第15回
情報科学ワークショップ

The 15th Workshop on Theoretical Computer Science,
Hiroshima, September 2019 (WTCS2019)

亀井　清華

はじめに

本論文集は 2019 年 9 月 18 日〜20 日，休暇村　大久野島（広島県竹原市）にて開催された 第 15 回情報科学ワークショップでの発表原稿をまとめたものである．本ワークショップは，日本の並列／分散計算の研究者が研究室以上の議論と研究会未満のフォーマルさを目指し，合宿形式でお互 いを徹底的に切りまくる「チャンバラ大会」を目的とし，広島大学の中野浩嗣先生の呼びかけで, 大阪大学, 九州大学, 九州工業大学, 京都工芸繊維大学, 名古屋工業大学，奈良先端科学技術大学院大学の研究者有志によって 2005 年に始まったものである. 2005 年 9 月に第 1 回を出雲で開催して以来， 第 2 回瀬戸, 第 3 回門司, 第 4 回長浜，第 5 回広島，第 6 回桑名，第 7 回福岡，第 8 回神戸，第 9 回 唐津，第 10 回福山，第 11 回北名古屋，第 12 回富士吉田，第 13 回奈良大和平群，第 14 回 飯塚に続き, 今回で 15 回目の開催となる. 今回は法政大学, 広島大学, 大阪大学, 京都大学, 奈良先端科学技術大学院大学，名古屋大学，名古屋工業大学，豊橋技術科学大学，東京工業大学，九州大学，九州工業大学の 11 大学と国立情報学研究所から 51 人の参加者があり，33 件の発表が行われた．今回の開催場所である広島県竹原市大久野島は，うさぎ島として知られるとともに, 第二次世界大戦に関する史跡も残る島である．このような歴史と自然に囲まれた環境で, 並列／分散計算の最先端研究について夜遅くまで活発な討論が行われ, 有意義な時間を共有することができた． 2015 年度から，学生をはじめとする若手研究者たちのモチベーションの向上を念頭に, 参加した大学教員らの審査により, 優れた研究に対して優秀研究賞を，さらに素晴らしいプレゼンテーションを行った学生を対象に優秀プレゼンテーション賞を授与している. 受賞者らには今後の益々の活躍を, そして未受賞者諸君には次の受賞者を目指して更なる研鑽を積んでいただきたいと思う. なお, 来年度は名古屋で開催の予定である． 最後に, 今回の開催にあたりご協力をいただいた休暇村 大久野島に感謝の意を表する. また, ワークショップ運営にご協力をいただいた参加者の皆様に厚くお礼申し上げる.

2019 年 9 月

亀井　清華

目次

Bloom Filterを用いた GPU上の高速パターンテスト

広島大学　林川 雅俊

研究概要

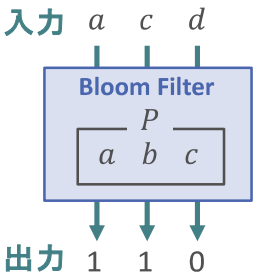
- Folded Bloom Filterの提案
  - 一般的なBloom Filterと比較して，有利な点
    1. 使用するハッシュのビット数が少ないため，ハッシュ値の計算コストが小さい
    2. メモリアクセスの局所化により，メモリアクセスコストを削減
- Bloom Filterを用いたパターンテストの効率的なGPU実装

研究概要（Bloom Filterについて）

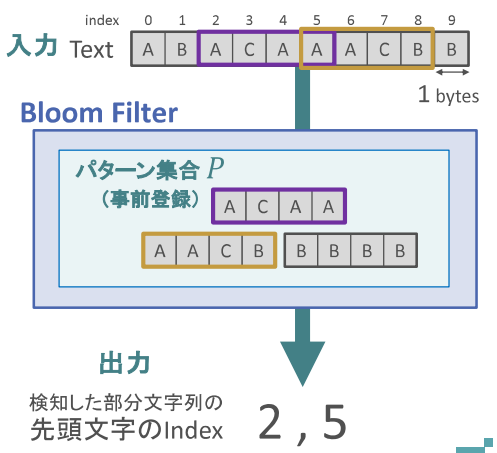
Bloom Filter

- クエリ要素xが集合Pに含まれているかどうかを判定することができるデータ構造

クエリ要素xと集合Pの関係	出力
$x \in P$	1
$x \notin P$	0



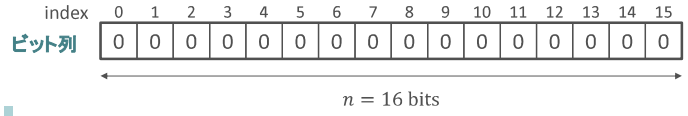
Bloom Filterを用いたパターンテスト



Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

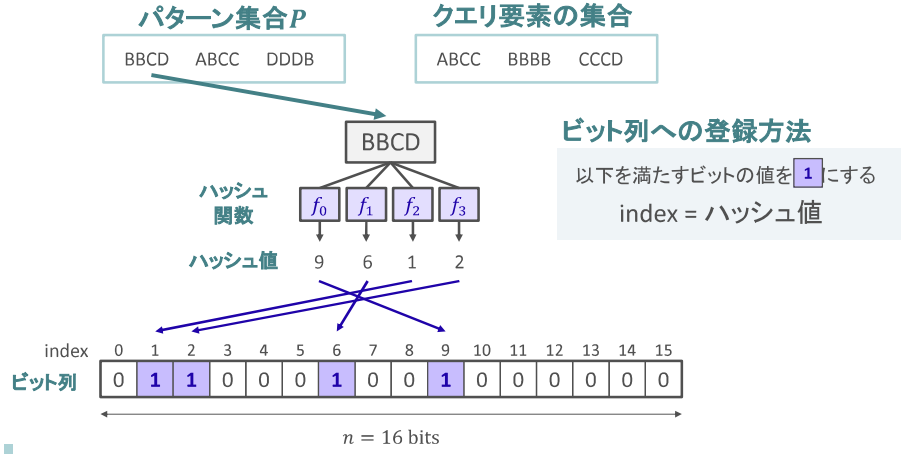
n ビットのビット列を使用
初期値 0



Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

❑ 複数のハッシュ関数を用いて，各要素をビット列に登録



ビット列への登録方法

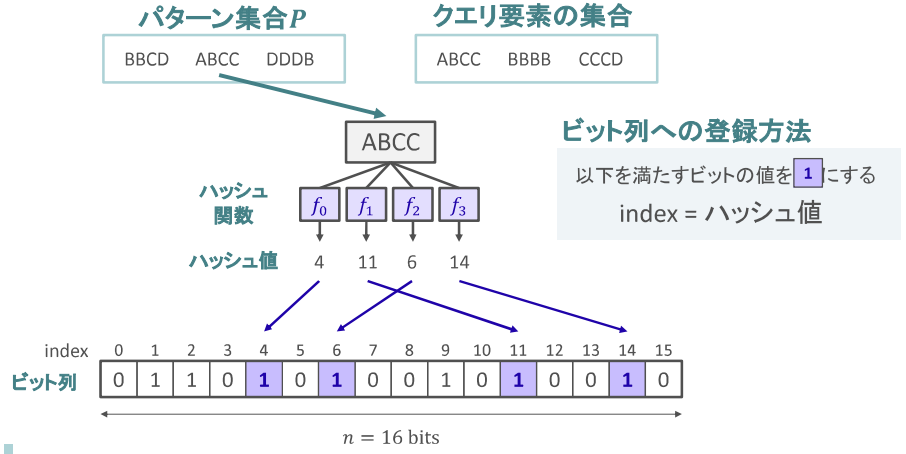
以下を満たすビットの値を 1 にする

index = ハッシュ値

Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

❑ 複数のハッシュ関数を用いて，各要素をビット列に登録



ビット列への登録方法

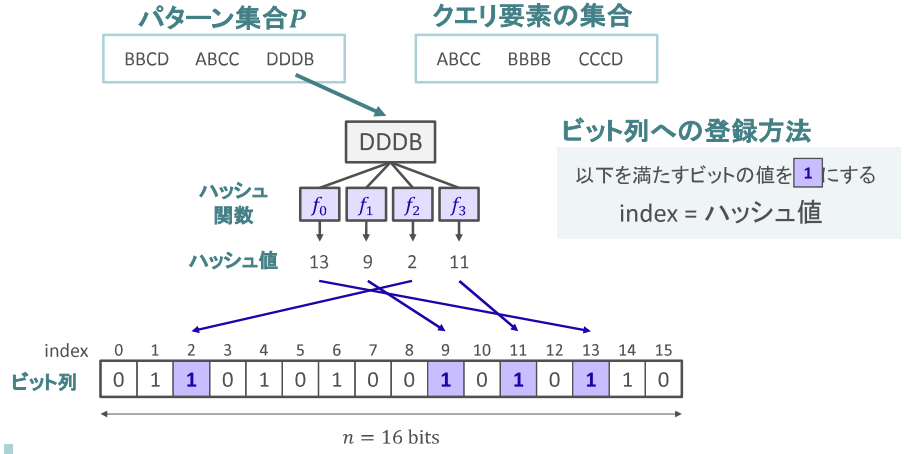
以下を満たすビットの値を 1 にする

index = ハッシュ値

Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

❑ 複数のハッシュ関数を用いて，各要素をビット列に登録



ビット列への登録方法

以下を満たすビットの値を 1 にする

index = ハッシュ値

Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

❑ 複数のハッシュ関数を用いて，各要素をビット列に登録

ビット列への登録方法

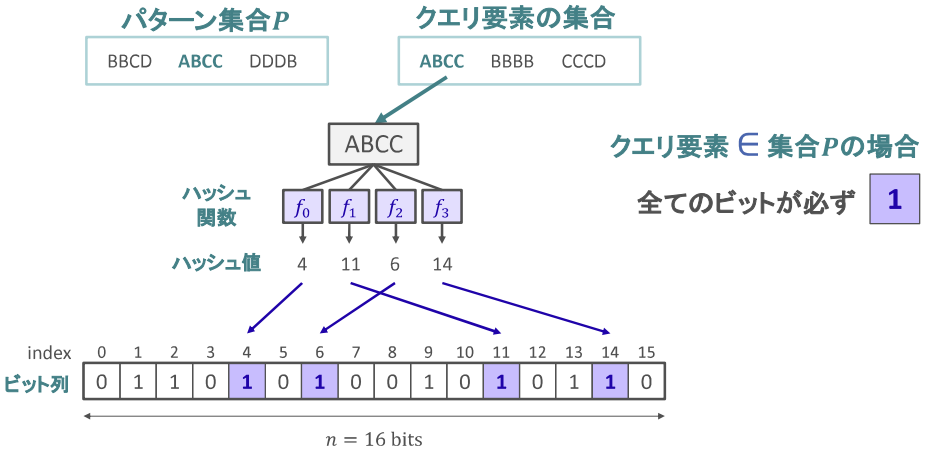
以下を満たすビットの値を 1 にする

index = ハッシュ値

Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

❑ クエリ要素が集合のメンバであるかチェック



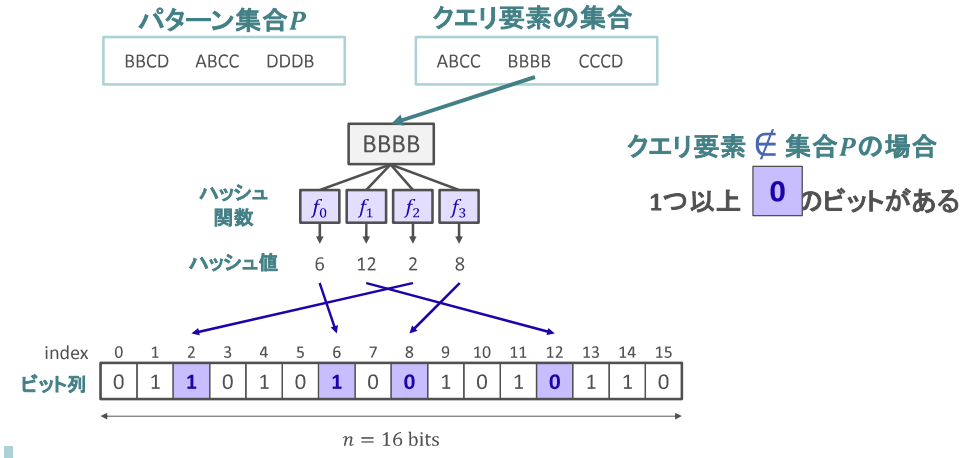
クエリ要素 $\in$ 集合Pの場合

全てのビットが必ず **1**

Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

❑ クエリ要素が集合のメンバであるかチェック



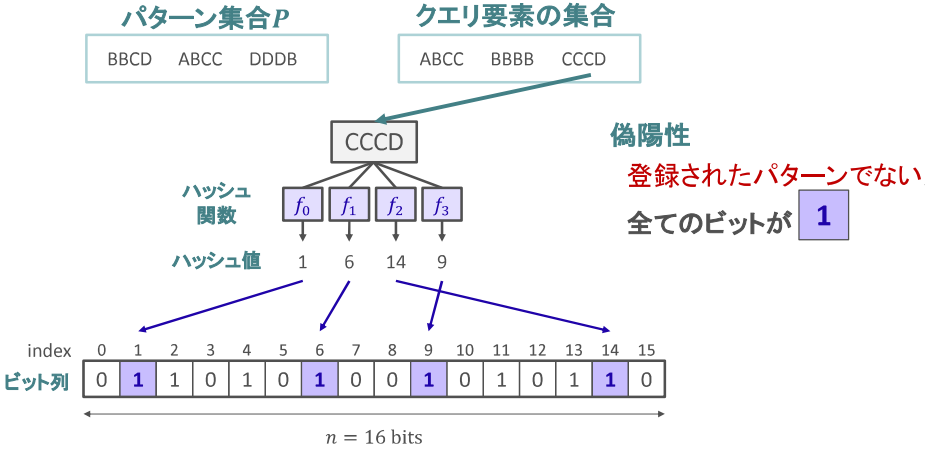
クエリ要素 $\notin$ 集合Pの場合

1つ以上 **0** のビットがある

Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

❑ クエリ要素が集合のメンバであるかチェック



偽陽性

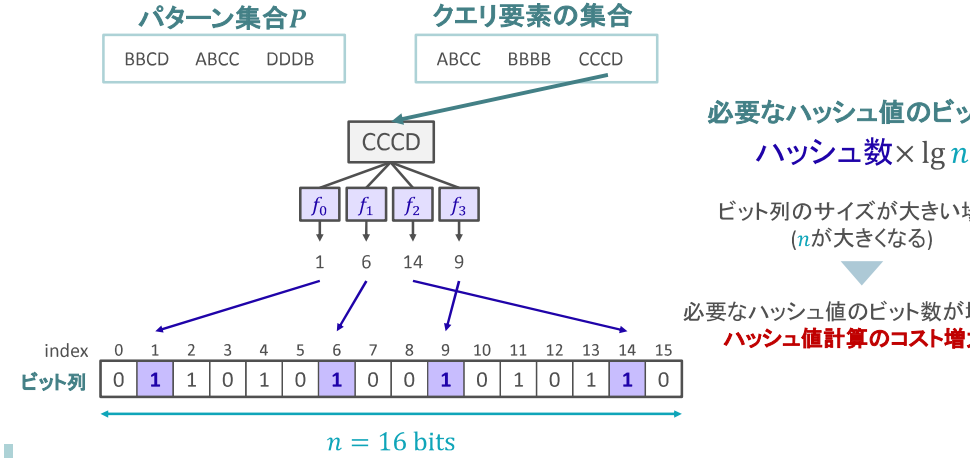
登録されたパターンでないが，

全てのビットが **1**

Bloom Filter

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

❑ クエリ要素が集合のメンバであるかチェック



必要なハッシュ値のビット数

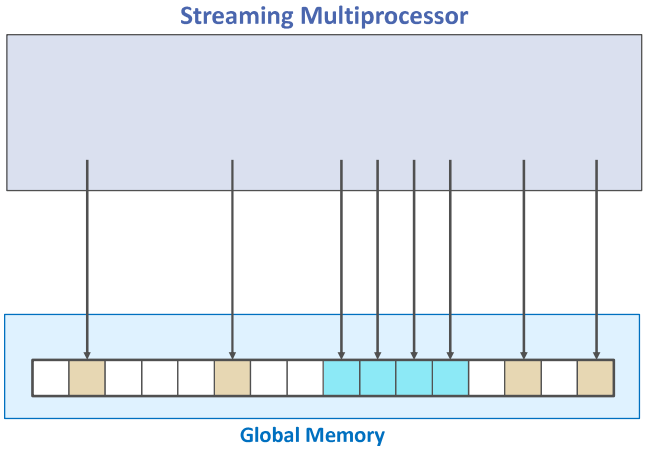
ハッシュ数$\times \lg n$

ビット列のサイズが大きい場合
(nが大きくなる)

必要なハッシュ値のビット数が増加し，
ハッシュ値計算のコスト増大

GPUとGlobal Memoryアクセス

- GPUはアクセス方法によって，振る舞いが異なる



コアレスドアクセス
（連続したメモリにアクセス）

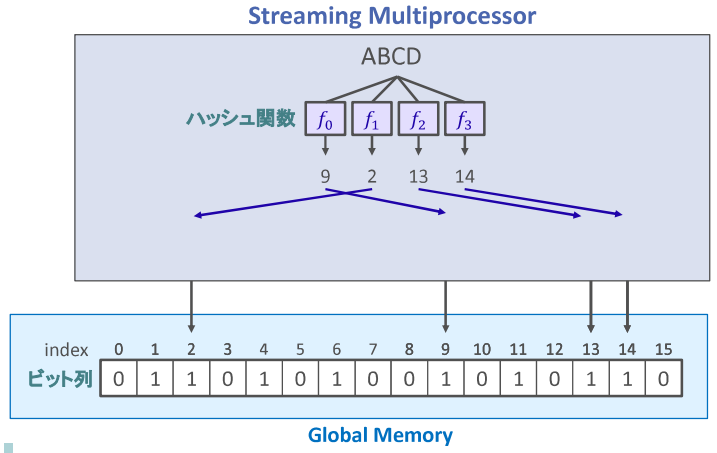
一度でアクセス

ストライドアクセス
（連続**でない**メモリにアクセス）

逐次的にアクセス

Bloom Filterのメモリアクセス

- クエリ要素が集合のメンバであるかチェック



Standard Bloom Filter
ビット列にランダムアクセス

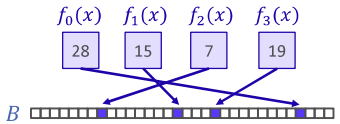
ストライドアクセス
逐次的に処理される

最悪の場合
ハッシュ数だけメモリアクセス

Bloom Filter

ここまでの内容

Standard
Bloom Filter



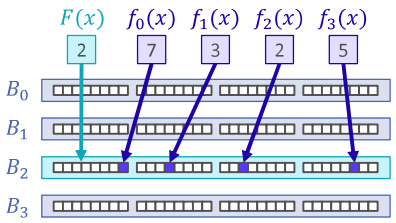
問題点

- ビット列へのランダムアクセス
- ハッシュ値の計算コストが大きい

改善

ここからの内容

Blocked
Bloom Filter



Blocked Bloom Filter

F. Putze, P. Sanders, and J. Singler, Cache-, hash-and space-efficient bloom filters,“ in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2007, pp. 108-121.

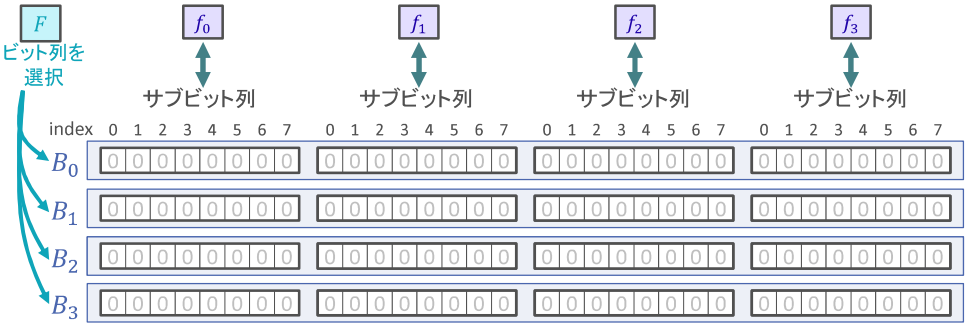
- アイディア： ビットを二種類のハッシュ関数で選択

ハッシュ関数 1

複数のビット列を用意し，
ハッシュ関数Fで選択

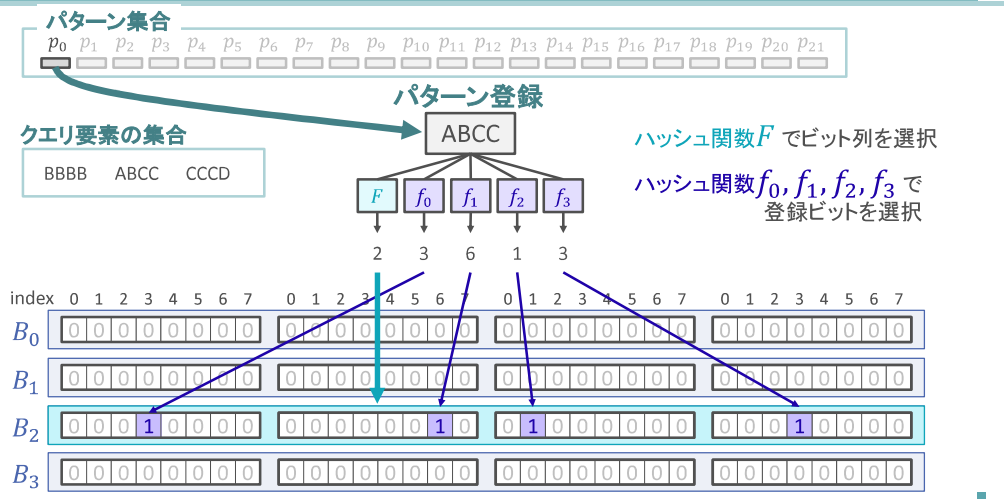
ハッシュ関数 2

各ビット列は，ハッシュ数分のサブビット列で構成
対応するハッシュ関数でビットを選択



Blocked Bloom Filter

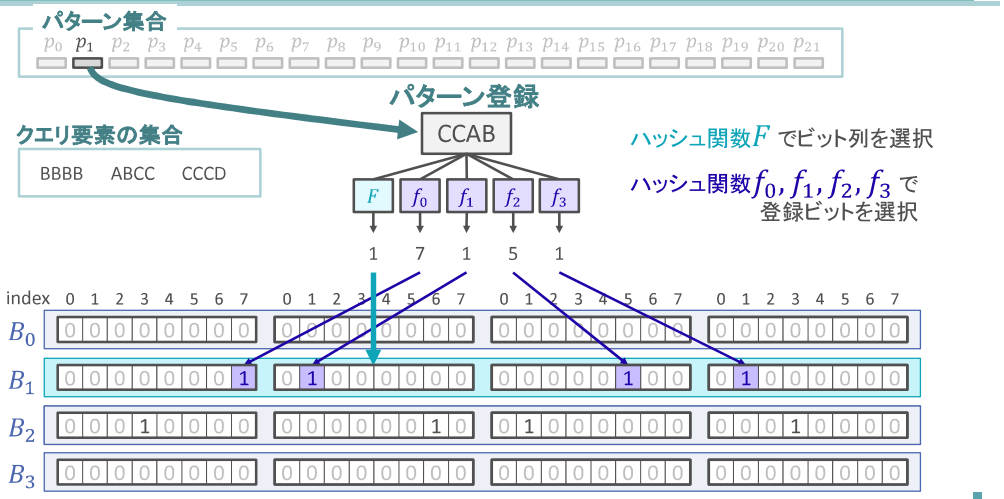
F. Putze, P. Sanders, and J. Singler, Cache-, hash-and space-efficient bloom filters,“ in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2007, pp. 108-121.



17

Blocked Bloom Filter

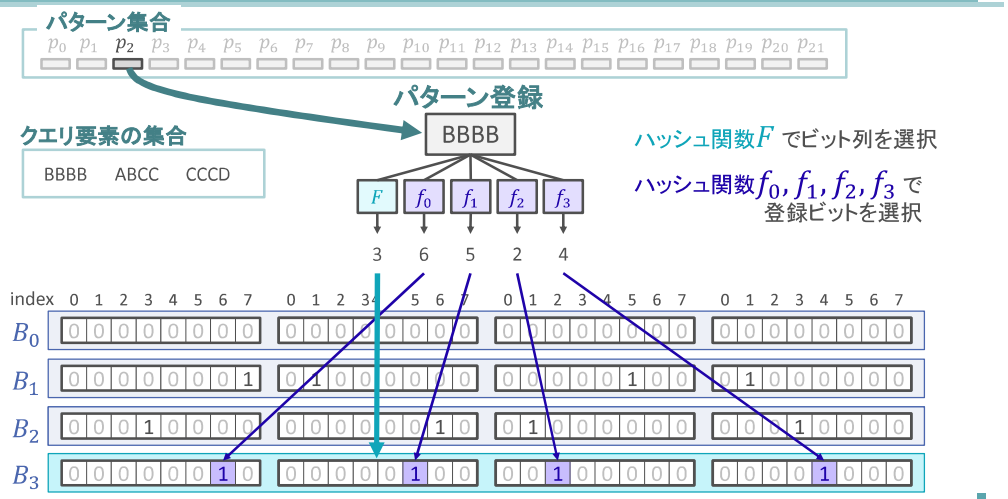
F. Putze, P. Sanders, and J. Singler, Cache-, hash-and space-efficient bloom filters,“ in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2007, pp. 108-121.



18

Blocked Bloom Filter

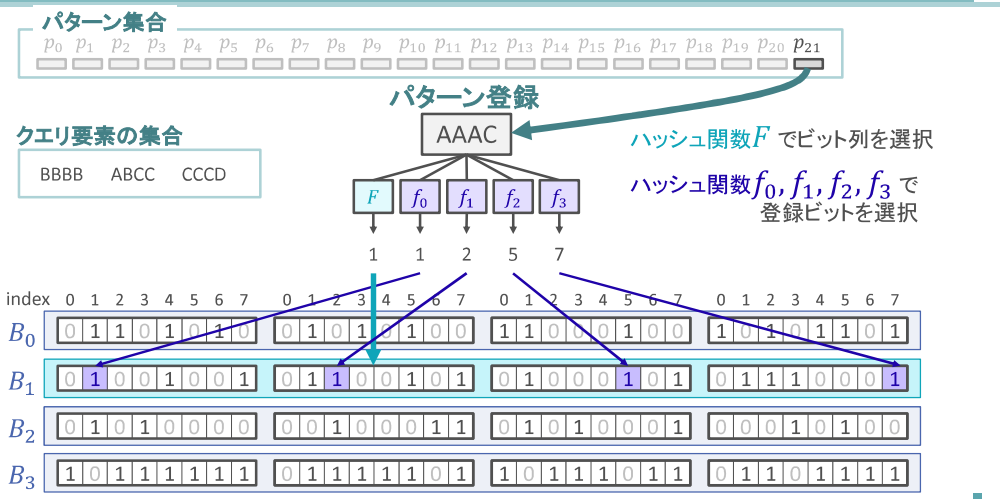
F. Putze, P. Sanders, and J. Singler, Cache-, hash-and space-efficient bloom filters,“ in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2007, pp. 108-121.



19

Blocked Bloom Filter

F. Putze, P. Sanders, and J. Singler, Cache-, hash-and space-efficient bloom filters,“ in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2007, pp. 108-121.



20

Blocked Bloom Filter

F. Putze, P. Sanders, and J. Singler, Cache-, hash-and space-efficient bloom filters,“ in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2007, pp. 108-121.

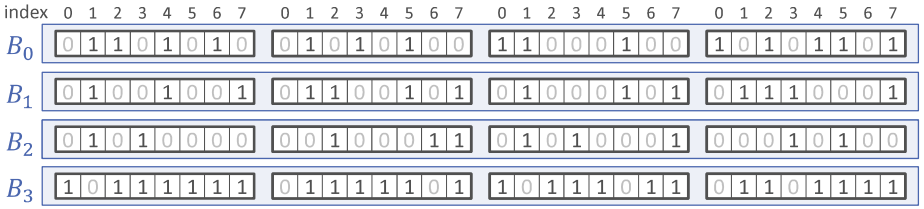
パターン集合

p_0 p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12} p_{13} p_{14} p_{15} p_{16} p_{17} p_{18} p_{19} p_{20} p_{21}

クエリ要素の集合

BBBB ABCC CCCD

パターン集合を登録後のビット列



Blocked Bloom Filter

F. Putze, P. Sanders, and J. Singler, Cache-, hash-and space-efficient bloom filters,“ in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2007, pp. 108-121.

パターン集合

p_0 p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12} p_{13} p_{14} p_{15} p_{16} p_{17} p_{18} p_{19} p_{20} p_{21}

クエリ

クエリ要素の集合

BBBB ABCC CCCD

CCCD

F f_0 f_1 f_2 f_3

2 3 6 3 5

ハッシュ関数F でビット列を選択

ハッシュ関数f_0, f_1, f_2, f_3 で
チェックビットを選択

Blocked Bloom Filter

F. Putze, P. Sanders, and J. Singler, Cache-, hash-and space-efficient bloom filters,“ in *International Workshop on Experimental and Efficient Algorithms*. Springer, 2007, pp. 108-121.

