value.

Function tf-idf, which is intended to represent the importance of t in S, is formally defined as follows:

$$\phi(t, r) \stackrel{\text{def}}{=} \mathsf{tf}(t, r) * \mathsf{idf}(t).$$

Using such notions, similarity between two multisets X and Y is defined as follows:

$$sim(X, Y) \stackrel{\text{def}}{=} \frac{\sum_{t \in T} \{\phi(t, X) * \phi(t, Y)\}}{\sqrt{\left(\sum_{t \in T} \phi(t, X)^2\right) * \left(\sum_{t \in T} \phi(t, Y)^2\right)}}$$

where T denotes the set of all U-tags.

3.3.2 Procedure

Without loss of generality, let us assume that our system has already known set $T(u)$ of U-tags associated with user u. After receiving a list of URLs as the result of query issued by user u, the system conducts a reordering of those URLs according to the similarity to $T(u)$; i.e., it sequentially calculates the similarity to $T(u)$ for each URL, and sorts those URLs in a non-increasing order of the similarity. Since $\phi(t, T(u))$ takes a value either zero or one for any t, in the calculation of similarity between X and $T(u)$, we may simply add $\phi(t, X)$'s for each $t \in T(u)$ (note that the denominator of the formula takes a constant value for given X and $T(u)$).

In order to speed up such selective additions, in the proposed system, we adopt Bloom filter [8] to avoid unnecessary search of non-existing elements (each vector is represented as a list of elements, since we could not bound the length of vector in advance).

3.4 Hierarchical Clustering of URLs

In addition to the reordering of a list of URLs, in the proposed system, we prepare a mechanism to navigate the search of a target page via indicating U-tags associated with each search result. Recall that the proposed system is designed to provide a higher rank to a URL if it is similar to the preference of the requester. In addition, by referring to the set of U-tags attached to the listed URLs, a user can recognize "which U-tag associated to him is actually used in the reordering." Ordering of URLs can change if another set of U-tags is used in the reordering, and it motivates a navigation of page ranking via the change of the reference set of U-tags which is initially set to the set of U-tags associated with the user.

In the proposed system, we realize such navigation via the change of reference set by introducing two new techniques, i.e., 1) hierarchical clustering of URLs and 2) identification of U-tags which characterize such clusterings. This is an extension of the method proposed in [2]. More concretely, a clustering of several subclusters corresponds to a *threshold* concerning to the similarity of those subclusters, where similarity of two subclusters is defined as follows:

$$sim(C_1, C_2) = \min_{x \in C_1, y \in C_2} \{sim(x, y)\}.$$

The maximum size of each cluster (i.e., the maximum number of subclusters contained in each cluster) is given as a parameter. Appropriate value of such parameter will be determined through extensive simulations, but it is left as a future work. Note that the computation time required for such calculation can be hidden in general situations, since we can calculate the similarity between any two URLs in advance. Although the similarity between two URLs should be recalculated periodically according to an increase of the number of U-tags associated with each URL, we have an intuition such that the period of such update is relatively long, e.g., few weeks and few months.

Calculated values of similarity among URLs are informed to the requester with a list of URLs, and actual clustering of URLs is conducted at the client side. At the side of the resulting hierarchical clustering, the system displays a set of U-tags associated with each cluster, in order to navigate the control of the search result towards a finding of a target page.

3.5 Observation

A key issue in our proposed scheme is how to collect a large number of U-tags from users having various preferences. As was described previously, the basic operation in our proposed system is to attach *all* U-tags to the selected URL. In other words, we do not consider any discrimination or filtering of U-tags in this operation, since it is di cult to realize a selective attachment of tags without the aid of voluntary users. Instead of taking such approach, we adopt an optimistic strategy such that an uncontrolled accumulation of U-tags causes an appropriate distribution of the frequency of U-tags re ecting the interest of the browsing users.

4 Prototype System

We implemented a prototype system to demonstrate the availability of the proposed scheme. The program is written in PHP5 (server) and JavaScript (client). The environment of the server is as follows: Intel(R)

表 4.1 Format of PF.

Item	Explanation
URL	Used as a page ID
Tag	Name of tag
Times	Number of annotations of the tag (i.e., tf value)

Core(TM)2 Duo CPU E7400, 2GB Memory, Ubuntu/8.10, Apache/2.2.11, and PHP/5.2.9. As the search engine, we used Yahoo! JAPAN Web API, which returns at most 50 URLs for each query.

4.1 Data Structure

In the prototype system, a set of U-tags associated to each user is stored in a cookie in his personal browser, and each user can register a U-tag relevant to him either by specifying a keyword in a free description form, or by clicking U-tags which have already been registered by the other users. We adopt XML as the basic format of transmitted messages, because of its popularity and the ease of manipulation. PF (preference footprint) of each user is also described in the form of XML, and an attachment of PF to a URL is simply done by merging two XML forms.