使用ハッシュのビット数について

ハッシュ関数とサブビット列を対応づけ

&

ビット列サイズの増加による
ハッシュのビット数増加は F のみ

使用ハッシュのビット数削減

クエリ

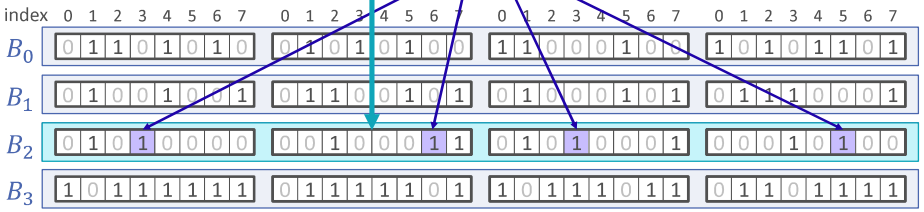
クエリ要素の集合

BBBB ABCC CCCD

CCCD

F f_0 f_1 f_2 f_3

2 3 6 3 5



Blocked Bloom FilterのGlobal Memoryアクセス

Streaming Multiprocessor

CCCD

F f_0 f_1 f_2 f_3

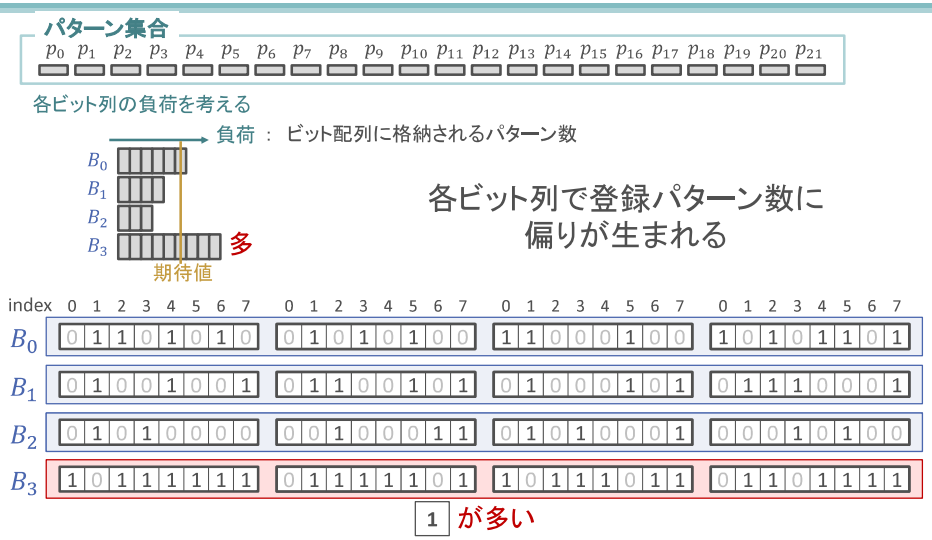
2 3 6 3 5

Blocked Bloom Filter
ビット列のまとまった
アドレスにアクセス

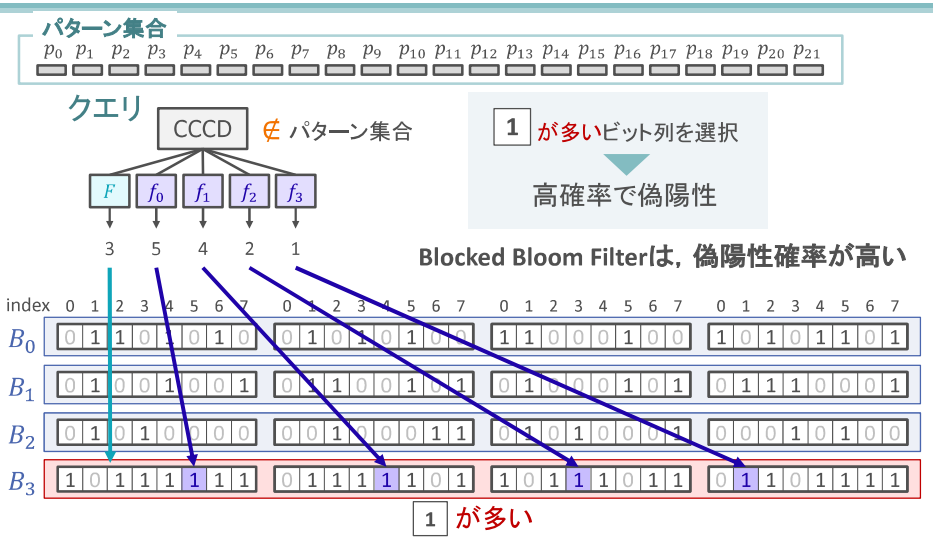
コアレスドアクセス
一度でアクセス

Global Memory

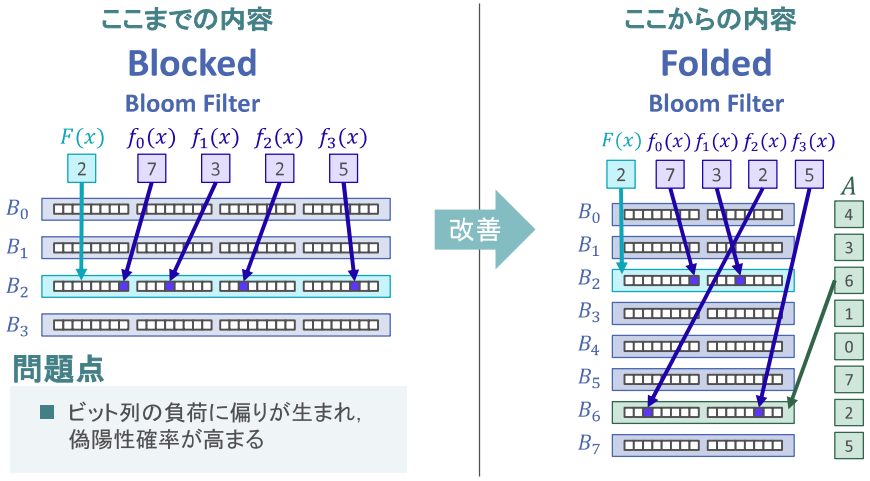
Blocked Bloom Filterの偽陽性確率



Blocked Bloom Filterの偽陽性確率



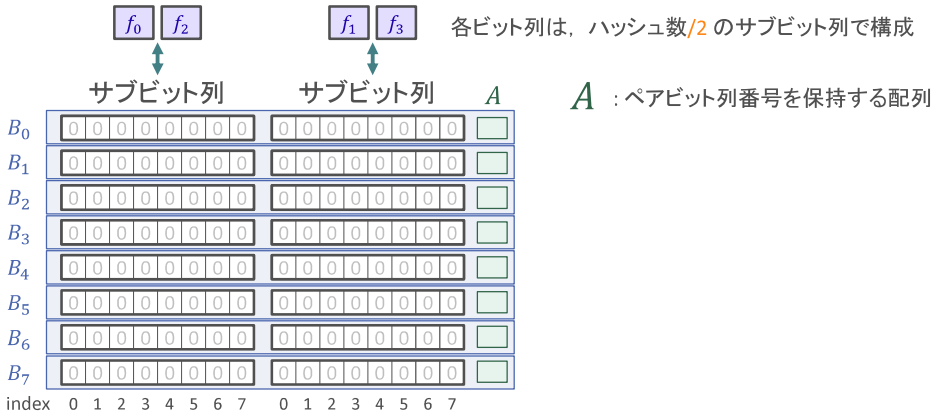
Bloom Filter



Folded Bloom Filter（提案手法）

アイディア 2つのビット列がビットを共有 → 負荷をバランス

複数のビット列（$B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7$）を用意

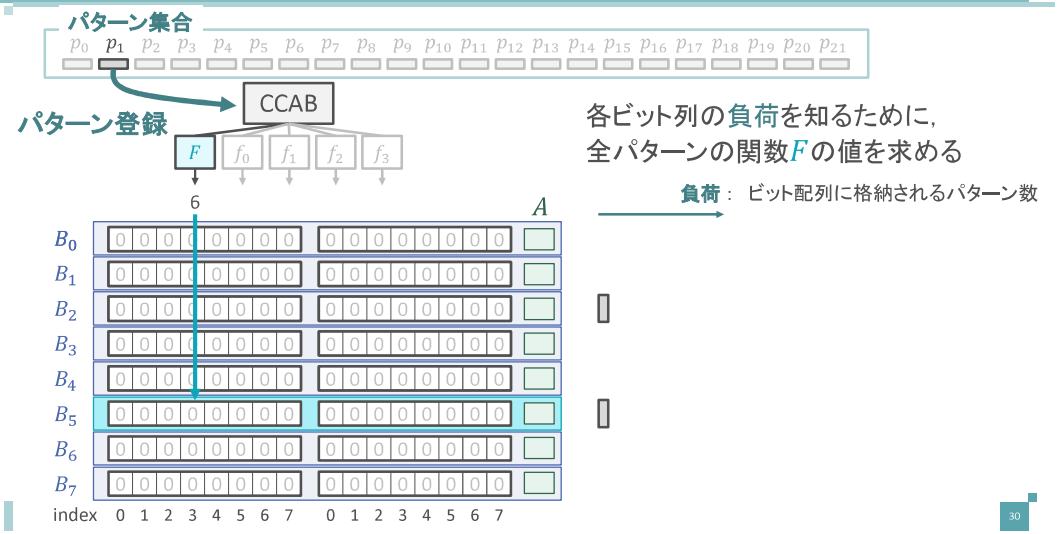


Folded Bloom Filter（提案手法）

各ビット列の負荷を知るために，
全パターンの関数Fの値を求める

負荷： ビット配列に格納されるパターン数

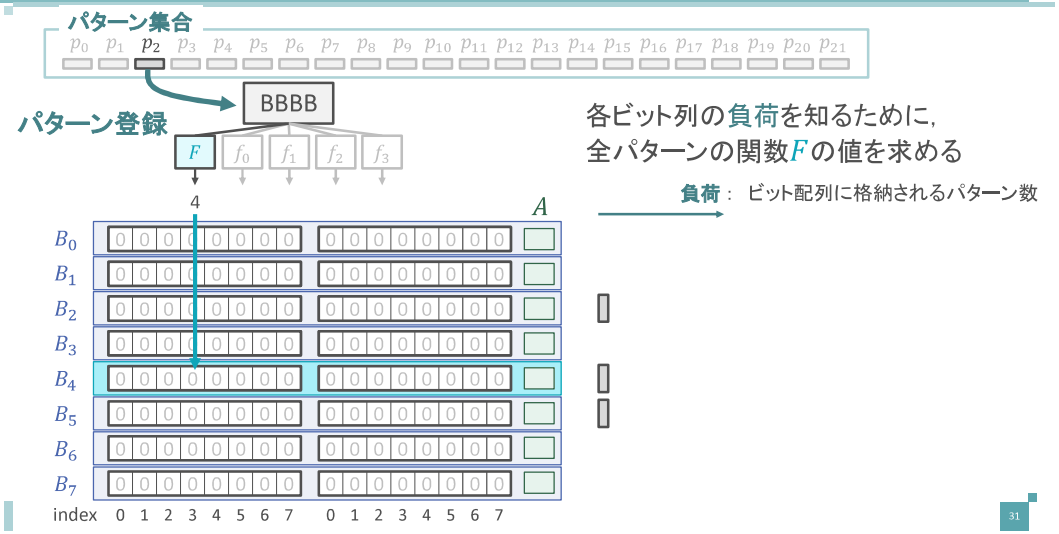
Folded Bloom Filter（提案手法）



各ビット列の負荷を知るために，
全パターンの関数Fの値を求める

負荷： ビット配列に格納されるパターン数

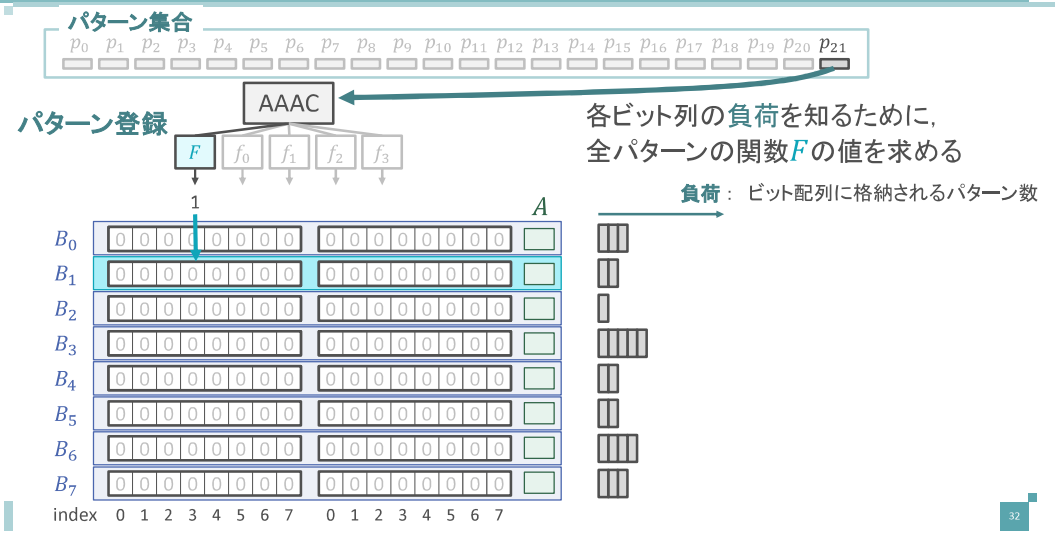
Folded Bloom Filter（提案手法）



各ビット列の負荷を知るために，
全パターンの関数Fの値を求める

負荷： ビット配列に格納されるパターン数

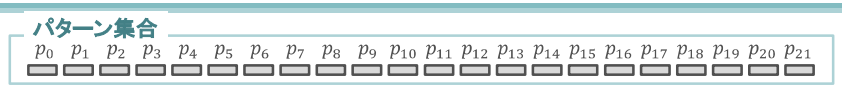
Folded Bloom Filter（提案手法）



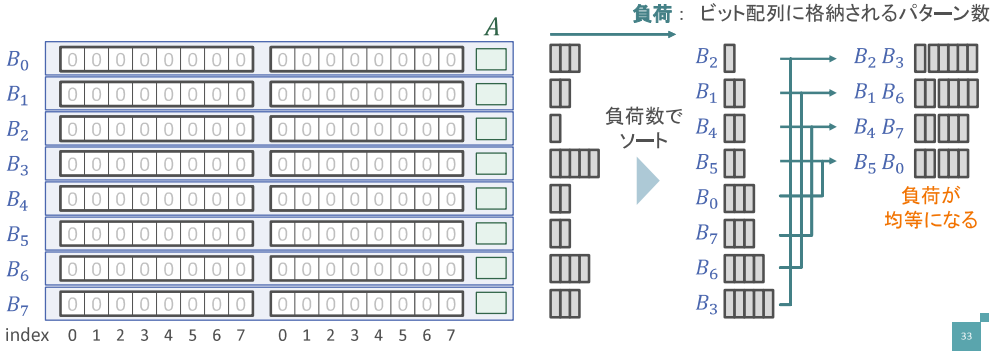
各ビット列の負荷を知るために，
全パターンの関数Fの値を求める

負荷： ビット配列に格納されるパターン数

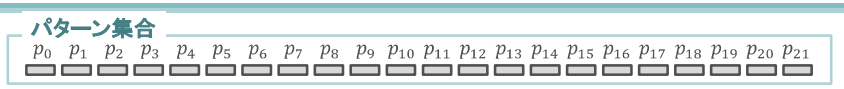
Folded Bloom Filter（提案手法）

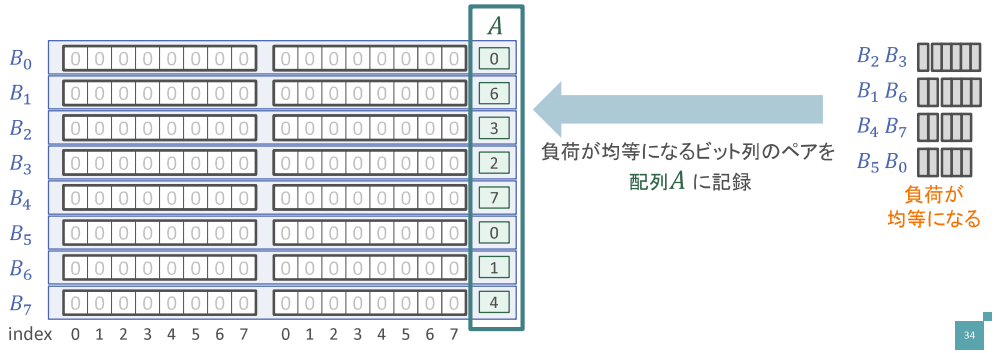


求めた各ビット列の負荷から，
負荷が均等になるビット列のペアを決定

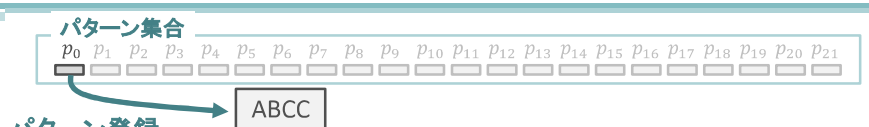


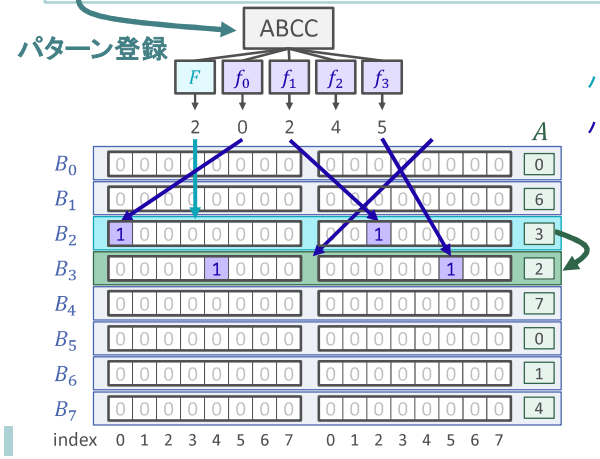
Folded Bloom Filter（提案手法）





Folded Bloom Filter（提案手法）

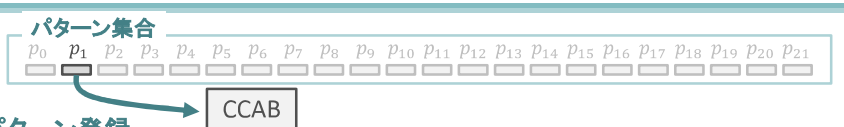


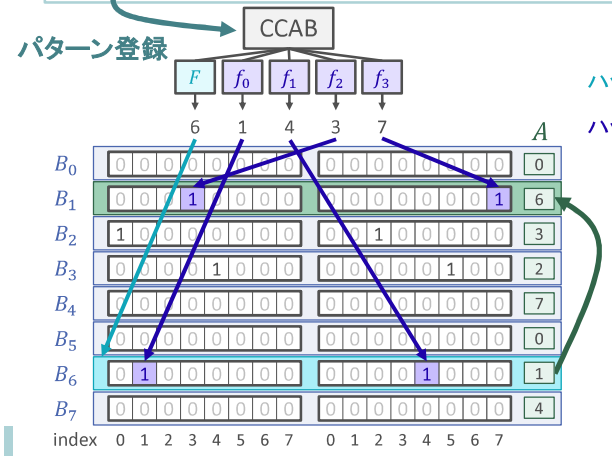


ハッシュ関数Fと配列Aでビット列を選択

ハッシュ関数f_0, f_1, f_2, f_3で登録ビットを選択

Folded Bloom Filter（提案手法）

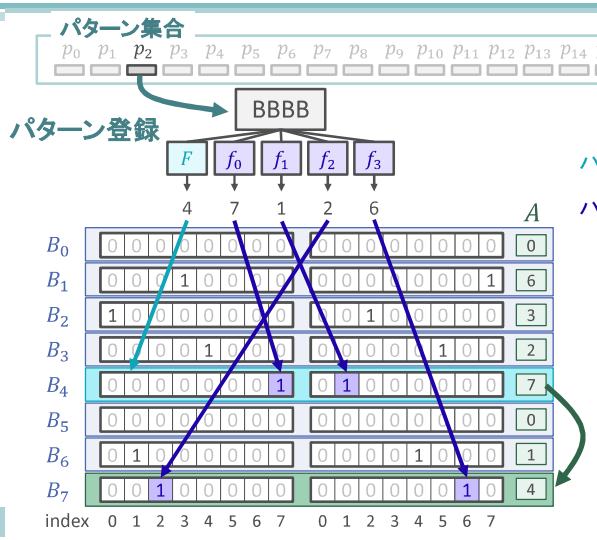




ハッシュ関数Fと配列Aでビット列を選択

ハッシュ関数f_0, f_1, f_2, f_3で登録ビットを選択

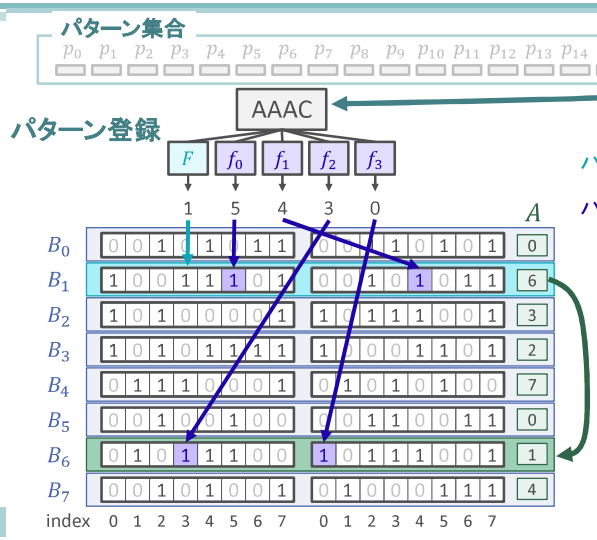
Folded Bloom Filter（提案手法）



ハッシュ関数F と配列A でビット列を選択

ハッシュ関数f_0, f_1, f_2, f_3 で登録ビットを選択

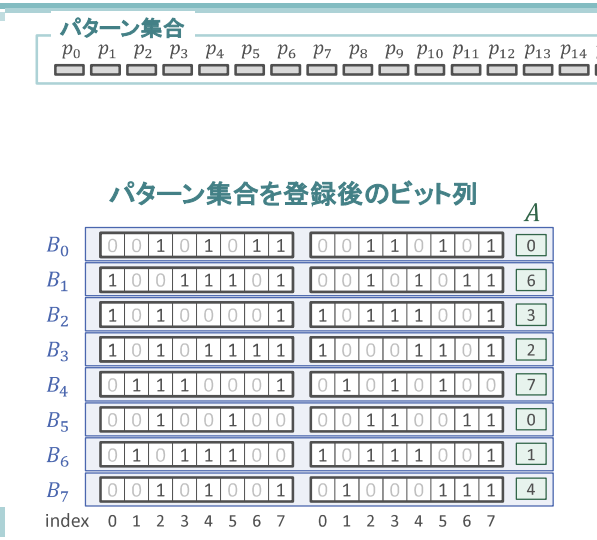
Folded Bloom Filter（提案手法）



ハッシュ関数F と配列A でビット列を選択

ハッシュ関数f_0, f_1, f_2, f_3 で登録ビットを選択

Folded Bloom Filter（提案手法）

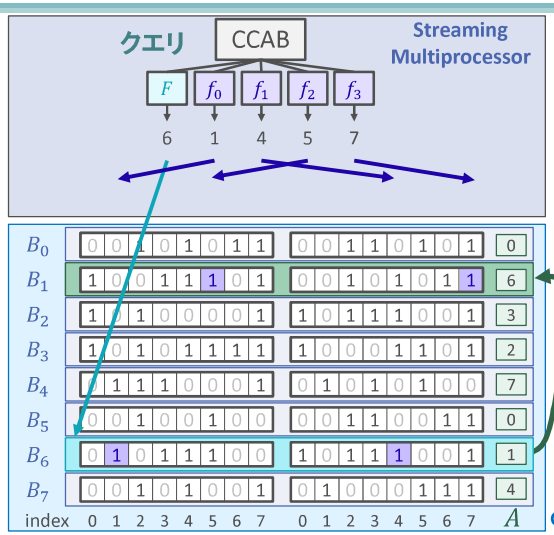


負荷を均等にする操作により，

1 が多すぎるビット列が減少

偽陽性確率減少

Folded Bloom FilterとGlobal Memoryアクセス



Folded Bloom Filter
ビット列のまとまった
アドレスにアクセス

コアレスドアクセス
一度でアクセス

ローリングハッシュ関数

ローリングハッシュ関数 （入力$s_0 s_1 \cdots s_{m-1}$） d, q :素数$(d < q)$

$$R(s_0 s_1 \cdots s_{m-1}) = (s_0 \cdot d^{m-1} + s_1 \cdot d^{m-2} + \cdots + s_{m-1} \cdot d^0) \bmod q$$

$a \otimes_q b = a \times b \bmod q$
$a \oplus_q b = a + b \bmod q$

素数を変えることで，一つの文字列から複数のハッシュ値を取得可能

例 入力文字列 A C B B

$$R_0(\mathrm{ACBB}) = (\mathrm{A} \cdot d_0^3 + \mathrm{C} \cdot d_0^2 + \mathrm{B} \cdot d_0^1 + \mathrm{B} + d_0^0) \bmod q_0$$

$$R_1(\mathrm{ACBB}) = (\mathrm{A} \cdot d_1^3 + \mathrm{C} \cdot d_1^2 + \mathrm{B} \cdot d_1^1 + \mathrm{B} + d_1^0) \bmod q_1$$

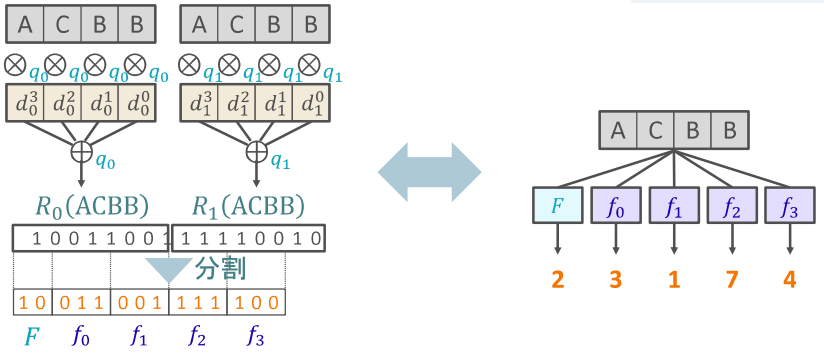
Bloom Filterのハッシュ生成方法

❑ **異なるRolling Hash関数を複数使用し，必要なビット数で分割**

例 入力文字列 A C B B

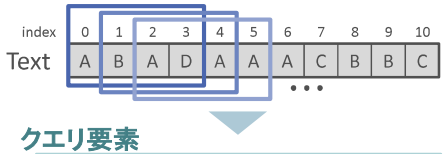
$a \otimes_q b = a \times b \bmod q$
$a \oplus_q b = a + b \bmod q$
d_i, q_i : 素数 $(i = 0,1,2, \ldots)$

ローリングハッシュ関数 と ハッシュの分割

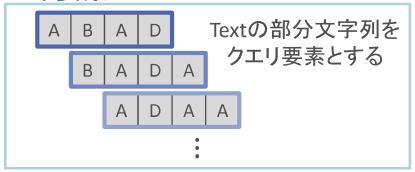


パターンテスト（GPU実装）

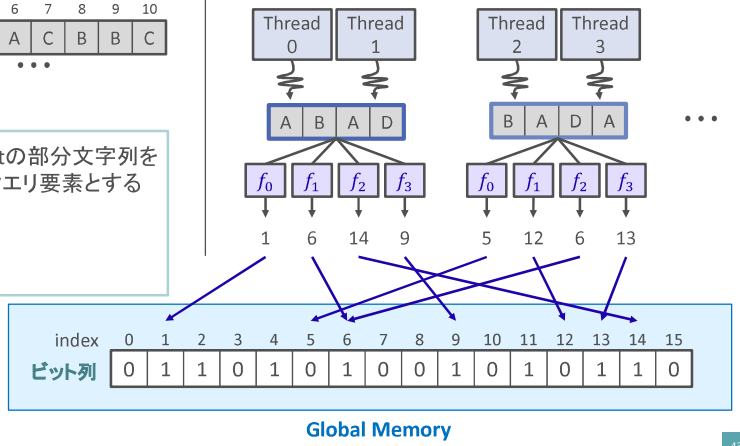
Text内にパターンがあるかどうか判定



クエリ要素



1 部分文字列に 2 Threads を割り当て



性能評価

GPU

Tesla V100

ブーストクロック	1370 MHz
L1キャッシュ	10 MB
L2キャッシュ	6 MB
CUDAコア	5120基
メモリ容量	16 GB

CPU

Intel(R) Xeon(R) Silver
4112 CPU @ 2.60GHz

動作周波数	2.60 GHz
L1 キャッシュ	64 KB/コア
L2 キャッシュ	4096 KB
L3 キャッシュ	8448 KB

実験結果

Pattern数：744261120
Pattern長：2^{10} [byte]
Text長：2^{26} [byte]

ハッシュ数 $h = 64$　　Bit列：2^{36} [bit]

Bloom Filter	Standard	Blocked	Folded（提案手法）
GPU実装 [Gbps]	9.01	25.2	25.6
逐次実装 [Gbps]	0.0124	0.0297	0.0298
高速化率 (GPU実装/逐次実装)	723.70	849.35	856.74
偽陽性確率（理論値）	5.42×10^{-20}	2.02×10^{-12}	1.20×10^{-18}

※ ハッシュ数 $h = 64$は，偽陽性確率が小さすぎるため実測値計測が困難

ハッシュ数 $h = 32$　　Bit列：2^{35} [bit]

Bloom Filter	Standard	Blocked	Folded（提案手法）
理論値	2.33×10^{-10}	9.21×10^{-8}	4.16×10^{-10}
実測値 （GPU実装で計測）	2.21×10^{-10}	9.23×10^{-8}	4.15×10^{-10}

理論値と実測値は非常に近い値であり，$h = 64$の理論値も正しいことが予想される

まとめ

❑ Folded Bloom Filterを提案

- ビット列への**ランダムアクセスを低減**
- 必要なハッシュ値のビット数を減少させ，**ハッシュ値の計算コストをカット**
- ビット列の負荷をバランスすることで，**偽陽性確率を低減**

❑ Bloom Filterを用いたパターンテストの効率的GPU実装

Folded Bloom FilterのGPU実装は，

- **Standard Bloom FilterのGPU実装**と比較して，約 2.84 倍の高速化を達成
- **Folded Bloom Filterの逐次実装**と比較して，約 850 倍の高速化を達成

Convolution-Poolingの GPUへの効率良い実装

Shunsuke Suita, Takahiro Nishimura, Hiroaki tokura, Koji Nakano, Yasuaki Ito
Hiroshima University

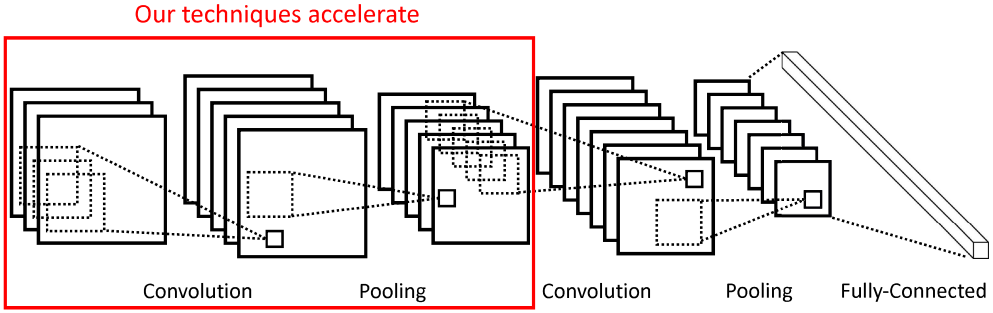
Akihiro Kasagi, Tsuguchika Tabaru
Fujitsu Labolatories

Contribution

- Present a very efficient implementation of the convolution-pooling in the GPU
  - 2 proposed methods, Fused Filter and Direct sum
    - ➢ Use 2 accelerate techniques
    - (1) Convolution interchange with direct sum
    - (2) Conversion to matrix multiplication
- Our proposed method is 9.49 times faster than the multiple convolution and pooling by cuDNN

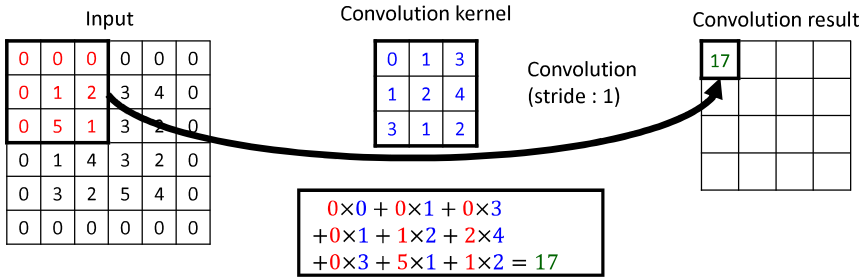
Convolutional Neural Network

Convolutional Neural Network (CNN) is commonly used for deep learning
CNN consists of some convolution operations and pooling operations



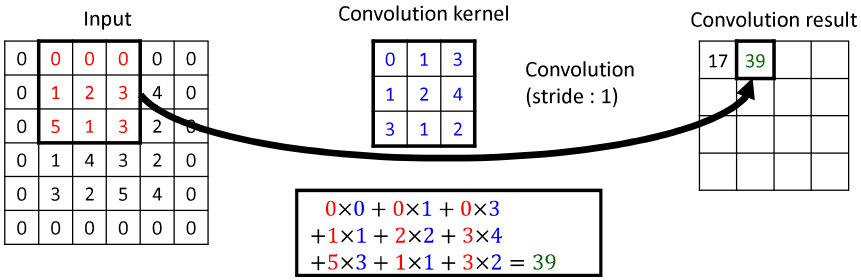
Convolution computation

Convolution is performed by the product-sum of input and kernel



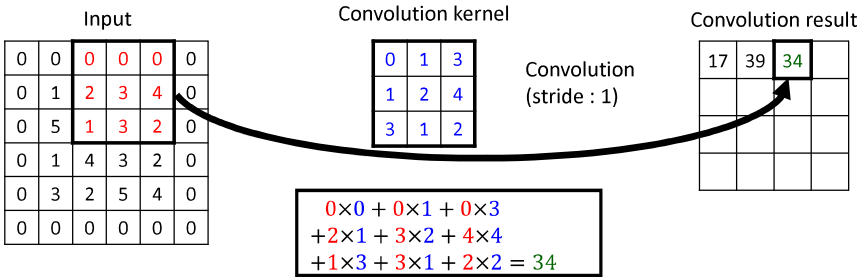
Convolution computation

Shift convolution kernel to the right and compute



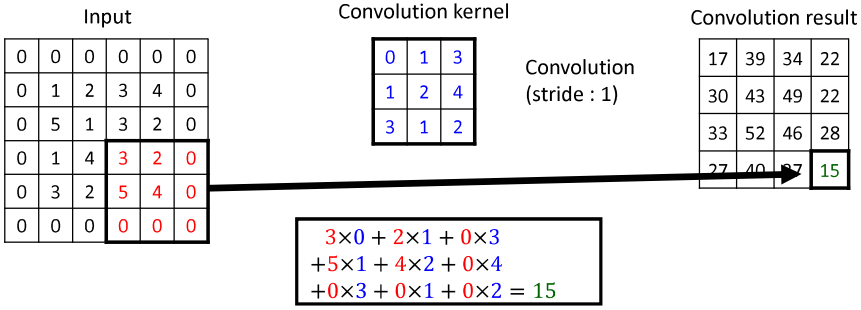
Convolution computation

Shift convolution kernel to the right and compute



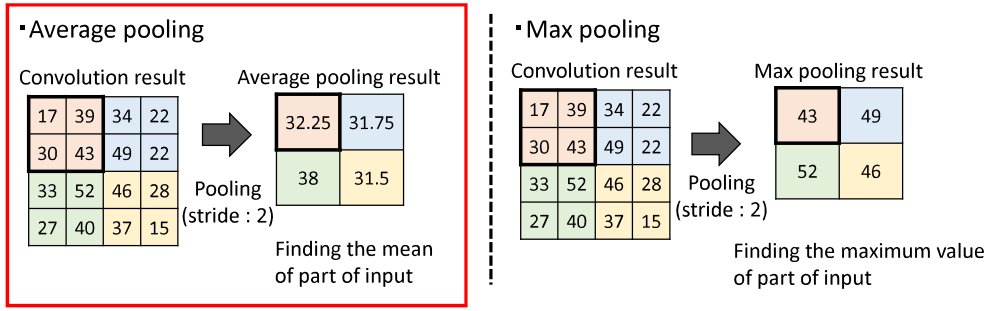
Convolution computation

Convolution is complete



Pooling computation

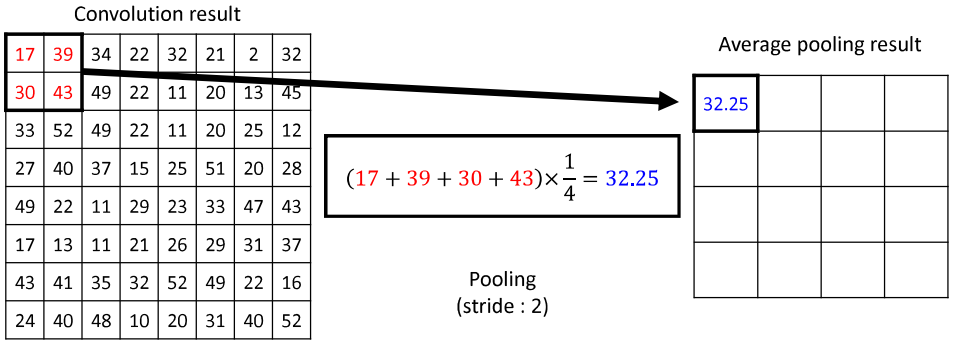
There are some pooling methods like Average pooling, Max pooling



Our techniques are used for average pooling

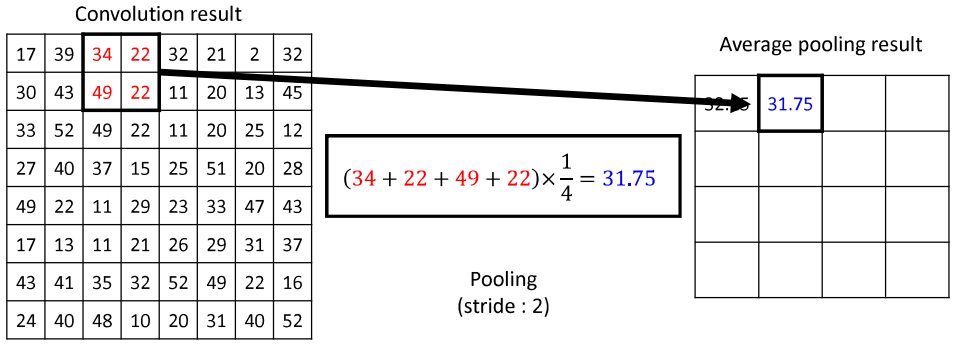
Average pooling computation

Average pooling is performed by finding the mean of part of convolution result



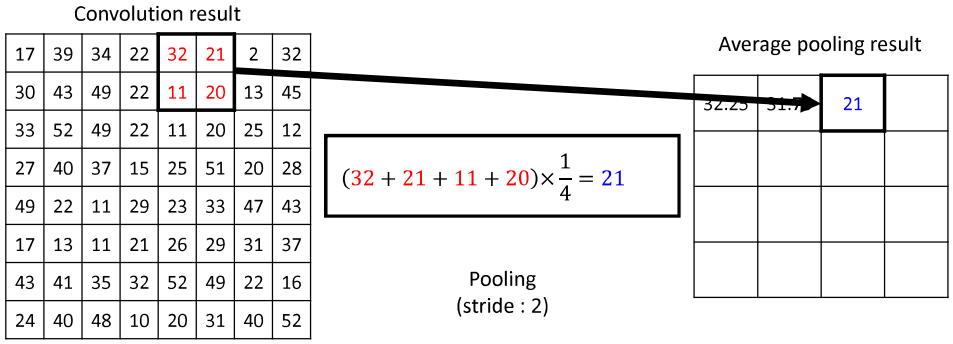
Average pooling computation

Shift pooling part 2 times to the right and compute



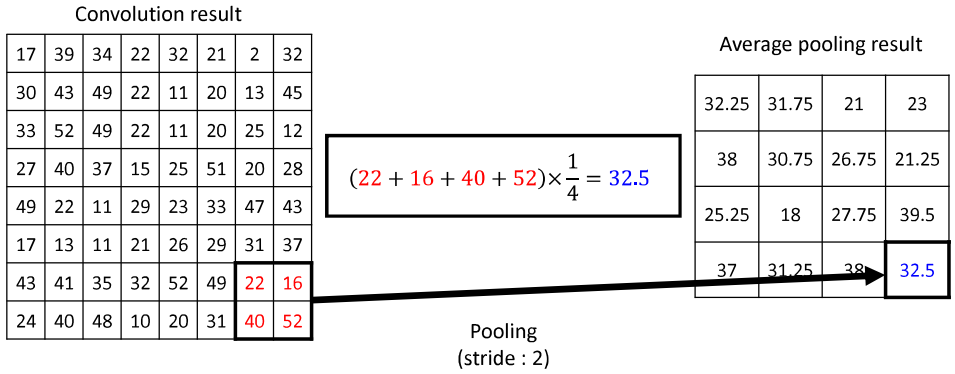
Average pooling computation

Shift pooling part 2 times to the right and compute



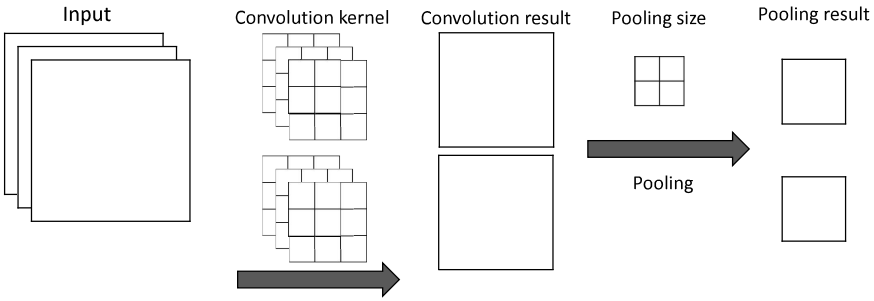
Average pooling computation

Average pooling computation is complete



Convolution and Pooling

The computational cost of the multiple convolution and pooling is high
→ By our technique, these costs are reduced

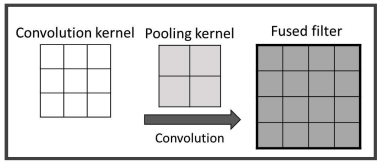


2 proposed methods

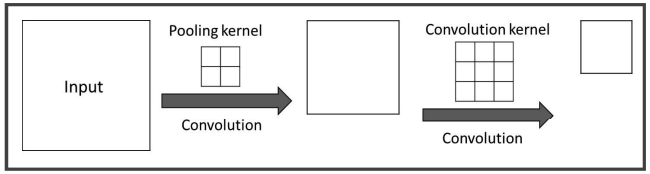
Both techniques use associativity in convolution

(1) Fused filter technique
we make fused filter from convolution kernel and pooling kernel

(2) Direct sum technique
Exchange convolution and pooling



(1) Fused filter



(2) Direct sum

Proposed methods
(1) Fused filter technique
(2) Direct sum technique

Fused filter algorithm

We can compute average pooling operation as convolution operation

Convolution result

17	39	34	22
30	43	49	22
33	52	46	28
27	40	37	15

Pooling kernel

0.25	0.25
0.25	0.25

Pooling
(stride : 2)

Average pooling result

32.25	31.75
38	31.5

Proposed methods
(1) Fused filter technique
(2) Direct sum technique

Fused filter algorithm

▪ Naive convolution and pooling

Input X

0	0	0	0	0	0
0	1	2	3	4	0
0	5	1	3	2	0
0	1	4	3	2	0
0	3	2	5	4	0
0	0	0	0	0	0

Convolution kernel W

0	1	3
1	2	4
3	1	2

Convolution
(stride : 1)

Convolution result $X * W$

17	39	34	22
30	43	49	22
33	52	46	28
27	40	37	15

Pooling kernel P

0.25	0.25
0.25	0.25

Pooling
(stride : 2)

Average pooling result $(X * W) * P$

32.25	31.75
38	31.5

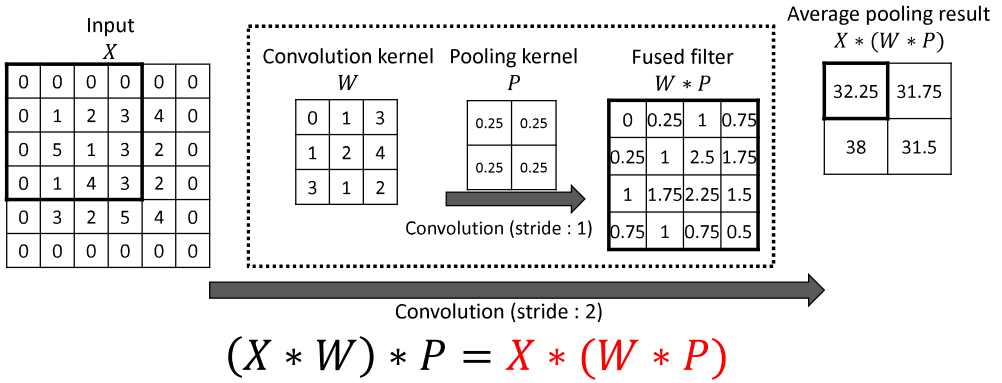
Convolution operation is associative ↔ We can compute the convolution $W * P$ first
(W: convolution kernel, P : pooling kernel)

$$(X * W) * P = X * (W * P)$$

Proposed methods
(1) Fused filter technique
(2) Direct sum technique

Fused filter algorithm

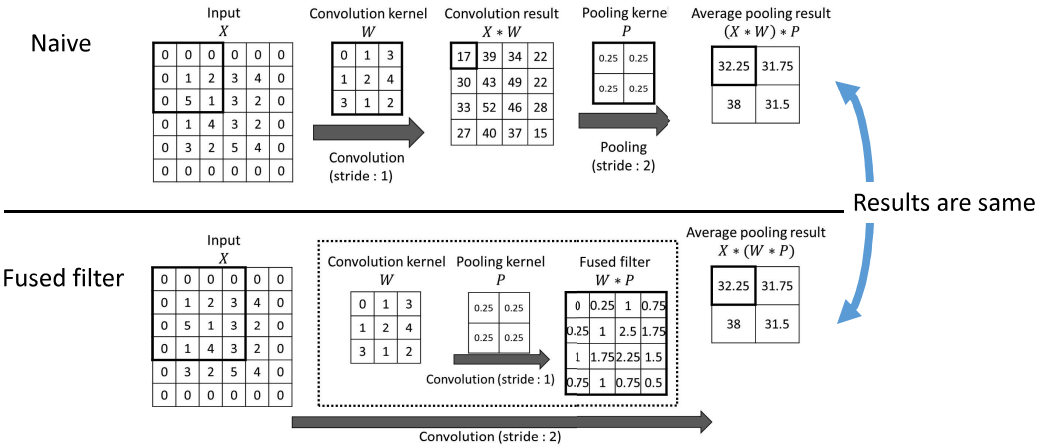
Step1 . Make Fused filter from convolution kernel and pooling kernel
Step2 . Convolution input and Fused filter



$$(X * W) * P = X * (W * P)$$

Proposed methods
(1) Fused filter technique
(2) Direct sum technique

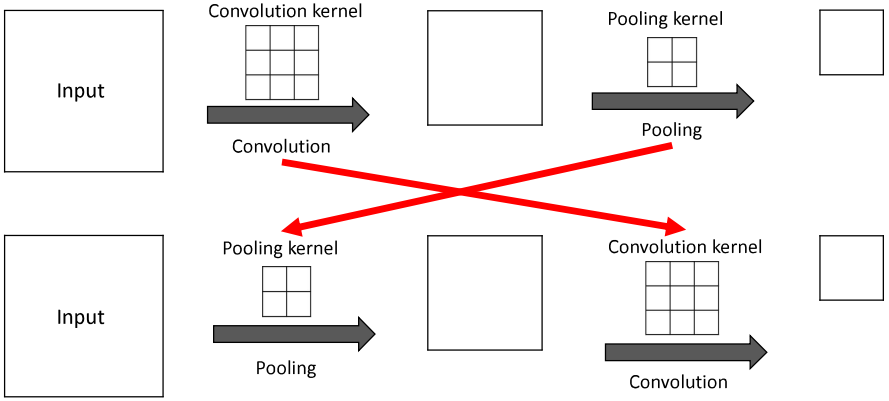
Fused filter algorithm



Proposed methods
(1) Fused filter technique
(2) Direct sum technique

Direct sum algorithm

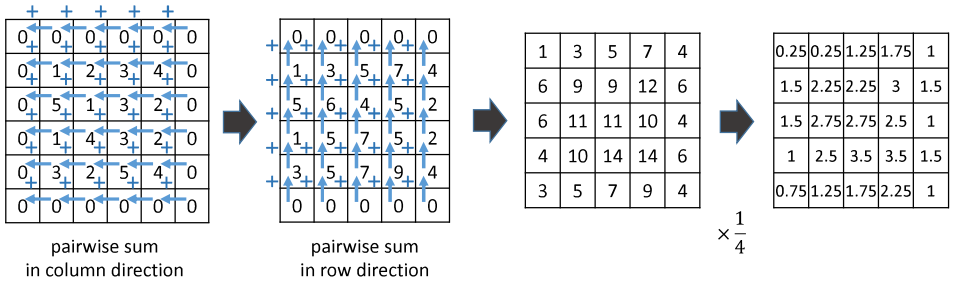
Convolution operation is associative, we can exchange convolution and pooling



Proposed methods
(1) Fused filter technique
(2) Direct sum technique

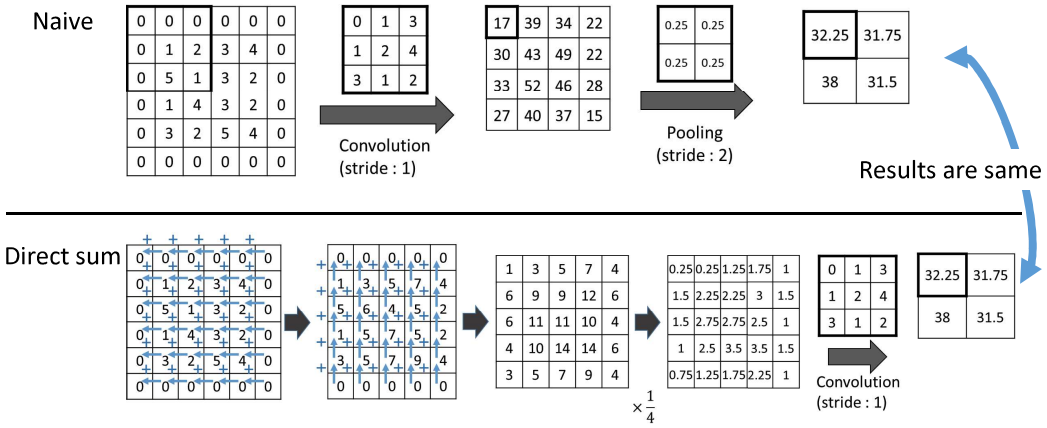
Direct sum algorithm

Average pooling operation is computed by 2 steps
Step1. Compute the pairwise sum in column direction and row direction
Step2. Multiply the result by $\frac{1}{4}$



Direct sum algorithm

Proposed methods
(1) Fused filter technique
(2) Direct sum technique

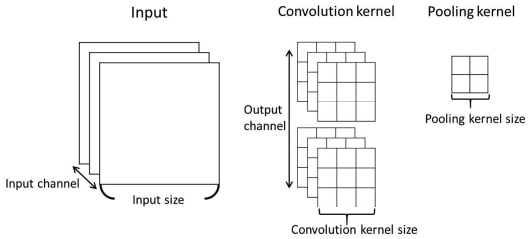


Computational cost

Parameters
Input size $n \times n$, Convolution kernel size $k \times k$,
Input channel I, Output channel R, Pooling kernel size $p \times p$

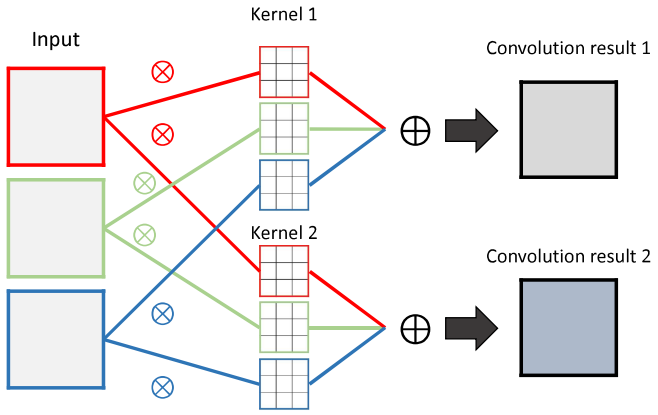
Naive	$O(n^2k^2IR)$
Fused Filter	$O(\frac{n^2(k+p)^2IR}{p^2})$
Direct Sum	$O(\frac{n^2k^2IR}{p^2} + n^2p^2I)$

About p^2 times faster



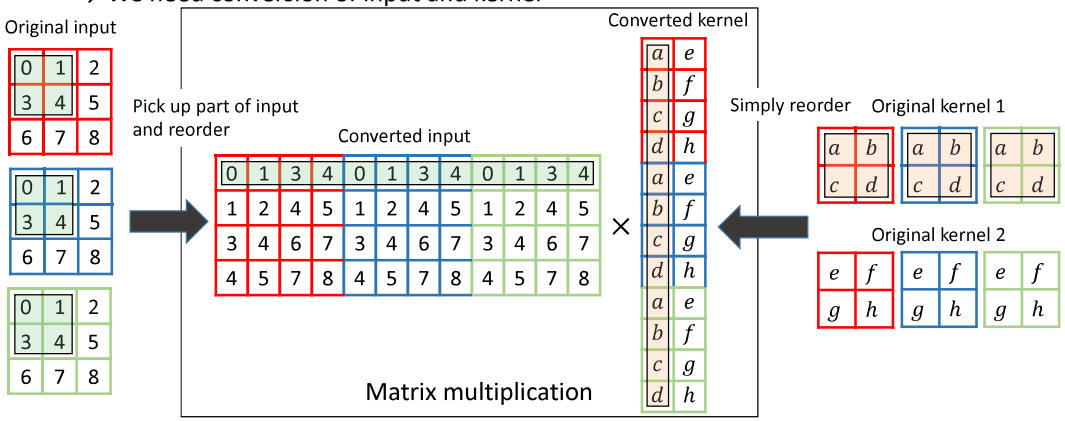
Matrix multiplication conversion for the convolution

Multiple convolution is computed by convolution between corresponding channels and sum of the results



Matrix multiplication conversion for the convolution

Convolution can be computed as a matrix multiplication
→ We need conversion of input and kernel



cuBLAS and cuDNN

These are commonly used libraries for computing convolution and pooling in DNN

cuBLAS

Linear algebra library including matrix computations
This is optimized for GPU
We can get the best performance for operations of linear algebra using cuBLAS

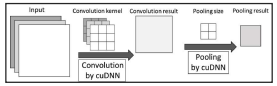
cuDNN

Popular library to implement the DNN on GPU
Provide efficient convolution and pooling
Machine learning frame work such as TensorFlow, Chainer
call cuDNN API to accelerate operations in DNN

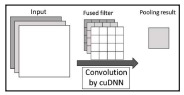
Implementation using cuDNN and cuBLAS

Five implementations

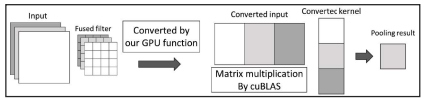
- Naive algorithm using cuDNN



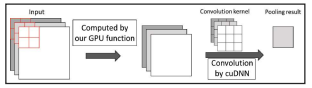
- Fused filter algorithm using cuDNN



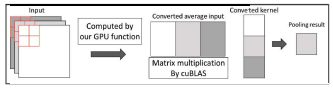
- Fused filter algorithm using cuBLAS



- Direct sum algorithm using cuDNN



- Direct sum algorithm using cuBLAS

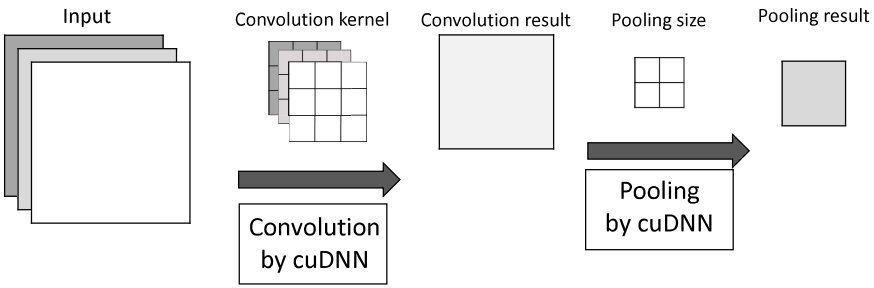


Implementation using cuDNN and cuBLAS

- cuDNN(naive)
- cuDNN(Fused filter)
- cuBLAS(Fused filter)
- cuDNN(Direct sum)
- cuBLAS(Direct sum)

Naive implementation using cuDNN

Convolution and pooling are computed by cuDNN
This is commonly used method in DNN

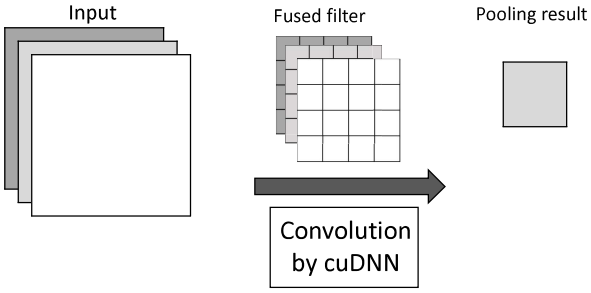


Implementation using cuDNN and cuBLAS

- cuDNN(naive)
- cuDNN(Fused filter)
- cuBLAS(Fused filter)
- cuDNN(Direct sum)
- cuBLAS(Direct sum)

Fused filter implementation using cuDNN

All fused filters are computed in advance
The convolution is computed by cuDNN



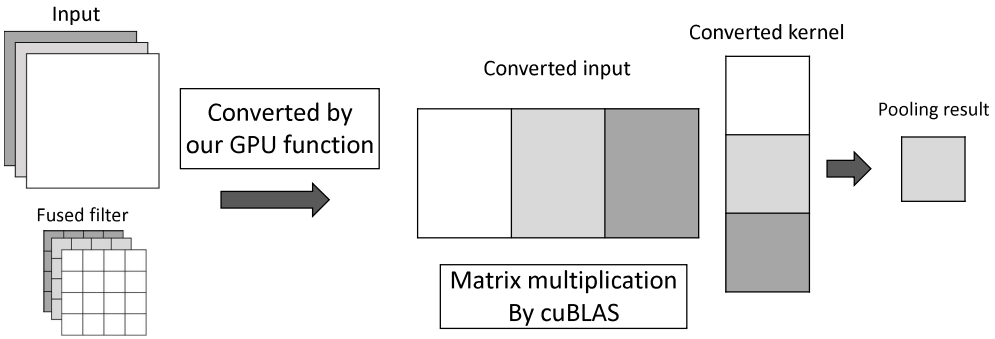
Implementation using cuDNN and cuBLAS

- cuDNN(naive)
- cuDNN(Fused filter)
- cuBLAS(Fused filter)
- cuDNN(Direct sum)
- cuBLAS(Direct sum)

Fused filter implementation using cuBLAS

All fused filters are computed in advance

Input and fused filter are converted for matrix multiplication, then computed by cuBLAS



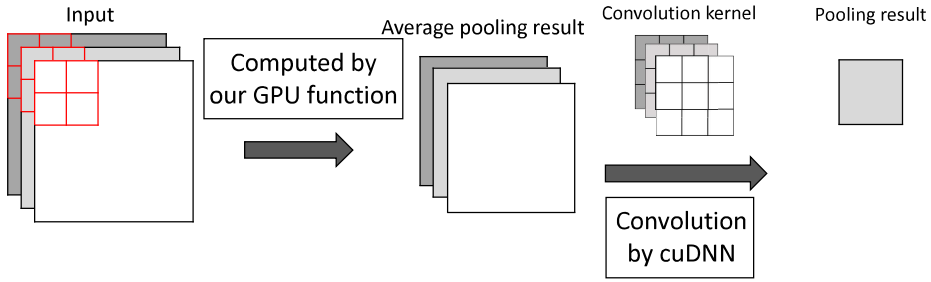
Implementation using cuDNN and cuBLAS

- cuDNN(naive)
- cuDNN(Fused filter)
- cuBLAS(Fused filter)
- cuDNN(Direct sum)
- cuBLAS(Direct sum)

Direct sum implementation using cuDNN

Compute the every 2×2 block average in input

Convolution is computed by cuDNN



Implementation using cuDNN and cuBLAS

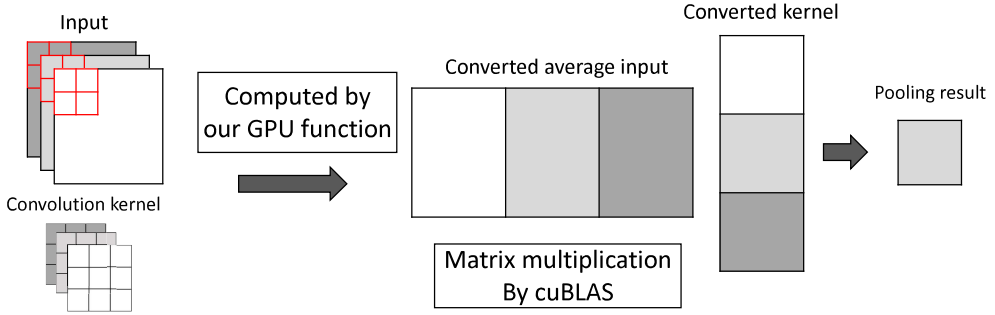
- cuDNN(naive)
- cuDNN(Fused filter)
- cuBLAS(Fused filter)
- cuDNN(Direct sum)
- cuBLAS(Direct sum)

Direct sum implementation using cuBLAS

Compute the every 2×2 block average in input

Pooling result and convolution kernel are converted for matrix multiplication

Matrix multiplication is computed by cuBLAS



Evaluation

GPU
- NVIDIA Tesla V100
- Cores: 5120
- SMs: 80
- Boost clock: 1.38GHz

cuDNN v7.1.4
cuBLAS v9.0

INPUT
- Data Type : float
- Batchsize : 64
- Input size : 8x8, 16x16, 32x32, 64x64
- (input channel, output channel) : (32,32), (64,64), (128,128), (256,256), (512, 512)

CONVOLUTION and POOLING
- Convolution kernel size 3x3
- Pooling kernel size 2x2

Evaluation

GPU
- NVIDIA Tesla V100 — cuDNN v7.1.4
- Cores: 5120 — cuBLAS v9.0
- SMs: 80
- Boost clock: 1.38GHz — Data Type : float

Execution time comparison

Input size of 8x8 [s]

(input channel, output channel)	(32, 32)	(64, 64)	(128, 128)	(256, 256)	(512, 512)
cuDNN(naive)	0.104	0.137	0.206	0.651	1.87
cuDNN(Fused Filter)	0.105	0.153	0.259	0.425	0.961
cuBLAS(Fused Filter)	0.103	0.141	0.281	0.929	3.17
cuDNN(Direct Sum)	0.107	0.133	0.340	0.341	0.701
cuBLAS(Direct Sum)	0.0473	0.0739	0.154	0.479	1.92

Input size of 16x16 [s]

(input channel, output channel)	(32, 32)	(64, 64)	(128, 128)	(256, 256)	(512, 512)
cuDNN(naive)	0.186	0.262	0.663	1.87	6.51
cuDNN(Fused Filter)	0.102	0.155	0.299	0.824	3.18
cuBLAS(Fused Filter)	0.110	0.174	0.335	0.922	3.34
cuDNN(Direct Sum)	0.112	0.146	0.342	0.616	1.92
cuBLAS(Direct Sum)	0.0488	0.101	0.192	0.577	1.89

Our Direct sum is faster than naive and fused filter

Evaluation

GPU
- NVIDIA Tesla V100 — cuDNN v7.1.4
- Cores: 5120 — cuBLAS v9.0
- SMs: 80
- Boost clock: 1.38GHz — Data Type : float

Execution time comparison

Input size of 32x32 [s]

(input channel, output channel)	(32, 32)	(64, 64)	(128, 128)	(256, 256)	(512, 512)
cuDNN(naive)	0.247	0.42	2.03	5.98	20.9
cuDNN(Fused Filter)	0.156	0.254	0.836	2.82	10.7
cuBLAS(Fused Filter)	0.294	0.538	1.05	3.3	12.5
cuDNN(Direct Sum)	0.163	0.255	0.626	1.85	7.25
cuBLAS(Direct Sum)	0.164	0.315	0.610	1.85	7.02

Input size of 64x64 [s]

(input channel, output channel)	(32, 32)	(64, 64)	(128, 128)	(256, 256)	(512, 512)
cuDNN(naive)	0.936	1.85	20.6	13.9	47
cuDNN(Fused Filter)	0.327	1.19	4.03	13.5	49.3
cuBLAS(Fused Filter)	0.989	1.96	4.06	12.8	44.3
cuDNN(Direct Sum)	0.355	0.685	2.17	7.03	24.2
cuBLAS(Direct Sum)	0.548	1.12	2.28	7.01	25.0

In some parameters, Fused filter is faster than Direct sum

Conclusion

- Proposed the efficient implementation of the convolution-pooling in the GPU
- Our proposed method is 9.49 times faster than the multiple convolution and pooling by cuDNN
  - Use 2 acceleration techniques

  (1) convolution interchange with direct sum

  (2) conversion to matrix multiplication

An asynchronous P system with branch and bound for graph coloring

Kotaro Umetsu Akihiro Fujiwara
Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502, Japan

Abstract—Membrane computing, which is a computational model based on cell activity, has attracted considerable attention as a new paradigm for computation. In general membrane computing, computationally hard problems have been solved in a polynomial number of steps using an exponential number of membranes. However, reduction of the number of membranes must be considered in order to make the P system a more realistic model.

In the present paper, we propose an asynchronous P system with branch and bound, which is a well-known optimization technique, in order to reduce the number of membranes. The proposed P system finds the minimum graph coloring for a graph with n vertices and works in $O(n^n)$ sequential steps or $O(n^2)$ parallel steps.

In addition, the number of membranes used in the proposed P system is evaluated using a computational simulation. The experimental results demonstrate the validity and efficiency of the proposed P system.

Index Terms—membrane computing, graph coloring, branch and bound

I. Introduction

A number of next-generation computing paradigms have been considered due to the limitations of silicon-based computational hardware. As an example of a computing paradigm, natural computing, which works using natural materials for computation, has attracted considerable attention. Membrane computing, which is representative of natural computing, is a computational model inspired by the structures and behaviors of living cells.

A basic feature of membrane computing was introduced in [10] as a P system. The P system consists mainly of membranes and objects. A membrane is a computing cell, in which independent computation is executed, and may contain objects and other membranes. Each object evolves according to the evolution rules associated with the membrane in which the object is contained.

The P system and most variants have been proved to be universal [12], and several P systems have been proposed for solving computationally hard problems [2], [3], [6], [7], [9], [11], [13]–[16], [19].

In addition, asynchronous parallelism has been considered on the P system. Asynchronous parallelism means that all objects may react to rules with different speeds, and evolution rules are applied to objects independently. Since all objects in a living cell basically work in an asynchronous manner, asynchronous parallelism makes a P system a more realistic computational model.

A number of asynchronous P systems have been proposed for the computationally hard problems [1], [4], [8], [17], [18]. For example, an asynchronous P system has been proposed for finding the minimum graph coloring [18]. The P system finds the minimum graph coloring for a graph with n vertices in $O(n^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^2)$ kinds of objects.

In all of the above P systems, computationally hard problems have been solved in a polynomial number of steps using an exponential number of membranes. The number of membranes is the number of living cells, and reduction of the number of membranes must be considered for the case in which the P system is implemented using living cells because living cells cannot be created exponentially.

Recently, an asynchronous P system using branch and bound has been proposed [5] for reducing the number of membranes. Branch and bound is a well-known optimization technique, and is used in the P system for omitting partial value assignments that cannot satisfy a given Boolean formula.

In the present paper, we propose an asynchronous P system for solving the minimum graph coloring problem with branch and bound. In the proposed P system, objects, which denote vertices, are colored one by one, and the adjacent vertices are checked for color. If the adjacent vertices have the same color, the partial color assignment of vertices is discarded as a bounding operation. Since the number of membranes increases according to the number of color assignments of vertices, the number can be reduced by omitting partial assignments that are not possible for the graph coloring. We show that the proposed P system finds the minimum graph coloring for a graph with n vertices and works in $O(n^n)$ sequential steps or $O(n^2)$ parallel steps using $O(n^2)$ kinds of objects.

In addition, the validity of the proposed system is evaluated through a computational simulation. In the simulation, various instances are executed on the previous P system [18] and the proposed P system, and the number of membranes are compared for the same instances. The experimental results demonstrate the validity and efficiency of the proposed P system.

The remainder of the present paper is organized as

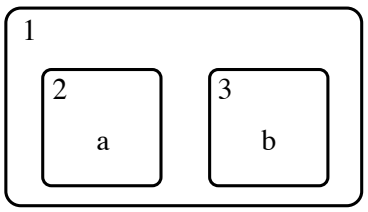


Fig. 1. Example of a membrane structure

follows. In Section 2, we give a brief description of the model for asynchronous membrane computing. In Section 3, we propose the P system with branch and bound for the minimum graph coloring, and experimental results for the proposed P system are shown in Section 4. Section 5 concludes the paper.

II. Preliminaries

A. Computational model for membrane computing

Several models have been proposed for membrane computing. We briefly introduce a basic model of the P system in this subsection.

The P system consists mainly of membranes and objects. A membrane is a computing cell, in which independent computation is executed, and may contain objects and other membranes. In other words, the membranes form nested structures. In the present paper, each membrane is denoted by a pair of square brackets, and the number on the right-hand side of each right-hand bracket denotes the label of the corresponding membrane. An object in the P system is a memory cell, in which each data is stored and can divide, dissolve, and pass through membranes. In the present paper, each object is denoted by finite strings over a given alphabet and is contained in one of the membranes.

For example, $[[a]_2[b]_3]_1$ and Fig. 1 denote the same membrane structure, which consists of three membranes. The membrane labeled 1 contains two membranes labeled 2 and 3, which contain objects a and b, respectively.

Computation of P systems is executed according to evolution rules, which are defined as rewriting rules for membranes and objects. All objects and membranes are transformed in parallel according to applicable evolution rules. If no evolution rule is applicable for objects, the system ceases computation.

We now formally define a P system and the sets used in the system as follows.

$$\Pi = (O, \mu, \omega_1, \omega_2, \cdots, \omega_m . R_1, R_2, \cdots, R_m)$$

- O: O is the set of all objects used in the system.
- μ: μ is a membrane structure that consists of m membranes. Each membrane in the structure is labeled with an integer.
- ω_i: Each ω_i is a set of objects initially contained only in the membrane labeled i.
- R_i: Each R_i is a set of evolution rules that are applicable to objects in the membrane labeled i.

In the present paper, we assume that input objects are given from the outside region to the outermost membrane, and computation is started by applying evolution rules. We also assume that output objects are sent from the outermost membrane to the outside region.

In membrane computing, several types of rules are proposed. In the present paper, we consider five basic rules of the following forms.

(1) Object evolution rule:

$$[\ a\ ]_h \to [\ b\ ]_h.$$

In the above rule, h is a label of the membrane and $a, b \in O$. Using the rule, an object a evolves into another object b. (We omit the brackets in each evolution rule, e.g., $a \to b$, for cases for which a corresponding membrane is obvious.)

(2) Send-in communication rule:

$$a[\ ]_h \to [\ b\ ]_h.$$

In the above rule, h is a label of the membrane, and $a, b \in O$. Using this rule, an object a is sent into the membrane and can evolve into another object b.

(3) Send-out communication rule:

$$[\ a\ ]_h \to [\ ]_h b.$$

In the above rule, h is a label of the membrane, and $a, b \in O$. Using this rule, an object a is sent out of the membrane and can evolve into another object b.

(4) Dissolution rule:

$$[\ a\ ]_h \to b$$

In the above rule, h is a label of the membrane, and $a, b \in O$. Using this rule, the membrane, which contains an object a, is dissolved, and the object can evolve into another object b. (The outermost membrane cannot be dissolved.)

(5) Division rule:

$$[\ a\ ]_h \to [\ b\ ]_h [\ c\ ]_h$$

In the above rule, h is a label of the membrane, and $a, b, c \in O$. Using this rule, the membrane, which contains an object a, is divided into two membranes that contain objects b and c, respectively.

We assume that each of the above rules is applied in a constant number of biological steps. In the following sections, we consider the number of steps executed in a P system as the complexity of the P system.

B. Maximal parallelism and asynchronous parallelism

In the standard model in membrane computing, which is a P system with maximal parallelism, all of the above rules are applied in a non-deterministic maximally parallel manner. In one step of computation of the P system, each object is evolved according to one of the applicable rules. (In cases that there are several possibilities, one of the applicable rules is non-deterministically chosen.) All objects for which no rules are applicable remain unchanged until the next step. In other words, all applicable rules are applied in parallel in each step of computation.

On the other hand, evolution rules are applied in a fully asynchronous manner on the asynchronous P system [17], and any number of applicable evolution rules are applied in each step of computation. In other words, the asynchronous P system can be executed sequentially and can also be executed in a maximally parallel manner.

We consider asynchronous parallelism in the present paper, based on the fact that every living cell acts independently and asynchronously. Since the standard P system ignores the asynchronous feature of living cells, the asynchronous P system is a more realistic computation model for cell activities.

In the asynchronous P system, all evolution rules can be applied completely in parallel, which is the same as the conventional P system, or all evolution rules can be applied sequentially. We define the number of steps executed in the asynchronous P system in the maximally parallel manner as the number of parallel steps. We also define the number of steps for the case in which the applicable evolution rules are applied sequentially as the number of sequential steps. The numbers of parallel and sequential steps indicate the best- and worst-case complexities for the asynchronous P system. In addition, the proposed asynchronous P system must be guaranteed to output a correct solution in any asynchronous execution.

III. An asynchronous P system with branch and bound for minimum graph coloring

In this section, we present an asynchronous P system with branch and bound for minimum graph coloring. We first explain an input and an output of the problem for the system and then present an outline and details of the P system. Finally, we discuss the complexity of the proposed P system.

A. Input and output for minimum graph coloring

The minimum graph coloring problem is a computationally hard graph problem. Given an undirected graph $G = (V, E)$, a coloring is an assignment of colors to all vertices such that no pair of vertices with the same color are bridged by an edge $e \in E$. For example, given the graph in Fig. 2, an assignment of colors, such that $c_1 = \{v_1\}, c_2 = \{v_2\}, c_3 = \{v_3, v_4\}$, is a coloring for the graph. The minimum graph coloring problem involves finding an assignment of colors for an input graph with the minimum number of colors.

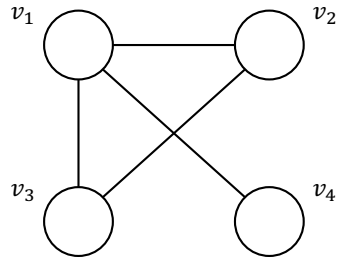


Fig. 2. Example of an input undirected graph

The input of the minimum graph coloring is the following set of objects O_E:

$$O_E \quad = \quad \{\langle e_{i,j}, W\rangle \mid 1 \le i \le n, 1 \le j \le n, W \in \{T, F\}\}$$

Each object $\langle e_{i,j}, W\rangle$ denotes an edge (v_i, v_j). If an edge (v_i, v_j) exists in the input graph, then W is set to T; otherwise, W is set to F.

For example, the following objects denote an input of the minimum graph coloring for the graph in Fig. 2:

$$\begin{array}{l}\langle e_{1,1}, F\rangle, \langle e_{1,2}, T\rangle, \langle e_{1,3}, T\rangle, \langle e_{1,4}, T\rangle,\\ \langle e_{2,1}, T\rangle, \langle e_{2,2}, F\rangle, \langle e_{2,3}, T\rangle, \langle e_{2,4}, F\rangle,\\ \langle e_{3,1}, T\rangle, \langle e_{3,2}, T\rangle, \langle e_{3,3}, F\rangle, \langle e_{3,4}, F\rangle,\\ \langle e_{4,1}, T\rangle, \langle e_{4,2}, F\rangle, \langle e_{4,3}, F\rangle, \langle e_{4,4}, F\rangle\end{array}$$

The output of the P system is denoted by the following set of n objects:

$$O_C \quad = \quad \{\langle V_i, h\rangle \mid 1 \le i \le n, 1 \le h \le n\}$$

Each object $\langle V_i, h\rangle$ means that vertex v_i is labeled with color h, and O_C denotes the minimum color assignment for the input graph.

For example, the following set of objects is an output of the minimum graph coloring for the graph in Fig. 2:

$$O_C \quad = \quad \{\langle V_1, 1\rangle, \langle V_2, 2\rangle, \langle V_3, 3\rangle, \langle V_4, 3\rangle\}$$

We assume that the set of input objects is given from the outside region to the skin membrane, and the output object is sent from the skin membrane to the outside region.

B. Branch and bound for minimum graph coloring

Branch and bound is a well-known computing paradigm for the optimization problem. On the existing P system for solving minimum graph coloring [18], all assignments of colors for vertices are created for an input graph with n vertices, and n^n assignments are checked as to whether each assignment of colors is valid. However, a partial assignment of vertices can be discarded if the valid assignment of vertices is determined.

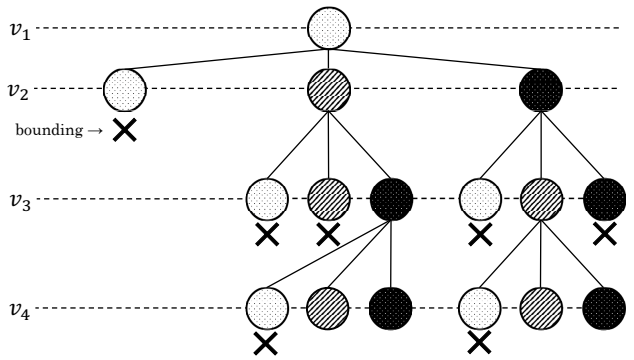


Fig. 3. Example of branch and bound for minimum graph coloring

Fig. 3 shows an example of a search tree for the above concept. Let the graph in Fig. 2 be an input graph. We consider a 3-coloring for the graph. In the search tree, a color assignment of vertices is denoted by a path from a root node to a leaf node. In this case, a path starting from v_1, which is colored with C_1, to v_2, which is also colored with C_1, can be bounded because there is an edge between v_1 and v_2 in the input graph.

We now explain an overview of the asynchronous P system with branch and bound for finding the minimum graph coloring. An initial membrane structure for the computation is $[\ [\ ]_1\ [\ ]_2\ \cdots\ [\ ]_n\ ]_0$. We refer to the membrane labeled 0 as the outer membrane and refer to the membranes labeled 1 through n as inner membranes.

The computation of the P system mainly consists of the following four steps.

Step 1: Move modified copies of input objects into all inner membranes.

Step 2: In each inner membrane labeled h, create an assignment of colors with h colors by dividing the inner membranes. Then, check whether adjacent vertices are colored with different colors with branch and bound. If adjacent vertices are colored with the same color in the assignment, stop assignment for vertices.

Step 3: Check whether an assignment of colors is completed for all vertices in each divided inner membrane and then output an object that denotes successful assignment of colors to the outer membrane without dissolving the inner membrane. Otherwise, dissolve the inner membrane and output an object that denotes failure of the coloring assignment. Then, determine the minimum number of colors in successful assignments in the outer membrane.

Step 4: Send out the minimum number of colors from the outer membrane.

C. Details of the proposed P system

We explain details of each step of the computation for minimum graph coloring. In Step 1, modified copies of the input objects are moved into all inner membranes. Each input object is modified so as to contain the label of a stored membrane.

Since the P system considered in the present paper is asynchronous, we cannot move the input objects in parallel, and input objects are moved one by one applying the following sets of evolution rules. (In the following description, a set of evolution rules $R_{i,j}$ indicates that the rules are used for membrane i in Step j.)

(Evolution rules for the outer membrane)

$$
\begin{aligned}
R_{0,1} \;=\; & \{\langle e_{i,j}, W\rangle \to \langle e_{i,j}, W, 1\rangle\langle f_{i,j}, W, 1\rangle \\
& \quad \mid 1 \le i \le n, 1 \le j \le n, W \in \{F, T\}\} \\
& \cup \{\langle f_{i,j}, W, h\rangle \to \langle e_{i,j}, W, h+1\rangle\langle f_{i,j}, W, h+1\rangle \\
& \quad \mid 1 \le i \le n, 1 \le j \le n, 1 \le h \le n-2, \\
& \quad W \in \{F, T\}\} \\
& \cup \{\langle f_{i,j}, W, n-1\rangle \to \langle e_{i,j}, W, n\rangle \\
& \quad \mid 1 \le i \le n, 1 \le j \le n, W \in \{F, T\}\} \\
& \cup \{\langle e_{1,1}, F, h\rangle[]_h \to [\langle M_{2,1}, h\rangle\langle e_{1,1}, F, h\rangle]_h \\
& \quad \mid 1 \le h \le n\} \\
& \cup \{\langle M_{i,j}, h\rangle\langle e_{i,j}, W, h\rangle[]_h \\
& \quad \to [\langle M_{i+1,j}, h\rangle\langle e_{i,j}, W, h\rangle]_h \mid 1 \le i \le n, \\
& \qquad 1 \le j \le n, 1 \le h \le n, W \in \{F, T\}\} \\
& \cup \{\langle M_{n+1,j}, h\rangle \to \langle M_{1,j+1}, h\rangle \mid 1 \le j \le n, \\
& \quad 1 \le h \le n\} \\
& \cup \{\langle M_{1,n+1}, h\rangle[]_h \to [\langle S_1, h\rangle]_h \mid 1 \le h \le n\}
\end{aligned}
$$

(Evolution rules for inner membrane h $(1 \le h \le n)$)

$$
\begin{aligned}
R_{h,1} \;=\; & \{[\langle M_{i,j}, h\rangle]_h \to []_h\langle M_{i,j}, h\rangle \\
& \quad \mid 1 \le i \le n+1, 1 \le j \le n+1\}
\end{aligned}
$$

In the above evolution rules, two kinds of objects, $\langle e_{i,j}, W, h\rangle$ and $\langle f_{i,j}, W, h\rangle$, are created from input object $\langle e_{i,j}, W\rangle$. The first objects, $\langle e_{i,j}, W, h\rangle$, are moved into a membrane labeled h using objects $\langle M_{i,j}, h\rangle$. At the end of Step 1, object $\langle S_1, h\rangle$ is created in the membrane labeled h, and the object triggers the computation of Step 2.

In Step 2, in each inner membrane labeled h, assignments of colors are created with h colors by dividing the inner membranes. Then, the partial assignment is checked as to whether adjacent vertices are colored with different colors using the bounding condition. If the adjacent vertices are colored with the same color, the membrane stops the partial assignment. The above step is executed by applying the following set of evolution rules.

(Evolution rules for inner membrane h $(1 \le h \le n)$)

$$
\begin{aligned}
R_{h,2} \;=\;& \{\langle S_1,h\rangle \to \langle S_2,h\rangle\langle V_1,h\rangle\} \\
& \cup\{[\langle S_i,k\rangle]_h \to [\langle L_{i,1}\rangle\langle V_i,k\rangle]_h[\langle S_i,k-1\rangle]_h \\
& \quad \mid 2\le i\le n, 2\le k\le h\} \\
& \cup\{\langle S_i,1\rangle \to \langle L_{i,1}\rangle\langle V_i,1\rangle \mid 2\le i\le n\} \\
& \cup\{\langle L_{i,j}\rangle\langle V_i,a\rangle\langle V_j,b\rangle\langle e_{i,j},F,h\rangle\langle e_{j,i},F,h\rangle \\
& \quad \to \langle L_{i,j+1}\rangle\langle V_i,a\rangle\langle V_j,b\rangle \\
& \quad \mid 2\le i\le n, 1\le j\le i, 1\le a,b\le h\} \\
& \cup\{\langle L_{i,j}\rangle\langle V_i,a\rangle\langle V_j,b\rangle\langle e_{i,j},T,h\rangle\langle e_{j,i},T,h\rangle \\
& \quad \to \langle L_{i,j+1}\rangle\langle V_i,a\rangle\langle V_j,b\rangle \\
& \quad \mid 2\le i\le n, 1\le j\le i, 1\le a,b\le h, a\ne b\} \\
& \cup\{\langle L_{i,j}\rangle\langle V_i,a\rangle\langle V_j,a\rangle\langle e_{i,j},T,h\rangle\langle e_{j,i},T,h\rangle \\
& \quad \to \langle Z_h,n-i\rangle\langle G_{i-1}\rangle\langle V_j,a\rangle \\
& \quad \mid 2\le i\le n, 1\le j\le i, 1\le a\le h\} \\
& \cup\{\langle L_{i,i}\rangle \to \langle S_{i+1},h\rangle \mid 2\le i\le n\}
\end{aligned}
$$

In the above evolution rules, vertex v_i is colored with color k in the second set of rules, and object $\langle V_i,k\rangle$ denotes the color. Then, the vertices of v_i $(1\le j\le i)$ are checked for v_i using object $\langle L_{i,j}\rangle$. If v_i and v_j are not adjacent or their colors are different, then object $\langle L_{i,j+1}\rangle$ is created. After all checked vertices are passed for v_i, object $\langle L_{i,i}\rangle$ is created, and object $\langle S_{i,j+1},h\rangle$, which is an object for checking the next vertex v_{i+1}, is created.

On the other hand, for the case in which v_i and v_j are adjacent and are colored with the same color, objects $\langle Z_h,n-i\rangle$ and $\langle G_{i-1}\rangle$, which denote a failure of coloring, are created. The object $\langle Z_h,n-i\rangle$ denotes that $n-i$ colors remain in the case of coloring failure, and $\langle G_{i-1}\rangle$ is an object used in Step 3 for deleting partial assignment of colors in the membrane. In addition, the two objects trigger the computation of Step 3.

In Step 3, the assignment in each membrane is checked as to whether colors are assigned for all vertices. For the case in which an assignment of colors is completed for all vertices, an object that denotes the success of the assignment of colors is sent to the outer membrane without dissolving the inner membrane. Otherwise, the inner membrane is dissolved, and an object that denotes a failure of coloring assignment is output to the outer membrane. Then, the minimum number of colors in successful assignments is determined in the outer membrane. Step 3 is executed by applying the following sets of evolution rules.

(Evolution rules for the outer membrane)

$$
\begin{aligned}
R_{0,3} \;=\;& \{\langle S_{n+1},h\rangle \to \langle S_{n+1},h,0\rangle \mid 1\le h\le n\} \\
& \cup\{\langle S_{n+1},h,k\rangle^h \to \langle S_{n+1},h,k+1\rangle \\
& \quad \mid 1\le h\le n, 0\le k\le n-2\} \\
& \cup\{\langle Z_h,k\rangle^h \to \langle Z_h,k+1\rangle \\
& \quad \mid 1\le h\le n, 0\le k\le n-2\} \\
& \cup\{\langle S_{n+1},h,k\rangle^{h-d}\langle Z_h,k\rangle^d \to \langle S_{n+1},h,k+1\rangle \\
& \quad \mid 1\le h\le n, 0\le k\le n-2, 1\le d\le h-1\} \\
& \cup\{\langle S_{n+1},h,n-1\rangle\langle Z_{h-1},n-1\rangle \to \langle A_h\rangle \\
& \quad \mid 1\le h\le n\}
\end{aligned}
$$

(Evolution rules for inner membrane h $(1\le h\le n)$)

$$
\begin{aligned}
R_{h,3} \;=\;& \{[\langle S_{n+1},h\rangle]_h \to []_h\langle S_{n+1},h\rangle\} \\
& \cup\{\langle V_i,k\rangle\langle G_i\rangle \to \langle G_{i-1}\rangle \\
& \quad \mid 2\le i\le n, 1\le k\le h\} \\
& \cup\{[\langle V_1,h\rangle\langle G_1\rangle]_h \to \langle G_0\rangle\}
\end{aligned}
$$

In Step 3, $R_{h,3}$ is applied in each divided membrane in Step 2. For the case in which the object that denotes success of the assignment of colors, $\langle S_{n+1},h\rangle$, is in the membrane, the object is sent to the outer membrane without dissolving the membrane. On the other hand, for the case in which the object that denotes failure of the assignment of colors, $\langle G_i\rangle$, is in the membrane, the membrane is dissolved using $\langle G_i\rangle$, and objects $\langle S_{n+1},h,k\rangle$ and $\langle Z_h,k\rangle$ are in the outer membrane.

The object $\langle S_{n+1},h,k\rangle$ denotes that there are h^k successful assignments of colors with h colors, and the object $\langle Z_h,k\rangle$ denotes that there are h^k failure assignments of colors with h colors. Then, if objects $\langle S_{n+1},h,n-1\rangle$ and $\langle Z_{h-1},n-1\rangle$ exist simultaneously in the outer membrane, coloring with h colors is successful, and coloring with $h-1$ colors has failed. Therefore, the object $\langle A_h\rangle$, which indicates that the minimum number for assignment of colors is h, is created in the outer membrane, and the object triggers the computation of Step 4.

In Step 4, one of the assignments that denote the minimum coloring is sent from the outer membrane. Step 4 is executed by applying the following sets of evolution rules.

(Evolution rules for the outer membrane)

$$
\begin{aligned}
R_{0,4} \;=\;& \{\langle B_i\rangle \to \langle C_i\rangle\langle B_{i+1}\rangle | 1\le i\le n-1\} \\
& \cup\{\langle B_n\rangle \to \langle C_n\rangle\} \\
& \cup\{[\langle C_i\rangle\langle V_i,h\rangle]_0 \to []_0\langle V_i,h\rangle | 1\le i\le n, 1\le h\le n\}
\end{aligned}
$$

(Evolution rules for inner membrane h $(1 \leq h \leq n)$)

$$\begin{aligned} R_{h,4} &= \{\langle A_h\rangle[]_h \rightarrow [\langle A_h\rangle]_h\} \\ &\quad \cup\{[\langle A_h\rangle]_h \rightarrow []_h\langle B_1\rangle\} \end{aligned}$$

The objects $\langle V_i, h\rangle$ are sent out from the outer membrane, and the computation is then finished.

We now summarize the asynchronous P system $\Pi_{\text{MIN\_COL}}$ for finding the minimum graph coloring as follows:

$$\Pi_{\text{MIN\_COL}} = (O, \mu, \omega_1, \omega_2, R_0, R_1, \cdots, R_n)$$

$$\begin{aligned} O &= \{\langle e_{i,j}, W\rangle | 1 \leq i \leq n, 1 \leq j \leq n, W \in \{F, T\}\} \\ &\quad \cup\{\langle e_{i,j}, W, h\rangle \\ &\qquad | 1 \leq i \leq n, 1 \leq j \leq n, 1 \leq h \leq n, W \in \{F, T\}\} \\ &\quad \cup\{\langle f_{i,j}, W, h\rangle \\ &\qquad | 1 \leq i \leq n, 1 \leq j \leq n, 1 \leq h \leq n, W \in \{F, T\}\} \\ &\quad \cup\{\langle M_{i,j}, h\rangle | 1 \leq i \leq n, 1 \leq j \leq n, 1 \leq h \leq n\} \\ &\quad \cup\{\langle S_i, h\rangle | 1 \leq i \leq n+1, 1 \leq h \leq n\} \\ &\quad \cup\{\langle V_i, h\rangle | 1 \leq i \leq n, 1 \leq h \leq n\} \\ &\quad \cup\{\langle L_{i,j}\rangle | 1 \leq i \leq n, 1 \leq j \leq n\} \\ &\quad \cup\{\langle Z_h, k\rangle | 1 \leq h \leq n, 0 \leq k \leq n-2\} \\ &\quad \cup\{\langle G_i\rangle | 0 \leq i \leq n-1\} \\ &\quad \cup\{\langle S_{n+1}, h, k\rangle | 1 \leq h \leq n, 0 \leq k \leq n-1\} \\ &\quad \cup\{\langle A_h\rangle | 1 \leq h \leq n\} \\ &\quad \cup\{\langle B_i\rangle | 1 \leq i \leq n\} \\ &\quad \cup\{\langle C_i\rangle | 1 \leq i \leq n\} \\ \mu &= [\ [\ ]_1\ [\ ]_2\ \cdots\ [\ ]_n\ ]_0 \\ \omega_1 &= \omega_2 = \cdots = \omega_n = \phi \\ R_0 &= R_{0,1} \cup R_{0,3} \cup R_{0,4} \\ R_h &= R_{h,1} \cup R_{h,2} \cup R_{h,3} \quad (1 \leq h \leq n) \end{aligned}$$

D. Complexity of the P system

We now consider the complexity of the asynchronous P system $\Pi_{\text{MIN\_COL}}$. Since $O(n^2)$ objects are moved from the outer membrane into the inner membrane sequentially, Step 1 can be executed in $O(n^2)$ parallel or sequential steps using $O(n^2)$ kinds of objects and $O(n^3)$ kinds of evolution rules.

In Step 2, $O(n^n)$ membranes are created in the worst case, and evolution rules are applied sequentially at most $O(n)$ times in each membrane. Therefore, Step 2 can be executed in $O(n^2)$ parallel steps and $O(n^n)$ sequential steps using $O(n^3)$ kinds of objects and $O(n^4)$ kinds of evolution rules.

Step 3 is executed in each divided membrane. Therefore, Step 3 can be executed in $O(n)$ parallel steps and $O(n^n)$ sequential steps using $O(n^3)$ kinds of evolution rules.

In the final step, the computation is executed in the outer membrane, and Step 4 can be executed in $O(n)$ parallel or sequential steps using $O(n^2)$ kinds of evolution rules.

Therefore, we obtain the following theorem for the asynchronous P system $\Pi_{\text{MIN\_COL}}$.

Theorem 1: The asynchronous P system $\Pi_{\text{MIN\_COL}}$, which finds the minimum graph coloring for a graph with n vertices, works in $O(n^n)$ sequential steps or $O(n^2)$ parallel steps by using $O(n^2)$ types of objects and evolution rules of size $O(n^4)$. □

IV. Experimental simulations

We compare the numbers of membranes used in executions of an existing P system [18] and the proposed P system for minimum graph coloring. The simulations are executed on a custom simulator, which is programmed in Python, for asynchronous P systems.

Since the simulator executes P systems with asynchronous parallelism, all of the evolution rules are applied in a fully asynchronous manner. In other words, any number of applicable evolution rules is applied in each step of execution in the simulator. Therefore, the applied evolution rules differ between executions on the simulator, and the output of the simulation may differ for the same input. We first implement the existing P system and the proposed P system for minimum graph coloring on the simulator and execute simulations for various inputs. In the simulation, valid results are obtained for all inputs.

Next, we compare the number of membranes used on the existing P system and the proposed P system for the minimum graph coloring. For each number of vertices n $(3 \leq n \leq 6)$, five random graphs are created so that each edge is connected with probability $\frac{1}{2}$, and the same input graphs are given to the existing P system and the proposed P system.

Fig. 4 shows average values of the number of membranes in the simulation. Since the simulation for the existing P system for a graph with six vertices did not finish in a valid execution time, the number of the membranes is omitted for this case. The number of membranes of the existing P system increases exponentially with respect to the number of vertices n. Although the number of membranes in the proposed P system also increases according to n, the number of membranes in the proposed P system is more than 56 percent less than the number of membranes in the existing P system for each n $(3 \leq n \leq 5)$.

V. Conclusions

In the present paper, we proposed the asynchronous P system with branch and bound for solving the minimum graph coloring. The P system with branch and bound reduces the number of membranes by discarding partial assignments of colors in which the colors of neighboring vertices are the same.

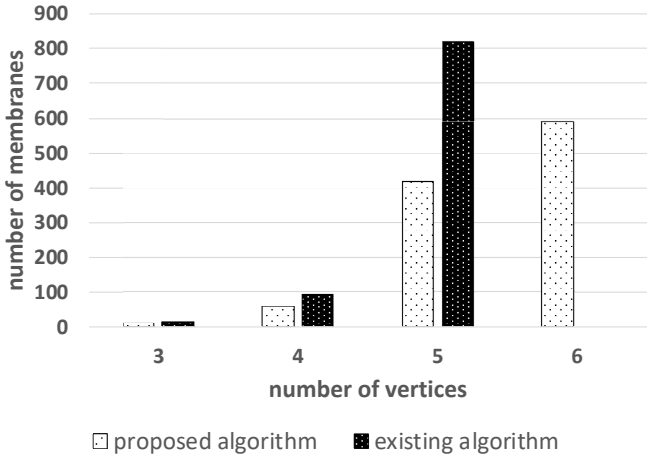


Fig. 4. Experimental results

The proposed P system is fully asynchronous and works in a polynomial number of steps in a maximally parallel manner and also works sequentially. Although the number of sequential steps is a power of the number of vertices, the result indicates that the proposed P system works for any combination of sequential and asynchronous applications of evolution rules and guarantees that correct solutions are output for any case in which any number of evolution rules are synchronized.

We also showed that the proposed P system outputs valid results and that the number of membranes in the proposed P system is as much as 48 percent less than the number of membranes in the existing P system for the minimum graph coloring.

In future research, we intend to consider a reduction in the number of objects and evolution rules used in the proposed P system. We also intend to consider asynchronous P systems with branch and bound for other computationally hard problems.

VI. Acknowledgments

This research was partially supported by JSPS KAKENHI, Grand-in-Aid for Scientific Research (C), 16K00021.

References

[1] R. Freund. Asynchronous P systems and P systems working in the sequential mode. In International workshop on Membrane Computing, pages 36–62, 2005.

[2] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A fast P system for finding a balanced 2-partition. Soft Computing, 9(9):673–678, 2005.

[3] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and F. J. Romero-Campero. A uniform solution to SAT using membrane creation. Theoretical Computer Science, 371(1-2):54–61, 2007.

[4] J. Imatomi and A. Fujiwara. An asynchronous P system for MAX-SAT. In 8th International Workshop on Parallel and Distributed Algorithms and Applications, pages 572–578, 2016.

[5] Y. Jimen and A. Fujiwara. Asynchronous P systems for hard graph problems. International Journal of Networking and Computing, 8(2):141–152, 2018.

[6] A. Leporati, C. Zandron, and G. Mauri. Solving the factorization problem with P systems. Progress in Natural Science, 17(4):471–478, 2007.

[7] V. Manca. DNA and membrane algorithms for SAT. Fundamenta Informaticae, 49(1-3):205–221, 2002.

[8] T. Murakawa and A. Fujiwara. Arithmetic operations and factorization using asynchronous P systems. International Journal of Networking and Computing, 2(2):217–233, 2012.

[9] L. Q. Pan and A. Alhazov. Solving HPP and SAT by P systems with active membranes and separation rules. Acta Informatica, 43(2):131–145, 2006.

[10] G. Păun. Computing with membranes. Journal of Computer and System Sciences, 61(1):108–143, 2000.

[11] G. Păun. P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics, 6(1):75–90, 2001.

[12] G. Păun. Introduction to Membrane Computing. Springer, 2006.

[13] M. J. Pérez-Jiménez and A. Riscos-Núñez. A linear-time solution to the knapsack problem using P systems with active membranes. Membrane Computing, 2933:250–268, 2004.

[14] M. J. Pérez-Jiménez and A. Riscos-Núñez. Solving the subset-sum problem by P systems with active membranes. New Generation Computing, 23(4):339–356, 2005.

[15] M. J. Pérez-Jiménez and F.J. Romero-Campero. Solving the BIN PACKING problem by recognizer P systems with active membranes. In The Second Brainstorming Week on Membrane Computing, pages 414–430, 2004.

[16] M. J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini. A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics, 11(4):423–434, 2003.

[17] H. Tagawa and A. Fujiwara. Solving SAT and Hamiltonian cycle problem using asynchronous p systems. IEICE Transactions on Information and Systems (Special section on Foundations of Computer Science), E95-D(3), 2012.

[18] K. Tanaka and A. Fujiwara. Asynchronous P systems for hard graph problems. International Journal of Networking and Computing, 4(1):2–22, 2014.

[19] C. Zandron, C. Ferretti, and G. Mauri. Solving NP-complete problems using P systems with active membranes. In Unconventional Models of Computation, pages 289–301, 2000.

Two swarm intelligence optimizations for the multi-objective knapsack problem

Yuta Hadachi Yuta Matsumoto Akihiro Fujiwara
Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502, Japan

***Abstract*—A multi-objective optimization problem involves a number of objective functions to be maximized or minimized, and no single solution exists for the problem because there is no solution that simultaneously satisfies all of the objective functions. Therefore, a set of Pareto-optimal solutions, which are non-dominated solutions for the problem, is defined for the multi-objective optimization problem.**

In the present paper, we consider the multi-objective knapsack problem, and propose two optimization algorithms using swarm intelligence. The first is based on the flower pollination algorithm (FPA), and the second is based on the artifival bee colony algorithm (ABC). The experimental results show that the proposed algorithms obtains a better set of Pareto solutions than the existing algorithm.

***Index Terms*—multi-objective knapsack problem, flower pollination algorithm**

I. Introduction

A multi-objective optimization problem involves a number of objective functions to be maximized or minimized. The multi-objective optimization problem has been considered in a number of fields, such as economics or engineering, and the most optimal solution is needed for the practical problems.

However, there is no single solution for the multi-objective optimization problem because no solution satisfies all of objectives simultaneously. Since there exists a trade-off between two or more conflicting objective functions, the non-dominated solutions, which are not inferior to the other solutions in all of the objective functions, are needed for the multi-objective optimization problem.