As a concrete set of U-tags, we used livedoor clip datasets[*2] in our experiments. This data set was published in December 2008 by livedoor clip[*3] which is one of the most popular SBSs in Japan. Each record in the data set consists of four fields, i.e., user ID, the date of bookmarking, URL, and attached tags. In the experiments, we converted it into a data set such that: 1) each record in the set is indexed by URL, and 2) the record concerned with a URL contains all U-tags associated with the URL (note that the original data set contains several records corresponding to each URL). The resultant database, which will be referred to as PF database hereafter, consists of 200 thousand URLs and 150 thousand U-tags.

4.2 Basic Functions

Basic functions used in the proposed scheme are implemented as follows (See Figure 4.1 for illustration): 1) calculation of tf-idf value for each U-tag associated with URL is executed after receiving a message from a user; The tf and idf values are periodically updated. 2) similarity between a set of U-tags associated with a user and URLs contained in a URL list received from the search engine, is also calculated by the server and each client

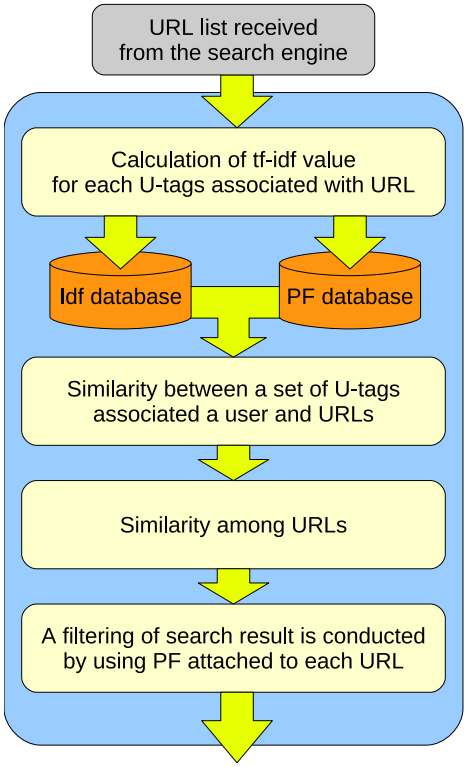


図 4.1 Basic functions used in the proposed scheme are implemented.

merely receives the result of such calculation, i.e., after receiving such information, each client conducts a reordering of the search result using JavaScript; 3) similarity among URLs, which is necessary to realize a hierarchical clustering of URLs at a client, is calculated by the server after completing the calculation of tf-idf (we are going to move this part to each client); and finally, 4) a filtering of search result is conducted by using PF attached to each URL.

4.3 Quick Retrieval of Record

In order to realize a quick access to a record in the database, in the prototype system, we divide the PF database into 256 files in the following manner: 1) each file is associated with two hex digits from 00 to , 2) each URL is mapped to a hex string by an appropriate hash function, and 3) each of those 256 files stores PF data concerned with URLs such that a prefix of the hex string corresponding to the URL matches hex digits associated with the file. In addition, as a way of detecting the non-existence of a particular data in the database, we adopt a Bloom filter of 164 bits (details of parameters are omitted in this extended abstract).

[*2] http://labs.edge.jp/datasets/
[*3] http://clip.livedoor.com/

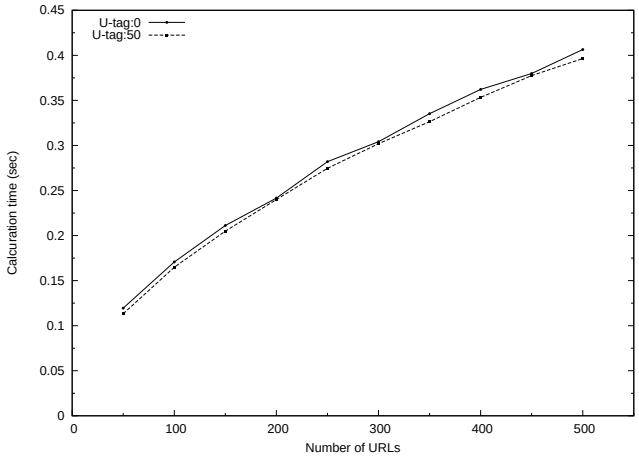


図 4.2 Average calculation time of the similarity to the received URLs for each user.

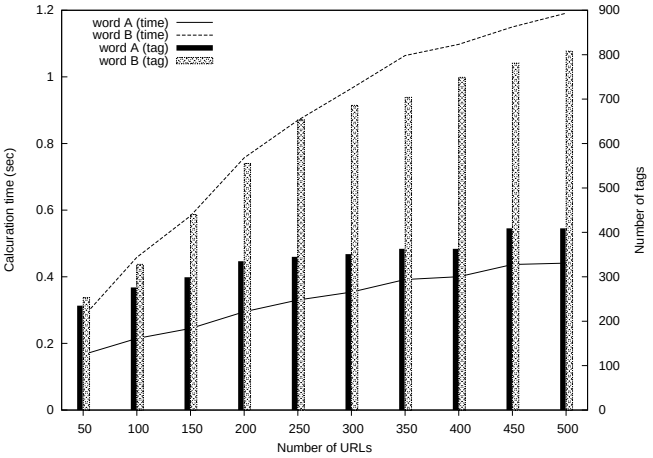


図 4.4 Comparison of two query words.