The non-dominated solution is called Pareto optimal solution [4], and a set of maximal Pareto optimal solutions is considered as an optimal solution for the multi-objective optimization problem. Since the problem for computing the maximal Pareto optimal solutions for the multi-objective optimization problem is generally computationally hard, a number of approximation algorithms have been proposed for the problem.

As an example of the multi-objective optimization problems, a number of approximation algorithms have proposed for a multi-objective 0-1 knapsack problem. A standard 0-1 knapsack problem [3] is generally given with a knapsack and multiple items, and value and weight are defined for each item. On the other hand, there exists a number of knapsacks in the multi-objective 0-1 knapsack problem, and values and weights of items are defined for each knapsack.

A number of approximation algorithms have been proposed for the multi-objective knapsack problem. In addition, some algorithms have been proposed based on a group intelligence optimizations, such as the ant colony algorithm [5] and the particle swarm optimization and the firefly algorithm [6].

In the present paper, we propose two approximation algorithms for the multi-objective 0-1 knapsack problem using flower pollination algorithm (FPA) [7] and artifical bee colony algorithm (ABC) [2].

The FPA is an optimization method based on the process of the flower pollination. There are two types of pollinations, which are global and local pollinations. In the global pollination, pollen is transferred by pollinators such as insects. The insects are attracted to flowers of bright color and strong smell. In the local pollination, pollen is transferred from one flower to the same flower by the wind.

On the other hand, the ABC is an optimization method based on behavior of honey bees. The bee gathers honey outside region, and shares information of gathered honey with other bees when the bee arrives at comb. Then, each bee decides a next way of exploration using exchanged information. In the ABC, each search is executed by exploration of the bees.

We implement our proposed algorithm and an existing algorithm [1] in experimental environment, and evaluate validity of the proposed algorithms. the experimental results show that our proposed algorithms obtain better sets of Pareto solutions than the existing algorithm.

II. Preliminaries

A. Multi-objective optimization problem

In this section, we first define the multi-objective optimization problem. We assume that an instance of the problem is k-dimensional decision vector $\boldsymbol{x}$. The multi-objective optimization problem consists of a set of n objective functions $\{f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \cdots, f_{n-1}(\boldsymbol{x})\}$ and a set of k constraint functions $\{g_0(\boldsymbol{x}), g_1(\boldsymbol{x}), \cdots, g_{k-1}(\boldsymbol{x})\}$. Then, each objective function is defined as image from $\boldsymbol{x}$ to n objective function vector $\boldsymbol{y}$. The definition is mathematically formulated as follows.

$$\max / \min \ \boldsymbol{y} = \{y_0, y_1, ..., y_{n-1}\} = \{f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), ..., f_{n-1}(\boldsymbol{x})\}$$
$$\text{such that } \boldsymbol{x} = (x_0, x_1, ..., x_{m-1}) \in X,$$
$$X = \{\boldsymbol{x} \mid \forall i \in \{0, 1, ..., k-1\}, g_i(\boldsymbol{x}) \leq 0\}$$

In the above definition, X is called as a set of feasible solutions for the problem.

B. Pareto-optimal solution

Since no solution satisfies all of the objective functions in the multi-objective optimization problem, a solution that is not inferior to the other solutions is needed for the multi-objective problem. The solution is called *Pareto optimal solution*, and we first define the dominance relationship of the solutions for defining the Pareto-optimal solution.

We assume that $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ are two feasible solutions for the problem and all objective functions are maximized. Then, $\boldsymbol{x_2}$ dominates $\boldsymbol{x_1}$ if and only if the following two conditions holds.

$$\forall i \in \{0, 1, ..., n-1\}, f_i(\boldsymbol{x}_1) \leq f_i(\boldsymbol{x}_2)$$
$$\exists j \in \{0, 1, ..., n-1\}, f_j(\boldsymbol{x}_1) < f_j(\boldsymbol{x}_2)$$

In this paper, $\boldsymbol{x}_1 \prec \boldsymbol{x}_2$ denotes $\boldsymbol{x}_2$ dominates $\boldsymbol{x}_1$.

In addition, a feasible solution $\boldsymbol{x}$ is called *Pareto optimal* if and only if there is no feasible solution $\boldsymbol{x'} \in X$ such that $\boldsymbol{x} \prec \boldsymbol{x'}$. Since the Pareto-optimal solution is a solution that cannot be improved in any of the objective function without degrading one of the other functions, a maximal set of the Pareto optimal solution is considered as an optimal solution for the multi-objective optimization problem.

C. The hypervolume indicator

There are various metrics for a set of Pareto-optimal solutions of the multi-objective optimization problem. In the paper, we evaluate a set of the Pareto-optimal solutions using *hypervolume indicator* [4].

Let v_x be the volume of the hypercube created by Pareto optimal solution $\boldsymbol{x}$ and the reference point $\boldsymbol{r}$. The hypervolume V for a set of Pareto optimal solutions is defined as follows.

$$V = \bigcup_{\boldsymbol{x} \in X} v_x$$

Fig. 1 shows an example of the hypervolume indicator in case that the problem is bi-objective. Let $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \boldsymbol{x_4}\}$ be a set of Pareto optimal solutions. The hypervolume for X is defined as an union, which is an gray-shaded area in the figure, of four rectangular regions whose opposite vertices are a reference point $\boldsymbol{r}$ and $(f_1(\boldsymbol{x_i}), f_2(\boldsymbol{x_i}))$.

III. The optimization algorithms for the multi-objective knapsack problem

In this section, we first introduce definition of the multi-objective 0-1 knapsack problem, which is a well-known multi-objective optimization problem. We next explain two common procedures for two optimization. We finally show our two proposed optimization algorithms, which are based on the flower pollination optimization and the artificial bee colony optimization.

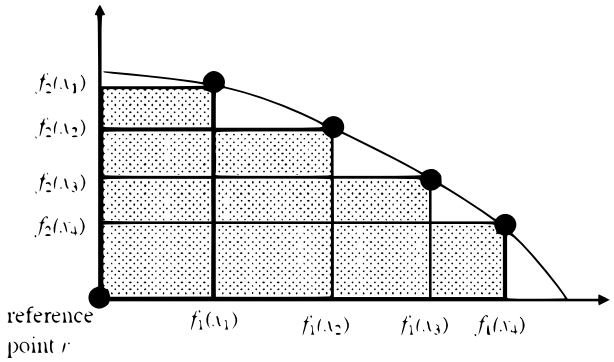


Fig. 1. An example of Hypervolume indicator

A. The multi-objective 0-1 knapsack problem

An input of the multi-objective 0-1 knapsack problem is given as follows.

- n knapsacks whose capacities are $c_0, c_1, \cdots c_{n-1}$.
- m items stored in the knapsacks. $v_{i,j}$ and $w_{i,j}$ denote value and weight of item j for knapsack i, respectively.

Let $_x = (x_0, x_1, \cdots, x_{m-1})$ are m-dimensional Boolean vector. Then, the multi-objective 0-1 knapsack problem is formulated as follows.

$$\max \ \boldsymbol{y} = \{f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), ..., f_{n-1}(\boldsymbol{x})\}$$
$$f_i(\boldsymbol{x}) = \sum_{j=0}^{m-1} p_{i,j} x_j$$
$$\text{s.t. } g_i(\boldsymbol{x}) = \left(\sum_{j=0}^{m-1} w_{i,j} x_j \right) - c_i \leq 0$$

In the above expression, $x_j = 1$ denotes that item j is stored in the knapsacks.

B. The common procedures for optimization algorithms

We explain two common procedures used in the proposed optimization algorithm. The first is a procedure that updates best Pareto optimal solutions, and the second is a procedure that repair infeasible solutions, which do not satisfy constraint functions, to feasible solutions.

1) A procedure for updating best solutions: In the proposed algorithm, Pareto optimal solutions are created at every iteration. Then, Pareto ranking sort is used as a procedure for obtaining a set of l best optimal solutions and a set of Pareto optimal solutions. Let P be a temporal Pareto optimal solutions, and F is a set of feasible solutions obtained in the algorithm. Then, a procedure, Pareto ranking sort, is summarized as follows.

A procedure of Pareto ranking sort:

Step 1: Set $P' = P \cup F$ and $r = 1$. Then, compute Pareto rank of each solution in P' by repeating the following sub-steps until $P' = \phi$.

(1-1) Select Pareto optimal solutions in P', and set Pareto ranks of the selected solutions to r.

(1-2) Remove the selected solutions from P', and set $r = r + 1$.

Step 2: Sort all solutions in $P \cup F\}$ according to the computed Pareto ranks. Then, select l best solutions $\{t_0, t_1, \cdots, t_{l-1}\}$ for the next iteration. In addition, all solutions whose rank is 1 is selected as a set of Pareto optimal solutions P.

2) A procedure for repairing infeasible solutions: We next show a procedure that repair infeasible solutions, which do not satisfy constraint functions, to feasible solutions. In the procedure, a number of items are removed from a tentative solution using a greedy method.

Let $x_i = (x_{i,0}, x_{i,1}, \cdots, x_{i,m-1})$ be an infeasible solution obtained the in the proposed algorithm. Then, the following procedure is executed and the solution is repaired so that all constraints are satisfied.

A procedure for repairing an infeasible solution:

Step 1: Sort all items in descending order according to the following value s_j $(0 \leq j \leq m-1)$.

$$s_j = \sum_{i=0}^{n-1} \frac{p_{i,j}}{w_{i,j}}$$

Step 2: For each item j $(0 \leq j \leq m-1)$ in solution x_i, the following sub-steps are repeated until x_i is feasible.

(2-1) Select $x_{i,j}$ that satisfies the following condition.

$$s_j = \min\{s_k \mid x_{i,k} = 1, 0 \leq k \leq m-1\}$$

(2-2) Set $x_{i,j} = 0$.

(2-3) Select $x_{i,j}$ that satisfies the following condition.

$$s_j = \max\{s_k \mid x_{i,k} = 0, 0 \leq k \leq m-1\}$$

(2-4) Set $x_{i,j} = 1$ and evaluate the solution. In case that the solution is infeasible, set $x_{i,j} = 0$ again.

C. An optimization algorithm based on flower pollination

The flower pollination algorithm (FPA) [7] is an optimization method inspired by the nature from the pollination process of flowers. Flower pollination is a transfer of pollen from a stamen to the stigma of a flower.

The pollination is classified into four kinds of pollination. Pollination that the pollen is transferred by pollinators such as insects or animals is called biotic pollination. In this case, animals and insects are attracted by flowers that have bright colors and strong smell. 90 % of pollination is classified under the biotic pollination. On the one hand, biotic pollination transfer pollen through the wind or diffusion in the water. 10 % of pollination is classified under the biotic pollination. In addition, pollination can be classified into self-pollination and cross-pollination. A transfer of pollen from one flower to the same flower is called as self-pollination, and a transfer of pollen from one flower to a different flower is called cross-pollination.

The pollination increases the quality and strength of flowers. To simulate the pollination process, the following four rules are used.

- In the global pollination, biotic and cross-pollination are considered. The pollinators move in a way which obeys a Levy flight distribution.
- In the Local pollination, biotic and self-pollination are considered.
- Flower constancy is considered as the reproduction probability, which is proportional to the similarity of two flowers involved.
- A choice between global pollination or local pollination is controlled by a switch probability $p \in [0, 1]$.

We now formulate the flower pollination as a optimization technique. In global pollination, flower pollen are carried over a long distance by the pollinators such as insects and animals. To simulate the above first and third rules, the following equation is used.

$$x_i^{t+1} = x_i^t + \gamma L(\lambda)(x_i^t - b^*) \tag{1}$$

where x_i^t is the solution vector x_i at iteration t, and b^* is the current best solution. γ is a scaling factor to control the step size. The parameter $L(\lambda)$ control the strength of the pollination. In [7], the parameters are suggested to set $\gamma = 0.1$ and $\lambda = 1.5$. To simulate move of pollinators over a long distance, the Levy flight distribution, which is given below, is used.

$$L(\lambda) \sim \frac{\lambda\Gamma(\lambda)\sin(\frac{\pi\lambda}{2})}{\pi} \frac{1}{S^{1+\lambda}} (S >> S_0 > 0) \tag{2}$$

$\Gamma(\lambda)$ is the standard gamma function. The step size S is generated by using Gaussian distributions through generating two random numbers U and V as follows.

$$S = \frac{U}{|V|^{\frac{1}{\lambda}}}, U \sim N(0, \sigma^2), V \sim N(0, 1) \tag{3}$$

$$\sigma^2 = \left[\frac{\Gamma(1+\lambda)}{\lambda\Gamma(\frac{1+\lambda}{2})} \frac{\sin(\frac{\pi\lambda}{2})}{2^{\frac{\lambda-1}{2}}} \right]^{\frac{1}{\lambda}} \tag{4}$$

To simulate the above second rule as local pollination, the following equation is defined.

$$x_i^{t+1} = x_i^t + U(x_j^t - x_k^t) \tag{5}$$

where x_j^t, x_k^t are pollen from different flowers of same plant and U is a uniform distribution in $[0, 1]$.

We now propose an optimization algorithm for the multi-objective knapsack problem using FPA. The algorithm consists of three steps.

An algorithm for the multi-objective knapsack problem using FPA

Step 1: Create random initial l solutions as m-dimensional Boolean vector $x_i^0 = (x_0, x_1, \cdots, x_{m-1})$ $(0 \leq i \leq l-1)$. Each x_k of Boolean vector is 0 or 1 with provability of $\frac{1}{2}$. Then, store the vector as a set of solution X. If x_j is an infeasible solution, execute a procedure for repairing for the infeasible solution.

In addition, find a set of Pareto optimal solutions P in the initial solutions, and choose a best solution b^* from P.

Step 2: Repeat the following sub-steps T times. (T is a parameter that denotes the number of generations.) We assume that t denotes the number of executed generations in the following.

(2-1) Generate solutions x_i^{t+1} $(0 \leq i \leq l-1)$ as follows.

(2-1-1) Generate a random value r from a uniform distribution in $[0, 1]$. In case that $p \leq r$, execute global pollination according to (1). Otherwise, execute local pollination according to (5).

(2-1-2) Convert real values obtained in (2-1-1) into binary values using the sigmoid function given below. (r is a random value in $[0, 1]$.)

$$x_{sig} = \frac{1}{1 + \mathrm{e}^{-x_i}} \tag{6}$$

$$x_{bin} = \begin{cases} 1 & r \leq x_{sig} \\ 0 & (\text{otherwise}) \end{cases} \tag{7}$$

(2-1-3) In case that the obtained solution is infeasible, execute a procedure for repairing the infeasible solution. Then, store the solution in a set of solutions F.

(2-3) Execute a procedure of Pareto ranking sort for P and F, and obtain a set of Pareto optimal solutions P.

Step 3: Output a final P as a set of Pareto optimal solutions.

D. An optimization algorithm based on artificial bee colony

The artificial bee colony (ABC) algorithm [2] is an optimization method inspired by foraging behavior of honey bees. There are three kinds of bees, which are employed onlooker and scout bees. Roles of the bees are described below.

- **Employed Bees:** Employed bees have food sources information and tried to detect a new food sources. After detecting a new food sources, the bees return to beehive and share information of the sources with onlooker bees.
- **Onlooker bees:** Onlooker bees evaluate information shared with employed bees, and the bees try to find a new food sources in the neighborhood of the food sources.
- **Scout Bees:** Scout bees try to find a new food sources randomly. After detecting a new food sources, a scout bee becomes a employed bee.

We now propose an optimization algorithm for the multi-objective knapsack problem using ABC. We assume that $EB = \{eb_0, eb_1, \cdots, eb_{m_1}\}$ and $OB = \{ob_0, ob_1, \cdots, ob_{m_2}\}$ denotes sets of employed, onlooker and scout bees, respectively, and the roles of the above bees are formulated in the algorithm. The algorithm consists of three steps.

A procedure for repairing an infeasible solution for ABC:

Step 1: Sort all items of each knapsack $i (0 \leq i \leq n-1)$ in descending order according to the following value $s_{i,j}$ $(0 \leq j \leq m-1)$.

$$s_{i,j} = \frac{p_{i,j}}{w_{i,j}}$$

Step 2: For each item j $(0 \leq j \leq m-1)$ in solution x_i, the following sub-steps are repeated until x_i is feasible. Then, each bee is responsable for a knapsack. Suppose that the number of bee $m = 100$ and knapsack $n = 2$, $i-th$ bee refer the value of the knapsack1 $s_{1,j}$ by $i\%$ accurancy. On the other hand, the bee refer the value of the knapsack2 $s_{2,j}$ by $100 - i\%$ accurancy.

(2-1) Select $x_{i,j}$ that satisfies the following condition.

$$s_{i,j} = \min\{s_{i,k} \mid x_{i,k} = 1, 0 \leq k \leq m-1\}$$

(2-2) Set $x_{i,j} = 0$.

(2-3) Select $x_{i,j}$ that satisfies the following condition.

$$s_{i,j} = \max\{s_{i,k} \mid x_{i,k} = 0, 0 \leq k \leq m-1\}$$

(2-4) Set $x_{i,j} = 1$ and evaluate the solution. In case that the solution is infeasible, set $x_{i,j} = 0$ again.

Algorithm: an algorithm for multi-objective 0-1 knapsack problem using ABC

Step 1: Create initial l solutions in the same as Step 1 of the algorithm using FPA.

Step 2: Repeat the following sub-steps T times. (T is a parameter that denotes the number of generations.)

(2-1) Each employed bee eb_i executes the following (2-1-1) $\sim$ (2-1-4).

(2-1-1) Select other two employed bee eb_{k_1} and eb_{k_2} randomly. Then, modify a solution $x_i = (x_{i,0}, x_{i,1}, \cdots, x_{i,m-1})$ as follows.

$$x_{i,j} = \begin{cases} x_{i,j} & (\text{if } x_{i,j} = x_{k_1,j} = x_{k_2,j}) \\ \overline{x_{i,j}} & (\text{otherwise}) \end{cases} \tag{8}$$

(2-1-2) In case that the obtained solution is infeasible, execute a procedure for repairing the infeasible solution.

(2-1-3) Execute a procedure of Pareto ranking sort for P and a obtained solution.

(2-1-4) Decide whether to transform to scout bee by a probabilistic choice. (The probability of the transformation is set to 0.01 in the paper.) In case of transformation to a scout bee, create a new solution in the same as Step 1. Then, transform to employed bee again.

(2-2) Each onlooker bee ob_i executes the following (2-2-1) $\sim$ (2-2-5).

(2-2-1) Decide whether to keep or destroy a solution x_i by a probabilistic choice. A probability p_i for the choice is calculated as

follows. In the expression, $fit(x)$ denotes an evaluation value for solution x.

$$\hat{p}_i = \frac{fit(x_i) - fit_{\min}(X)}{fit_{\max}(X) - fit_{\min}(X)} \quad (9)$$

$$fit_{\max}(X) = \max\{fit(x_k) \mid 0 \leq k \leq m_2\}$$
$$fit_{\min}(X) = \min\{fit(x_k) \mid 0 \leq k \leq m_2\}$$

(2-2-2) Each onlooker bee that destroys a solution search a new solution x_i as follows.

$$x_i = \begin{cases} x_{gb}, & r < 0.5 \\ x_k, & r \geq 0.5 \end{cases} \quad (10)$$

In the above expression, x_{gb} is a Pareto optimal solution whose humming distance is closest to the solution x_i, and r is a random number generated from a uniform distribution in $[0, 1]$. In addition, x_k is a solution of an onlooker bee selected randomly.

(2-2-3) Execute binary swap for the solution x_i as follows.

$$x_{i,u} = \overline{x_{iu}} \quad (11)$$

$$x_{i,v} = \begin{cases} x_{iv}, & (\text{if} x_{i,u} = x_{i,v}) \\ \overline{x_{i,v}}, & (\text{otherwise}) \end{cases} \quad (12)$$

(2-2-4) In case that the obtained solution is infeasible, execute a procedure for repairing the infeasible solution.

(2-2-5) Execute a procedure of Pareto ranking sort for P and a obtained solution.

Step 3: Output P as a set of Pareto optimal solutions.

IV. Experimental results

Our proposed algorithm and an existing algorithm [1] are implemented using C++, and we compare Pareto optimal solutions and hypervolume indicators.

First, we describe parameters of the multi-objective 0-1 knapsack problem. The values of the variables used in the simulation are as follows.

- The number of knapsacks n: 2
- The number of items m: 500
- A value of item $p_{i,j}$ $(1 \leq i \leq n, 1 \leq j \leq m)$: a randomly generated integer in [10, 100]
- A weight of item $w_{i,j}$ $(1 \leq i \leq n, 1 \leq j \leq m)$: a randomly generated integer in [10,100]
- A capacity of a knapsack c_i $(1 \leq i \leq n)$: $c_i = \frac{1}{2}\sum_{j=1}^{m} w_{i,j}$

Fig. 2 shows a part of our experimental results. Pareto optimal solutions obtained by the proposed algorithms are distributed in wider range than the solutions obtained by the existing algorithm.

Table I shows hypervolume indicators of the algorithms. The hypervolumes of the proposed algorithms are better than the existing algorithm.

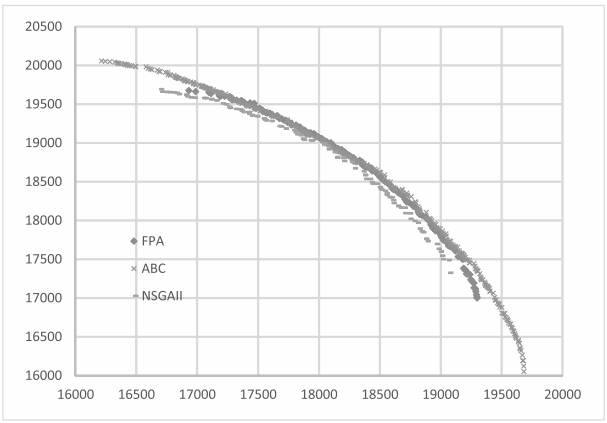


Fig. 2. Pareto optimal solutions of the algorithms

TABLE I
HYPERVOLUME INDICATORS OF THE ALGORITHMS

FPA	ABC	The existing algorithm
3.78×10^8	3.90×10^8	3.74×10^8

V. Conclusions

In this paper, we proposed an approximation algorithm for the multi-objective 0-1 knapsack problem using FPA and ABC. We implemented our proposed algorithms and an existing algorithm in experimental environment, and showed validity of the proposed algorithm from the viewpoint of hypervolume indicator.

As our future research, we are considering improvement of our proposed algorithm for diversity of the Pareto optimal solutions, and also considering reduction of the execution time of the proposed algorithms.

References

[1] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation (IEEE-TEC)*, 2002.

[2] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39, 2007.

[3] H. Kellerer, U. Pferschy, and D. Pisinger. Knapsack problems. *Springer*, 2004.

[4] J. D. Knowles, L. Thiele, and E. Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. *TIK-Report No.214, Computer Enfineering and Networks Laboratory*, 2005.

[5] D. Simian R. Jovanovic, M. Tuba. Ant colony optimazation applied to minimum weight dominating set problem. In *Proc. 12th WSEAS International Conference on AUTOMATIC CONTROL*, 2010.

[6] X. Yang. *Nature-Inspird Metahuristic Algorithms, second edition*. Luniver Press, 2010.

[7] X.-S. Yang. Flower pollination algorithm global optimization. In *Proc. Unconventional Computation and Natural Computation*, 2012.

局所性を考慮した辺交換ゲームの収束性と均衡の効率

吉村　正太郎 *　　山内　由紀子 *

概要

Peer-to-Peer 型システムのように，不特定多数の参加者によって形成されるネットワークの性質の解析には利己的な参加者によゲームによるモデル化が有効であると考える．さらに，それらのゲームで得られた知見によって各プロセスの利己的．局所的な最適化による分散アルゴリズムの設計方針が得られると期待する．

Alon らは辺交換ゲーム（Swap Game, SG）を提案した [1]．これは Fabrilkant らの提案したネットワーク構成ゲーム（Network Creation Game, NCG）[3] の特殊な状況を取り出した戦略形ゲームである．SG では，最初に n 個の頂点をプレイヤーとする無向グラフ G_0 が与えられ，プレイヤーは辺交換，すなわち自身の接続辺を別のプレイヤーに繋ぎ変える操作を行うことで新たなグラフを生成し，他のプレイヤーとの距離に依存する自身のコストを改善していく．このゲームは特に G_0 を木としたときに，均衡への収束性を持つことが知られている [4, 5]．Bilò らは，NCG において，各プレイヤーが全てのプレイヤーとの距離といった大域的情報を得られるという仮定を緩和し，各プレイヤーが自身の k-近傍，すなわち自身から距離 k 以内にいるプレイヤーによる誘導部分グラフのみを観測できるという条件の下での NCG を提案した [2]．このゲームでは，k をある程度小さく設定した際に，元の NCG には現れない直径がプレイヤー数に依存して大きくなる均衡が数種類見つかっており，例えば $k \leq n/2-1$ の時には直径が $\lfloor n/2 \rfloor$ となる均衡が示されている．

本稿では Bilò らの局所性の制約を G_0 を木とする SG に取り入れたゲームを提案し，このゲームにおける収束性や均衡を示す．この制約を加えた SG は元の SG と同様に収束性を持ち，$k \leq 3$ の場合に大域的情報を知ることができる場合 [1] には表れない，直径がプレイヤー数 n に比例するような均衡が出現する一方で，$k \geq 4$ の場合には，大域的情報を知ることができる場合と同様に直径が高々 3 の均衡しか存在しないことを示す．グラフの直径が大きいほどプレイヤー同士の距離が離れて各プレイヤーのコストが増大するので，この結果は全プレイヤーのコストを最適にするためには，視界の大きさが $k = 4$ であれば十分であることを示している．

参考文献

[1] N. Alon, E. D. Demaine, M. T. Hajiaghayi, and T. Leighton. Basic network creation game. J. on Discrete Math, 27, 2, pp. 656–668, 2013.

[2] D. Bilò, L. Gualà, S. Leucci, and G. Proietti. Locality-based network creation games. J. on TOPC, 3, 1, pp. 6:1–6:26, 2016.

[3] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimiriou, and S. Shenker. On a networrk creation game. Proc. of PDOC 2003, pp. 347–351, 2003.

[4] B. Kawald and P. Lenzner. On dynamics in selfish network creation. Proc. of SPAA 2013, pp. 83–92, 2013.

[5] P. Lenzner. On dynamics in basic network creation games. Proc. of SAGT 2011, pp. 254–265, 2011.

* 九州大学大学院システム情報科学府

個体群プロトコルにおける半数分割の可解性の解明

安見　嘉人,　大下　福仁,　井上　美智子
奈良先端科学技術大学院大学

概要

本研究では，個体群プロトコルモデルにおける半数分割問題に着目する．個体群プロトコルモデルとは，個体と呼ばれる低能力のデバイスで構成されるネットワーク上の計算を表現する抽象モデルである．半数分割問題とは，その全個体を二つの同サイズのグループに分けることを目的とした問題である．先行研究では，半数分割問題について，４つの仮定（基地局の存在，公平性，デバイスの初期状態，プロトコルの対称性）の下での空間計算量を考えていた．本研究での主な成果は以下の二つである．１）上記の４つのいくつかの仮定の下で，より厳密な空間計算量を示した．この結果と先行研究の結果から，その４つの仮定の下での半数分割問題に対する空間計算量を完全に明らかにした．２）新たにグラフトポロジの仮定（デバイス間の通信に制限を加える仮定）を考え，既存の仮定との各組合せに対して問題を考え，その多くの組合せで空間計算量を明らかにした．

1　はじめに

個体群プロトコルモデル (population protocol model)[1] とは，低能力のデバイス（個体，agent）で構成されるネットワーク上の計算を表現する抽象モデルである．このモデルでは，個体がそれぞれ状態を持ち，他の個体と交流を繰り返すことで状態を更新して計算を進める．このとき，どの個体がどの順序で交流するかをシステム自体はコントロールできない．このモデルの応用として，低性能な分子ロボットを用いた分子スケールのシステムが挙げられる．このシステムでは，多数の分子ロボットが協力して体内の情報を収集したり，病気の細胞へ薬を届けたりする応用が考案されている．また，別の応用として，鳥や小動物に取り付けたセンサで構成されるセンサネットワークなども挙げられる．

これまでに個体群プロトコルモデルに対する多くのアルゴリズムが研究されている [2]．例えば，全体の個体数を求めるアルゴリズム [3]，リーダー選挙アルゴリズム [4] などが提案されている．

本稿では，個体群を同サイズに 2 分割する半数分割問題 [5] に注目する．これは個体がもつ状態を最終的に 2 状態に収束させ，それぞれの状態の個体数を同数にするという問題である．半数分割問題の応用例として，分割した一方の個体群だけを動作させることによるエネルギー消費の低減，分割した個体群による別々のタスクの同時実行などが考えられる．

本稿では半数分割問題についての可解性を考察する．その際，システムの仮定として基地局の

表 1　完全グラフで半数分割問題を解くための最小状態数，ここで P は個体数の上界.

基地局	公平性	初期値一定		初期値任意	
		非対称遷移あり	非対称遷移なし	非対称遷移あり	非対称遷移なし
あり	全体公平性	3	3	4	4
	弱公平性	3	3	$\boldsymbol{P}$*	$\boldsymbol{P+1}$*
なし	全体公平性	3	4	不可	不可
	弱公平性	3	不可	不可	不可

* **Our contribution**

表 2　任意グラフで半数分割問題を解くための最小状態数，ここで P は個体数の上界.

基地局	公平性	初期値一定		初期値任意	
		非対称遷移あり	非対称遷移なし	非対称遷移あり	非対称遷移なし
あり	全体公平性	**3***	**3***	**$\geq$ 4?***	**$\geq$ 4?***
	弱公平性	**$\leq 3\boldsymbol{P}+1$***	**$\leq 3\boldsymbol{P}+1$***	**不可 ***	**不可 ***
なし	全体公平性	**4***	**5***	不可	不可
	弱公平性	**不可 ***	不可	不可	不可

* **Our contribution**

有無，公平性，グラフトポロジ，実現するアルゴリズムの性質として，非対称遷移の有無，基地局以外の個体の初期値の設定を考える．基地局とは強力な能力をもつ特別な個体で，1 台の基地局を仮定することで良い性質を実現できる可能性がある．弱公平性は各個体間で交流が無限回発生することのみを仮定している．詳細は後述するが，全体公平性は弱公平性よりも強い仮定である．完全グラフでは，全ての個体が交流可能であることを仮定している．対して任意グラフでは，交流できる個体に制限を加えている．非対称遷移のあるプロトコルは，個体間で対称性を破壊する仕組が必要であり，分子ロボットのような低性能な個体では実現が難しい．そのため非対称遷移のないプロトコルが望まれる．初期値が一定の場合，実行開始時に個体の初期化が必要となる．初期値が任意の場合，基地局以外の個体の初期化が必要ない．そのため，一時故障が発生して個体が任意の状態になったとしても，基地局を初期化するだけで目的の状況を達成することができる．

本研究では，完全グラフについては，基地局あり，弱公平，初期値一定の仮定の下での半数分割問題の可解性を明らかにした（表 1）．完全グラフの場合，この結果と先行研究の結果を合わせて，全仮定の組合せについて，可解性を完全に明らかにした．さらに，任意グラフの場合も多くの設定において可解性を明らかにした（表 2）．自然数 $n(=3,4,5,P,P+1)$ の項目は，状態数 n のアルゴリズムが存在し，状態数 $n-1$ のアルゴリズムが存在しないことを示す．ここで，P は個体数の上界を表す．$\leq 3P+1$ の項目では個体数の上界 P に対して $3P+1$ 状態で問題を解くアルゴリズムが存在することを表す．$\geq$4?の項目は状態数 3 では不可能であることのみを表す．

また，不可は状態数によらずアルゴリズムの実現が不可能であることを示す．

2 諸定義

個体群プロトコルモデルでは，個体群と呼ばれる連結グラフ $G(V, E)$ によってシステムが構成される．ここで，V は個体の集合であり，$E \subseteq V \times V$ は辺の集合である．各辺は交流可能かどうかを表している．つまり，もし個体 $u \in V$ と $v \in V$ が交流可能であれば，$(u, v) \in E$ である．

プロトコル P は，各個体の取りうる状態 (state) の集合 Q と Q 上の遷移の集合の組で定義される．各遷移は，$(p, q) \to (p', q')$ の形で表され，これは状態 p を持つ任意の個体 x と状態 q を持つ任意の個体 y が交流したとき，x は状態 p から状態 p' に変わり，y は状態 q から状態 q' に変わることを表す．プロトコル P において，すべての状態のペア (p, q) に対して，ただひとつの遷移 $(p, q) \to (p', q')$ が存在するならば，P は決定性のプロトコルであるという．本稿では決定性のプロトコルのみを考慮する．遷移 $(p, q) \to (p', q')$ に対して，$p = q$ かつ $p' \neq q'$ ならば，その遷移を非対称遷移と呼ぶ．そうでない遷移は対称遷移と呼ぶ．プロトコルが対称遷移のみを含む場合，そのプロトコルを対称なプロトコルと呼ぶ．プロトコルが対称遷移と非対称遷移を含む場合，そのプロトコルを非対称なプロトコルと呼ぶ．

個体群の大域的な状態を状況 (configuration) と呼び，すべての個体の状態を要素としたベクトルで表す．ある状況 C と C' において，C' が個体のペアの一回の交流によって C から得られるとき，$C \to C'$ と表す．これは，状況 C において状態 p の個体 x と状態 q の個体 y が存在し，ある遷移 $(p, q) \to (p', q')$ によって，個体 x と y の状態が p' と q' へ変わった状況 C' に遷移することを表す．

無限状況列 $E = C_0, C_1, C_2, \ldots$ がそれぞれの $i (i \geq 0)$ において $C_i \to C_{i+1}$ を満たすとき，E をプロトコルの実行という．ある実行 E において，全ての個体同士が無限回交流するとき，E を弱公平な (weakly fair) 実行という．ある実行 E において，C が無限回発生するならば，$C \to C'$ が成り立つすべての C' も無限回発生するとき，E を全体公平な (globally fair) 実行という．

本稿では基地局ありモデルと基地局なしモデルの 2 つを考える．基地局ありモデルでは，個体集合の中に，他の個体と区別可能な基地局 (base station) が一つ存在することを仮定する．他の個体の能力（メモリ量など）が非常に弱いのに対し，基地局は強力な能力を仮定することができる．基地局なしモデルでは，基地局が存在しない．

状態集合 Q は，基地局の状態集合 Q_b と基地局以外の状態集合 Q_p に分割される．すなわち，$Q = Q_b \cup Q_p$, $Q_b \cap Q_p = \emptyset$ を満たす．なお，基地局なしモデルでは $Q_b = \emptyset, Q_p = Q$ となる．基地局を除いた個体の数を n と表す．また，初期状態では基地局も含めて全ての個体が n を知らないものとする．

A_p を基地局以外の全ての個体の集合とし，個体 $x \in A_p$ に対して，s_x を x の状態とする．$f : Q_p \to \{red, blue\}$ を基地局以外の個体の状態を $red, blue$ へ関連付ける関数とする．実行 $E = C_0, C_1, C_2, \ldots$ が半数分割問題を解くとは，以下の条件を満たす t と A_p の分割 R, B が存在することである．

1. $||R| - |B|| \leq 1$
2. 全ての $t' \geq t$ で状況 $C_{t'}$ において，$\forall x \in R : f(s_x) = red$ かつ $\forall x \in B : f(s_x) = blue$ が

成立

プロトコル P の全ての実行 E が半数分割問題を解くとき，プロトコル P は半数分割問題を解くと定義する．

3 結果

本稿では，基地局がなく，全体公平性を仮定したシステムにおいて，任意グラフ上で，非対称遷移があり，各個体の初期値が一定の仮定の下での半数分割問題を考える．具体的には，その仮定の下で，4 状態で問題を解くアルゴリズムと 3 状態の不可能性の結果を示す．

3.1 アルゴリズム

本節では，基地局がなく，全体公平性を仮定したシステムにおいて，任意グラフ上で，非対称遷移があり，各個体の初期値が一定，状態数が 4 の半数分割アルゴリズムを示す．

各個体の状態集合は $Q = \{r^\omega, b^\omega, r, b\}$ とする．ここで $f(r^\omega) = f(r) = red$, $f(b^\omega) = f(b) = blue$ とし，個体の初期状態は r^ω とする．アルゴリズムの遷移規則を Algorithm 1 に示す．

アルゴリズムの基本戦略は以下の通りである．

トークン (r^ω と b^ω) を持つ個体同士が交流した際に，トークンを 2 つ消して，red の個体数を 1 減らし $blue$ の個体数を 1 増やすような遷移（遷移 1 と 6）を行う．トークンは状態の交換によりグラフ中を移動するため（遷移 2，3，4，5），全体公平性から，トークンが 2 つ以上存在すればいつかトークンを持つ個体同士の交流が発生する．遷移 1 と 6 では，トークンは 2 つずつ消えるため，全トークンが消えるまでにこのような遷移は $\lfloor n/2 \rfloor$ 回発生する（n は個体数）．結果として，red の個体数が $\lfloor n/2 \rfloor$ 個減り，$blue$ の個体数が $\lfloor n/2 \rfloor$ 個増える．個体は初期状態として red を持つため，全トークンが消えたとき，red の個体数が $\lceil n/2 \rceil$ 個，$blue$ の個体数が $\lfloor n/2 \rfloor$ 個となり，半数分割が達成される．

Algorithm 1 Uniform bipartition protocol

Transition rules

1. $(r^\omega, r^\omega) \to (r, b)$
2. $(r^\omega, r) \to (r, r^\omega)$
3. $(b^\omega, b) \to (b, b^\omega)$
4. $(r^\omega, b) \to (r, b^\omega)$
5. $(b^\omega, r) \to (b, r^\omega)$
6. $(r^\omega, b^\omega) \to (b, b)$

3.2 不可能性

本節では，基地局がなく，全体公平性を仮定したシステムにおいて，任意グラフ上で，非対称遷移があり，各個体の初期値が一定のとき，個体の状態数が 3 以下では問題を解くことが不可能

であることを示す.

まず，red の個体数と $blue$ の個体数に関する記法を以下に定義する.

Definition **1.** ある個体群 V に対し，V に属する個体の中で red である数を $\#red(V)$，$blue$ である数を $\#blue(V)$ と表す．文脈から明らかであれば，$\#red$, $\#blue$ と表す.

証明は以下の手順で行う．背理法から，3 状態で問題を解くアルゴリズム Alg が存在するとする．まず，個体数 $n < P/2$ が奇数の任意のグラフに対する Alg の全体公平実行で，各個体は状態を無限回別の状態に遷移させることを示す．その後，そのような実行では，個体が 3 状態しか持てなければ，1 状態しか存在しない色が存在し，その状態は安定状況以降に別の色に遷移することになるため Alg が問題を解くことと矛盾することを示す.

Theorem **1.** 基地局がなく，全体公平性を仮定したシステムにおいて，任意グラフ上で，非対称遷移があり，各個体の初期値が一定のとき，個体の状態数が 3 で半数分割問題を解くアルゴリズムが存在しない.

背理法から，そのようなアルゴリズム Alg が存在するとする.

まずは，個体数 $n < P/2$ が奇数の任意のグラフに対する実行で，各個体は状態を無限回別の状態に遷移させることを示す.

Lemma **1.** 個体数 $n < P/2$ が奇数の任意のグラフ $G(V, E)$ に対して半数分割アルゴリズム Alg を適用した任意の全体公平実行 Ξ について，各個体は状態を無限回別の状態に遷移させる.

Proof. 証明の概略は，以下の通りである．背理法から，個体数 $n < P/2$ が奇数の任意のグラフ $G(V, E)$ に対する半数分割アルゴリズム Alg を適用した任意の全体公平実行 Ξ について，ある状況 C 以降にある個体 v_α が状態を変化させない．ここで，v_α に隣接する任意の個体を v_β とする．状態数が有限であるため，状況 C 以降に無限回発生する状況 C_t が存在する．つづいて，グラフ $G(V, E)$ と同型な 2 つのグラフを $G_1(V_1, E_1)$ と $G_2(V_2, E_2)$ とする．さらに，$G_1(V_1, E_1)$ で $G(V, E)$ の v_α に対応する個体を v_1 とし，$G_2(V_2, E_2)$ で $G(V, E)$ の v_β に対応する個体を v_2 とする．$G_1(V_1, E_1)$ と $G_2(V_2, E_2)$ をただ 1 つの辺 (v_1, v_2) で繋いだグラフ $G'(V', E')$ を考える．$G'(V', E')$ に対して，$G_1(V_1, E_1)$ と $G_2(V_2, E_2)$ でそれぞれ実行 Ξ で C_t に到達するまでと同様の交流を行い，その後は $G'(V', E')$ が全体公平を満たすように交流する実行 Ξ' を考える．Ξ が全体公平であることから，以下がいえる．Ξ' では,

- Ξ で C_t 以降に v_β が持つ状態を v_2 が持つ限り，v_1 は状態を変えない.
- v_1 が状態を変えない限り，Ξ で C_t 以降に v_β が持つ状態以外を v_2 は持たない.

これらの事実から，Ξ' で v_1 は状態を変化させず，Ξ で C_t 以降に v_β が持つ状態以外を v_2 は持たない．よって，Ξ' で，$G_1(V_1, E_1)$ の各個体は $G_2(V_2, E_2)$ の存在に気付けず，$G_2(V_2, E_2)$ の各個体も $G_1(V_1, E_1)$ の存在に気付けない．したがって，収束状況の V に対して，収束状況の V_1 と V_2 では，$\#red(V) = \quad \#red(V_1) = \#red(V_2)$ かつ $\#blue(V) = \#blue(V_1) = \#blue(V_2)$ である．$G(V, E)$ は奇数であるため，その収束状況では $\#red(V) - \#blue(V) = 1$ もしくは $\#blue(V) - \#red(V) = 1$ である．よって，Ξ' の $G'(V', E')$ では色の差が 2 となる状況が無限

に続く．これは，矛盾．

以下から詳細な証明を示す．個体数 $n < P/2$ が奇数の任意のグラフ $G(V, E)$ を考える．ここで，$V = \{v_1, v_2, v_3, \ldots, v_n\}$ とする．背理法から，ある実行 $\Xi = C_0, C_1, C_2, C_3, \ldots$ で有限回しか状態を遷移させない個体 v_α が存在するとする．仮定から，実行 Ξ には v_α がそれ以降にある状態 p から状態を遷移させない無限回発生する収束状況 C_t が存在する．C_t では一般性を失わず $\#red(V) - \#blue(V) = 1$ とする．また，v_α に隣接する任意の個体を v_β とする．

つづいて，以下のようなグラフ $G'(V', E')$ を考える．

- $V' = \{v'_1, v'_2, v'_3, \ldots, v'_{2n}\}$ である．さらに，$V'_1 = \{v'_1, v'_2, v'_3, \ldots, v'_n\}$, $V'_2 = \{v'_{n+1}, v'_{n+2}, v'_{n+3}, \ldots, v'_{2n}\}$ とする．
- $E' = \{(v'_x, v'_y), (v'_{x+n}, v'_{y+n}) \in V' \times V' \mid (v_x, v_y) \in E\} \cup \{(v'_\alpha, v'_{n+\beta})\}$ である．

$G'(V', E')$ に対して，以下のような実行 $\Xi' = C'_0, C'_1, C'_2, C'_3, \ldots$ を考える．

- $i \leq t$ に対して，v_x と v_y が $C_i \to C_{i+1}$ で交流するとき，v'_x と v'_y が $C'_{2i} \to C'_{2i+1}$ で交流し，v'_{x+n} と v'_{y+n} が $C'_{2i+1} \to C'_{2i+2}$ で交流する．
- C'_{2t} 以降は，全体公平性を満たすように交流する．

E' の定義から，V'_1 の 2 個体間、V'_2 の 2 個体間にはそれぞれ E と同様に辺が存在するため，このような実行は可能である．実行の定義から，ある $1 \leq x \leq n$ に対して，$s(v'_x, C'_{2t}) = s(v'_{x+n}, C'_{2t}) = s(v_x, C_t)$ である．

C'_{2t} 以降の任意の状況 C'_m に対して，$\forall v'_x \in V'_1$ と $\forall v'_y \in V'_2$ に関してそれぞれ $s(v'_x, C'_m) = s(v_x, C_a)$ である C_a と $s(v'_{y+n}, C'_m) = s(v_y, C_b)$ である C_b が Ξ で無限回発生していることを状況のインデックスを使った帰納法で示す．仮定から，そのような C'_m では常に $s(v'_\alpha, C'_m) = s(v'_{\alpha+n}, C'_m) = p$ が成り立つ．

まず，$m = 2t$ のとき，つまり，C'_{2t} のときは，C_t が条件を満たすため成り立つ．つづいて，$C'_m (m \geq 2t)$ のとき成り立つとして，C'_{m+1} で成り立つかどうか考える．$C'_m \to C'_{m+1}$ での交流が $v'_\alpha, v'_{\beta+n}$ 間の交流である場合とそうでない場合の二つの場合を考える．まず，$v'_\alpha, v'_{\beta+n}$ 間の交流でない場合を考える．$v'_i \in V'_1, v'_j \in V'_2$ であるような v'_i, v'_j 間に存在する辺は $(v'_\alpha, v'_{\beta+n})$ のみである．よって，$C'_m \to C'_{m+1}$ で交流する個体 v'_i, v'_j は v'_i, $v'_j \in V'_1$ か v'_i, $v'_j \in V'_2$ である．仮定から，$\forall v'_x \in V'_1$ と $\forall v'_y \in V'_2$ に関してそれぞれ $s(v'_x, C'_m) = s(v_x, C_a)$ である C_a と $s(v'_{y+n}, C'_m) = s(v_y, C_b)$ である C_b が Ξ で無限回発生している．v'_i, $v'_j \in V'_1$ ならば，C_a で v_i, v_j が交流することで条件を満たす $C_{a'}$ に遷移可能である．v'_i, $v'_j \in V'_2$ ならば，C_b で v_{i-n}, v_{j-n} が交流することで条件を満たす $C_{b'}$ に遷移可能である．Ξ は全体公平性を満たすため，そのような $C_{a'}$ と $C_{b'}$ は Ξ で無限回発生するため，この場合は成り立つ．つづいて，$v'_\alpha, v'_{\beta+n}$ 間の交流である場合を考える．仮定から，$s(v'_{\beta+n}, C'_m) = s(v_\beta, C_b)$ を満たす C_b が Ξ で無限回発生するため，v_α は状態を C_t 以降に遷移させないことから遷移規則 $(s(v_\beta, C_b), p) \to (s, p)$ が存在する (s は任意の状態)．よって，v'_α が $v'_{\beta+n}$ と交流した後も v'_α は状態 p を維持する．したがって，$\forall v'_x \in V'_1$ に関して $s(v'_x, C'_{m+1}) = s(v'_x, C'_m) = s(v_x, C_a)$ であり，そのような C_a は Ξ で無限回発生している．また，仮定から $s(v'_\alpha, C'_m) = s(v'_{\alpha+n}, C'_m) = p$ であるため，$v'_{\beta+n}$ は v'_α と交流したとき，$v'_{\alpha+n}$ と交流したときの遷移と同様の遷移を行う．よって，仮定から $\forall v'_y \in V'_2$ に関し

て $s(v'_{y+n}, C'_m) = s(v_y, C_b)$ である C_b が Ξ で無限回発生しているため，そのような C_b で v_α, v_β が交流したとき条件を満たす $C_{b'}$ に遷移可能である．Ξ は全体公平性を満たすため，そのような $C_{b'}$ は Ξ で無限回発生する．

したがって，$\forall v'_x \in V'_1$ と $\forall v'_y \in V'_2$ に関してそれぞれ $s(v'_x, C'_m) = s(v_x, C_a)$ である C_a と $s(v'_{y+n}, C'_m) = s(v_y, C_b)$ である C_b が Ξ で無限回発生している．

この事実と C_t 以降に $\#red(V) - \#blue(V) = 1$ であることから，C'_t 以降は $\#red(V'_1) - \#blue(V'_1) = 1$ かつ $\#red(V'_2) - \#blue(V'_2) = 1$ である．よって，C'_t 以降は $\#red(V') - \#blue(V') = 2$ を常に満たす．Ξ' は全体公平性を満たすため，これは矛盾． □

つづいて，補題 1 から，状態数 3 で半数分割問題を解くアルゴリズムが存在しないことを示す．

Alg が取りうる状態集合を $S = \{s_1, s_2, s_3\}$ とする．ここで, 一般性を失わず, $f(s_1) = f(s_2) = red$, $f(s_3) = blue$ とする．個体数 $n < P/2$ が奇数の任意のグラフ G に対する Alg の実行 Ξ を考える．補題 1 から, Ξ のある状況 C_t 以降に各個体は状態を無限回別の状態に遷移させる．よって，C_t 以降の収束状況以降にも状態を無限回別の状態に遷移させるため，状態 s_3 を持つ個体も red の状態に遷移する．これは，収束状況であることと矛盾．

4 結論

本稿では，完全グラフ，基地局あり，弱公平，初期値一定の仮定の下での半数分割問題の可解性を明らかにした．この結果と先行研究の結果を合わせて，完全グラフの場合の全仮定の組合せについて，可解性を完全に明らかにした．

また，任意グラフの場合，多くの仮定で可解性を解明した．

今後の課題として，以下の解明が考えられる．

- 今回解明できなかった残りの設定についての空間計算量
- 分割数が任意の場合の可解性
- 分割問題と他の問題 (リーダー選挙やカウンティングなど) との関係
- 分割問題の時間計算量

参考文献

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

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

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

[4] Sudo, Y., Ooshita, F., Izumi, T., Kakugawa, H., Masuzawa, T.: Logarithmic expected-time leader election in population protocol model. arXiv preprint arXiv:1812.11309 (2018)

[5] Yasumi, H., Ooshita, F., Yamaguchi, K., Inoue, M.: Constant-space population protocols for uniform bipartition. the 21st International Conference on Principles of Distributed Systems (2017)

ビッグデータを用いた為替予測問題について

羽柴和摩[1] 和田幸一[2]

法政大学大学院 理工学研究科 応用情報工学専攻[1]

法政大学 理工学部 応用情報工学科[2]

概要 機械学習の中でも時系列分析に向いている LSTM を用いて外国為替価格の変動予測を行う．実際の現場で設定されている取引時間から問題を定義し，最適解を導く．本研究では，為替の変動データに対し，どのようにアプローチをかけ，最適な形で LSTM に入出力させればより精度の高いものになるのかを考え，検証する．

1.はじめに

ビッグデータとは，従来のデータベース管理システムなどでは記録や保管・解析が難しいような巨大なデータ群のことである．明確な定義があるわけではなく，企業向け情報システムメーカーのマーケティング用語として多用されている．多くの場合，ビッグデータとは単に量が多いだけでなく，様々な種類・形式が含まれる非構造化データ・非定型的データであり，さらに，日々膨大に生成・記録される時系列性・リアルタイム性のあるようなものを指すことが多い．今までは管理しきれないため見過ごされてきたそのようなデータ群を記録・保管して即座に解析することで，ビジネスや社会に有用な知見を得たり，これまでにないような新たな仕組みやシステムを産み出す可能性が高まるとされている．

時系列データを入力に用いたニューラルネットワーク(Neural Network : NN)を構築する際には，過去のデータを記憶しておく必要があるため，隠れ層の計算結果を自身に再び入力する再帰的な構造をもったリカレントニューラルネットワーク(Recurrent Neural Network : RNN)を用いる[5]．しかし，長期間の時間依存性が失われてしまうデメリットを抱えているので，内部に値を保持するためのメモリセルを導入した Long Short-Term Memory(LSTM)と呼ばれるものにニューロンを置き換える．NN は隠れ層の数や学習回数(epochs)を増やすことでより細かく解析することができる．

本研究のこれまでの成果としては，外為オプションから定義した外為レート予測問題を定義し，それに対してどのような購入をすれば利益が得られるのかを自作のアルゴリズムで検証してきた[2][3][4]．また，機械学習を用いて波形そのものを予測することも行なった[1]．今回は LSTM の隠れ層の数や epochs などのパラメータを調整し，より精度の高い予測を目的とする．用いるデータは Google BigQuery を用いて GMO クリック証券で実際に使われている為替取引データである．取得済みの 1 年分のデータで解析と予測を行い，精度を検証する．

以降, 2 章で外為レート予測問題について定義する. 3 章と 4 章では RNN と LSTM の構造について紹介する. 5 章で解析に用いているデータについて説明する. 最後に, まとめと今後の展望について述べる.

2.外為レート予測問題

GMO クリック証券では 1 回の取引区間が 3 時間となっており, 1 日に 10 回設けられている. 3 時間のうちに何回でも購入と売却を繰り返すことができるが, 空売りすることはできず, 終了 2 分前まで取引を行うことができる. 取引画面を以下の図 1 に示す.

図 1:GMO クリック証券の取引画面

任意の基準値を選択するとその基準値に応じた円安・円高の購入・売却価格が表示される. 外国為替の変動予測を解くに当たって, 以下のように問題を定義する.

$T_n = \{0,1,2,\dots,n,n+1,n+2\}$とし,外為レート関数を $f_x: T_n \to R+$とする. ここで, $R+$は正の実数の集合である. 外為レート予測問題は, 時刻$t(0 \leqq t \leqq n)$において, ある基準レートvに対し,

$$f_x(n+2) \geqq v(\text{レート安})$$
$$f_x(n+2) < v(\text{レート高})$$

これらのうち, どちらになるかを予測する問題である. この問題は以下の 2 つのパターンが存在する.

(1) 購入のみを行う
(2) 購入と売却の両方を行う

外為レート予測問題は基準レートvに対して, 時刻$0 \sim n$の間に購入と売却を行い, 利益を最大にする問題である. 購入・売却には円安・円高の 2 種類が存在し, 以下のように購入・売却価格関数を定義する. また, 購入可能な単位は 1 とする.

円安購入関数

$$B_L(t, f(t), v): T_n \times R + \times R + \to [0..2]$$

円高購入関数

$$B_H(t, f(t), v): T_n \times R + \times R + \to [0..2]$$

円安売却関数

$$S_L(t, f(t), v): T_n \times R + \times R + \to [0..2]$$

円高売却関数

$$S_H(t, f(t), v): T_n \times R + \times R + \to [0..2]$$

このとき, 以下の式が成り立つ.

$$\begin{cases} B_L(t, f(t), v) + S_H(t, f(t), v) = 2 \\ B_H(t, f(t), v) + S_L(t, f(t), v) = 2 \end{cases}$$

次に利益について考える. 時刻tにおいて, 基準レートvで円安購入し, 時刻$s(t < s \leq n, s \neq t+1)$した場合の利益$P_L(t, s, f, v)$は以

下のようになる．

$$P_L(t,s,f,v) = S_L(s,f(s),v) - B_L(t,f(t),v)$$

満期$s = n+2$のとき

$$P_L(t,s,f,v) = \begin{cases} 2 - B_L(t,f(t),v) \cdots (f(n+2) \geq v) \\ -B_L(t,f(t),v) \cdots (f(n+2) < v) \end{cases}$$

円高の場合も同様に定義する．時刻tにおいて，基準レートvで円高購入し，時刻$s(t < s \leq n, s \neq t+1)$した場合の利益$P_H(t,s,f,v)$は以下のようになる．

$$P_H(t,s,f,v) = S_H(s,f(s),v) - B_H(t,f(t),v)$$

満期$s = n+2$のとき

$$P_H(t,s,f,v) = \begin{cases} 2 - B_H(t,f(t),v) \cdots (f(n+2) < v) \\ -B_H(t,f(t),v) \cdots (f(n+2) \geq v) \end{cases}$$

これら 2 つの最大値を利益Pとし，これを最大化する．

3.RNN

通常の NN では時系列データの処理をすることができないので，隠れ層の状態を再帰的に入力する RNN が考案された[5]．これにより，RNN は時刻tにおける隠れ層の値が時刻tの入力だけでなく，時刻$t-1$の隠れ層の値も用いて計算される．これによって過去のデータを考慮した学習がなされる．

RNN の概要を以下の図 2 に示す．

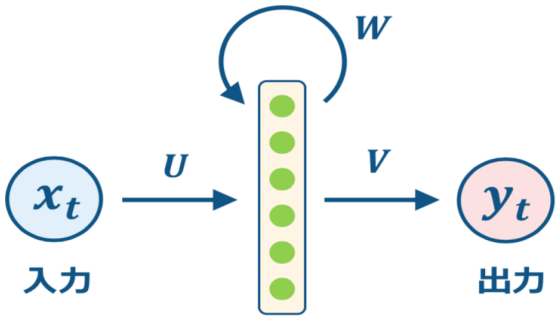


図 2:RNN の概要

時刻tにおける入力x_tが係数行列Uを通過し，隠れ層へ伝達される．また，係数行列U, Vは隠れ層に戻るときと出力されるときに通過する係数行列である．このとき，時刻tにおける隠れ層の値h_tと出力y_tは以下のように定義される．

$$h_t = \tanh\big(W(h_{t-1}) + U(x_t)\big)$$
$$y_t = V \cdot h_t$$

これらを何層も重ねて RNN を構成する．入力データのサイズに合わせて RNN を展開することによって，通常の NN と同様に誤差逆伝播法を適用することができる．以下の図 3 に展開した RNN の概要を示す．

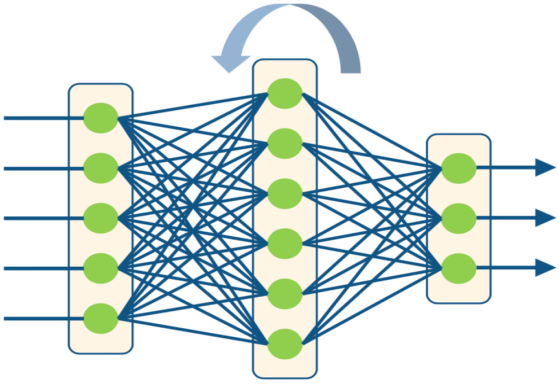


図 3:展開した RNN の概要

4.LSTM

通常の RNN では 1 つ前の隠れ層の値を再帰的に入力していたが，この方法では入力データが長くなると計算時間が極端に長くなってしまう欠点がある．それに加え，深いネットワークになると誤差逆伝播のアルゴリズムでは勾配が消失(発散)する問題が生じてしまう[7]．また，離れたデータ間で誤差が拡散してしまい，うまく相関が取れないといった問題も挙げられる．これらの問題を解決するため，隠れ層をデータの記憶に特化したユニットである Long Short-Term Memory(LSTM)に置き換えることが提案された[6]. LSTM の概要を以下の図 4 に示す

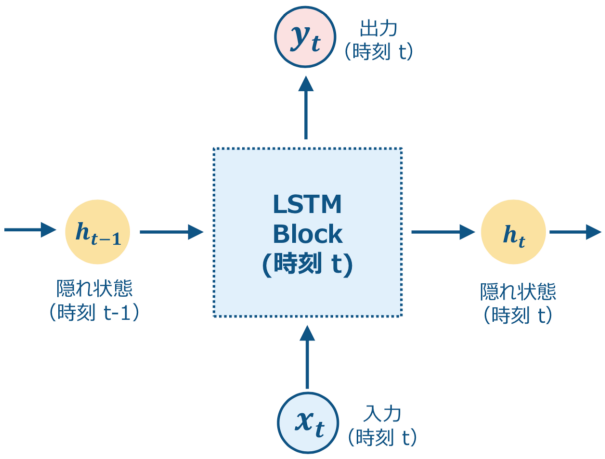


図 4:LSTM の概要

見かけ上は通常の RNN と同じだが，内部に入力ゲート，出力ゲート，忘却ゲートの 3 つのゲートと値を保持するメモリセルを持った構造をしている．それぞれのゲートの役割と内部構造をそれぞれ以下の表 1 と図 5 に示す.

表 1:LSTM 内の役割

名称	役割
入力ゲート	入力された情報をどの程度受け取るかを決める
出力ゲート	保存した情報をどの程度出力するかを決める
忘却ゲート	メモリセル内に過去の情報をどの程度残すかを決める
メモリセル	過去の情報を保持する

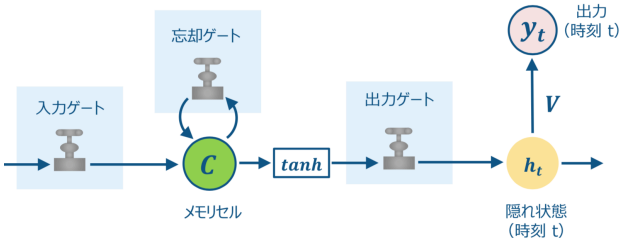


図 5:LSTM の内部構造

LSTM を用いた時系列予測の関連研究に RNN の改良として，係数行列に単位行列(Identity matrix)を用いる IRNN と呼ばれる手法がある[6]． 通常の RNN よりも離れたデータ間での相関が得られやすいとされている．これは活性化関数に Rectified Linear Unit(ReLU)関数を用いており，シンプルな構造になっている．しかし ReLU 関数の特性として演算結果がマイナス値になった場合には 0 を返してしまうので内部での演算が楽になる反面，重要な値を取りこぼしてしまう可能性がある．

5.用いるデータ

本研究では GMO クリック証券で実際に使われている為替取引データを用いている．このデータを Google 社の Big Query で取得

している．取得データが持っているパラメータを以下の図 6 に示す．

Row	time	open	high	low	close	created
1	2017-11-01T00:00:00	113.434	113.444	113.432	113.438	2017-11-01 01:10:01 UTC
2	2017-11-01T00:01:00	113.438	113.469	113.436	113.461	2017-11-01 01:10:01 UTC
3	2017-11-01T00:02:00	113.461	113.469	113.454	113.465	2017-11-01 01:10:01 UTC
4	2017-11-01T00:03:00	113.465	113.474	113.46	113.466	2017-11-01 01:10:01 UTC
5	2017-11-01T00:04:00	113.466	113.486	113.466	113.478	2017-11-01 01:10:01 UTC
6	2017-11-01T00:05:00	113.479	113.512	113.478	113.506	2017-11-01 01:10:01 UTC
7	2017-11-01T00:06:00	113.506	113.52	113.506	113.513	2017-11-01 01:10:01 UTC
8	2017-11-01T00:07:00	113.513	113.526	113.509	113.514	2017-11-01 01:10:01 UTC

図 6:Google BigQuery での取得データ

Google BigQuery で取得できるデータは 1 分刻みとなっており，以下の 5 つの項目を取得することができ，パラメータの詳細を以下の表 2 に示す．本研究では入力値に open を用いている．

表 2:データのパラメータ

パラメータ	内容
time	データの時刻
open	xx 時 xx 分 00 秒の価格
high	xx 時 xx 分 00 秒～xx 時 xx 分 59 秒の最高値
low	xx 時 xx 分 00 秒～xx 時 xx 分 59 秒の最低値
close	xx 時 xx 分 59 秒の価格

6.解析結果と今後の展望

LSTM を用いた RNN を構成し，1 年分のデータを入力として読み込ませた．LSTM の層数を 500 にして epochs とバッチサイズ(データの入力サイズを決めるパラメータ)を変えて解析・予測を行い，結果として 1 ヶ月分の予測出力された波形を以下の図 7, 8 に示す．(横軸がデータ数，縦軸が価格．青線が予測波形，橙線がテストデータ．)

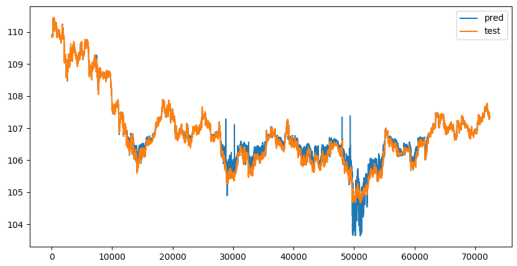


図 7:epochs = 1000, batch_size = 100

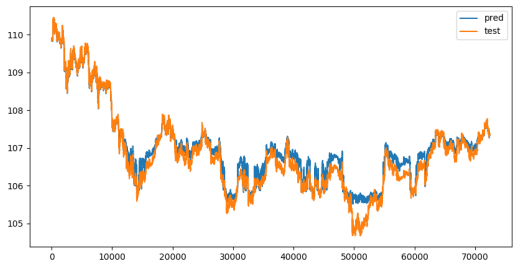


図 8:epochs = 5000, batch_size = 200

予測値とテストデータの誤差の計測には二乗平均平方根誤差(RMSE)を使用している．RMSE は損失関数として多くの場面で使われており，最小二乗法に対して最適である．

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(T_i - y_i)^2}$$

T_i:テストデータ
y_i:予測値
n:データ数

学習時のログから，RMSE としては小さい 0.04 まで下がったことが確認できた．しかし，青線の波形と橙線の波形がほとんど

一致してしまっているので過学習してしまった．よって，今後の展望として以下の方針を定める．

・過学習してしまったので Early Stopping を導入しておく．
・データの区切り方を変えて，現状より精度の高い予測を行う．(5日分のデータで1日を予測や，1ヶ月分のデータで1週間を予測等)
・季節変動や世界経済の影響を大きく受けてしまうので対策を考える．
・定義した問題にフィットするように波形の出力を調整する．

7.参考文献

[1] 羽柴和摩 法政大学理工学部応用情報工学科 卒業論文 「Deep Learning を用いた外国為替の変動予測に関する研究」 2018 年 1 月 27 日

[2] 後藤将仁 法政大学理工学部応用情報工学科 卒業論文 「外為レート予測問題を解決するオンラインアルゴリズムについて」 2018 年 1 月 27 日

[3] 小林裕次郎 法政大学理工学部応用情報工学科 卒業論文 「オンラインアルゴリズムを用いた外国為替の価格変動の予測に関する研究」 2018 年 1 月 27 日

[4] 高橋愛恵 法政大学理工学部応用情報工学科 卒業論文 「外国為替レート予測問題に対するオンラインアルゴリズムについて」 2019 年 1 月 26 日

[5] 巣籠悠輔 著 『詳解 ディープラーニング Tensorflow・Keras による時系列データ処理』 マイナビ出版 2017 年 5 月 25 日

[6] 水川徳之，松原崇，上原邦昭 「再帰を用いた深層学習による時系列データの学習」 情報処理学会関西支部 支部大会 2015 年

[7] 新納浩幸 著 『Chainer V2 による実践深層学習』 オーム社 2017 年 9 月 20 日

A Parallel Branch-and-Bound Method using MapReduce and HBase

Ryo Yahagi*, Yonghwan Kim†, and Yoshiaki Katayama‡

Department of Computer Scienc of Graduate School of Engineering,
Nagoya Institute of Technology, Aichi, Japan
*yahagi@moss.elcom.nitech.ac.jp, {†kim, ‡katayama}@nitech.ac.jp

Abstract: A *combinatorial optimization problem* is a problem to find the solution in a large discrete set of states such that an objective function is optimized (maximized or minimized). There are many combinatorial optimization problems, e.g., 0-1 Knapsack problem, Traveling salesperson problem, etc., and many studies are presented to solve these problems efficiently. These problems can be modeled as *a state space* which is a set of states that a problem can be in. It is clear that if we can check all the states in a state space, the optimal solution (optimal state) can be easily found, however, to check all the states usually requires an unacceptable computational cost.

A *branch-and-bound* is a general technique to search a state space efficiently. It basically searches branches of tree which represents the subsets of states, but before searching, it calculate the upper or lower bound of the branch to decide searching or not. Even a branch-and-bound can drastically decrease the searching time, if a state space becomes extremely large, it also requires a huge amount of computational cost.

As a naive solution, a branch-and-bound can be parallelized by searching branches concurrently. However, even the number of states in every branch is the same, the actual searching number of state hardly becomes same because many of states are locally dumped by a branch-and-bound. If the nodes constantly exchange their information among them to parallelize, the total number of searching becomes small, but an enormous communication cost becomes necessary. This implies that a load balancing among nodes is difficult in a parallel branch-and-bound, thus the utilization among nodes is a very important issue to parallelize a branch-and-bound effectively.

In this paper, we propose a framework to parallelize a branch-and-bound using *MapReduce* [1], which is distributed programming model based on Hadoop. The proposed system can adjust the amount of both the (local) computation and communication to optimize the performance of a parallel branch-and-bound. These adjustment can be determined by two parameters α and β, which represent the amount of local computation and communication respectively. Moreover, we use HBase to share information among nodes because the nodes cannot exchange any information during the execution of MapReduce. As a result, we show that our system consisting of five nodes realizes at most 4.01 speed-up of the execution time to solve 0-1 Knapsack Problem when two parameters are determined by the preliminary experiment.

Keywords: MapReduce, HBase, Branch-and-Bound, Parallelization, Optimization Problem, State Space Search

1 Introduction

Combinational Optimization problem [5, 6] is a discrete optimization problem to find the optimal solutions within a discrete set of possible solutions. Combinatorial optimization problem is usually modeled as *a state space* consisting of a large number of states, and a state space forms a graph, especially a tree. One famous example of a combinatorial optimization problem is *0-1 Knapsack Problem*, where the value of the goods put into the knapsack with limited capability has to be maximized. In this case, the number of states in the state space becomes $\mathcal{O}(2^n)$, where n is the number of items, thus, an exhaustive searching of the state space requires an unacceptable computational cost.

A branch-and-bound is a well-known general technique for an effective searching of a state space. In every step in a branch-and-bound method, the most promising state is determined as its next examining state among all possible states from its current states. This determination is called *branch* and this can be realized based on the potential value, called *bound*, which is the upper or lower bound of the feasible solutions in the subtree rooted at the current examining state.

A branch-and-bound can be parallelized by searching branches concurrently using many computational nodes, however, even if the state space is evenly partitioned into the subsets with the same size, the actual number of searched states hardly becomes the same, because many of states may be discarded by a local branch-and-bound. This may occur a bad load balancing among the nodes which causes a performance degradation. Even if the nodes constantly exchange the information among them for an effective branch-and-bound, it may cause a huge amount of communication cost, consequently, the entire performance decreases. These implies that a load balancing to increase utilization among nodes between the local computational cost and the communication cost among nodes is important issue to realize a parallel branch-and-bound.

In this paper, we propose a framework to parallelize a branch-and-bound using MapReduce [1], which provides a simple implementation for designing a parallel algorithm by implementing a *map* and a *reduce* method in a very affordable manner. The proposed system can adjust the amount of the local computation and the communication by changing two system parameters α and β respectively. We aim for an realization of an effective parallel branch-and-bound using the appropriate values of these two parameters. Moreover, we use HBase, which is a distributed database based on Hadoop Distributed File System (HDFS), to share some information among the nodes because the nodes cannot exchange any information among them during the execution of MapReduce. Finally, as a case study, we implement the proposed system and evaluate the performance to solve the combinatorial optimization problem, 0-1 Knapsack Problem, with some input data sets. And we show that the proposed system can realize an efficient parallel branch-and-bound in this case study, when the parameters are set to the good values.

The rest of this paper is organized as follows: we introduce some related works in Section 2. And Section 3 presents some preliminaries of our work. Our proposed system is described in Section 4 and the implementation of the proposed system and experimental evaluations are given in Section 5. We conclude our work and give some future works in Section 6.

2 Related Work

L. Barreto et al. present a comparion of an extened version of the regular branch-and-bound algorithm with a new parallel implementation using OpenMP and MPI in [3]. In this paper,

[3] shows that at most 9.1 times and 2.1 times speed-up in the experimental results using their implemented system with MPI and OpenMP respectively consisting of 100 processes.

Mahmoud M. Ismail et. al. proposed a parallel branch-and-bound algorithm for solving large scale integer programming problems [2] using a supervisor process which controls multiple process sets. The proposed algorithm in [2] consists of four hierarchical processes, supervisor, master, sub-master, and worker to realize a parallel branch-and-bound. The result of [2] showed the proposed algorithm improves the efficiency in time at most 7 times comparing with the master-worker algorithm.

J. M. Jansen and F. W. sijstermans proposed a parallel algorithm for branch-and-bound designed to run on MIMD machines [8]. In [8], the authors mainly focus on the load-balancing to improve the efficiency of a parallel algorithm making the idle node to examine the different sub-tress of the state space tree, and consider 0-1 Knapsack Problem to evaluate the proposed algorithm. As a test for the performance of the algorithm, the branch-and-bound algorithm for the knapsack problem is programmed in POOL and tested on a simulator for the DOOM machine. The result in [8] has an almost linear speed-up for a small number of processors (up to 32) and the speed-up slowly goes to saturate for a large number of them.

3 Preliminaries

3.1 Apache Hadoop

Hadoop is a distributed processing framework for reliable, scalable, and distributed computing. Hadoop consists of a single master node and many worker nodes. Hadoop basically includes some core components and there are many related projects. One of the components is *Hadoop Distributed File System (HDFS)* which is a software-based distributed file system that provides high-reliability and high-throughput access to application data. Another component is MapReduce which is a programming framework that is used for writing applications to process huge amount of data on large clusters.

MapReduce has two computational steps, map phase and reduce phase. In map phase, each worker node performs sorting, filtering, or some special computation defined by user, and sends the result to the nodes in reduce phase. In reduce phase, some worker nodes performs a summary operation such as counting, summation, or calculating average value using the output data of map phase, and write the results to HDFS (if necessary).

3.2 Apache HBase

Apache HBase, which is one of the related projects of Hadoop, is distributed database working on HDFS. HBase stores each data record as Key/Value pairs which is a basic data unit of MapReduce. All the records are sorted based on their Key in the lexicographically order. HBase provides strictly consistent reads and writes. It is well-suited for faster read and write operations on large datasets with high throughput and low input/output latency.

4 Proposed Method

4.1 Basic Strategy

In this subsection, we present a basic strategy for parallelize a branch-and-bound. To realize a branch-and-bound, every worker node has to know the (global) configuration at all time, however, this requires a huge communication cost. On the other hand, if there is no communication among worker nodes, every worker node has to completely search the subtree, this may cause a performance decrement due to bad load balancing.

Our strategy is as follows: at first, master node partitions the state space into some subtrees and each worker node searches the different subtree. Each worker node periodically put some unexamined states (middle-states) to HBase and these states may be examined by the other worker nodes which becomes idle. Not only the middle-states, but the current optimal value is also periodically put to HBase for an effective branching by sharing the optimal value. Figure 1 illustrates an overview of the proposed system.

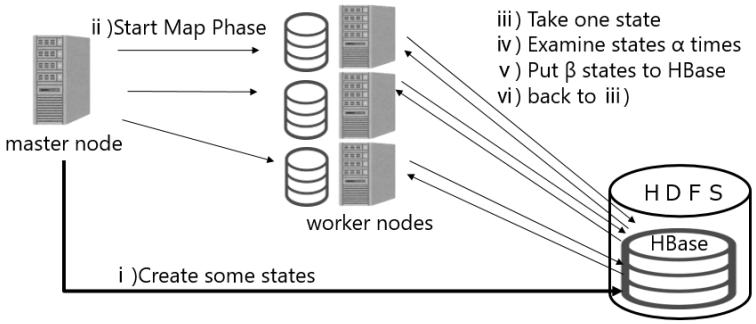


Figure 1: An overview of the proposed method

4.2 Master Node

In our proposed method, one master node executes the following steps:

1. Read the data set.
2. Create the root state (i.e., the root node) of the entire state space tree T.
3. Examine the states of T from the root state until the number of the (un-examined) states becomes larger than the number of worker nodes.
4. Put all the unexamined states to HBase.
5. Send start messages to all the worker nodes.

4.3 Worker Node

Each worker node executes the following procedures when it receives a start message from its master node.

1. Get the state from HBase. Note that the state becomes the most promising state in HBase. This step is operated atomically with mutual exclusion by the method of HBase.

2. If there is no (promising) state in both HBase and its local queue, which is a priority queue based on the potential value of the states), wait for the other (executing) worker nodes until they finish. If no other promising state is provided even all the worker nodes becomes idle, terminate the algorithm.

3. Put the state to its local queue.

4. Pick one state from its local queue, and examine the state, i.e., generates the child states frome the state. After that, put all the generated child states which are more promising that its current local optimal (may not be the global optimal) solution to its local queue again. This step will be repeated α times.

5. Write its updated local optimal solution to HBase (if necessary). And if there is the better solution written by another worker node in HBase, update its local optimal solution to the solution in HBase.

6. Put the β states to HBase to share the states with the other worker nodes.

7. Return to the first step.

There are two important parameters, α and β. α is the number of executions to examine the state: when α becomes larger, each worker node locally examine many states, and otherwise, each worker node communicates with HBase frequently. This means that a larger α causes a lot of local computations (even some of them are unnecessary computations) with a little communications, whereas a smaller α causes the frequent communications for effective examinations of the states. β is the number of the states which are put to HBase at last of every execution. This implies that β becomes larger, each worker node shares many states with the other worker nodes in every execution.

5 Experimental Evaluations

5.1 Experiment Environment

We arrange cluster consisting one master node and five worker nodes. Tables 1 and 2 represent the specification of master node and worker node respectively, and we use Hadoop verison 2.9.2 and HBase version 2.2.0 at all nodes in the cluster. We implement the prototype system based on the proposed method. And we consider a well-known combinatorial optimization problem, 0-1 Knapsack Problem for experimental evaluations. As the input of the problem, we arrange five data sets as Table 3. *Necessary number of search* in Table 3 means the total number of the examined states to find the optimal solution using a branch-and-bound on a single node.

5.2 Preliminary Experiments for Two Parameter α and β

Before the performance evaluations, we determine two important parameters by some preliminary experiments., Figure 2 illustrates the effect of parameter α. X-axis represents the value of α, and Y-axis shows the execution time to solve 0-1 Knapsack Problem with five data sets.

Table 1: Specification of master node

CPU	Intel(R) Xeon(R) (4 cores / 4 threads)
RAM	8GB
Storage	128GB (SSD)
OS	CentOS 7.4.1810

Table 2: Specification of worker node

CPU	Intel Celeron (2 cores / 2 threads)
RAM	4GB
Storage	500GB (SSD)
OS	CentOS 7.4.1708

Table 3: Input data sets for 0-1 Knapsack Problem

Data set	input60	input80	input90	input110a	input110b
Number of items	60	80	90	110	110
Necessary number of search	2,301,419	3,840,643	24,771,399	49,765,046	146,109,001

When α is small, the execution time is becomes very high in any input data sets. On the other hand, execution time which is exceeding a value goes down drasticallly, and the time converges. So, we can gain knowledge that big enough α is good. The reason is that when α is small, worker nodes communicate with HBase frequently and this causes the high communication cost. Otherwise, parameter α becomes large, the communication cost decreases, as a result, the execution time also decreases. Even the amount of the local computation increases when α is high, the execution time hardly changes because the number of the states is finite. This implies that each worker node may finish the searching of the subtree less than parameter α times.

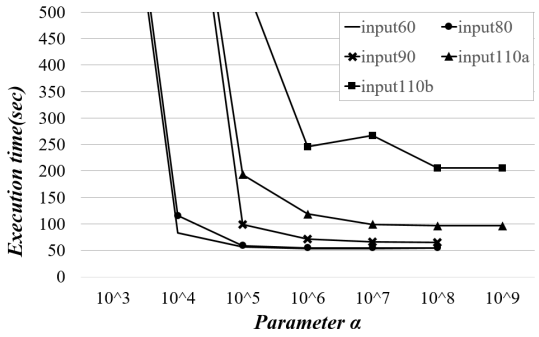


Figure 2: Effect of parameter α

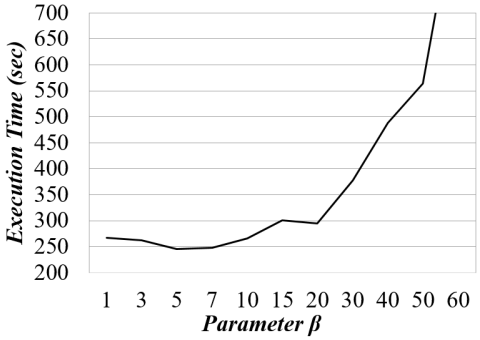


Figure 3: Effect of parameter β

Figure 3 represents the effect of parametr β when parameter α is is fixed to 1,000,000. And this experiment is evaluated using input110b as its input. Parameter β becomes larger, the time execution time significantly increases. A notable result is that when β is the same number of the worker nodes, the execution time becomes the shortest. The reason is that if β is too big, there are many useless states in HBase and increase communication cost.

5.3 Performance Comparison between single node and cluster

Now we evaluate the performance comparison between a single node and a cluster. Figure 4 represent the execution times to solve 0-1 Knapsack Problem when value of α is 1,000,000 and

β is 5 from the preliminary experiments in Section 5.2.

X-axis represents the data sets, and Y-axis shows the execution time to solve 0-1 Knapsack Problem on a single node and a cluster. The result of a cluster is partitioned into two parts, the black part represents the actual computational time to solve the problem, and the gray part describes the overhead of the MapReduce which is required to maintain the resource, or broadcast the program to every worker node, and so on. Note that the overhead of MapReduce is almost fixed regardless the size of the input or processing data. This means that this overhead hardly changes even the input data becomes larger. From the results, our proposed method is at most 4.01 times faster than single node when the input data set is input110a without the fixed-time overhead. Note that there is no result of input110b on a single node because a single node could not solve the problem due to the lack of the memory.

Table 4 represents the total number of the examined states to solve the problem by the cluster. Even if the cluster searchs the more states comparing with a single node (at most 114.92% states when input110b), the actual execution time becomes shorter.

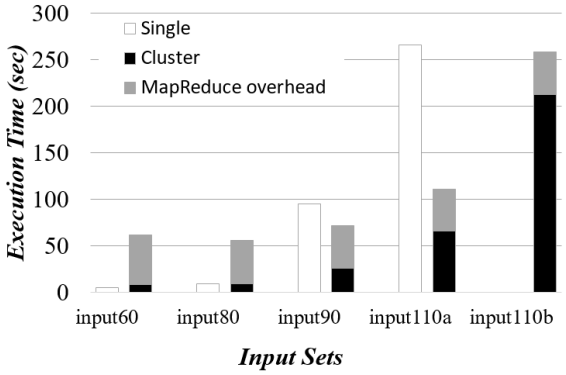


Figure 4: Perfomance comparison between single node and cluster

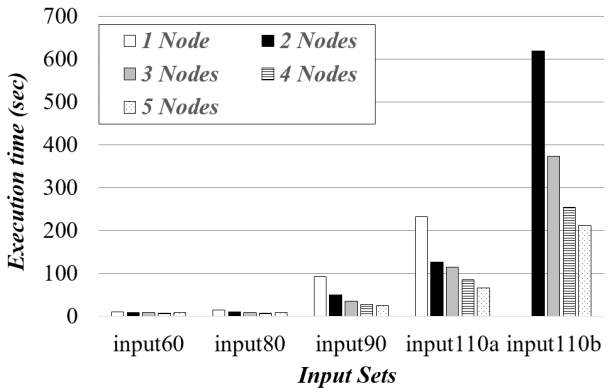


Figure 5: Scalability of our proposed method

Table 4: The total number of the examined states by the cluster

Data set	input60	input80	input90	input110a	input110b
Number of examined states	2,399,460	4,213,910	25,739,168	565,14,926	167,908,934

5.4 Scalability

In this subsection, we evaluate the scalability of our proposed method. Figure 5 describes the execution times when the number of the nodes in the cluster changes. Note that the execution time included fixed time preparing map phase by Master Node.

When the input data set is small, input60 or input80, there is no noticeable difference among the clusters. However, the input data set becomes larger, the difference among the clusters becomes significant. When the input is input110b (the largest input in this experiment), the execution time on five nodes is fastest, and one-node cluster cannot solve the problem. This result shows that the proposed method is scalable.

6 Conclusion and future Works

In this paper, we proposed a parallel branch-and-bound method using MapReduce and HBase. In the proposed system, each worker node locally searches the subtree and periodically shares necessary information with the other worker nodes to perform a branch-and-bound technique. We implemented the prototype system consisting one master node and six worker nodes for experimental evaluations, and we showed at most 4.01 speed-up of the computational time to solve 0-1 Knapsack Problem when the input data set is large. Moreover, we confirmed that our proposed system is scalable.

In this paper, we consider only 0-1 Knapsack Problem, however, there are many other combinatorial optimization problems. Hence we have to solve many other problems using the proposed system to evaluate the performance in various cases. The scalability of the proposed system consisting of more than five worker nodes is another future work. Finally, we have some other ideas so that it may improve the performance of the system: to use some distributed (priority) queue services instead of HBase, to add new parameter γ which presents the frequency of the sharing information during local searching, and so on.

References

[1] Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM - 50th anniversary issue, Volume 51 Issue 1, January 2008 Pages 107-113*

[2] Mahmoud M. Ismail, Osama abd el-raoof, and Waiel F. Abd EL-Wahed, A Parallel Branch and Bound Algorithm for Solving Large Scale Integer Programming Problems, *Applied Mathematics & Information Sciences, Vol.8, No.4, pp. 1691-1698* (2014)

[3] Lucio Barreto and Michael Bauer, Parallel Branch and Bound Algorithm - A comparison between serial, OpenMP and MPI implementations, *J. Phys.: Conf. Ser. 256 012018* (2010)

[4] R. Neapolitan and K. Naimipour, Foundations of Algorithms, *The Fourth Edition, Jones and Bartlett Publishers* (2009)

[5] C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, *Odver Publications, ISBN0486402584* (1998)

[6] M. Z. Zgurovsky and A. A. Pavlov, Combinatorial Optimization Problems in Planning and Decision Making – Theory and Applications, *Springer, ISBN3319989766* (2018)

[7] T. G. Crainic, et. al., Parallel Combinatorial Optimization, Ch.1. Parallel Branch-and-Bound Algorithms, *Wiley* (2006)

[8] J.M. JANSEN and F.W. SIJSTERMANS, Parallel Branch-and-Bound Algorithms, *Future Generation Computer Systems, Vol. 4, Issue 4, pp. 271-279* (1989)

[9] R. Neapolitan and K. Naimipour, Foundations of Algorithms, *The Fourth Edition, Jones & Bartlett Publishers* (2009)

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest , Clifford Stein, Introduction to Algorithms, *The Third Edition, The MIT Press* (2009)

[11] The Hadoop Distributed File System, Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (2010)

非同期自律分散ロボット群に対する集合問題について

亀田 勇気[1] 和田 幸一[2]
[1]法政大学大学院理工学研究科応用情報工学専攻
[2]法政大学理工学部応用情報工学科

概要：定数ビットの記憶領域(ライト)を持つ，非同期モデルにおける複数の自律分散ロボットによる集合問題について考察している．ライトを持たない基本的なロボットモデルにおいては，集合問題はたとえ半同期モデルであっても可解でないことが知られている．しかし，複数の色を表示できるライトを持つことで集合問題が可解となる場合がある．自身と他のロボットのライトを観測できる場合(full-light)，非同期モデルで移動性が non-rigid の場合，１０色のライトを用いることで集合問題を可解にできる．ここで，non-rigid とはロボットが計算した目的地に到達しない可能性があることであり，この場合，少なくとも最小移動距離 $\delta>0$ は移動する．計算された目的地に必ず到達する場合を rigid という．

本稿では，非同期モデルに制限を与えたクラスを提案し，そのクラスにおいて，full-light，rigid，2 色のライトで集合問題が可解になるアルゴリズムを示す．

1. はじめに

複数の計算機群がそれぞれに計算し，互いに通信を行うことで全体としてある問題の解決を達成するシステムを分散システムと呼ぶ．その中で自律的に動作する複数のロボットが協調的に動作することにより全体でひとつの問題を達成する自律分散ロボット群の研究が盛んである．

この自律分散ロボット群においては全体を効率的に動作させるアルゴリズムの設計が重要とされている．研究を行うにあたって，ロボットは文献[1]で紹介される理論モデルとして扱われる．ロボットは平面上を動く点として考えられる．加えて，ロボットは外見によって識別することはできず，ロボット群に対する集中制御はない．また他のロボットに情報を伝える直接の通信手段はないものと仮定する．

ロボットは任意の時刻において動作と待機を行うことができる．ロボットが動作すると，Look 命令，Compute 命令，Move 命令の 3 つの命令サイクルを実行する．Look 命令では，自身のセンサを使用し周囲のロボットを観測し，ロボットの位置座標をスナップショットとして保存する．Compute 命令では，Look 命令で得たスナップショットを入力としてアルゴリズムを実行し，次の目的地の座標を計算する．そして Move 命令では，計算された座標に向かって移動する．Look 命令と Compute 命令は瞬間的に実行されるものとするが，Move 命令はロボットの移動速度に応じた時間がかかるものとする．この移動速度に一貫性はないものとする．スナップショットはサイクル実行毎に削除され，ロボットは過去の履歴を持たないものとする．これを無記憶性という．

ロボットは3 つの命令を実行する時刻を決定するスケジューラに従って動作する．各命令の同期の程度によって3 つのスケジューラを定義することができる(図 1)．全てのロボットの命令を行う時刻が共通であるものを全同期(FSYNC)，FSYNC と同様に命令を行う時刻は共通であるが，サイクルを実行しないロボットの存在を1 台以上許すものを半同期(SSYNC)，全てのロボットが独立して動作しており，同期の仮定を考えないものを非同期(ASYNC) という．

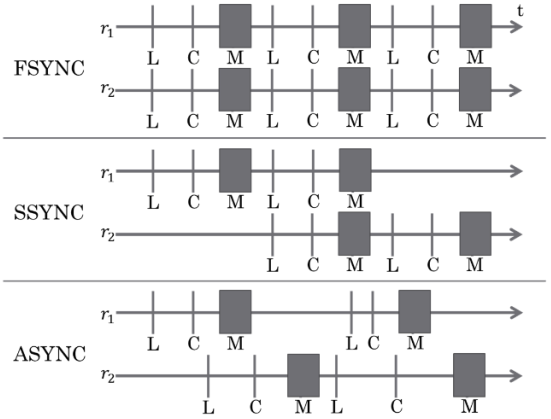


図 1．ロボットのスケジューラの実行例．
図における L，C，M はそれぞれ Look 命令，Compute 命令，Move 命令を表している．
また，t は時刻，r はロボットを表している

複数のロボットが任意の初期配置から有限時間内にあらかじめ決められていない一点に集合する問題を集合問題といい，ロボットの台数が 2 台の集合問題をランデブー問題という．本稿では，ASYNC モデルで集合問題を可解とするために必要最低限の能力や仮定について考察している．集合問題(ランデブー問題)は，基本的なロボットの場合，FSYNC モデルで可解であるが，SSYNC モデル，ASYNC モデルでは非可解であることが知られている[1].