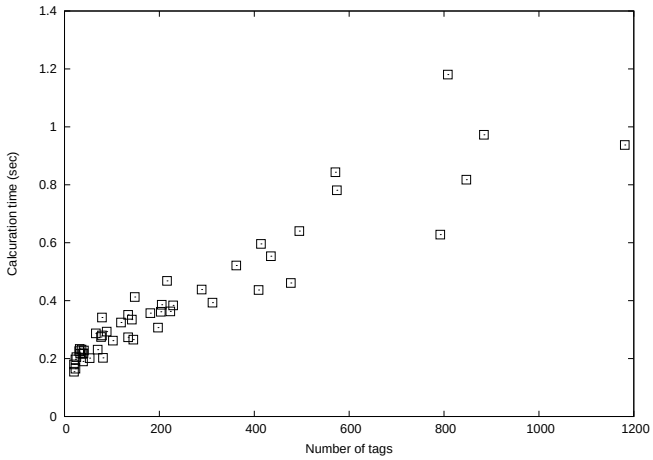


図 4.3 Relationship between the number of U-tags and the overall calculation time.

4.4 Evaluation

We conducted preliminary experiments to evaluate the performance of the prototype system. In the experiments, we consider two users u_1 and u_2, where user u_1 is attached 50 U-tags which are identical to "top 50" of the search word ranking of Yahoo! Japan[*4], and user u_2 is attached no U-tags. We measure the time required for calculating the similarity between the set of U-tags attached to each user and the set of U-tags attached to URLs received from the search engine, by varying the number of received URLs from 50 to 500.

Figure 4.2 summarizes the result. The vertical axis of the figure represents an average calculation time over 20 runs, where "U-tag:50" indicates the calculation time for user u_1 and "U-tag:0" indicates the calculation time for user u_2. From the figure, we can observe that the calculation time monotonically increases as increasing the number of URLs contained in the search result (e.g., it takes 0.25 sec for 200 URLs and 0.4 sec for 500 URLs and), but it is not a ected by the number of U-tags attached to each user. The rate of increasing the calculation time gradually decreases as increasing the number of URLs, which is due to the reduction of the percentage of URLs attached U-tags; i.e., in the data set used in the experiments, U-tags are attached to a limited number of (popular) URLs received from the search engine.

Figure 4.3 shows the calculation time of the scheme for each query, where each point in the figure corresponds to a query which is averaged over 20 runs. The vertical axis represents the total number of U-tags associated with the resulting URLs, and the vertical axis represents the total calculation time. This result indicates that the calculation time of the scheme depends on the number of U-tags contained in the search result independent of the number of URLs in the search result. In order to observe the relationship between those factors in more detail, we selected two query words w_A and w_B, and compared the calculation time and the number of U-tags concerned with those words by changing the number of URLs in the search result. Figure 4.4 shows the result. The horizontal axis is the calculation time (left hand side) and the number of U-tags contained in the search result (right hand side). As shown in the figure, the di erence of the calculation time is certainly proportional to the di erence of the number of U-tags.

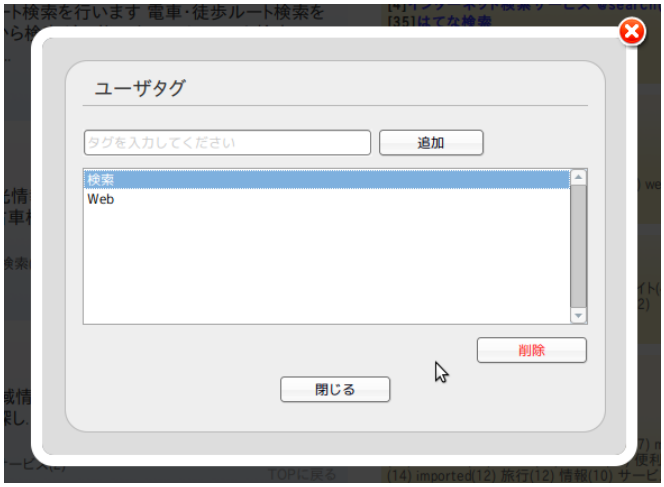
Screen shots of the prototype system are shown in Figures 4.5 and 4.6.

[*4] The period of counting the search words is from January 1 to June 30, 2008. See `http://searchranking.yahoo.co.jp/ranking2008firsthalf/` for the details

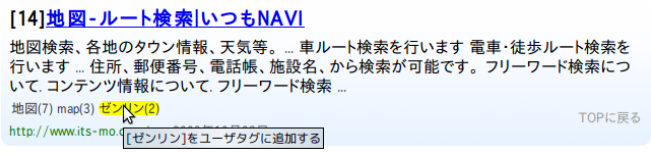
(b) Before a clustering.

(b) After clustering.

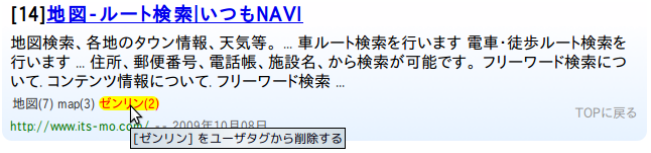
図 4.5　Screen shot (1): Clustering.



(a) Edit of a tag.



(b) Deletion of a tag.



(c) Insertion of a tag.

図 4.6　Screen shot (2): Manipulation of tags.

5　Concluding Remarks

In this paper, we proposed a new scheme for web page recommendation which re ects the preference of each user to the recommended result in an e cient and effective manner. We conducted preliminary experiments to evaluate the performance of the scheme and showed the feasibility of the scheme.

A future work is to improve the e ciency of the proposed scheme, by adopting a selective attachment of U-tags to the browsed pages, and/or by tuning parameters used in the scheme (e.g., the cluster size). We are also planning to open the prototype system for public use, in order to demonstrate the e ectiveness of our web page recommendation scheme in the real world.

参考文献

[1] Okkyung Choi, Sangyong Han, and Ajith Abraham. Integration of Semantic Data Using a Novel Web Based Information Query System. In *International Journal of Web Services Practices*, Vol1, No.1-2, page 21–29, 2005.

[2] Shepitsen Andriy, Gemmell Jonathan, Mobasher Bamshad, and Burke Robin. Personalized recommendation in social tagging systems using hierarchical clustering. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 259–266, 2008.

[3] Tak W. Yan, and Hector Garcia-Molina SIFT—A tool for Wide-Area Information Dissemination. In *Proceedings of the 1995 Usenix Technical Conference*, pages 177–186, 1995.

[4] Takehiro Yamamoto, Satoshi Nakamura, and Katsumi Tanaka. Rerank-By-Example: E cient Browsing of Web Search Results. In *Proceedings of the 18th International Conference on Database and Expert Systems Applications*, pages 801–810, 2007.

[5] Akira Sasaki, Miyata Takamichi, Inazumi Yasuhiro, Aki Kobayashi and Sakai Yoshinori. Web Content Recommendation System Based on Similarities among Contents Cluster of Social Bookmark [in Japanese] . In *Information Processing Society of Japan*, pages 14–27, 2007.

[6] Delicious http://delicious.com/

[7] Netvouz http://www.netvouz.com/

[8] Burton H. Bloom Space/time trade-o s in hash coding with allowable errors. In *Communications of the ACM*, Vol 13, pages 422–426, 1970.

[9] Zhou Xujuan, Wu Sheng T., Li Yuefeng, Xu Yue, Lau Raymond Y. K., and Bruza Peter D. Utilizing Search Intent in Topic Ontology-Based User Profile for Web Mining. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 558–564, 2006.

[10] Alexander Pretschner, and Susan Gauch. Ontology based personalized search. In *Proceedings of the 11th IEEE International Conference on Tools with Artici al Intelligence*, pages 391–398, 1999.

[11] Dennis Fetterly, Mark Manasse, and Marc Najork. Spam, Damn Spam, and Statistics: Using statistical analysis to locate spam web pages. In *Proceedings of the Seventh International Workshop on the Web and Databases*, pages 1–6, 2004.

[12] Timothy Jones, Ramesh Sankaranarayana, David Hawking, and Nick Craswell. Nullification test collections for web spam and SEO. In *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web*, pages 53–60, 2009.

Web サービスミドルウェアに対応可能なチェックポイントの作成方法

中井亮† 中村純哉† 櫟粛之†† 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

††NTT コミュニケーション科学基礎研究所

概要 Web サービスとは，複数の計算機が連携してサービスを提供する分散システムであり，近年その利用に注目が集まっている．一方で，Web サービスは多数の計算機から構成されており，その計算機の一部に故障が生じることは避けられない．そこで，一部の計算機に故障が生じても Web サービス全体に重大な影響を及ぼさないように，Web サービスに故障耐性をもたせることが重要となる．分散システムの故障耐性を実現する手段として，チェックポイント法がよく知られている．既存のチェックポイント法では，システム内の各プロセスが任意のタイミングでチェックポイントを作成できることを仮定している．しかし現実の計算機では，その実行状態は複雑で容易にチェックポイント作成することはできない．そのため，チェックポイント法を現実の分散システムに対して実装する際には，ローカルでのチェックポイント作成方法が問題となる．本稿では，既存の Web サービスのミドルウェアに対する実装を考慮し，メッセージログを利用したチェックポイントの作成方法を提案する．

1 はじめに

Web サービスとは，複数の計算機が通信により連携してサービスを提供するシステムである．図 1 の例では，顧客は欲しい商品のリクエストを送信するだけで，システムが商品の探索や配達の手続きといった作業を人間に代わり自動で行う．これにより，受発注の効率化や在庫の削減といった効果が期待される．