そこで基本的なロボットに，自身の状態を記録できる定数ビットの記憶領域(ライト)を搭載したモデルが提案されている[2]．この状態は Look 命令で観測し，Compute 命令で更新できるものとする．自身のライトのみを観測できるものを internal-light，相手のライトのみを観測できるものを external-light, 双方のライトを観測できるものを full-light と呼ぶ(表 1).

表 1．ライトモデルを用いた場合の状態の見え方

状態の見え方	自分の状態	相手の状態
full-light	○	○
external-light	×	○
internal-light	○	×

また，ロボットの移動性についても考える．Move 命令時に，計算された目的地に確実に移動できるものを rigid という．計算された目的地にたどり着かない場合があり，少なくとも最小移動距離δ>0 は移動するものを non-rigid という．また，移動性が non-rigid であり，ロボットが最小移動距離δに関する知識を持っている場合，non-rigid(+δ)と表す[2].

ロボットが目的地を計算する際，ライトの色のみを用いて目的地を決定するアルゴリズムが存在し，それらはクラス L に属するアルゴリズムという[2，6]．すべてのロボットの初期状態が同じ状態から開始するとき，任意の同じ初期状態から開始すれば正しく動作するアルゴリズムを準自己安定という。また、任意の初期状態から開始しても正しく動作するアルゴリズムを自己安定という．

full-light の ASYNC モデルの場合，自己安定，non-rigid で 2 色のライトを用いたランデブーアルゴリズムが存在する．表 2，3 にそれぞれランデブー問題，集合問題に対する従来のアルゴリズムを示す．表 2，3 において，*が付いているものはクラス L であることを表している．自己安定であるものには(S)，準自己安定であるものには(QS) を記述している．斜線は，それよりさらに弱い仮定で可解である，もしくは，さらに強い仮定で非可解であることを示している．また，?は可解となるアルゴリズムが明らかでないことを示している．

ASYNC の集合問題に対しては，10 色の full-light で解けることがわかっている．この結果は，SSYNC における full-light の色数 k のアルゴリズムを ASYNC において色数 5k で模倣できることが示されており[9]，この結果に SYSNC における full-light で色数 2 のアルゴリズム[4]を適用することにより得られる．本稿では，各ライトモデルに対して，ASYNC において色数を最適にするようなアルゴリズムの開発を目標とする．最適な色数を達成するために，ASYNC の部分クラスを定義し，どのクラスで最適色数が達成可能かを考察する．その第一歩として，LCM サイクルにおける Compute 命令と Move 開始命令と Move 終了命令が瞬時に行われるモデル（CM-atomic と呼ぶ）においては，full-light，rigid を仮定すれば最適な色数で集合問題が解けることを示した．

表 2．ライトを持つロボットによるランデブー問題

scheduler	movement	full-light	external-light	internal-light	no-light
FSYNC					○
SSYNC	rigid			6	×
	non-rigid	2*(S)	3*(S)	3(+δ)	
LC-Atomic	rigid		3*	?	
ASYNC	non-rigid	2*(S)	4*(QS),5*(S)	?	
ASYNC	rigid	2*	12	?	
	non-rigid	2(S)	3(+δ)	?	

表 3．ライトを持つロボットによる集合問題

scheduler	movement	full-light	external-light	internal-light	no-light
FSYNC					○
SSYNC	rigid		2	?	×
	non-rigid	2	?	2(+δ)	
ASYNC	rigid	?→2(CM)	?	?	
	non-rigid	10	?	?	

2. ロボットのモデル

システムは，平面空間$\mathbb{R}^2$に存在するロボットの集合 $\mathcal{R}=\{r_1, \ldots, r_n\}$から成る．ロボット$r_i$は，ライトと呼ばれる定数ビットの記憶領域 $\ell(r_i)$を持つ．

ロボットr_iはそれぞれ，常に自身が原点となる独自の座標系を持っている．それぞれの座標系には向きや単位距離による共通の合意はないものとする．

ロボットには active と inactive の 2 つの状態がある．active の場合，以下の命令サイクル Look-Compute-Move(LCM)サイクルを実行する．

①Look 命令
センサを使用し周囲のロボットを観測し，
座標を得る．
②Compute 命令
Look 命令で得た観測結果をもとに，
移動先の座標を計算する．
③Move 命令
実際に計算された位置に移動する. Move 命令は Move 開始命令(M_B)で始まり，Move 終了命令(M_E)で終了するものと仮定する．このとき，一度の Move 命令で目的地に確実に到達できる場合，その移動性を rigid という．ロボットが一度の Move 命令で目的地にたどり着けず，その場合は少なくとも最小移動距離 δ>0 は移動するものとする. この移動性を non-rigid という. また，

移動性が non-rigid で, かつロボットが最小移動距離 δ の値に関する知識を持っている場合，その移動性を non-rigid(+δ)という[2].

LCM サイクルが終了する度に，Look 命令で得た情報は消去され，次サイクルでは使用できないものとする(無記憶).

各命令の同期の程度によって，3 種類のスケジュールが考えられている.

・FSYNC(全同期)
すべてのロボットの LCM サイクルの各命令を行う開始時刻が一致している.
・SSYNC(半同期)
FSYNC 同様，各動作は同期しているが，サイクルを実行しないロボットの存在を 1 台以上許す.
・ASYNC(非同期)
すべてのロボットが独立して LCM サイクルを実行していて，各動作が同期していない.

本稿では，ASYNC に制限を与えたクラスとして次のスケジューラを提案する．ASYNC の制限は，Look 命令(L)，Compute 命令(C)，Move 開始命令(M_B)，Move 終了命令(M_E)に対して，複数個の連続する命令を単一命令と考えることによって以下の 11 個が定義される. ①～④は 2 つの命令を，⑤～⑧は 3 つの命令を，⑨～⑪は 4 つの命令を，それぞれ単一化することによって得られる.

① LC-Atomic
各サイクルにおいて，Look 命令と Compute 命令が同時刻に実行される.
② CM_B-Atomic
各サイクルにおいて，Compute 命令と Move 命令が同時刻に実行される.
③ Move-Atomic (M_BM_E-Atomic)
各サイクルにおいて，Move 命令が瞬間的に実行される.
④ M_EL-Atomic
各サイクルにおいて, Move 命令が終了する時刻に次のサイクルの Look 命令が実行される.
⑤ LCM_B-Atomic
各サイクルにおいて，Look 命令と Compute 命令と Move 命令が同時刻に実行される.
⑥ CM-Atomic (CM_BM_E-Atomic)
各サイクルにおいて，Compute 命令と Move 命令が同時刻に実行され，かつ，Move 命令が瞬間的に実行される.
⑦ ML-Atomic
各サイクルにおいて, Move 命令が瞬間的に実行され，かつ，Move 命令と次のサイクルの Look 命令を同時に実行する.
⑧ M_ELC-Atomic
各サイクルにおいて, Move 命令が終了する時刻に Look 命令と Compute 命令が実行される.
⑨ LCM-Atomic (LCM_BM_E-Atomic)
各サイクルにおいて，Look 命令と Compute 命令と Move 命令が同時刻に実行され, かつ, Move 命令が瞬間的に実行される.
⑩ CML-Atomic (CM_BM_EL-Atomic)
各サイクルにおいて, Move 命令が瞬間的に実行され，かつ，Compute 命令と Move 命令と次のサイクルの Look 命令が同時刻に実行される.
⑪ MLC-Atomic (M_BM_ELC-Atomic)
各サイクルにおいて, Move 命令が瞬間的に実行され，かつ，Move 命令と次のサイクルの Look 命令と Compute 命令が同時刻に実行される.

①LC-atomic と③Move-atomic はこれまでにも定義されている．定義から②，④，⑦は ASYNC と，⑤，⑧は LC-atomic と同等である．⑨は SSYNC と同等である．これらの間の関係を明らかにすることが本研究の目的であるが，まず，CM-atomic において集合問題を考える.

ロボットは自身の内部状態を記録できる定数ビットの記憶領域(ライト)を搭載しているものとする. ライトの色は Look 命令で観測でき, Compute 命令で更新できるものとする．ライトの見え方によって，以下の 3 つのモデルが考えられる.
・full-light
Look 命令時に，自分と他のロボットの状態を観測できる.
・external-light
他のロボットの状態は観測できるが，自分の状態は観測できない.
・internal-light
自分の状態は観測できるが，相手の状態は観測できない.

n(≥2)台のロボットが有限時間内に，予め決められていない一点に集合する問題を集合問題といい, n=2 の場合を特別にランデブー問題という.

3. 先行研究

ランデブー問題は FSYNC モデルでは解くことができるが，SSYNC モデルでは解くことができない.

定理 1. [1] ランデブー問題は，SSYNC モデルのロボットでは解くことができない.

また，ランデブー問題は ASYNC モデルでは解くことができない.

定理 2. [1] ランデブー問題は，ASYNC モデルのロボットでは解くことができない.

ロボットにライトを持たせた場合，ランデブー問題が可解となる条件を以下の定理で示す(表 2).

定理 3. [2，5，6]

① SSYNC，full-light，non-rigid，2 状態でランデブー問題を解くクラス L に属する自己安定なアルゴリズムが存在する.
② SSYNC，external-light，non-rigid，3 状態でランデブー問題を解くクラス L に属する自己安定なアルゴリズムが存在する.
③ SSYNC，internal-light，rigid，6 状態でランデブー問題を解くアルゴリズムが存在する.
④ SSYNC，internal-light，non-rigid(+δ)，3 状態でランデブー問題を解くアルゴリズムが存在する.
⑤ ASYNC，full-light，rigid，2 状態でランデブー問題を解くクラス L に属するアルゴリズムが存在する.
⑥ ASYNC，full-light，non-rigid，2 状態でランデブー問題を解く自己安定なアルゴリズムが存在する.
⑦ ASYNC，external-light，rigid，12 状態でランデブー問題を解くアルゴリズムが存在する.
⑧ ASYNC，external-light，non-rigid(+δ)，3 状態でランデブー問題を解くアルゴリズムが存在する.

また，ロボットにライトを持たせ，ASYNC モデルに制限を与え，LC-Atomic ASYNC モデルを用いた場合，ランデブー問題が可解となる条件を以下の定理で示す(表 2).

定理 4 [5]

① LC-Atomic ASYNC，full-light，non-rigid，2 状態でランデブー問題を解くクラス L に属する自己安定なアルゴリズムが存在する.
② LC-Atomic ASYNC，external-light，rigid，3 状態でランデブー問題を解くクラス L に属するアルゴリズムが存在する.
③ LC-Atomic ASYNC, external-light, non-rigid, 4 状態でランデブー問題を解くクラス L に属する準自己安定なアルゴリズムが存在する.
④ LC-Atomic ASYNC, external-light, non-rigid, 5 状態でランデブー問題を解くクラス L に属する自己安定なアルゴリズムが存在する.

集合問題は FSYNC モデルでは解くことができるが，SSYNC モデルでは解くことができない.

定理 5. [1] 集合問題は，SSYNC モデルのロボットでは解くことができない.

また, 集合問題は ASYNC モデルでは解くことができない.

定理 6. [1] 集合問題は，ASYNC モデルのロボットでは解くことができない.

ロボットにライトを持たせた場合，集合問題が可解となる条件を以下の定理で示す(表 3).

定理 7. [4]

① SYNC，full-light，non-rigid，2 状態で集合問題を解くアルゴリズムが存在する.
② SYNC，external-light，rigid，2 状態で集合問題を解くアルゴリズムが存在する.
③ SYNC，internal-light，non-rigid(+δ)，2 状態で集合問題を解くアルゴリズムが存在する.
④ ASYNC，full-light，non-rigid，10 状態で集合問題を解くアルゴリズムが存在する.

4. ライトを持つ CM-Atomic ASYNC モデルの集合アルゴリズムについて

ASYNC を CM-Atomic ASYNC に制限した場合に，full-light，rigid，2 状態(A，B)で集合問題を解くアルゴリズムを Algorithm 1 に示す.

全てのロボットはライトの初期状態 A から開始する．全てのロボットが A であることを観測した場合，状況に応じた目的地を決定し，自身の状態を B に更新する．また，自身の状態が A であり，状態が B のロボットを観測した場合，目的地を状態が B のロボットが存在する点とし，自身の状態を B に更新する．自身の状態が B である場合，目的地を現在の自身の位置とし，状態の更新は行わない.

今回のアルゴリズムにおいては，全てのロボットが成す最小包含円を目的地決定の際に利用している．全てのロボットが A であることを観測し，最小包含円の中点にロボットが存在する場合，中点に存在するロボットは状態を B に更新し，他のロボットは状態が B のロボットを観測し，状態を B に更新し，目的地を状態が B のロボットが存在する点とする．全てのロボットが A であることを

観測し，最小包含円の中点にロボットが一台も存在しない場合，4 つのパターンを除いて，ロボットは状態を B に更新し，最小包含円の中点を目的地とする(4 つのパターンにおいては，最小包含円を用いずに目的地を決定することができる).

Algorithm 1 CM-Atomic Gathering

```
Assumptions: full-light, rigid, two colors ( A and B )
If me.light = B Then
   me.destination ← me.position
Else if me.light = A Then
   If other.light = B Then
      me.state ← B
      me.destination ← B
   Else if all others.light = A Then
      SEC: =Smallest Enclosing Circle Of All Robots;
      c: = Center of SEC;
      If there are robots at point c Then
         If I am at point c Then
            me.light ← B
      Else if no robot is at point c Then
         If there are 3 points and all 3 points are on a line Then
            If I am at the median point on the line Then
               me.light ← B
         Else if there are 3 points and the triangle contains an angle of at least 120°
            η: = the point at a vertex whose angle is at least 120°
            If I am at η Then
               me.light ← B
         Else if there are 4 points and there 3 points on a line Then
            If I am at the median point on the line Then
               me.light ← B
         Else if there are 4 points and one point is strictly inside CH Then
         CH: = Convex Hull of 4 points;
         r: = point inside CH;
            If I am at r Then
               me.light ← B
         Else
            me.light ← B
            me.destination ← c
```

定理 8. CM-Atomic ASYNC，full-light，rigid，2 状態で集合問題を解くことができる.

5. 結論と展望

本稿では, ASYNC モデルを CM-Atomic モデルに制限し，ライトモデルを full-light，移動性を rigid と仮定し，最適な状態数で集合問題を解くアルゴリズムを示した. 残された未解決問題として，移動性の制限とライトの能力を緩めた集合問題を解くアルゴリズムの開発，制限された ASYNC の部分クラス間の能力を明らかにすることなどがある.

参考文献

1. P. Flocchini，G. Prencipe，N. Santoro，Distributed Computing by Oblivious Mobile Robots，Synthesis Lectures on Distributed Computing Theory，Morgan & Claypool，2012.
2. P. Flocchini，N. Santoro，G. Viglietta，M. Yamashita，Rendezvous with Constant Memory，Theoretical Computer Science，621，57-72，2016.
3. M. Cieliebak，G. Prencipe，Gathering Autonomous Mobile Robots，In Proceedings of 9th International Colloquium on Structural Information and Communication Complexity (SIROCCO)，pages 57-72，2002.
4. Y. Katayama，S. Terai，K. Wada，Gathering Problems for Autonomous Mobile Robots with Lights，arXiv:1811. 12068，(Nov. 2018).
5. X. Defago，T. Okumura，K. Wada，Optimal Rendezvous L-Algorithms for Asynchronous Mobile Robots with External-Lights，OPODIS 2018，LIPICS，vol. 125，pp. 24:1-24:16，(Dec. 2018).
6. G. Viglietta，Rendezvous of two robots with visible bits，10th International Symposium on Algorithms and Experiments for Sensor Systems，Wireless Networks and Distributed Robotics (ALGOSENSORS)，pages 291—306，2013
7. G. Prencipe，Impossibility of gathering by a set of autonomous mobile robots，Theoretical Computer Science，384(2-3):222{231，2007.
8. X. Defago，A. Heriban，S. Tixeuil，Optimally gathering two robots，In Proceedingsof 19th ICDCN，3:1-10，2018.
9. S. Das，P. Flocchini，G. Prencipe，N. Santoro，M. Yamashita，Autonomous mobile robots with lights，Theoretical Computer Science，609，171{184，2016.

誤り許容・高バンド幅の光通信を用いた不確実容認コンピューティング

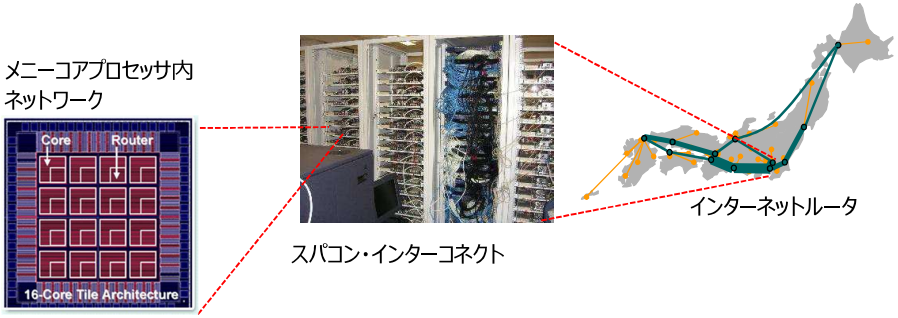
鯉渕 道紘 （こいぶち みちひろ）
国立情報学研究所

本発表は、JSPS科研費19H01106の助成とNEDO委託業務の結果得られたものです。

NII

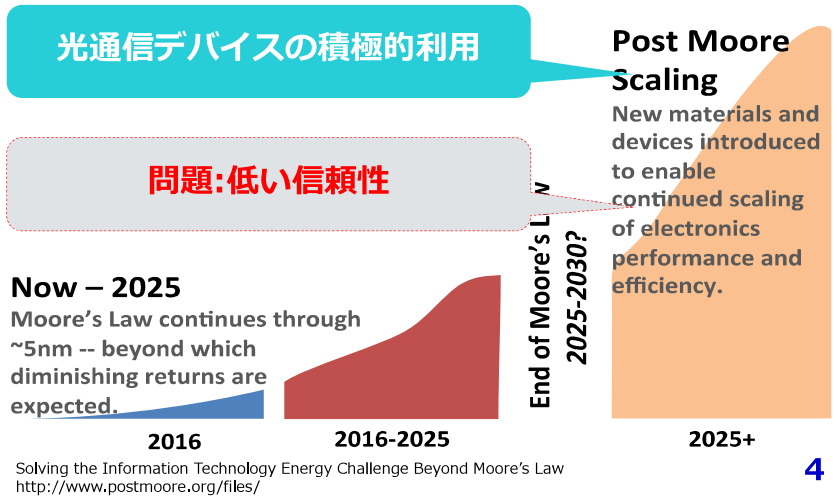
自己紹介（計算機システム・ネットワーク）

- 2005年　国立情報学研究所 鯉渕（こいぶち）研、現在に至る
- メニーコアプロセッサネットワーク (センチメートル)
  - バスからパケットネットワーク
- スパコンネットワーク (数十メートル)
- インターネットバックボーン (数百キロメートル以上)



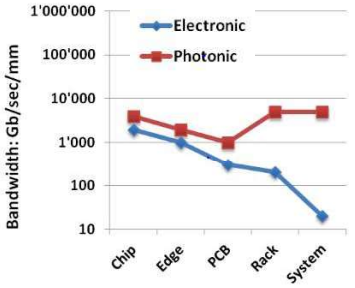
- ❑ **背景**
- ❑ 不確実容認コンピューティング
- ❑ まとめ

Old Moore's Law:
2x Transistor Density every 12-24 Months



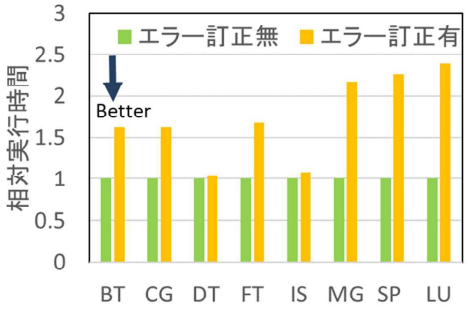
Solving the Information Technology Energy Challenge Beyond Moore's Law
http://www.postmoore.org/files/

光通信デバイスのジレンマ



既存(Electronic) vs. 光通信性能

[DOE/ASCAC TopTen Exascale Research Challenge]

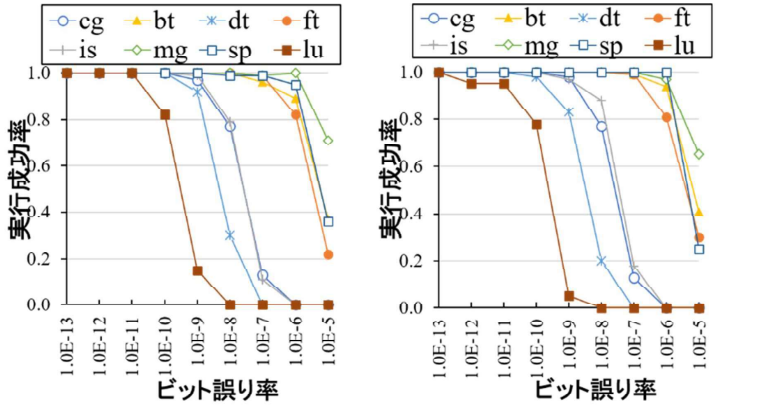


HPC並列計算性能に対するに
通信のエラー訂正の影響

[T.T.Nguyen, H.Matsutani, M.Koibuchi NCA2018]

頻発するビット化け
(冗長化/再送)
→ アプリ性能が劣化

HPCでも必要となる通信のビット誤り率は実は低い



1.ランダムエラー　　2.バーストエラー(Gilbert-Elliot model)

(a) ビット誤り率と実行成功率の解析結果(256プロセス)

[NCA18] T. T. Nguyen, H. Matsutani, M. Koibuchi, Low-Reliable Low-Latency Networks Optimized for HPC Parallel Applications, IEEE Int. Symp. on Network Computing and Applications(NCA) 2018

- ❑ 背景
- ❑ **不確実容認コンピューティング**
- ❑ まとめ

不確実容認コンピューティング

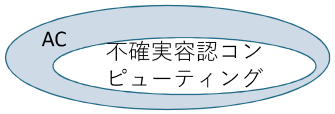
- 定義:不確実であるが高性能な計算機システムの上で、精度保証*のある計算処理を実行
  - 前述のフォトニクスを用いた計算機システムの設計技術
  - 本計算機システムの不確実な動作を許容する計算処理アルゴリズム設計
    - 精度保証とは、ある確率で正しい解もしくは一定の近似率の解が得られることを保証

	現在	2020年代後半以降
先進的並列計算	科学技術計算 精度保持	ビッグデータ解析、AI/脳 必要最低限の耐故障性
通信アーキテクチャ	エラーフリー	不確実容認 X10 性能向上 & ビット誤り容認
通信回路設計	ムーア則による集積度向上	単チップの性能向上が困難 新通信デバイスの利用

関連研究: Approximate Computing (AC)

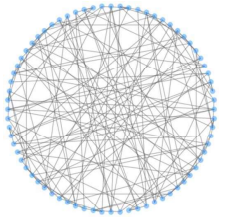
アプリケーションから要求される「正確さ」と供給される「正確さ」のギャップを埋めて
性能・エネルギー効率化を達成
[ACM Computing Surveys,2016, Mittal]

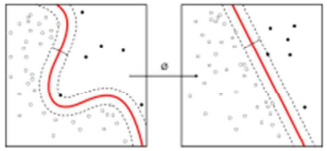
ACは近似アルゴリズム+完璧な計算システムを含む**広い概念**



不確実容認コンピューティングの計算対象

- 完全な答えを得るのは難しいが、解の判定は可能
  - 古典近似アルゴリズムの対象
    - ヒューリスティックアルゴリズム
    - 潜在的な不正確さ SAT、**巡回セールスマン問題**、GraphGolf
  - 数値計算
    - BlackScholes方程式(金融)、共役勾配法
- 全体としての結果が重要
  - データクラスタリング
  - 画像信号処理
    - FFT、デジタルフィルタ(FIR)(信号処理)
    - JPEG デコーダ

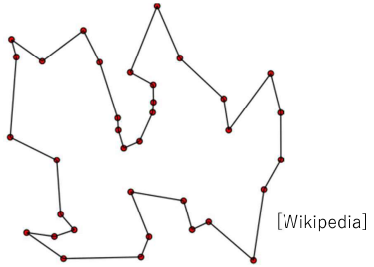




[Wikipedia, K-means clustering]

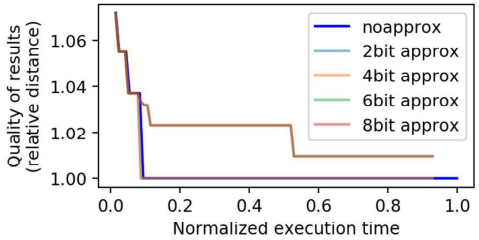
不確実容認コンピューティングの成果例

Example: Traveling Salesman Problem
"Given a list of cities and the distances between cities, what is the shortest route?"



[Wikipedia]

Ant-colony heuristics for better route
"A bit error does not affect best-known route"



[ETNET19] 丹羽、鯉渕、天野,巡回セールスマン問題の近似解法へのApprox. Computingの適用, ETNET2019(信技報)

発展:耐故障プログラミングにより幅広い計算に対する不確実容認コンピューテイグを実現可能

- チェックポイント
  - Un/coordinated + multilevel / flat
  - システムの巨大化が進むと
    - 保存に要する時間 > 平均故障間隔
- ABFT
  - 例:線形代数（行列計算）のチェックサム
  - 投機実行(課題A)を利用した検算遅延隠蔽
  - アプリ依存のロシー通信対応メカニズム (API)

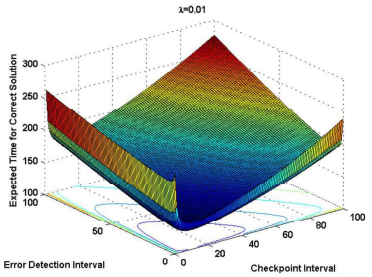
$$A^r := \begin{bmatrix} A \\ e^T A \end{bmatrix}$$
$$B^c := \begin{bmatrix} B & Be \end{bmatrix}$$
$$C^f := A^r B^c = \begin{bmatrix} AB & ABe \\ e^T AB & e^T ABe \end{bmatrix} = \begin{bmatrix} C & Ce \\ e^T C & e^T Ce \end{bmatrix}$$

	Backward Recovery	Forward Recovery
一般的な方法	チェックポイント	多重化
特化した方法		**ABFT、反復計算**

例:Fault-tolerant CG法
チェックポイントとABFTを効率良く併用

1 : Compute $r^{(0)} = b - Ax^{(0)}, z^{(0)} = M^{-1}r^{(0)}, p^{(0)} = z^{(0)}$, and $\rho_0 = r^{(0)^T} z^{(0)}$ for some initial guess $x^{(0)}$
2 : checkpoint: A, M, and b
3 : for $i = 0, 1, \ldots$
4 : if ((i>0) and ($i\%d = 0$))
5 : if ($\frac{p^{(i+1)^T} q^{(i)}}{||p^{(i+1)}||.||q^{(i)}||} > 10^{-10}$
or $\frac{||r^{(i+1)} + Ax^{(i+1)} - b||}{||A||.||x^{(i+1)}|| + ||b||} > 10^{-10}$)
6 : recover: A, M, b, i, ρ_i, $p^{(i)}$, $x^{(i)}$, and $r^{(i)}$
7 : else if ($i\%(cd) = 0$)
8 : checkpoint: i, ρ_i, $p^{(i)}$, and $x^{(i)}$
9: endif
10: endif
11: $q^{(i)} = Ap^{(i)}$
12: $\alpha_i = \rho_i / p^{(i)^T} q^{(i)}$
13: $x^{(i+1)} = x^{(i)} + \alpha_i p^{(i)}$
14: $r^{(i+1)} = r^{(i)} - \alpha_i q^{(i)}$
15: solve $Mz^{(i+1)} = r^{(i+1)}$, where $M = M^T$
16: $\rho_{i+1} = r^{(i+1)^T} z^{(i+1)}$
17: $\beta_i = \rho_{i+1} / \rho_i$
10: $p^{(i+1)} = z^{(i+1)} + \beta_i p^{(i)}$
19: check convergence; continue if necessary
20: end

チェックポイントの回復
オーバヘッド大



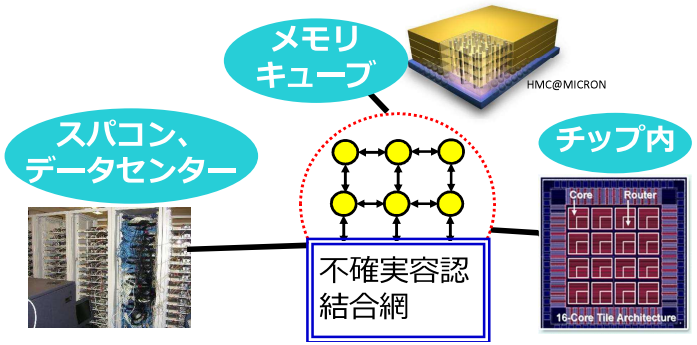
Z.Chen, Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods, PPoPP2013

- ❑ 背景
- ❑ 不確実容認コンピューティング
- ❑ まとめ

まとめと議論(1/2)

Imperfect is the New Perfect

- 高性能低信頼の通信デバイスの積極的利用
  → 不確実容認コンピューティング
- 協調設計
  - 各プログラムが耐故障性をどう有するか？(計算科学)
  - 不確実容認Comp.システムとは？(計算機科学)



まとめと議論(2/2)

1. 不確実容認コンピューティングに応用できる並列処理をご存知ないでしょうか？
近似計算、Algorithm Based Fault Tolerant計算、マルチメディア、数値計算、どのレベルのコメントもwelcome

2. 精度保証をどうするか？
ある確率で正しい解もしくは一定の近似率の解が得られることを保証
- 実行期待値は可能

Analyzing Novem, a Two-Player Multi-Stage Simultaneous Game

François Bonnet
School of Computing
Tokyo Institute of Technology
bonnet@c.titech.ac.jp

Introduction and motivation Novem is an abstract strategic game designed by Gil Druckman and published by Tactic in 2008. It won the "Årets Spel Best Adult Game" award in 2008 [1]. To the best of our knowledge, this game was not yet studied or solved.

While (board) games are usually played sequentially (e.g. Chess, Go, Hex, Poker, ...), Novem consists of simultaneous moves. The most famous simultaneous game is certainly Rock-Paper-Scissor (RPS), in which two (or more) players have to choose one of three hand gestures. RPS is trivially solved and the optimal strategy is the uniform strategy. There exist other simultaneous games that are (1) *interesting*, in the sense that they are played by real (human) players, and (2) *non-trivial*, which makes them worth studying. We know at least three such games that have already been studied; Colonel Blotto, Goofspiel, 10 000yens. The main difference between them and Novem lies in the duration of a game. These three games run for a finite number of rounds; 1, 10, and 13 round(s) respectively.[1] On the contrary, Novem is not a bounded game; ending a game may require an unbounded number of rounds. Note that, in Novem, there is no terminating rule similar to the 50-move rule which exists in Chess.

Rules of Novem The game is played using a 3×3 grid where tiles numbered from 1 to 9 are disposed on two layers (left of Figure 1 for initial board). At each round, the first player (P1) selects a row; A, B, or C, and the second player (P2) selects a column; 1, 2, or 3. Both choices are revealed simultaneously by the players. In the odd (resp. even) rounds, P1 (resp. P2) collects the visible tile located on the cell designed by the combined choices of row-column. If the designed cell is empty, no tile is collected in the round. The game ends as soon as one row or column is completely empty. The winner is the player whose sum of collected tiles is higher. Note that the game may end in a draw if both players collected the same total. Complete rules can be found on the publisher website [2].

We also consider a simplified version played with a single layer of tiles (right side of Figure 1).

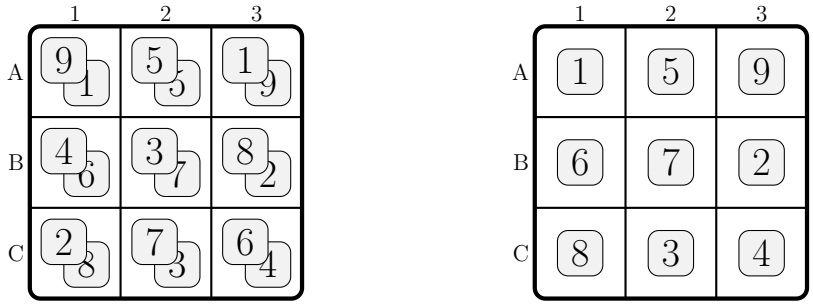


Figure 1: Initial boards of Novem (superimposed tiles are shifted to allow reading of both values)

Contributions Our goal is to *solve* the game. For Novem, it means finding *optimal strategies* and the corresponding *expected outcome*, i.e. computing Nash Equilibrium. We define the outcome as 1 if P1 wins, 0.5 if the game is drawn, and 0 if P2 wins. Our results are summarized below.

[1] Goofspiel and 10 000yens can be generalized, but the number of rounds will always be fixed.

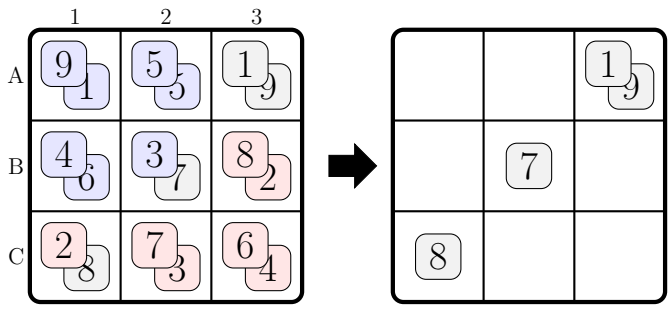


Figure 2: Deadlock configuration. Current score for P1(blue) is 33 and for P2(red) is 32.

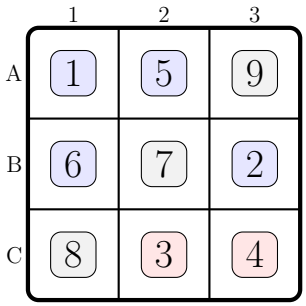


Figure 3: Configuration with irrational optimal strategies. Current score for P1(blue) is 14 and for P2(red) is 7.

1. Design issues We observed that Novem has some design issues which may lead to deadlock configurations.[2] Figure 2 depicts an execution of the game leading to a deadlock. P1(blue) collected 33 points, P2(red) collected 32 points, and there remain only four tiles on the board. To avoid an immediate loss, both players should obviously prevent their opponent to collect the tiles 7 or 8. We can also show that each player should avoid collecting the tile 1 because it would favor their opponent. The intuition is that collecting the tile 1 makes the tile 9 become visible and thus increase the winning chance of next collecting player. Therefore, no player will ever collect a tile in this configuration!

There is no such deadlock in the simplified single-layer variant, hence our decision to study it.

2. Not rationally solvable Optimal strategies cannot be expressed as rational mixing of pure strategies, not even for the single-layer version of the game. Figure 3 represents a configuration with only three tiles remaining on the board. The game ends as soon as any player collects one more tile; P1 wins if she collects any tile, while P2 wins with tiles 8 or 9, but achieves only a draw with tile 7.

Assuming P1 is next to collect, the (irrational) optimal outcome is $\frac{5+\sqrt{5}}{10} \approx 0.72$. The following (irrational) mixed strategies are optimal and unique:

- When P1 is collecting: {A:$\frac{1}{3}$, B:$\frac{1}{3}$,C:$\frac{1}{3}$} and {1:$\frac{1}{3}$, 2:$\frac{1}{3}$,3:$\frac{1}{3}$}.
- When P2 is collecting: {A:$\frac{3-\sqrt{5}}{4}$, B:$\frac{-1+\sqrt{5}}{2}$,C:$\frac{3-\sqrt{5}}{4}$} and {1:$\frac{3-\sqrt{5}}{4}$, 2:$\frac{-1+\sqrt{5}}{2}$,3:$\frac{3-\sqrt{5}}{4}$}.

Here, there is no optimal rational mixing. It is not a big problem from a theoretical point of view, but it makes exact computation much harder, that is why we computed only numerical approximations.

3. Numerical computations We computed numerical approximations of optimal strategies and expected outcome. Detailed results will appear in the longer version of the paper. For the single-layer version, when both players play optimally, the expected outcome is ≈ 0.686 which means that P1 is favored (not a surprise). With a komi of 3 ("free points" initially given to the second player), the single-layer game is almost fair; the expected outcome is approximately 0.499. With a komi of -7.5 (i.e. 7.5 points given to P1), P1 has a simple winning strategy. Conversely, to guarantee a win for P2, the game should be played with a komi of 10.5 (or $10 + \epsilon$ for any $\epsilon > 0$).

References

[1] Novem on BoardGameGeek website. `https://boardgamegeek.com/boardgame/38678/novem`. Accessed: 2019-05-24.

[2] Rule of novem. `https://web.archive.org/web/20190524073019/http://www.tactic.net/site/rules/UK/02582.pdf`. Accessed and archived: 2019-05-24.

[2] It does not really disturb real players. The game is still fun to play! But it is a problem when trying to solve the game.

Model-Checking Robot Algorithms in Euclidean Space

Xavier Défago
School of Computing
Tokyo Institute of Technology
Tokyo, Japan

jointly with

Adam Heriban
Sorbonne, LIP6
Paris, France

Sébastien Tixeuil
Sorbonne, LIP6
Paris, France

Koichi Wada
Hosei University
Tokyo, Japan

Abstract

The presentation provides an overview of recent developments on using model-checking to verify the correctness of cooperative mobile robots algorithms in a Euclidean environment.

In short, using model checking to verify an algorithm consists of the following. The behavior of the system, the algorithm, and an adversary are expressed as a transition system, typically using a dedicated language (e.g., Promela in the case of the SPIN model-checker [10]). The correctness is itself expressed as temporal logic predicates (e.g., using LTL syntax). The model checker (e.g., SPIN) checks the model by performing an exhaustive search starting from every initial state and branching at every non-deterministic choice. If all executions satisfy the correctness predicates, then the model is verified and the model-checker reports a success. In contrast, if some execution leads to a state that violates a correctness predicate, the model-checker reports as a counter-example the steps that lead to the violating state.

Model-checking was originally motivated and is often being used to verify software programs and concurrency algorithms. The approach has also been used to verify [2, 6, 7, 12] or synthesize [3, 11] algorithms for mobile robots moving in very simple graphs, such as a ring or a line. Using an automated approach for complex exhaustive proofs has helped identify (and correct) very subtle errors made in earlier work on that topic.

In ongoing work on the problem of rendezvous of robots with light, we have applied model-checking to automate the process of verifying the correctness of many algorithms in various system models (e.g., FSYNC, SSYNC, ASYNC, and variants) [5, 8]. The motivation to automate the process came from checking the tedious proofs of a rendezvous algorithm in ASYNC optimal in the number of colors [9].

By its exhaustive nature, model-checking cannot possibly deal with infinite domains. In fact, to avoid combinatorial explosion, the domain of all variables must be restricted to a very small number of discrete values. A fortiori, the verification of algorithms in Euclidean space poses the challenge of representing robots' locations with a small number of discrete values, thus leading to a huge loss of information. The method consists in defining an intermediate model (verification model) using a discrete encoding of robots' positions and proving that it conserves important properties of the original system model. The verification model is then verified by the model checker. The design of the verification model opts for a *conservative* model (i.e., a verification model that never reports success for a faulty algorithm but may instead report failure for a correct one).

In this presentation, we will review some of the key design issues that came when developing a verification model for rendezvous of two robots with light. We will then discuss what could

possibly be done to develop a more general verification model to address problems with a larger number of robots.

The model-checking approach is in contrast with the formal verification based on interactive theorem provers such as Coq or Isabelle/HOL. That approach has been used effectively to verify mobile robots in a continuous environment [4, 1]. The approach, which still involves writing the proofs manually (or at least provide tactics), should be seen as complementary to model-checking. The former provides more general results whereas the later is more automatic. The two approaches have different objectives.

Acknowledgments

This work was supported by JST SICORP Grant Number JPMJSC1606, JST SICORP Grant Number JPMJSC1806, and JSPS KAKENHI Grant Number 17K00019.

References

[1] Thibaut Balabonski, Pierre Courtieu, Robin Pelle, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Brief announcement: Continuous vs. discrete asynchronous moves: A certified approach for mobile robots. In *Proc. 20th Intl. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 404–408, November 2018.

[2] Béatrice Bérard, Pascal Lafourcade, Laure Millet, Maria Potop-Butucaru, Yann Thierry-Mieg, and Sébastien Tixeuil. Formal verification of mobile robot protocols. *Distributed Computing*, 29(6):459–487, 2016.

[3] François Bonnet, Xavier Défago, Franck Petit, Maria Potop-Butucaru, and Sébastien Tixeuil. Discovering and assessing fine-grained metrics in robot networks protocols. In *Proc. 33rd IEEE Intl. Symp. on Reliable Distributed Systems Workshops, (SRDS Workshops)*, pages 50–59, October 2014.

[4] Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Certified universal gathering in $\mathbb{R}^2$ for oblivious mobile robots. In *Proc. 30th Intl. Symp. on Distributed Computing (DISC)*, pages 187–200, September 2016.