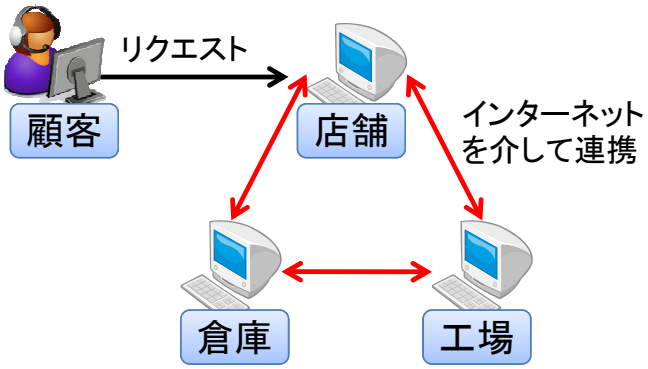


図 1 Web サービス（マーケットプレース）の例

多くの Web サービスでは，計算機同士の通信に SOAP(Simple Object Access Protocol) と呼ばれる技術が利用されている．この SOAP による通信を行うためのツールとして Web サービスミドルウェアが存在する．Web サービスはこのミドルウェア上で提供される．代表的な Web サービスミドルウェアとして Apache Axis2 [1] が存在する．

一方で，Web サービスは複数の計算機から構成される分散システムであり，システムの大規模化・複雑化により，一部の計算機が故障することは避けられない．そこで，計算機に故障が生じた際にシステム全体を正常な状態に復元する故障耐性を，Web サービスに持たせることが重要となる．

分散システムで故障耐性を実現する方法として，チェックポイント・ロールバックリカバリが存在する．この手法では，各計算機はその状態をチェックポイントとして安定記憶 (故障が生じてもその内容が失われない記憶) に保存しておく．故障が生じた際には，計算機状態をチェックポイントまでロールバックさせ実行を再開することで正常な状態に復旧する．しかし，分散システムにおいては，各計算機を独立にロールバックさせると，複数の計算機の間で矛盾が生じる場合がある．そこでシステムに矛盾が生じないように，各計算機がチェックポイントを作成するタイミングとロールバックする手順を定めたものがチェックポイント法である．

Web サービスでは，インターネットに接続された数多くの計算機がサービスに参加するという点からシステムの規模が大きく，またサービスに対して新たな計算機の参加や離脱が生じるという点から動的という特徴がある．これらの特徴に適合する手法として，守屋らの手法 [2] がある．そこで，この手法を Web サービスミドルウェア [1] に対して適用することで故障耐性を実現することを考える．Web サービスミドルウェアに対して故障耐性を付加することにより，Web サービスの提供者は，信頼性の高いサービスを提供することが可能となる．これにより，Web サービスの利用の拡大と新たなサービスの創出が期待される．

手法 [2] では，各計算機は任意のタイミングでチェックポイントを作成できることと，そのチェックポイントへのロールバックが可能であることを仮定している．しかし，既存の Web サービスミドルウェアにはそのようなチェックポイント作成やロールバックの機能は存在しない．そこで，既存のミドルウェアに対してチェックポイント作成とロールバックの機能追加が必要となる．

本稿では，既存の Web サービスミドルウェアに対して

実装可能なチェックポイント作成方法を考える．Web サービスにおけるリクエストの実行処理が，受信メッセージの内容と順番により決定される点に着目し，受信メッセージのログを利用してチェックポイントを作成する方法を提案する．また，保存したログに基づいてリクエストの実行を再現することで，チェックポイント作成時の状態にロールバックが可能である．

2　Web サービスのシステム構成とリクエスト処理モデル

ここでは，故障耐性を付加する対象となる Web サービスのシステム構成とリクエスト処理について述べる．

2.1　システムモデル

各計算機に 1 つの Web サービスミドルウェアが存在し，ミドルウェア上では複数のスレッドが並行してリクエスト処理を実行する．また，各計算機はローカルに 1 個以上の DB を持つ．これらはローカルスレッド間の共有メモリとしての役割を果たす．システム構成図を図 2 に示す．

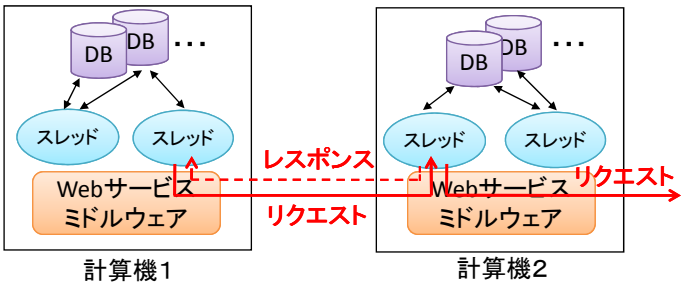


図 2　システム構成

リクエストの送信は，図 2 で示すように Web サービスミドルウェアを介して行われる．またリクエスト送信には，レスポンス無しとレスポンス有りの 2 種類が存在する．レスポンス有りのリクエストを送信したスレッドは，レスポンスを受け取るまで実行を中断する．

2.2　スレッドによるリクエスト処理

Web サービスにおいて，スレッドによるリクエスト処理の実行は次のように行われる．まずリクエストの受信により新たなスレッドが生成され，リクエスト処理が開始される．スレッドは決定的な動作を行いリクエストを処理する．その実行の途中で他の計算機へのリクエスト送信，ローカル DB へのアクセスを行うことがある．リクエスト送信に対するレスポンスの受信や，DB の読み出しを行ったときには，スレッドはその内容に従い決定的な実行を行う．最初に受信したリクエストがレスポンス有りの場合には，リクエストの送信元に対してレスポンスを送信する．すべての処理が終了するとスレッドは消滅する．スレッドによるリクエスト処理の流れを図 3 に示す．