[5] Xavier Défago, Adam Heriban, Sébastien Tixeuil, and Koichi Wada. Using model checking to formally verify rendezvous algorithms for robots with lights in euclidean space. Technical report, 2019.

[6] Ha Thi Thu Doan, François Bonnet, and Kazuhiro Ogata. Model checking of a mobile robots perpetual exploration algorithm. In *Proc. 6th Intl. Workshop on Structured Object-Oriented Formal Language and Method (SOFL+MSVL), Revised Selected Papers*, pages 201–219, November 2016.

[7] Ha Thi Thu Doan, François Bonnet, and Kazuhiro Ogata. Model checking of robot gathering. In *Proc. 21st Intl. Conf. on Principles of Distributed Systems, (OPODIS)*, pages 12:1–12:16, December 2017.

[8] Xavier Défago, Adam Heriban, Sébastien Tixeuil, and Koichi Wada. Brief announcement: Model checking rendezvous algorithms for robots with lights in euclidean space. In *DISC 2019*, October 2019.

[9] Adam Heriban, Xavier Défago, and Sébastien Tixeuil. Optimally gathering two robots. In *Proc. 19th Intl. Conf. on Distributed Computing and Networking, ICDCN*, pages 3:1–3:10, January 2018.

[10] Gerard Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.

[11] Laure Millet, Maria Potop-Butucaru, Nathalie Sznajder, and Sébastien Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In *Proc. 16th Intl. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 237–251, September 2014.

[12] Arnaud Sangnier, Nathalie Sznajder, Maria Potop-Butucaru, and Sébastien Tixeuil. Parameterized verification of algorithms for oblivious robots on a ring. In *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pages 212–219, 2017.

10秒でわかる Graph Golf

- プログラミングコンテストの一種
  - NII で 2015～毎年開催
- お題: **直径の小さいグラフを探せ！**
- 成果
  - 実在のスパコンやプロセッサの性能向上可能性を提示
  - グラフ理論の有名問題の最善解を更新
  - 高速なグラフ解析アルゴリズムの開発を促進

低遅延インターコネクトへの期待

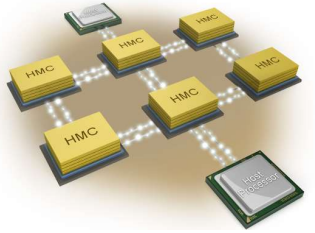
配線長　　期待される通信遅延

~10m

~1μs

K. Scott Hemmert et al, Report on Institute for Advanced Architectures and Algorithms, Interconnection Networks Workshop 2008

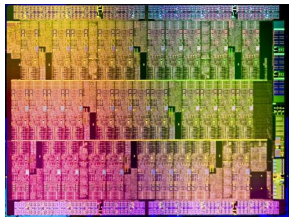
~10cm



~10ns

https://www.electronicdesign.com/memory/hybrid-memory-cube-shows-new-direction-high-performance-storage

~10mm

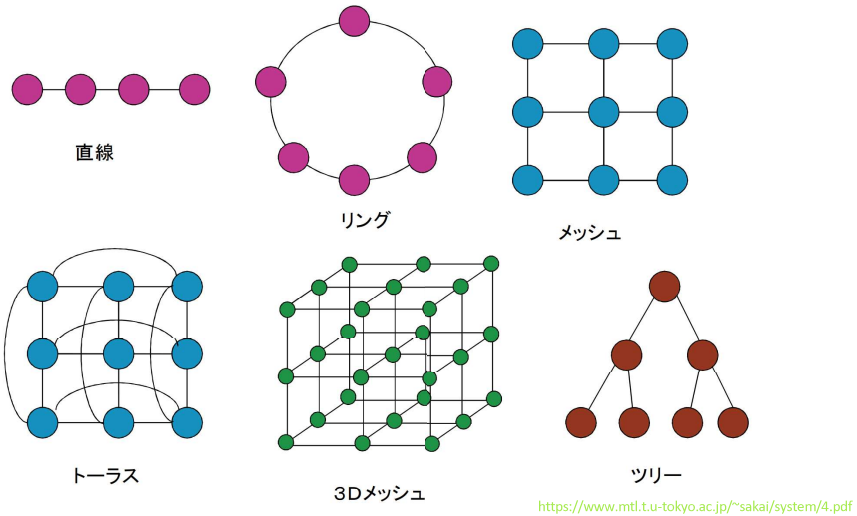


~1ns

https://www.extremetech.com/extreme/171678-intel-unveils-72-core-x86-knights-landing-cpu-for-exascale-supercomputing

相互結合網をグラフとしてモデル化

- 相互結合網の通信遅延 ∝ グラフの直径・平均距離
  - 直径の小さいグラフを見つければ、低遅延な相互結合網を作れる



https://www.mtl.t.u-tokyo.ac.jp/~sakai/system/4.pdf

直径の小さいグラフを探せ！

- Order/degree problem:

Given 頂点数 n 次数 d

Minimize 直径 k 平均距離 l

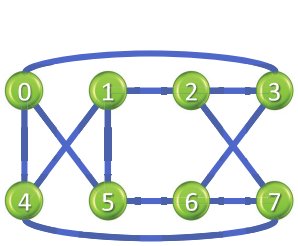
直径が小さいグラフを探す。
直径が同じ場合は、平均距離が小さいグラフを探す。

- ムーア限界（後述）にもとづく平均距離の下界 L との差

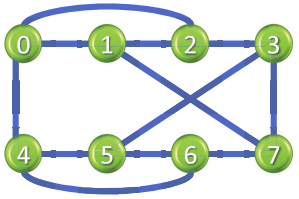
$$gap = L - l$$

例題

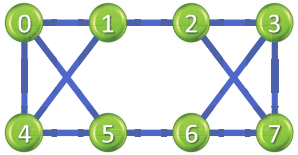
- $n = 8, d = 3$ で、直径・平均距離が最小のグラフは？



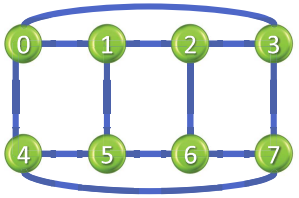
$k = 2$
$l = 1.571$
$gap = 0$



$k = 3$
$l = 1.643$
$gap = 0.072$



$k = 3$
$l = 1.786$
$gap = 0.215$



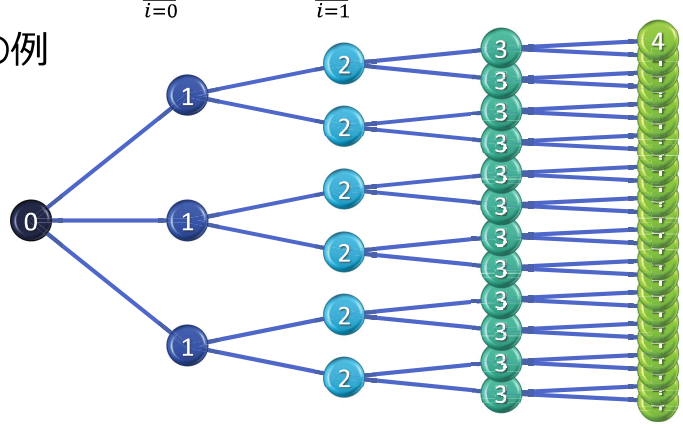
$k = 3$
$l = 1.714$
$gap = 0.143$

ムーア限界

- 与えられた直径 k, 次数 d に対する頂点数 n の**上界** $N_{d,k}$

$$N_{d,k} = \sum_{i=0}^{k} n_i = 1 + d\sum_{i=1}^{k}(d-1)^{i-1}$$

- 次数 d=3 の例



ノード数 n_k =	1	3	6	12	24
上界 $N_{d,k}$ =	1	4	10	22	46

ムーア限界にもとづく下界

- 頂点数 n, 次数 d に対する直径 k の**下界** $K_{n,d}$

$$K_{n,d} = \begin{cases} \left\lceil \frac{n-1}{2} \right\rceil & \text{if } d = 2 \\ \left\lceil \log_{d-1}\left(\frac{(n-1)(d-2)}{d} + 1\right) \right\rceil & \text{if } d > 2 \end{cases}$$

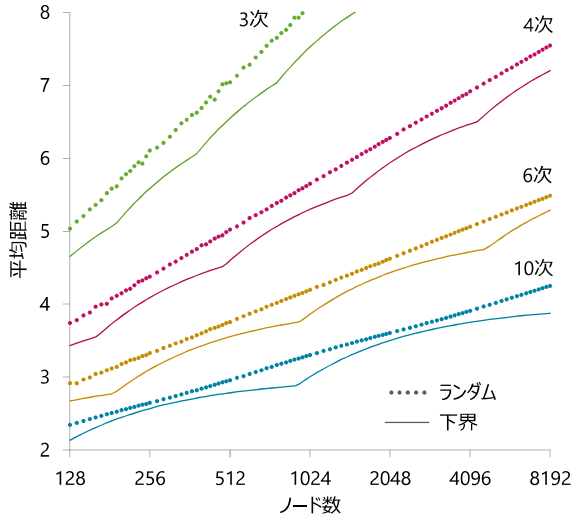
- 頂点数 n, 次数 d に対する平均距離 l の**下界** $L_{n,d}$

$$L_{n,d} = \begin{cases} 1 & \text{if } K_{n,d} = 1 \\ \dfrac{\sum_{i=1}^{K_{n,d}-1} id(d-1)^{i-1} + K_{n,d}\left(n - 1 - \sum_{i=1}^{K_{n,d}-1} d(d-1)^{i-1}\right)}{n-1} & \text{if } K_{n,d} \geq 2 \end{cases}$$

- 平均距離の**ギャップ**

$$gap = l - L_{n,d}$$

ランダムグラフじゃだめなの？



ランダムより良いグラフが存在するかもしれない

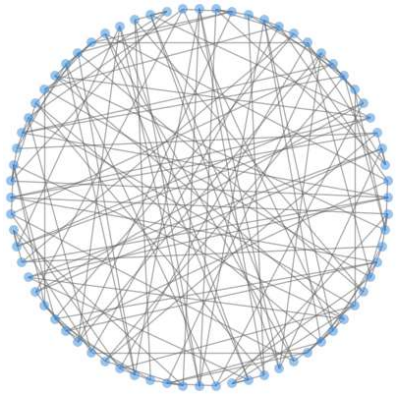
すでに誰かが解いてないの？

- 有向グラフの Order/degree problem
  - 今瀬・伊藤グラフは、直径がムーア限界より高々1だけ大きい
  - 無向グラフについて、同様の構成法は知られていない

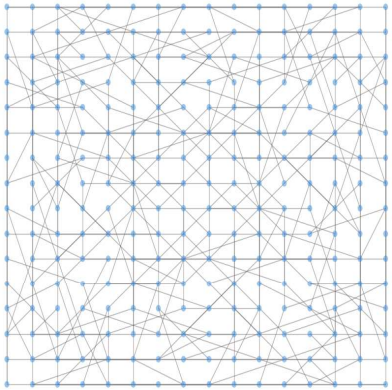
- Degree/diameter problem
  与えられた直径 k, 次数 d をもつグラフの中で、**頂点数** n が**最大**のグラフを求める問題
  - 既知の解が蓄積された Wiki がある
  - 得られる頂点数は飛び飛び。ネットワークトポロジとしては使いにくい

一般グラフと格子グラフ

- 相互結合網は実装上、配線の長さに制約がある
  - 短配線なら省電力／高クロック／安価な電気ケーブルを使える
- 格子グラフ部門を設置 (2017～)
  - 辺の長さ $\leq r$



辺の長さはマンハッタン距離
（図は見やすいように斜めに描いている）



一般グラフ部門

頂点数 n	次数 d	出題意図
16	3,4	①
32	5	①
36	3	①
64	3,4,8,16	①⑧
72	4	②
96	3	①
256	3,4,5,8,10,16,18,23	⑧
576	30	③ *Cori*
1024	3,8,11,32	⑧
1344	30	③ *Piz Daint*
1560	40	?
1800	7	?
2300	10	⑥
3019	30	③ *Cori*
3250	57	⑦
4096	3,4,16,23,60,64	⑧
4855	30	③ *Trinity*
4896	24	③ *Tianhe-2A*
9344	10	③ *Titan*
10000	3,4,7,11,16,20,23,60,64	⑧
12000	7	⑥
20000	11	⑥
40000	8	⑥
77000	6	⑥
88128	12	③ *K Computer*
98304	10	③ *Sequoia*
100000	7,11,20,32,64	④
132000	8	⑥
200000	32,64	⑤
400000	32	⑤

※一部省略

出題意図

① 入門用の小規模問題
② 実在のチップに即した問題
③ 実在のスパコンに即した問題
④ 将来のスパコンに即した問題
⑤ マシンいじめ用の大規模問題
⑥ Degree/diameter問題の記録更新を狙うもの
⑦ グラフ理論の未解決問題に挑むもの
⑧ キリの良い数

格子グラフ部門

頂点数 n	次数 d	辺長制約 r	出題意図
4×4	3	2	①
4×16	4	4	①
16×16	3	3	②
16×16	3	4	④
16×16	3	15	③
16×16	6	3	②
16×16	6	4	④
16×16	6	15	③
16×16	15	3	②
16×16	15	4	④
16×16	15	15	③
32×32	4	3	②
32×32	4	4	④
32×32	4	5	③
16×64	4	4	②
16×64	4	5	④
16×64	4	7	③
4×256	4	12	②
4×256	4	18	④
4×256	4	24	③
100×100	3	6	②
100×100	3	18	④
100×100	3	33	③
100×100	9	6	②
100×100	9	18	④
100×100	9	33	③
100×100	28	6	②
100×100	28	18	④
100×100	28	33	③

出題意図

① 入門用の小規模問題
② 辺長の制約が支配的なもの
③ ムーア限界が支配的なもの
④ 両方の制約が拮抗しているもの

賞

- 最多数の最善解を発見した人
- 最善解とは、各 n, d において直径 k が最小のグラフ
- 直径が同じ場合は平均距離 l で比較

- すべての n, d にわたって最小の平均距離ギャップを達成した人
- 事実上、平均距離ギャップ $gap = 0$ を達成した人全員

受賞者

		Widest Improvement Award	Deepest Improvement Award
2015		● Nobutaka Shimizu, Ryo Ashida, Ryuhei Mori (Tokyo Tech)	● Ryosuke Mizuno, Yawara Ishida (Kyoto Univ.)
2016		● Takayuki Matsuzaki, Teruaki Kitasuka, Masahiro Iida (Kumamoto Univ.)	● Yawara Ishida, Ryosuke Mizuno (Kyoto Univ.)
2017	一般	● Takayuki Matsuzaki, Teruaki Kitasuka, Masahiro Iida (Kumamoto Univ., Hiroshima Univ.)	● Ryuhei Mori (Tokyo Tech)
	格子	● Ibuki Kawamata (Tohoku Univ.)	● Takayuki Matsuzaki, Teruaki Kitasuka, Masahiro Iida (Kumamoto Univ., Hiroshima Univ.)
2018	一般	● Masahiro Nakao (RIKEN)	● Toru Koizumi (Univ. of Tokyo) ● Masahiro Nakao (RIKEN) ● Teruaki Kitasuka Masahiro Iida (Hiroshima Univ., Kumamoto Univ.)
	格子	● EvbCFfp1XB (unknown affiliation)	● EvbCFfp1XB (unknown affiliation)

ユニークユーザー 54名、有効投稿 2143件 （2019/7/22現在）

スペシャルゲスト

2017年ワークショップ基調講演

渡辺治先生（東工大）

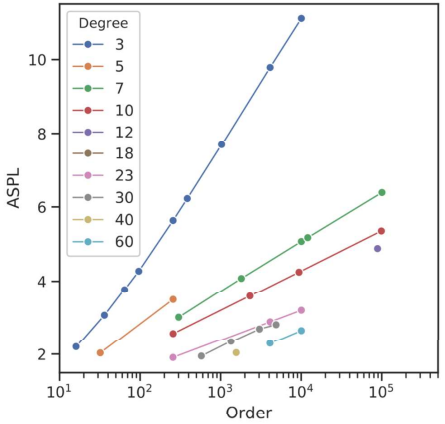
夏の電脳甲子園 SuperCon 2016 に Graph Golf ベースの問題を採用

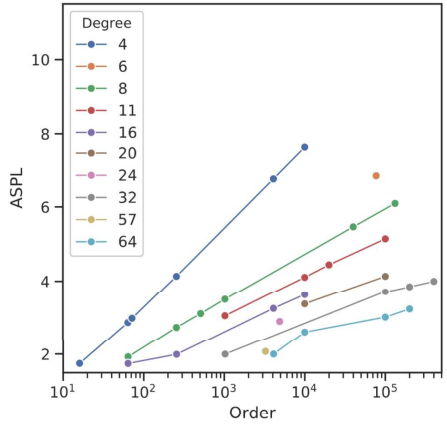
2018年ワークショップ基調講演

井上武さん（NTT研）

グラフ数え上げソフト Graphillion の作者

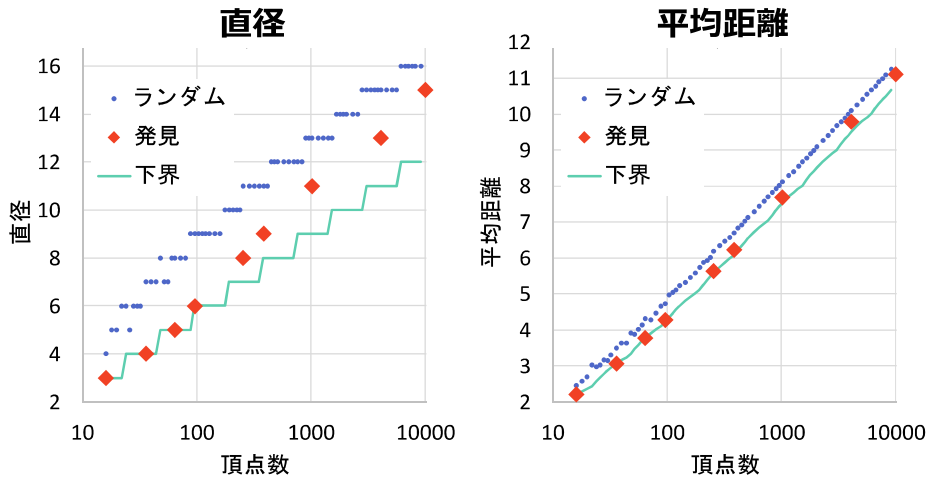
最善解の平均距離





次数３の例

※ランダムは5試行の中央値



直径は n<100 で下界に一致。平均距離も下界から約 3% 以内

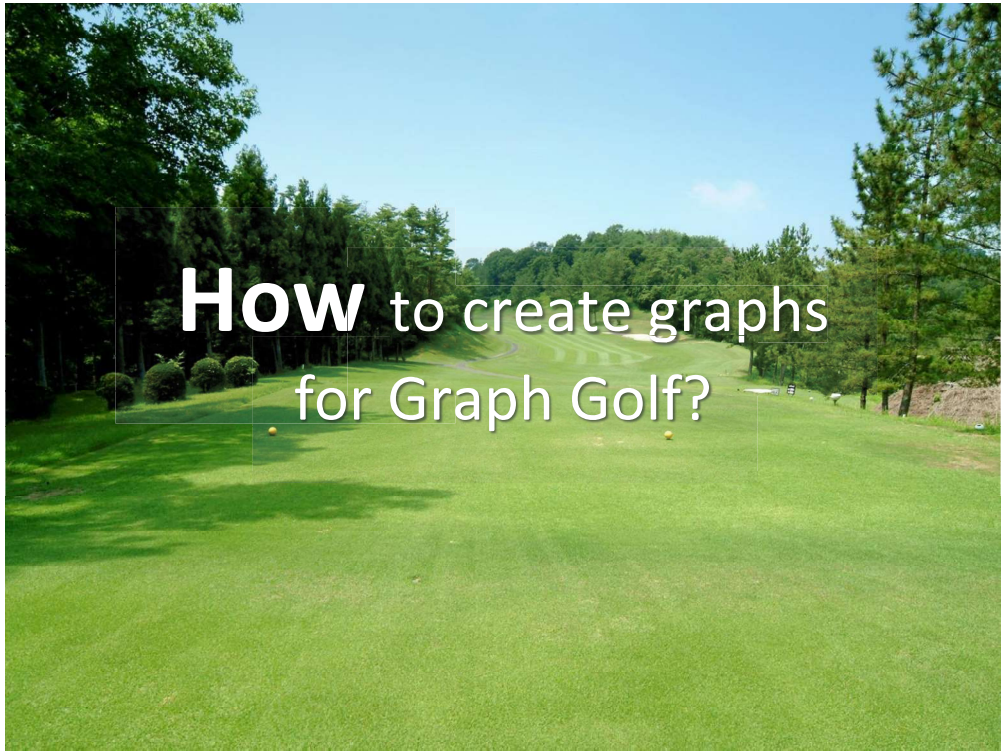
主な成果

- 実在のプロセッサに即した問題で、直径・平均距離が下界に等しいグラフを発見
  - $(n, d) = (72, 4)$: Intel Xeon Knights Landing
- 実在のスパコンに即した問題で、直径が下界に等しいグラフを発見
  - $(n, d) = (576, 30), (3019, 30)$: Cori
  - $(n, d) = (1344, 30)$: Piz Diant
  - $(n, d) = (9344, 10)$: Titan
- そのほか複数の問題で、直径・平均距離が下界に等しいグラフを発見
  - $(n, d) = (16, 3), (32, 5), (64, 16), (256, 10), (256, 23)$

副次的な成果と情報発信

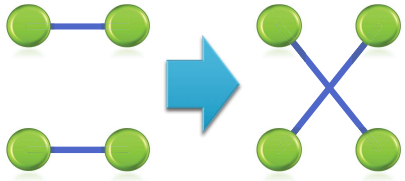
- Degree/diameter problem の既知の最善解を更新
  - $(n, d) = (2394, 10), (20468, 11), (80050, 6), (137745, 8)$
- 第22回夏の電脳甲子園 (SuperCon2016) で関連問題

- 情報発信
  - 第10回 IEEE/ACM International Symposium on Networks-on-Chip (NOCS 2016) の Graph Golf 特別セッション
  - 第15回情報科学技術フォーラム (FIT2016) の企画イベント
  - INTERNET WATCH、日刊工業新聞、科学新聞
  - 2015～2019
    - 国立情報学研究所ニュースリリース 毎年2回（開催と競技報告）
    - 電子情報通信学会情報・システムソサイエティ誌 毎年2月号（競技報告）



2-opt とメタヒューリスティクス

- 2-opt
  - 2つの辺を入れ替えてみる
  - 直径・平均距離を再計算する
  - 小さくなっていれば確定、さもなくばキャンセル

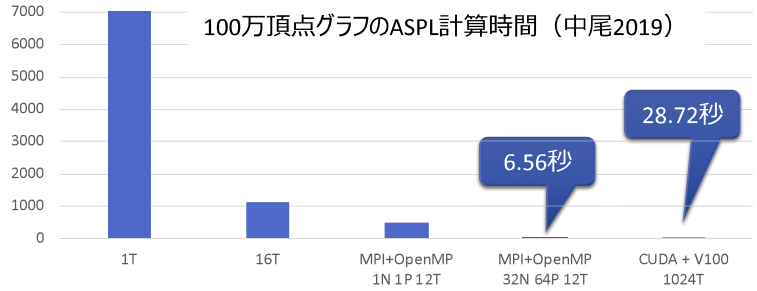


- 素朴なローカルサーチでは限界がある

  ⇒ Simulated Annealing などのメタヒューリスティクス

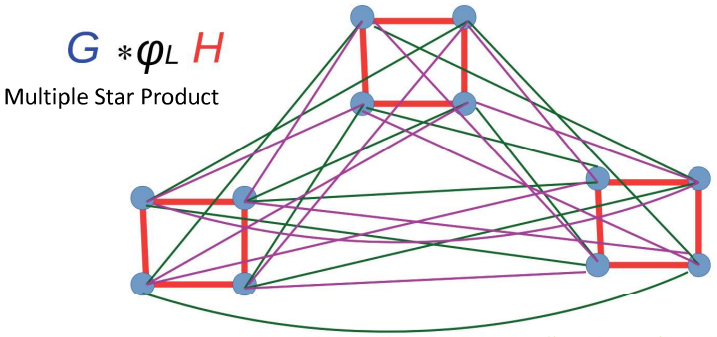
平均距離の計算を速くする

- 素直に幅優先探索すると $O(n^2 d)$
- 2-optによる変化量を $O(1)$ で概算（清水2015）
  - ただし直径3に限る
- Path count index（井上2016）
- グラフ対称性の利用（中尾2018）
- MPI+OpenMP ハイブリッド並列化（中尾2018）



既知のグラフを組み合わせる

- Multiple star product（水野2015）
- 複数の Petersen graph（北須賀2015）
- 直径 3、頂点数 2^{2N}、次数 2^N の構成法（水野2016）
- Brown's construction（松崎2017）
- Voltage graph（川又2017）



http://research.nii.ac.jp/graphgolf/2016/fit/fit2016-mizuno.pdf

開催スケジュール (2019)

5/13 00:00 UTC　　7/22 00:00 UTC　　10/14 23:59 UTC

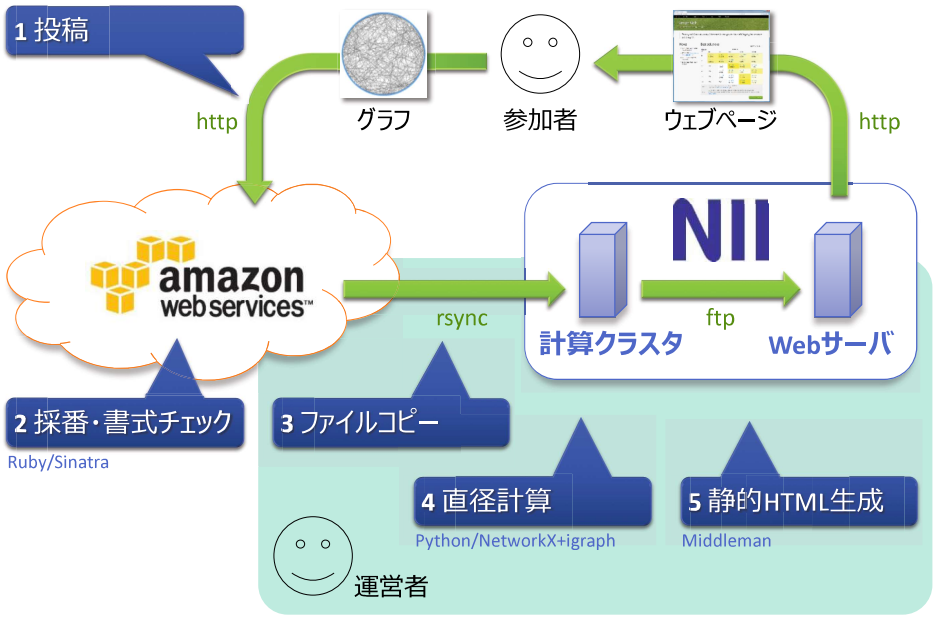
Closed submission period | **Open submission period**

- 運営が作ったランダムグラフがベースラインとして公開されている
- 投稿は非公開

- 投稿が毎週月曜日に公開される（グラフは非公開も選択可）
- 公開された解をベースに改良も可
- 同順位の場合、表彰されるのは最初に公開されたもののみ

11/26 **Workshop in CANDAR** — 表彰式・講演会

http://research.nii.ac.jp/graphgolf/



成果と展望

- Graph Golf 4年間の成果
  - 実在のスパコンやプロセッサの相互結合網に対し、直径・平均距離を削減し、さらなる低遅延化の見込みを提示
  - Degree/diameter problem の既知の最善解を更新
  - 高速な平均距離計算アルゴリズムの開発を促進
  - 1000個以上の有用なグラフを収集
    - 2019年はすでに1,245件！（累計2,000件超）
- 応用上の展望
  - 数百万～数千万コア/スパコンの登場 → 通信遅延重視が顕著に
  - 旧来のアプローチのコスト高
    - システムの複雑化 → 並列計算アプリが既存のネットワーク構造に最適化する工程の複雑化
    - トラフィックが規則的でないアプリの台頭により、小直径ネットワークが有利
  - **GraphGolf の成果が活用される日は近づいていると確信!!**

最小次数全域木問題の近似について

岡村空 北村直暉 泉泰介

本研究では，最小次数全域木問題の近似について検討する．この問題の目的は入力として頂点数 n のグラフ $G = (V, E)$ が与えられたとき，グラフ G における全域木 T の頂点の最大次数を最小にすることである．
既存研究において，最小次数全域木問題は NP 完全であることが知られているが，最適な木の次数を Δ^* としたとき，$O(mn\alpha(m, n) \log n)$(m はグラフ G における辺の本数) で $\Delta^* + 1$ の全域木を求める集中型のアルゴリズムが存在することが知られている．このアルゴリズムは局所探索に基づくアルゴリズムであり，$O(n \log n)$ 回の解更新を繰り返すことで所望の近似解を得る．本研究は，CONGEST モデルにおける最小全域木問題の近似を考える．この問題の絶対近似 1 のアルゴリズム関しては，これまでに $O(n^2)$ ラウンドの自明なアルゴリズムより良い上界は知られていなかった．本研究では，上述の局所探索における 1 回の解の更新が最小全域木の構成 1 回で行えることを示し，それに基づく $\tilde{O}(n^{1.5})$ ラウンドの CONGEST モデル上のアルゴリズムを示す．

メモリ消費量を削減した適応型分散スライシングプロトコル

上辻 利奈 *　　首藤裕一 *　　角川裕次 †　　増澤利光 *

概要

分散スライシング問題とは，コンピュータネットワーク中の各ノードが持つ属性値に従って大域的に順位付けを行い，ノードを k 個のグループに分割する問題である．本研究では，既存の分散スライシングプロトコルのメモリ消費量を削減する手法を提案し，実験でこれらの手法の性能を比較評価する．

1　はじめに

分散システムとは，ネットワークに接続された多くの計算機 (ノード) で構成されるシステムである．複数の計算機間で相互通信が可能であり，協調してタスクを実行する．近年，システムの大規模化や複雑化が進んでいるため，システムを効率的に利用するプロトコル設計が必要となっている．

ノードが持つ固有の属性値に従って，ノードをグループに分割する操作は分散スライシングと呼ばれる [2]．分散スライシングプロトコルは，各ノードは自身がどのグループに属するかの指標であるスライス値を決定する．k 個のグループ (スライス) に分割する場合，属性値の大小によって上位 $1/k$，次の $1/k$，さらにその次の $1/k$ というように分割する．各ノードは周期的なサイクルで動作し，識別子や属性値の情報を他のノードと交換することによって自身のスライス値の決定を行う．すべてのノードの情報を集めればスライシングは行えるが，ここでは効率化のために各サイクルで一様ランダムに選んだノードとのみ情報を交換する手法を考えている．

重要度や処理性能に応じてノードにタスクを割り当てる際に分散スライシングを利用できる．例えば，ノードを処理能力に応じてグループ分けをしておき，タスク処理に必要な計算量を考慮して，タスクを適切な処理能力のグループのノードに割り当てることが考えられる．あるいは，ノードを安定性に応じてグループ分けをしておき，ストリーミングアプリケーションの性能を向上させるために，より安定したノードにより特権のある役割を割り当てることを行うことが考えられる [1, 7]．また，エネルギー量に応じてノードをグループに分割し，エネルギー量の多いグループに属するノードがエネルギー量の少ないグループに属するノードにエネルギーを渡すような仕組みも考えられる．

分散スライシングを実現するプロトコルとして，Ranking[2] が提案されている．このプロトコルでは，各ノードはランダムサンプリングにより周期的に一定数のノードの属性値を調べ，直近の M 個の属性値の中で自身の属性値が何番目に大きいかによって，自身のスライス値を決定する．このため，各ノードは直近の M 個の属性値それぞれについて自身の属性値との大小比較結果をリストに管理するので，各ノードは $O(M)$ のメモリ領域を必要とする．

本稿では，分散スライシングプロトコル Ranking に比べ，各ノードが持つメモリの消費量を削減した新たな手法を提案する．提案手法では，直近の M 個の属性値に関する履歴を正確に管理するのではなく，サンプリングが一様に行われていることを前提に，直近の M 個の属性値の中で自身の属性値より小さいものの数を推定する．これにより，各ノードのメモリ領域を $O(\log M)$ に削減する．

提案手法ではランダムサンプリングによって得た属性値の履歴を用いないため，プロトコル Ranking に比べて分散スライシングの精度が悪化する可能性がある．そこでシミュレーションにより，メモリ削減の提案手法が，分散スライシングの精度に及ぼす影響，また，ノードの属性値の変化に対する追随性に及ぼす影響を評価する．

本稿の構成は次の通りである．まず，2 節で本研究に関する諸定義を示す．次に，3 節で既存の分散スライシングプロトコル Ranking について，4 節で提案手法について説明する．5 節でこれら 2 つの手法を比較するための性能評価の実験の内容と結果を示す．最後に，6 節で本研究についてまとめる．

2　諸定義

本節では，本研究に関する定義を示す．

* 大阪大学大学院情報科学研究科
† 龍谷大学理工学部

システム全体のノードの総数を N，スライスの数を k とする．k は入力としてすべてのノードに与えられる．

各ノード $i(0 \leq i < N)$ は固有の識別子 id_i と属性値 v_i と呼ばれる変数を持ち，属性値 v_i は特定の値に設定されている．異なるノードに同じ属性値を与えてもよい．また，各ノードはそれぞれ (局所) メモリを持つが，最初各ノードのメモリには何も情報は保存されていないとする．

スライス値 s_i は，各ノード i がどのスライスに属するかを表す指標である．各ノードはスライス値 s_i を出力とする．但し，$0 \leq s_i < k$ とする．また，正しいスライス値とは，全ノードを属性値の降順に並べた場合に割り当てられるスライス値である．つまり，l 番目 $(0 \leq l < N)$ に大きい属性値を持つノードの正しいスライス値は $\lfloor lk/N \rfloor$ である．直感的に言うと，分散スライシング問題は，各ノードの属性値に基づいて，各スライスに属するノードの数が N/k になるように各ノード i にスライス値 s_i を割り当てる問題である．分散スライシングプロトコルでは，各ノードは収集した他ノードの属性値と自身の属性値 v_i の大小関係によって，各ノードがノード全体のどのスライス s_i に属するかを推定する．すなわち，分散スライシング問題の目的は，各ノード i に自身の属性値 v_i の相対的位置を正しく推定させることである．例えば，$N = 9$，k=3，$v_0 = 14$，$v_1 = 7$，$v_2 = 7$，$v_3 = 25$，$v_4 = 1$，$v_5 = 20$，$v_6 = 9$，$v_7 = 28$，$v_8 = 25$ とすると，$s_2 = 0$，$s_5 = 1$ となるようにスライシングを行う．

各ノードはサイクルとよばれる一連の動作を周期的に実行する．各サイクルでは，各ノード i はいくつかのノードと情報交換を行い，その情報と自身のメモリの内容に基づいて，自身のスライス値 s_i およびメモリの内容を更新する．なお，各ノード i は全ノードの属性値を収集することなく s_i を決定するため，分散スライシングを正確に実現できるとは限らない．また，シミュレーション実験では，ノード数や属性値の変更に対する分散スライシングプロトコルの追随性を評価するが，これらの変更はサイクル間に生じるものとする．

3 既存手法：プロトコル Ranking

本節では，既存の分散スライシングプロトコル Ranking[2] について説明する．擬似コードをアルゴリズム 1 に示す．分散スライシングプロトコル Ranking では，各ノードが自身の属性値よりも小さな属性値を持つノードの数を推定して，そこから自身のスライス値を決定することである．

プロトコル Ranking には，能動スレッドと受動スレッドがあり，能動スレッド，受動スレッドの順に動作する．能動スレッドでは，各サイクルであらかじめ決めておいた定数個のノードの情報 (識別子と属性値) を一様ランダムに取得する (9 行目)．このノードの集合を view と呼ぶ．1 サイクルですべてのノードと情報を交換するのではないことに注意する．一様ランダムサンプリングには，例えば，Cyclon[5]，Lpbcast[6] のような動的なランダムサンプリングを使用する [2]．一方，受動スレッドでは，view の中で自身の属性値よりも小さい属性値を持つノードの数 *smaller* を数え (34，35 行目)，ノードの属性値の相対的位置を推定し，スライスを決定する (38 行目)．属性値が同じ場合は，比較するノードの識別子で属性値の大小を判断する (28 行目)．

各ノードは view から取得したノードの属性値の情報をメモリに記憶する．具体的には，複数の Boolean 値を保存するリスト構造を用意し，取得したノードの属性値が自身の属性値より小さいときに True，大きいときに False をリストに格納する (28〜31 行目)．但し，このリストの長さには上限があり，リストの長さが上限に達すればリスト中で古い情報から破棄し (23，24 行目)，新しいノードの情報を格納する．この手法はスライディングウィンドウ手法と呼ばれる．各ノードが格納できるリストの要素の最大数を指定することにより，各ノードが必要とするメモリ量を調整できる．さらに，この手法を取り入れることによって，古い情報は破棄され常に新しい情報を記憶することができるので，ノード数やノードの属性値の変化に対する追随性を実現できる．各ノードが持つメモリには識別子の情報が記憶されていないため，リストに同じノードの情報が重複して記憶されている可能性がある．リストの要素の最大数を M とすると，各ノードが持つメモリ量は $O(M)$ となる．

4 提案手法

本節では，提案手法について説明する．提案手法の擬似コードをアルゴリズム 2 に示す．提案手法の主な動作は，既存の分散スライシングプロトコル Ranking[2] と同様に各ノードが自身の属性値よりも小さな属性値を持つノードの数を推定して，そこから自身のスライス値を決定することである．提案手法にも，プロトコル Ranking と同様に能動スレッドと受動スレッドがある．

プロトコル Ranking と提案手法の主な違いは，各ノードがメモリに記憶するデータ構造にある．プロトコル Ranking では，メモリには複数の Boolean 値を保存するリスト構造を使用していて，直近 M 個の属性値に関する

アルゴリズム 1　プロトコル Ranking[2]

```
1  // myIDは自身の識別子
2  // myAttributeValueは自身の属性値
3  // attributeListは各ノードが収集した属性値のリスト
4  // kはスライスの数
5
6  // 能動スレッド
7  def activeThread()
8      // 一様ランダムに特定の数のノードを取得する
9      view ← getView()
10     return view
11
12
13 // 受動スレッド
14 def passiveThread(view)
15     // 自身の属性値よりも小さい属性値を持つノードの数
16     smaller ← 0
17     // 比較したノードの総数
18     total ← 0
19
20     for each ID in view :
21         // スライディングウィンドウ手法
22         // 記憶するビット列の長さがM
23         if attributeList.full :
24             attributeList.removeOldest()
25
26         // 自身の属性値が小さい場合，あるいは属性値が同じで自身の識別子が小さい場合，真を追加する
27         // 上記以外は偽を追加する
28         if attributeValue < myAttributeValue or (attributeValue == myAttributeValue and ID <
             myID) :
29             attributeList.add(True)
30         else :
31             attributeList.add(False)
32
33     // smaller, totalの値を数える
34     smaller ← attributeList.count(True)
35     total ← attributeList.size()
36
37     // スライス値を決定する
38     sliceValue ← k * ( smaller / total )
```

大小比較の履歴を保持している．一方で，提案手法では，各ノードが持つメモリには直近 M 個の属性値に対する *smaller* と *total* の 2 つの推定値のみを格納する．各ノードのメモリは現在のサイクルにおける自身の属性値より小さいノードの数 *smaller*，比較したノードの数の合計 *total* それぞれに，過去のサイクルにおけるこれらの値をそれぞれ α 倍にした値を加えたものを推定値として記憶する (32，33 行目)．ここに，α は提案アルゴリズムのパラメータであり，$0 < \alpha < 1$ とする．α の値を変更することで，現在のサイクルに持ち越す過去のサイクルの情報の割合を調整できる．

total を α 倍している目的は，格納されている情報のうちの $(1-\alpha)$ を破棄するためである．一様ランダムサンプリングをしているなら，*smaller* もそのうちの $(1-\alpha)$ が破棄されると推定でき，*smaller* も α 倍している．新たに取得する属性値の数を x とすると，$x*(1+\alpha+\alpha^2+...) = x/(1-\alpha)$ 個の属性値から求めた *total* と *smaller* の推定値を保持していることとなり，$x/(1-\alpha) = M$ となるように α を設定すれば，提案手法における各ノードが持つメモリ量は $O(\log M)$ になる．また，直近の M 個のサンプルに基づいてスライス値を推定するという手法を使用するので，ノード数や属性値の変化に対する追随性を実現できる．

次の節で，分散スライシングプロトコル Ranking の手法と提案手法の比較実験を α の値を変えながら行い，提案手法を評価する．

アルゴリズム 2　提案手法

```
1  // myIDは自身の識別子
2  // myAttributeValueは自身の属性値
3  // attributeListは各ノードが収集した属性値のリスト
4  // kはスライスの数
5
6  // 能動スレッド
7  def activeThread()
8      // 一様ランダムに特定の数のノードを取得する
9      view ← getView()
10     return view
11
12
13 // 受動スレッド
14 def passiveThread(view)
15     // 自身の属性値よりも小さい属性値を持つノードの数
16     smaller ← mySmaller
17     // 比較したノードの総数
18     total ← myTotal
19
20     for each ID in view :
21         // 自身の属性値が小さい場合，あるいは属性値が同じで自身の識別子が小さい場合，smallerを1
             増やす
22         if attributeValue < myAttributeValue or (attributeValue == myAttributeValue and ID <
             myID) :
23             smaller ← smaller + 1
24
25         total ← total + 1
26
27     // スライス値を決定する
28     sliceValue ← k * ( smaller / total )
29
30     // smaller, totalの値を圧縮してメモリに記憶する
31     // メモリ量はO(log M)
32     mySmaller ← α * smaller
33     myTotal ← α * total
```

5　性能評価実験

本節では，分散スライシングプロトコル Ranking[2] と提案プロトコルの性能を α の値を変えながらそれぞれの性能を調べる．プロトコル Ranking も提案プロトコルもともにランダムサンプリングに基づく手法であり，必ずしも分散スライシングを正しく実現できるとは限らない．そこで，シミュレーション実験により，平均誤差，誤り率，スライス変化率，グループのノード数の標準偏差を評価する．さらに，ノードの数と各ノードの属性値が変化する場合について，これら 4 つの評価値に関する追随性を評価する．

5.1　評価測度

この節では，実験の評価測度について説明する．

平均誤差とは，推定スライス値と正しいスライス値の平均誤差を表す．N をノードの数，ノード i の推定スライス値を s_i，正しいスライス値を a_i として，平均誤差を以下の式で定義する．

$$\frac{\sum_{i=0}^{N-1} |s_i - a_i|}{N}$$

平均誤差が小さいほど，各ノードのスライス値が正確に割り当てられていると判断できる．

誤り率とは，間違っているスライス値を持つノードの割合を表す．N をノードの数，ノード i の推定スライス値を s_i，正しいスライス値を a_i として，誤り率を以下の式で定義する．

$$\frac{\sum_{i=0}^{N-1} b_i}{N}$$

但し，

$$b_i = \begin{cases} 1 & (s_i \neq a_i) \\ 0 & (s_i = a_i) \end{cases}$$

とする．平均誤差と同様に，誤り率が低いほど各ノードのスライス値が正確に割り当てられていると判断できる．

スライス変化率とは，1 サイクルでスライス値を変更するノードの割合を表す．両プロトコルともサンプリングに基づく手法であり，ノードの属性値が変化しなくても，サンプリングの影響で各ノードのスライス値が変化する場合がある．N をノードの数として，スライス変化率をを以下の式で定義する．

$$\frac{\sum_{i=0}^{N-1} c_i}{N}$$

但し，

$$c_i = \begin{cases} 1 & \text{(識別子 } i \text{ のノードがスライスを変更する場合)} \\ 0 & \text{(識別子 } i \text{ のノードがスライスを変更しない場合)} \end{cases}$$