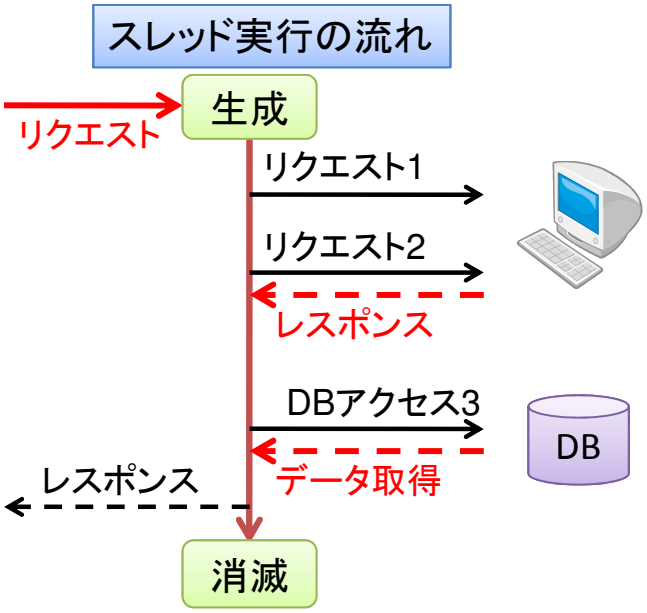


図 3　スレッドによるリクエスト処理

最初に受信したリクエストの内容により，スレッドの初期状態が決まる．その後，スレッドはリクエスト 1 を送信するまで決定的な処理を行う．リクエスト 1 はレスポンス無しのリクエスト送信であり，送信後スレッドは実行を中断することなく処理を続ける．次にスレッドはリクエスト 2 の送信を行う．リクエスト 2 はレスポンス有りのリクエスト送信であり，送信後，レスポンスを受信するまでスレッドは実行を中断する．レスポンスを受信したスレッドは，その内容により以降の実行が決定的される．実行を再開したスレッドは DB アクセス 3 を行うまで決定的に処理を行う．DB からデータを取得したスレッドは，その内容により以降の実行では決定的な処理を行う．最後にスレッドの実行結果をレスポンスとしてリクエスト送信元に返信し，すべての処理が終わるとスレッドは消滅する．

スレッドの実行は，スレッドの初期状態を決めるリクエストメッセージの内容と実行の途中で送信したリクエストに対するレスポンスの内容，DB アクセスによりに取得したデータの内容とそれらの受信順にのみ依存して決定される．

3　チェックポイントの要件

3.1　記録内容

チェックポイント作成時に記録しなければならないのは，作成時における計算機の実行状態を完全に再現することができるだけの情報である．各計算機の実行状態は，ローカルで実行中の全スレッド状態と DB の内容により決まる．したがって，これらを取得できればチェックポイントの作

成が可能となる．

一方，Web サービスミドルウェアから取得可能な情報は，DB の内容とそのアクセス履歴，送受信したリクエスト・レスポンスの内容と順番，DB アクセスの内容と取得したデータである．そこで，これらの情報を利用し，かつ実用的な時間でチェックポイントの作成とロールバックが可能なチェックポイントを作成する必要がある．

3.2 記録場所

チェックポイントは，故障が生じた際にその記録内容を利用して計算機の状態を復元させる．したがって，故障が生じた際にチェックポイントとして保存した記録内容が失われては意味がない．そこで，チェックポイントの内容は，安定記録と呼ばれる，故障が生じてもその内容が失われない記憶領域に保存する．しかし，安定記憶はデータアクセスが遅いという欠点がある．そこで，普段はデータアクセスが速い揮発性記憶を利用し実行を行い，チェックポイントを作成する際には，その内容を安定記録に保存する．

4 提案手法

Web サービスに対して耐故障性を付加するため，Web サービスミドルウェアに対して，チェックポイント法を利用した耐故障機能の追加を考える．しかし，Web サービスミドルウェアには，直接チェックポイントを作成する機能は存在しない．また各計算機上では，複数のスレッドが並行してリクエスト処理を実行しているが，それらの状態を直接取得することや，ある状態に直接復元することは困難である．そこで，Web サービスでは，リクエストを処理するスレッドの実行が受信メッセージの内容とその順番により決定的に決まるという点に着目し，メッセージログを利用することでスレッド状態の取得や復元を行うことを考える．

4.1 メッセージログを利用したチェックポイントの作成

Web サービスでリクエストを処理するスレッドの状態は，1) スレッドを生成したリクエストメッセージ，2) スレッド実行中に受信したレスポンスメッセージ・DB アクセスにより取得したデータとそれらの受信もしくは取得した順番によって決定される．そして，これらの情報はともにミドルウェア上で取得可能である．そこで，この 2 つの情報を利用してチェックポイントを作成する．

4.1.1 メッセージログ

チェックポイントの作成に利用するメッセージログには，以下の情報を記録する．

- 受信したリクエスト
- スレッド実行中のリクエスト送信・DB アクセス回数
- 受信したレスポンス，DB アクセスにより取得したデータとその受信順

4.1.2 メッセージログの作成

以下ではチェックポイント作成時に利用する，スレッドのメッセージログ作成の手順を示す．

1. スレッド生成時にメッセージログ (以降ログ) を作成する
2. スレッドが実行途中にリクエスト送信・DB アクセスを行う場合，それをログに記録する
3. リクエスト送信に対するレスポンスを受信した場合，レスポンス内容をログに記録する
4. DB アクセスによりデータを取得した場合，そのデータをログに記録する
5. スレッド実行終了時にログを消去する

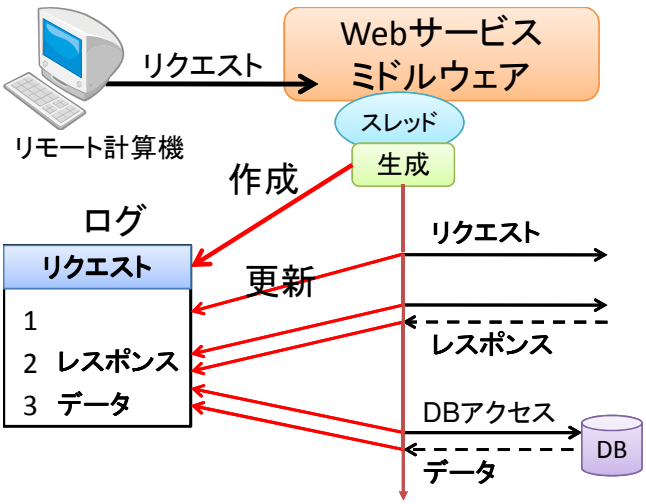


図 4　メッセージログ作成

図 4 では，Web サービスミドルウェアがリクエストを受信した際にスレッド生成する．リクエスト処理を開始する前に揮発性記録ログを作成し，リクエストをログに追加する．スレッドがリクエスト処理を開始し，リクエスト送信を行う前に，ログにリクエスト送信回数を記録する．ただし，送信するリクエスト内容に関してログには記録しない．リクエスト送信に対してレスポンスを受信する場合は，そのレスポンスの内容をログに記録する．このとき，そのレスポンスが何番目のリクエスト送信に対応しているかも記録しておく．DB に対するアクセスにおいてもリクエスト送信の場合と同様に，アクセス記録をログに追加する．DB からデータの読み出しを行った場合には，そのデータもログに記録しておく．スレッドによるリクエスト処理が完了したら，作成したログを消去する．

4.1.3 チェックポイント作成

チェックポイントの作成は，リクエスト処理実行中の全スレッドのメッセージログと，ローカルの全 DB の内容を安定記録に保存することで行う．

4.2 ロールバック

システムに故障が生じた際には，保存しておいたログに基づいてリクエスト処理を再実行し，チェックポイント作成時の全スレッドの実行状態とローカルの全 DB の内容を復元することでロールバックを行う．

ログには，スレッドの実行状態を再現するために必要なリクエストメッセージ，リクエスト送信回数，リクエスト送信に対応したレスポンスメッセージとその受信順が含まれており，これらを用いることでオリジナルの実行と同一の実行を再現することが可能である．リプレイ実行の際にはリクエストの送信は行わない．レスポンス有りのリクエスト送信の場合には，ログからそのレスポンスの内容を取得する．また，ログには実行中のリクエスト送信回数，DB アクセス回数も含まれている．これにより，どの地点までリプレイ実行を行えばよいか判断することができ，チェックポイント作成時の状態を再現することができる．

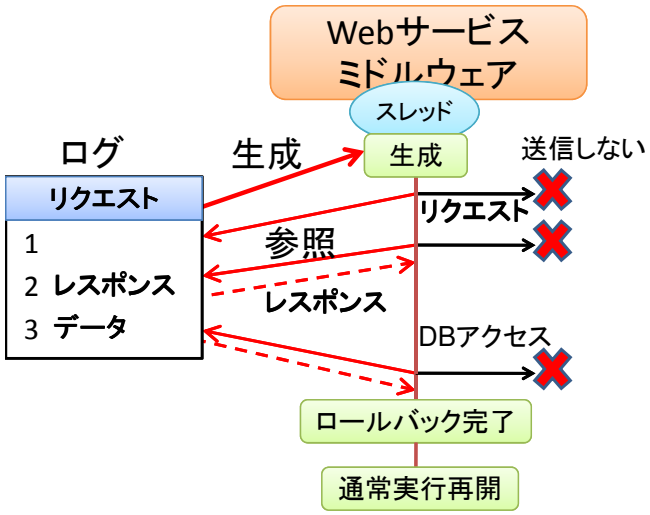


図 5　ロールバック

図 5 では，ログに記録されているリクエストを利用してスレッドを生成し，リプレイ実行を開始する．リプレイ実行中にリクエスト送信を行う場合，実際にはリクエスト送信は行わない．また，レスポンス有りのリクエスト送信を行う場合，通常の実行であればレスポンスを受信するまでスレッドは実行を中断するが，リプレイ実行ではレスポンスをログから取得して実行を続ける．ログに記録されたリクエスト送信，DB アクセスをすべて実行した箇所までリプレイを行ったとき，チェックポイント作成時のスレッド実行状態を再現したことになり，ロールバックが完了する．

ロールバック完了後は，通常のスレッド実行を再開する．

5 まとめ

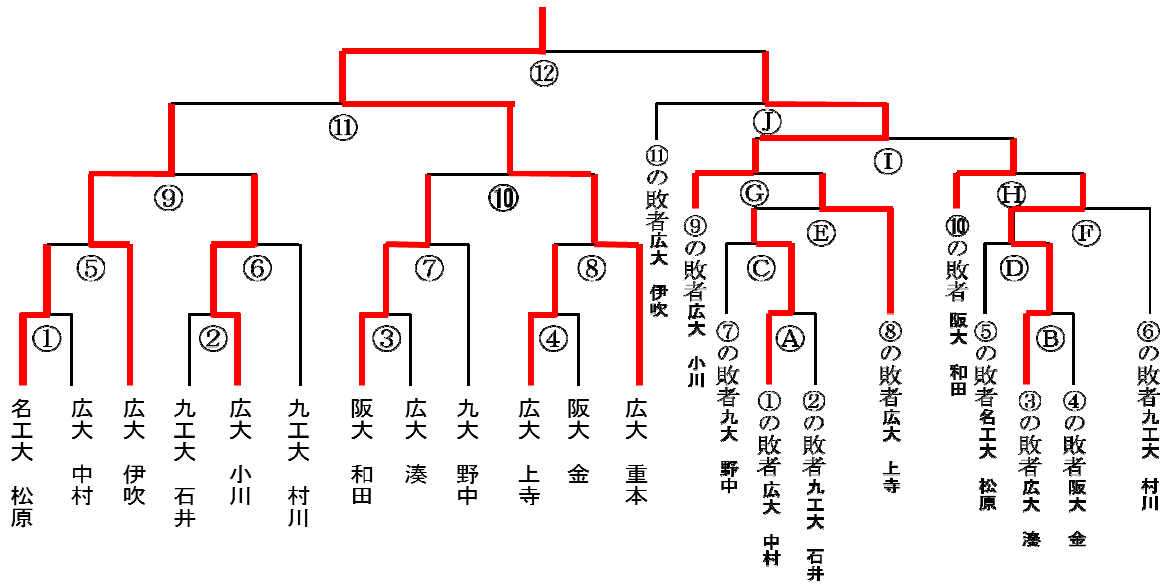
本稿では，メッセージログを利用したチェックポイント作成方法を提案した．これを利用することで，チェックポイント法を既存の Web サービスミドルウェアに対して適用することが可能である．今後の課題としては，提案した方法を用いて Web サービスミドルウェア [1] に耐故障機能を実装すること，実機を用いた性能評価 (チェックポイント作成によるオーバーヘッド，ロールバック時間)，さらに Web サービスの特性により適したチェックポイント法の考案が挙げられる．

参考文献

[1] Apache Axis2
http://ws.apache.org/axis2/

[2] . 守屋宣・櫟粛之：「インターネットエージェントのための動的スナップショットアルゴリズムと部分ロールバックアルゴリズム」. 電子情報通信学会論文誌, Vol. J86-D-I, No.5, p.p.301-317, 2003.

１００００円ゲーム大会結果



順位

- １位　広島大学　重本さん
- ２位　広島大学　小川さん
- ３位　広島大学　伊吹さん

結果詳細

試合	結果		
(1)	名工大　松原	744-256	広島大　中村
(2)	九工大　石井	355-645	広島大　小川
(3)	大阪大　和田	643-457	広島大　湊
(4)	広島大　上寺	548-451	大阪大　金
(A)	広島大　中村	688-312	九工大　石井
(B)	広島大　湊	685-315	大阪大　金
(5)	名工大　松原	475-523	広島大　伊吹
(6)	広島大　小川	733-267	九工大　村川
(7)	大阪大　和田	782-218	九州大　野中
(8)	広島大　上寺	313-687	広島大　重本
(C)	九州大　野中	416-584	広島大　中村

試合	結果		
(D)	名工大　松原	219-772	広島大　湊
(E)	広島大　中村	445-555	広島大　上寺
(F)	広島大　湊	1000-0	九工大　村川
(9)	広島大　伊吹	510-490	広島大　小川
(10)	大阪大　和田	423-527	広島大　重本
(G)	広島大　小川	607-393	広島大　上寺
(H)	大阪大　和田	423-527	広島大　重本
(I)	広島大　小川	603-397	大阪大　和田
(11)	広島大　伊吹	240-760	広島大　重本
(J)	広島大　伊吹	476-524	広島大　小川
(12)	広島大　重本	596-404	広島大　小川