とする．スライス変化率が低いほど，プロトコルは安定していると判断できる．

各グループにおけるノードの数 (スライスの大きさの分布) の**標準偏差**も評価する．分散スライシングでは N 個のノードを N/k 個のノードずつ k 個のグループに分割することを目的としている．しかし，各ノードは自身が行うサンプリングに基づいて，スライス値を推定するため，必ずしも N/k 個ずつのグループに分割されるとは限らない．

N をノードの数，k をスライスの数，スライス値が j であるノードの数を d_j として，標準偏差を以下の式で定義する．

$$\sqrt{\frac{1}{k}\sum_{j=0}^{k-1}\{\frac{d_j}{N} - \frac{1}{k}\}^2}$$

標準偏差が小さいほど，ノードがスライスに等しく全体的に分布していると判断できる．なお，ノード数の変化による影響を排除するため，通常の標準偏差を $\sqrt{N}$ で割った値を評価する．

平均誤差，誤り率，スライス変化率，標準偏差はサイクルごとで変化し，サイクルが進むにつれて収束していくので，これら 4 つの値の評価は収束した値で評価する．

追随性は，ノードの総数やノードの属性値が変化したときに，その変化に適応して平均誤差，誤り率，スライス変化率，標準偏差の 4 つの数値が変化し始めて収束するまでの時間を表す．収束時間が短いほど，追随性があると判断できる．

5.2 評価実験の概要

この節では，実験の概要について説明する．

本研究では，以下の 2 つの実験を行う．

- ノード数および各ノードの属性値が一定の場合の実験 (5.3 節)
- ノードのスライス値が安定した後，一定期間 1 サイクルごとにノードの数を増やす場合の実験 (5.4 節)

すべての実験で，スライスの数は $k = 10$，プロトコル Ranking のリストの要素数は $M = 100$，提案プロトコルの α の値は 1/2，2/3，3/4，4/5 とする．また，view の大きさは 20 とする．すなわち，各ノードは 1 サイクルで一様ランダムに選ばれた 20 個の他のノードから情報を受信する．Ranking では直近の 100 個の属性値の大小比較の履歴を保持しているのに対し，$\alpha = 1/2, 2/3, 3/4, 4/5$ はそれぞれ直近の 40，60，80，100 個の属性値から推定していることに相当する．各ノードの属性値は 0〜999999 の間で一様ランダムに選んだ値に設定されている．各ノードのスライスの初期値は 0 に設定されている．実験の結果は 10 回の試行の平均である．

5.3 ノード数および各ノードの属性値が一定の実験

この節では，ノードの数および各ノードの属性値が一定の実験を行う．ノードの数を $N = 10000$ とする．

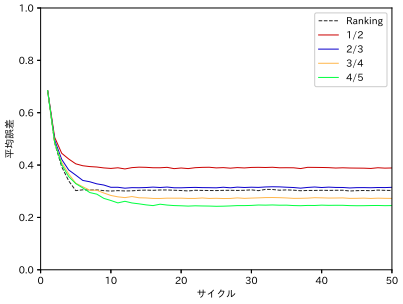


図 1　平均誤差

図 1 は，平均誤差についてのグラフである．$\alpha = 2/3$ の場合，Ranking の収束する値に近い結果となる．α の値を大きくすると，平均誤差の収束値が小さくなり，精度が向上する．

図 2 は，誤り率についてのグラフである．$\alpha = 2/3$ の場合，Ranking の収束する値に近い結果となる．α の値を大きくすると，誤り率の収束値が小さくなり，精度が向上する．

図 3 は，スライス変化率についてのグラフである．$\alpha = 3/4$ の場合，Ranking の収束する値に近い結果となる．α の値を大きくすると，スライス変化率の収束値が小さくなり，精度が向上する．

図 4 は，標準偏差についてのグラフである．$\alpha = 1/2$

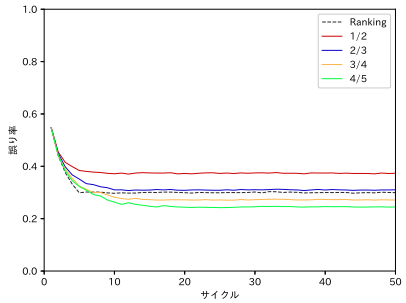


図 2　誤り率

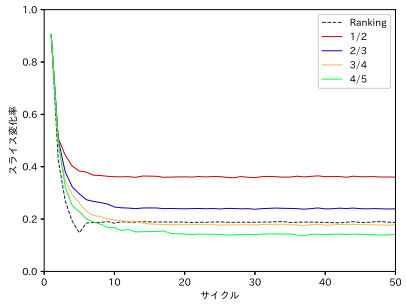


図 3　スライス変化率

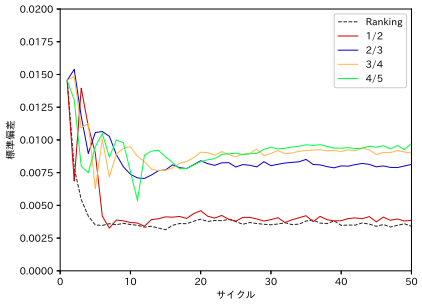


図 4　標準偏差

の場合，Ranking の収束する値に近い結果となる．α の値を大きくすると，標準偏差の収束値が大きくなる．

図 1〜4 におけるそれぞれの値の収束時間は，提案プロトコルよりもプロトコル Ranking の方が短いという結果になった．α の値を大きくすると，提案手法の収束時間が長くなる．

5.4　ノード数を動的に増やす実験

この節では，ノードのスライス値が安定した後，一定期間 1 サイクルごとにノードの数を増やした場合の分散スライシングプロトコルの追随性を評価するための実験を行う．最初のノードの数を $N = 5000$ とする．40 サイクル目から 49 サイクル目まで 1 サイクルごとにノードの数を 500 個ずつ増やす．最終的に，ノードの数は 10000 個になる．図 5〜8 の各グラフにおいて，(a) は最初からあるノード，(b) は追加されるノードについて表している．

図 5 は，平均誤差についてのグラフである．図 5(a) より，ノードの数が増えた 40 サイクル目以降でも元のノードは影響を受けないことがわかる．これは，追加したノードの属性値も一様ランダムに選んだため，元のノードの正しいスライス値もあまり変化しないためである．図 5(b) より，追加されたノードの収束時間は，$\alpha = 1/2$ の場合の提案プロトコルとプロトコル Ranking がほぼ同じという結果になった．α の値を大きくすると，ノードを増やした後における平均誤差の収束時間が徐々に遅くなる．

図 6 は，誤り率についてのグラフである．図 6(a) より，ノードの数が増えた 40 サイクル目以降でも元のノードは影響を受けないことがわかる．図 6(b) より，ノードの数を増やしてから，$\alpha = 1/2$ の場合の提案プロトコルとプロトコル Ranking がほぼ同じサイクル数で収束することがわかる．α の値を大きくすると，ノードを増やした後における誤り率の収束時間が徐々に遅くなる．

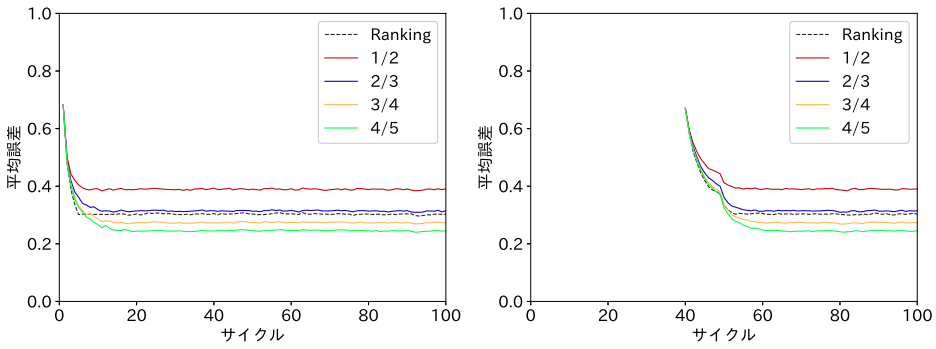
図 7 は，スライス変化率についてのグラフである．図 7(a) より，ノードの数が増えた 40 サイクル目以降でも元のノードは影響を受けないことがわかる．図 7(b) より，ノードの数を増やしてからの値の収束時間は，$\alpha = 1/2$ の場合の提案プロトコルとプロトコル Ranking がほぼ同じという結果になった．α の値を大きくすると，ノードを増やした後におけるスライス変化率の収束時間が徐々に遅くなる．

図 8 は，標準偏差についてのグラフである．図 8(a) より，ノードの数が増えた 40 サイクル目以降でも元からあったノードに関する標準偏差がわずかに増えており，ノードの数が増えると元のノードに影響することがわかる．図 8(b) より，ノードの数を増やしてからの値の収束時間は，$\alpha = 1/2$ の場合の提案プロトコルとプロトコル Ranking がほぼ同じという結果になった．α の値を大きくすると，ノードを増やした後における標準偏差の収束時間が徐々に遅くなる．

5.5　考察

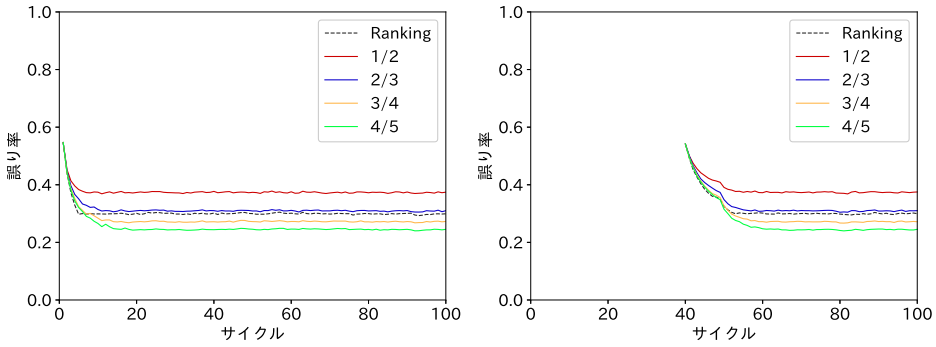
この節では，本実験を考察する．

α の値を大きくすると，提案手法の各ノードのメモリは過去のサイクルの情報を多く持っているため，提案手法におけるスライス変化率，平均誤差，誤り率の値は小さくなり正しいスライス値を割り当てられるノードの数が多くなる．一方で，α の値を大きくすると，環境変化に対する



(a) 最初からあるノード　　(b) 追加されるノード

図 5　平均誤差



(a) 最初からあるノード　　(b) 追加されるノード

図 6　誤り率

提案手法の追随性が悪くなり収束時間が遅くなる．環境変化が起きた際に，既存プロトコルではスライディングウィンドウ手法を用いて過去サイクルの情報をそのまま捨てることにより変化に対応しているが，提案手法では全体のうちの割合 $(1-\alpha)$ しか捨てていないので変化に対する反応が遅れると推測する．追随性の結果を改善するような手法として，過去のサイクルと現在のサイクルにおけるスライス値の差が大きい場合に *smaller*，*total* を 0 にするような手法を提案し，ノード数を動的に増やす実験を行った結果，大きく追随性の向上が見られなかった．

正しいスライス値を割り当てたいという目的には既存手法より提案手法が優れている．誤差の大きさへの要求が厳しくなく，各スライスに属するノードの数が N/k に近づくようにノードを均等に割り当てたいという目的や収束時間を短くしたい目的に対しては，既存手法や α の値を小さい提案手法を使うとよい．

6　おわりに

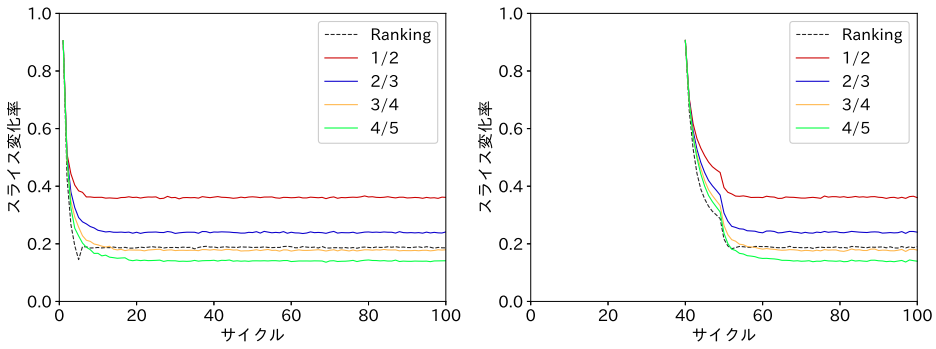
既存手法におけるリストの要素の最大数を M とすると，提案手法は既存手法よりメモリ消費量を $O(M)$ から $O(\log M)$ へと改善した．

既存の分散スライシングプロトコル Ranking[2] と提案手法の比較実験を行い，4 つの値と追随性を基準にし評価した．5 章の実験より，提案手法は既存手法より平均誤差，誤り率，スライス変化率において優れていることがわかる．

今後，メモリ消費量を抑えつつ追随性の結果を良くするような手法を考案し，実験することに取り組む．

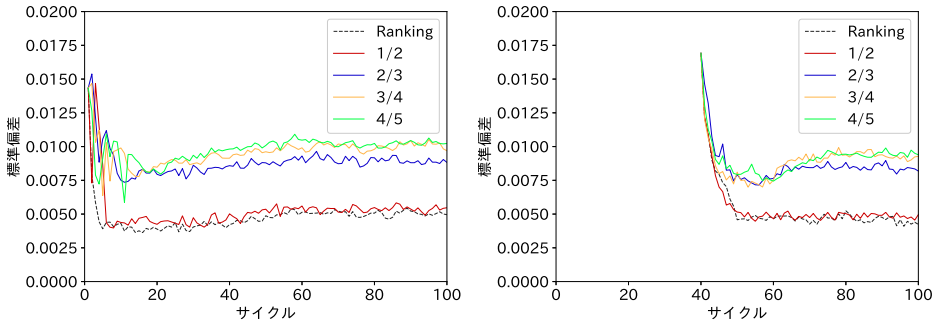
参考文献

[1] Maia, F., Matos, M., Riviere, E., Oliveira, R. Slead: Low-Memory, Steady Distributed Systems Slicing. In Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Sys-



(a) 最初からあるノード　　(b) 追加されるノード

図 7　スライス変化率



(a) 最初からあるノード　　(b) 追加されるノード

図 8　標準偏差

tems, pages 1-15, 2012

[2] Fernández, A., Gramoli, V., Jiménez, E., Kermarrec, A.-M., Raynal, M. Distributed Slicing in Dynamic Systems. In Proceedings of the International Conference on Distributed Computing Systems, pages 66-66, 2007

[3] Jaszkowski, P., Sienkowski, P., Iwanicki, K. Decentralized Slicing in Mobile Low-Power Wireless Networks. In the International Conference on Distributed Computing in Sensor Systems, pages 177-186, 2016

[4] Gramoli, V., Vigfusson, Y., Birman, K., Kermarrec, A.-M., van Renesse, R. A fast distributed slicing algorithm. In Proceedings of the twenty-seventh ACM Symposium on Principles of Distributed Computing, pages 427-427, 2008

[5] Voulgaris, S., Gavidia, D., Van Steen, M. Cyclon: Inexpensive membership management for unstructured p2p overlays. In Journal of Network and Systems Management, 13(2), pages 197-217, 2005

[6] Jelasity, M., Guerraoui, R., Kermarrec, A. M., Van Steen, M. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, pages 79-98, 2004

[7] Wang, F., Xiong, Y., Liu, J. mtreebone: A Collaborative Tree-Mesh Overlay Network for Multicast Video Streaming. In IEEE Transactions on Parallel and Distributed Systems, 21(3), pages 379-392, 2010

1-極大独立集合問題を解く反復合成を用いた自己安定アルゴリズム

田中 秀幸 * 首藤 裕一 * 角川 裕次 † 増澤 利光 *

内容梗概

ネットワークの 1-極大独立集合を求める自己安定分散アルゴリズムを提案する．1-極大独立集合とは，無向グラフ (つまり，ネットワーク)$G = (V, E)$ の極大独立点集合であって，任意の頂点 $u \in S, v, w \notin S$ $(v \neq w)$ について $S \cup \{v, w\} \setminus \{u\}$ が独立点集合とならないような頂点集合 S を求める問題である．この問題に対して，グラフの各頂点が一意の識別子を持つという仮定のもと動作するサイレント (silent) な自己安定分散アルゴリズムを提案する．提案アルゴリズムは弱公平分散デーモンのもとで動作する．n をグラフの頂点数，D をグラフの直径とすると，収束時間は $O(nD)$ であり，1 頂点あたりの空間計算量は $O(\log n)$ ビットである．これらのアルゴリズムの設計にあたって，反復合成と呼ばれる手法を用いる．

1 はじめに

分散システムとは，多数の計算機 (**ノード**) とそれらを繋ぐ通信リンクから構成されるシステムである．各ノードは互いに通信を行い，自律的かつ協調的に動作することにより問題を解決する．分散システムにおける問題を解くアルゴリズムを**分散アルゴリズム**と呼ぶ．

近年分散システムの大規模化が進むにつれて局所的な故障が無視できない頻度で発生することが多く，多くの分散システムで避けられなくなっている．そのため，分散システムに耐故障性を与えることが重要であると考えられている．また，モバイルノードを含む動的ネットワークでは，ネットワーク形状の動的変化に対する適応性を実現することが重要である．これらの耐故障性や適応性を実現する手法の 1 つとして，自己安定（分散）アルゴリズムとよばれる効果的な解決策が存在し，多くの研究がなされている．

自己安定アルゴリズムとは，任意の初期状況からアルゴリズムの実行を開始しても，やがて問題の要求を満たす状況に収束するという性質を持つアルゴリズムである．自己安定アルゴリズムは，故障やネットワーク形状の変化が発生することによって分散システムがいかなる状況に遷移しても，やがて正常な状況に回復し，次の故障や形状変化が発生するまでの間，正常な状況を保持することができる．したがって，自己安定アルゴリズムは極めて高い耐故障性を持つといえる．

本稿では，極大独立集合問題を拡張した 1-極大独立集合問題を考える．グラフ (ネットワーク)$G = (V, E)$ において，S に属するどの 2 頂点 (2 ノード) も隣接しないような集合 $S \subseteq V$ を独立集合と言う．分散システムの様々なアプリケーションにおいて，より要素数の大きい独立集合を求めることは重要である．独立集合の利用例として，無線ネットワークにおけるクラスタリングなどが挙げられる [1]．しかしながら，**最大独立集合**を求めることは NP 困難である [11]．したがって，**極大独立集合** (MIS) を求める研究が多くなされている．極大独立集合とは，どの頂点 $v \in V \setminus S$ についても $S \cup \{v\}$ が独立集合とならないような集合 S のことである．残念なことに，極大独立集合は常に大きな要素数を持つとは言えない．例えば，スターグラフ (直径 2 の木) において，極大独立集合の要素数は 1 になりうる．したがって，Bollobás et al. [2] により導入された，より強い極大性を持つ 1-極大独立集合 (1-MIS) を考える．極大独立集合 $S \subseteq V$ は，任意の頂点 $u \in S, v, w \notin S$ $(v \neq w)$ について $S \cup \{v, w\} \setminus \{u\}$ が独立集合とならないならば，1-極大独立集合である．スターグラフの場合，n 個の頂点からなるスターの 1-極大独立集合の要素数は $n - 1$ となる．

1.1 関連研究

極大独立集合問題はグラフ理論や分散システムにおいて基本的な問題の 1 つであるため，数多くの研究がなされてきた．表 1 に，MIS や 1-MIS に対する自己安定アルゴリズムの最近の結果を示す．1995 年に，Shukla et al.[10] により，任意のグラフに対する自己安定 MIS アルゴリズムが提案された．そのアルゴリズムは，集中デーモンを仮定しており，収束時間は $O(n)$ ステップ，1 頂点あたり 1 ビット (2 状態) で最適である．ここで，集中デーモンとは，各ステップにおいてただひとつの頂点にのみアクションを実行させるスケジューラである．このアルゴリズム匿名ネットワーク上で動作する．すなわち，頂点が一意な識別子を持つ必要がない．Ikeda et al.[7] では，MIS に対する別の自己安定アルゴリズムが提案された．そのアルゴリズムは，各頂点に識別子が存在することを仮定するが，分散デーモンのもとで正しく動作する．ここで，分散デーモンとは，各ステップにおいて複数の頂点がアクションを実行でき

*大阪大学大学院 情報科学研究科
†龍谷大学 理工学部 数理情報学科

表 1: MIS と 1-MIS に対する自己安定アルゴリズム．n と D はそれぞれグラフの全頂点数と直径を表す．

	問題	グラフ	ID	スケジューラ	収束時間 ステップ	収束時間 ラウンド	空間計算量
[10]	MIS	任意	なし	集中デーモン	$O(n)$	$O(n)$	$O(1)$ bits
[7]	MIS	任意	あり	分散デーモン	$O(n^2)$	$O(n)$	$O(1)$ bits
[11]	MIS	任意	あり	分散デーモン	$O(n)$	$O(n)$	$O(1)$ bits
[9]	1-MIS	木	なし	集中デーモン	$O(n^2)$	$O(n)$	$O(1)$ bits
[8]	1-MIS	任意	あり	集中デーモン	-	$O(n^2)$	$O(n \log n)$ bits
Proposed	1-MIS	任意	あり	分散デーモン	-	$O(nD)$	$O(\log n)$ bits

るようなスケジューラである．1 頂点あたりの空間計算量は 1 ビット (2 状態) で最適であるが，収束時間は $O(n^2)$ ステップである．Turau[11] は空間計算量を $O(1)$ ビット (3 状態) に増やすことで，収束時間を $O(n)$ ステップに改善している．

Shi et al.[9] は，初めて 1-MIS に対する自己安定アルゴリズムを提案した．そのアルゴリズムはグラフが木であることを仮定しており，集中デーモンのもとで正しく動作する．収束時間は $O(n^2)$ ステップであり，1 頂点あたりの空間計算量は $O(1)$ ビットである．難波 [8] は，任意のグラフにおいて，集中デーモンのもとで正しく動作する，1-MIS に対する自己安定アルゴリズムを提案した．論文中で，収束時間に関する解析は示されていないが，収束時間は $O(n^2)$(非同期) ラウンドである．難波のアルゴリズムは，部分アルゴリズムを並列に n 個実行するため，1 頂点あたりの空間計算量は $O(n \log n)$ ビットである．

各頂点に識別子が存在しない場合 (匿名グラフ)，初期状況における対称性を壊すことができないため，任意のグラフにおいて，分散デーモンのもとで動作する，MIS(1-MIS) に対する決定性アルゴリズムは存在しない．

1.2 本研究の成果

本稿では，極大独立集合よりも極大性の高い，1-極大独立集合問題を考える．この問題に対して，各頂点が一意な識別子を持つ，任意な連結無向グラフで動作するサイレントな自己安定アルゴリズムを提案する．n をグラフの頂点数，D をグラフの直径とすると，収束時間は $O(nD)$(非同期) ラウンドであり，1 頂点あたりの空間計算量は $O(\log n)$ である．このアルゴリズムの設計にあたって，反復合成と呼ばれる手法を用いる．

2 諸定義

ネットワークは単純連結無向グラフ $G = (V, E)$ で表される．V はネットワーク内の頂点の集合，E はネットワーク内の辺の集合とする．ネットワークの頂点数を $n = |V|$ とする．各頂点は，非負整数の集合 ID から選ばれる一意な識別子 id を持つ．頂点 v に接続する辺と繋がっている頂点を，頂点 v の隣接頂点と呼び，頂点 v の隣接頂点の集合を $N_v = \{w \in V \mid \{w, v\} \in E\}$ と表す．

通信モデルとして局所共有メモリモデル [5] を用いる．頂点は有限状態機械によりモデル化され，その状態は頂点が有する変数の値により決まる．ある頂点は自身とその隣接頂点の変数の値を同時に読み込むことができるが，書き換え可能なのは自身の変数のみである．

各頂点のアルゴリズムをアクション $< label >< guard > \longrightarrow < statement >$ の有限集合により定義する．$label$ は参照のため，およびアクションの優先度を表すために使用する．$guard$ は頂点 v とその隣接頂点の変数と識別子に関する述語である．$statement$ は頂点 v の 1 つ以上の変数の値を更新する処理を定める．頂点 v の 1 つ以上のアクションの $guard$ が真であるとき，v は**実行可能**であるという．$gurad$ の真偽の決定と対応する $statement$ の実行は 1 原子ステップで起こると仮定する [6]．

ネットワークの**状況**は各頂点の状態から成るベクトルで表される．状況 γ における，頂点 v の変数 x の値を $\gamma(v).x$ で表す．**ステップ**と呼ばれる，各状況の変遷は**デーモン**により引き起こされる．本稿では**分散型デーモン**を仮定する．分散型デーモンとは，各ステップで 1 つ以上の任意の実行可能な頂点のアクションを同時に実行できるデーモンである．実行される頂点に 2 つ以上の実行可能なアクションがある場合，最も優先度が高いラベルのアクションを実行する．アルゴリズム $\mathcal{A}$ の 1 つのステップにより状況が γ から γ' に変わることを $\gamma \mapsto_{\mathcal{A}} \gamma'$ と表す．アルゴリズム $\mathcal{A}$ の**実行**とは，全ての $i \geq 0$ に対して $\gamma_i \mapsto_{\mathcal{A}} \gamma_{i+1}$ となるような状況の系列 $\gamma_0, \gamma_1, \cdots$ である．デーモンは**弱公平**であることを仮定する．すなわち，連続して実行可能な頂点はいずれ実行されることが保証される．

自己安定アルゴリズムとは，ネットワークのどのような状況から実行を開始しても，いずれ正しい動作に復帰することを保証するアルゴリズムである．つまり，各頂点はどのような状態からアルゴリズムの実行を開始してもよい．状況 γ において実行可能な頂点が存在しないとき，γ を**最終状況**と呼ぶ．無限長の実行および有限長で末尾の状況が最終状況であるような実行を**極大実行**と呼ぶ．$\mathcal{L}$ を状況に対する述語とする．アルゴリズム $\mathcal{A}$ の極大実行 $\gamma_0, \gamma_1, \cdots$ が次の (i),(ii) を満たすとき，$\mathcal{A}$ は $\mathcal{L}$ に対して自己安定である．(i) ある $i \geq 0$ に対して，$\mathcal{L}(\gamma_i)$，(ii) 全ての $j \geq i$ に対して，$\mathcal{L}(\gamma_j)$．(i) は**収束性**，(ii) は**閉包性**と呼ばれる．$\mathcal{A}$ の任意の極大実行が有限であるとき，$\mathcal{A}$ は**サイレント**であるという．

実行の収束時間は非同期ラウンドで測定する．頂点 v が状況 γ_i で実行可能で，状況 γ_{i+1} で実行可能でないとき，頂点 v はステップ $\gamma_i \mapsto \gamma_{i+1}$ で**無効化**されたという．実行 $\varrho = \gamma_0, \gamma_1, \ldots$ の最初のラウンドを，γ_0 で実行可能な全頂点が一度は実行される，もしくは無効化される最小の接頭辞とし，$\gamma_0 \ldots \gamma_s$ と表す．ϱ の 2 番目のラウンドを，実行 $\gamma_s, \gamma_{s+1}, \ldots$ の最初のラウンドとする．それ以降のラウンドも同様に定義する．ϱ の実行時間をそのラウンドの数として測定する．

2.1 問題定義

本稿では 1-極大独立集合問題を考える．連結無向グラフ $G = (V, E)$ において，頂点の部分集合 $S \subset V$ の互いに異なる 2 つの頂点が隣接しないとき，S を**独立集合**と呼ぶ．また，どの頂点 $v \in V \setminus S$ についても $S \cup \{v\}$ が独立集合とならないような独立集合 S を**極大独立集合**と呼ぶ．さらに，極大独立集合 S に対して，S に含まれるどの頂点を 1 つ取り除き，S には含まれないどの 2 頂点を追加しても独立集合にはならないとき，集合 S を *1*-**極大独立集合**と呼ぶ．つまり，1-極大独立集合 S とは，任意の頂点 $u \in S, v, w \notin S$ $(v \neq w)$ について $S \cup \{v, w\} \setminus \{u\}$ が独立集合とならないような極大独立集合である．*1*-**極大独立集合問題**では，各頂点 v が変数 $v.\texttt{mis} \in \{\textbf{true}, \textbf{false}\}$ を持ち，頂点集合 $\{v \in V \mid v.\texttt{mis}\}$ が 1-極大独立集合となるように各頂点 v の $v.\texttt{mis}$ を設定する問題である．ここで変数 `mis` が真の頂点を独立頂点と呼ぶこととする．$\mathcal{L}_{1\mathrm{MIS}}$ を，状況 γ において，$\{v \in V \mid v.\texttt{mis} = \textbf{true}\}$ が 1-極大独立集合であるとき，かつそのときのみ $\mathcal{L}_{1\mathrm{MIS}}(\gamma) = \textbf{true}$ となる，状況に対する述語とする．本稿では $\mathcal{L}_{1\mathrm{MIS}}$ に対してサイレントな自己安定であるアルゴリズムを与える．

3 反復合成

反復合成 [3] とは，アルゴリズム $\mathcal{A}$，$\mathcal{P}$ と述語 E を入力として述語 $\mathcal{L}$ に対するサイレントな自己安定アルゴリズム $\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ を生成するフレームワークである．反復合成を利用するためには，与えられた述語 $\mathcal{L}$ に対して，後述する条件を満たすように $\mathcal{A}$，$\mathcal{P}$，述語 E を設計しなければならない．アルゴリズム $\mathcal{A}$ は，繰り返し実行される基盤アルゴリズムである．$\mathcal{A}$ は後述する 3 つの条件 (*shiftable convergence*，*loop convergence*，*correctness*) を満たす必要がある．述語 $E : V \mapsto \{\textbf{false}, \textbf{true}\}$ は，ローカルにエラー検出を行う述語である．ある頂点 $v \in V$ で $E(v)$ が成り立つような状況 γ を，**エラー状況**と呼ぶ．全ての頂点で $v \in V$ で $\neg E(v)$ が成り立つような状況 γ を，**非エラー状況**と呼ぶ．アルゴリズム $\mathcal{P}$ は，任意の状況から，ネットワークを非エラー状況にするサイレントな自己安定アルゴリズムである．大まかに言うと，$\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ は次のような動作をする．ここで，先述の通り，状況 γ において実行可能な頂点が存在しないとき，かつそのときのみ γ は $\mathcal{A}$ の最終状況である．

1: **repeat**
2: **if** 現在の状況がエラー状況である **then**
3: $\mathcal{P}$ を実行 (これにより，ネットワークが非エラー状況になる)
4: **else**
5: $\mathcal{A}$ を実行 (これにより，ネットワークは $\mathcal{A}$ の最終状況になる)
6: $\mathcal{A}$ の出力を $\mathcal{A}$ の入力にコピー
7: **end if**
8: **until** 現在の状況が $\mathcal{A}$ の最終状況かつ，$\mathcal{A}$ の入力と出力が同じである

以下では，$\mathcal{A}$，$\mathcal{P}$ が満たすべき条件と，$\mathcal{A}$ の出力を $\mathcal{A}$ の入力へ**コピー**することの意味について説明する．$\mathcal{A}(\mathrm{resp.}\mathcal{P})$ のアクションにより値が更新される変数の集合を $O_{\mathcal{A}}(\mathrm{resp.}O_{\mathcal{P}})$ とする．$\mathcal{A}(\mathrm{resp.}\mathcal{P})$ のアクションにより値が更新されることはなく，値が参照されるだけの変数の集合を $I_{\mathcal{A}}(\mathrm{resp.}I_{\mathcal{P}})$ とする．$O_{\mathcal{A}} \cap O_{\mathcal{P}} = \emptyset$，$I_{\mathcal{P}} = \emptyset$ と仮定する．エラー検出述語 $E(v)$ は頂点 $v \in V$ により評価され，その真偽が自身と自身の隣接頂点の $I_{\mathcal{A}} \cup O_{\mathcal{P}}$ の変数にのみ基づく述語である．$\mathcal{E}$ を，状況 γ で $\bigvee_{v \in V} E(v)$ が成り立つとき，かつそのときのみ $\mathcal{E}(\gamma)$ が成り立つような，状況に対する述語とする．アルゴリズム $\mathcal{A}$ は全ての変数 $x \in O_{\mathcal{A}}$ に対して，**コピー変数** $\overline{x} \in I_{\mathcal{A}}$ を持つと仮定する．γ^{copy} を，状況 γ で，全頂点 v とその全ての変数 $x \in O_{\mathcal{A}}$ に対して $v.\overline{x}$ の値を $v.x$ の値で置き換えて得られる状況とする．$\gamma \in \neg\mathcal{E}$，$\gamma^{copy} = \gamma$ が成り立ち，どの頂点でも全アクションが実行可能でないとき，状況 γ は述語 $\mathcal{C}_{goal}(\mathcal{A}, E)$ を満たすとする．アルゴリズム $\mathcal{A}$ は，ある整数 $R_{\mathcal{A}}$ と $L_{\mathcal{A}}$ に対して，次の 3 つの条件を満たすように設計されている必要がある．

Shiftable Convergence 状況 $\neg\mathcal{E}$ から開始される $\mathcal{A}$ の任意の極大実行は，$\gamma^{copy} \in \neg\mathcal{E}$ であるような状況 γ で終了する．

Loop Convergence $\varrho_0, \varrho_1 \ldots$ が，各 $i \geq 0$ に対して $\varrho_i = \gamma_{i,0}, \gamma_{i,1}, \ldots, \gamma_{i,s_i}$，$\gamma_{0,0} \in \neg\mathcal{E}$，$\gamma_{i+1,0} = \gamma_{i,s_i}^{copy}$ であるような，アルゴリズム $\mathcal{A}$ の無限の系列ならば，ある $j < \mathcal{L}_{\mathcal{A}}$ に対して，$\gamma_{j,s_j} \in \mathcal{C}_{\mathrm{goal}}(\mathcal{A}, E)$ と $R(\varrho_0) + R(\varrho_1) + \cdots + R(\varrho_j) \leq R_{\mathcal{A}}$ が成り立つ．

Correctness 任意の状況 γ に対して，$\gamma \in \mathcal{C}_{goal}(\mathcal{A}, E) \Rightarrow \gamma \in \mathcal{L}$ が成り立つ．

$L_{\mathcal{A}}$ は $\mathcal{A}$ の実行回数の上界を表し，$R_{\mathcal{A}}$ は $\mathcal{A}$ の繰り返しの実行における全ラウンド数の上界を表す．$\mathcal{P}$ の任意の極大実行は，$\neg\mathcal{E}$ であるような状況で $T_{\mathcal{P}}$ ラウンド以内に終了するように設計されている必要がある．

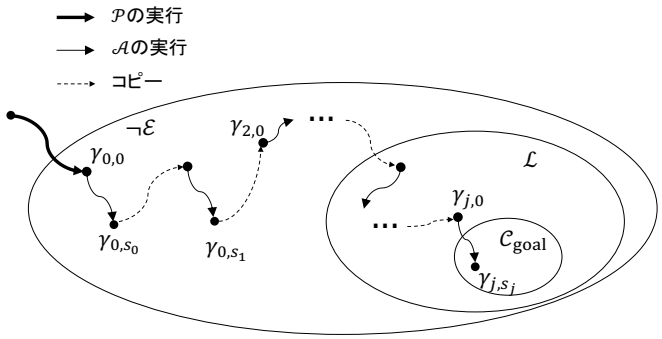


図 1: $\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ の実行

上記の条件を満たすように $\mathcal{A}$，$\mathcal{P}$，述語 E を設計したならば，作成されたアルゴリズム $\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ において以下の定理が導かれる

定理 3.1 ([4][1])．アルゴリズム $\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ は述語 $\mathcal{L}$ に対してサイレントであり，自己安定である．任意の $\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ の実行は $O(n + T_{\mathcal{P}} + R_{\mathcal{A}} + L_{\mathcal{A}}D)$ ラウンドで終了する．$\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ の 1 頂点あたりの空間計算量は $O(S_{\mathcal{A}} + S_{\mathcal{P}} + \log n)$ ビットである．$S_{\mathcal{A}}$，$S_{\mathcal{P}}$ はそれぞれ，アルゴリズム $\mathcal{A}$，$\mathcal{P}$ の 1 頂点あたりの空間計算量である．

以降で $\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ の動作について簡単に説明する．$\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ の実行において，各頂点はアルゴリズム $\mathcal{A}$ かアルゴリズム $\mathcal{P}$ のどちらかを実行する．任意の状況から始まったとしても，やがて全頂点は $\mathcal{A}$ と $\mathcal{P}$ のどちらを実行するのかについて合意を達する．現在の状況が $\neg\mathcal{E}$ を満たさないと検知されたとき，全頂点はアルゴリズム $\mathcal{P}$ を実行し，$\neg\mathcal{E}$ を満たす状況にする (図 1 の $\mathcal{P}$ の実行の矢印)．現在の状況が $\neg\mathcal{E}$ を満たすとき，全頂点はアルゴリズム $\mathcal{A}$ を実行する (図 1 の $\mathcal{A}$ の実行の矢印)．$\mathcal{A}$ の実行が終了する度に，状況 $\gamma \in \mathcal{C}_{goal}(\mathcal{A}, E)$ に到達しているかを確認する．もし，到達していたならば，その後何も動作しない．到達していないならば，全ての出力変数の値を対応するコピー変数にコピーする．それにより，現在の状況から別の状況へ変遷する (図 1 のコピーの矢印)．その後，再び新たに $\mathcal{A}$ を実行する．$\mathcal{A}$ を設計する上での条件である Shiftable Convergence と Loop Convergence より，やがて $\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ の実行は状況 $\gamma \in \mathcal{C}_{goal}(\mathcal{A}, E)$ に到達し，終了する．その後，$\mathcal{A}$ を設計する上での条件である Correctness により述語 $\mathcal{L}$ は常に成り立つ．

4 自己安定 1-極大独立集合アルゴリズム

本節では，前節で示した反復合成を用いて，述語 $\mathcal{L}_{1MIS}$ に対するサイレントな自己安定であるアルゴリズムを提案する．具体的には，反復合成アルゴリズム $\mathbf{Loop}(Inc, E_{\mathrm{MIS}}, Init)$ が述語 $\mathcal{L}_{1MIS}$ に対するサイレントな自己安定アルゴリズムとなるような基盤アルゴリズム Inc，初期化アルゴリズム $Init$，エラー検出述語 E_{MIS} を本節で与える．$\mathrm{Loop}(Inc, E_{MIS}, Init)$ において，1 頂点あたりの空間計算量は $O(\log n)$ ビットであり，収束時間は $O(nD)$ ラウンドである．

基盤アルゴリズム Inc において，各頂点 v は変数 $v.\mathtt{mis} \in \{\mathbf{false}, \mathbf{true}\}$ とそれに対応するコピー変数 $v.\overline{\mathtt{mis}} \in \{\mathbf{false}.\mathbf{true}\}$ を持つ．$v.\overline{\mathtt{mis}}$ は $Init$ の出力変数であり，Inc の入力変数である．$v.\mathtt{mis}$ は $Init$ では使用されず，Inc の出力変数である．ここで，$S_I = \{v \in V \mid v.\overline{\mathtt{mis}}\}$，$S_O = \{v \in V \mid v.\mathtt{mis}\}$ とする．S_I は Inc の入力における全ての独立頂点からなる集合であり，S_O は Inc の出力における全ての独立頂点からなる集合である．$\mathcal{L}_{input}$ を，状況 γ において S_I が MIS である，かつそのときのみ $\mathcal{L}_{input}(\gamma) = \mathbf{true}$ が成り立つような，状況に対する述語とする．ここで，$\mathcal{L}_{input}$ は前節における $\neg\mathcal{E}$ に対応する．

本節では以下を満たすような Inc，E_{MIS}，$Init$ を与える．

- S_I が MIS でないならば，少なくとも 1 つの頂点 v において $E_{MIS}(v) = \mathbf{true}$ が成り立つ．すなわち，状況 γ において $\neg\bigvee_{v\in V} E_{MIS}(v)$ が成り立つとき，かつそのときのみ $\mathcal{L}_{input}(\gamma) = \mathbf{true}$ が成り立つ．
- 任意の状況から開始する $Init$ の全ての極大実行は，$\mathcal{L}_{input} = \mathbf{true}$ が成り立つような状況で $O(n)$ ラウンド以内に終了する．
- $\mathcal{L}_{input} = \mathbf{true}$ であり，S_I が 1-MIS でないような状況から開始する Inc の極大実行 ϱ は，S_O が MIS であり $|S_O| \geq |S_I| + 1$ が成り立つような状況で $O(\epsilon + 1)$ ラウンド以内に終了する．ここで，ϵ は実行 ϱ の最終状況における $|S_O| - |S_I|$ である．
- $\mathcal{L}_{input} = \mathbf{true}$ であり，S_I が 1-MIS であるような状況から開始する Inc の極大実行 ϱ は，S_I が 1-MIS であり，$S_O = S_I$ であるような状況で $O(1)$ ラウンド以内に終了する．

[1]反復合成のフレームワーク $\mathbf{Loop}(\mathcal{A}, E, \mathcal{P})$ 自体は [3] で提案されたが，[4] でその計算量の解析が改善され，定理 3.1 が証明された．

上記が全て成り立てば，Inc，E_{MIS}，$Init$ は，前節の反復合成を使用する上での条件を全て満たし，$\mathcal{L} = \mathcal{L}_{1MIS}$，$T_{Init} = O(n)$，$R_{Inc} = O(n)$，$L_{Inc} = n$ となる．したがって，$\mathbf{Loop}(Inc, E_{\mathrm{MIS}}, Init)$ は述語 $\mathcal{L}_{1MIS}$ に対してサイレントな自己安定アルゴリズムであり，収束時間は $O(n + T_{Init} + R_{inc} + L_{inc} \cdot D) = O(nD)$ ラウンドである．

4.1 エラー検出述語 E_{MIS} と初期化アルゴリズム $Init$

エラー検出述語を次の通り与える．

$$E_{\mathrm{MIS}}(v) \equiv (v.\overline{\texttt{mis}} \wedge \exists u \in N_v : u.\overline{\texttt{mis}}) \vee (\neg v.\overline{\texttt{mis}} \wedge \forall w \in N_v : \neg w.\overline{\texttt{mis}}).$$

補題 4.1. 任意の状況 γ において，$\neg \bigvee_{v \in V} E_{\mathrm{MIS}}(v)$ が成り立つとき，かつそのときのみ $\mathcal{L}_{\mathrm{input}}(\gamma)$ が成り立つ．

Proof. まず $\neg E_{\mathrm{MIS}}(v) \equiv (v.\overline{\texttt{mis}} \Rightarrow \forall u \in N_v : \neg u.\overline{\texttt{mis}}) \wedge (\neg v.\overline{\texttt{mis}} \Rightarrow \exists w \in N_v : w.\overline{\texttt{mis}})$ である．ある状況 γ で $\neg \bigvee_{v \in V} E_{\mathrm{MIS}}(v)$ が成り立つとする．それは，状況 γ において全ての頂点 $v \in V$ で $\neg E_{\mathrm{MIS}}(v)$ が成り立つということを意味する．そのとき，S_I に属す全ての頂点は，S_I に属すような隣接頂点を持たず，S_I に属さない全ての頂点は S_I に属すような隣接頂点を少なくとも 1 つ持つ．したがって，状況 γ において S_I は MIS であり，$\mathcal{L}_{\mathrm{input}}(\gamma)$ が成り立つ．ある状況 γ で $\bigvee_{v \in V} E_{\mathrm{MIS}}(v)$ が成り立つとする．その場合，S_I に属すある頂点が，S_I に属す隣接頂点を持つ，もしくは，S_I に属さないある頂点が S_I に属す隣接頂点を持たない．したがって，状況 γ において S_I は MIS でなく，$\mathcal{L}_{\mathrm{input}}(\gamma)$ は成り立たない． □

初期化アルゴリズム $Init$ のアクションを表 1 に示す．$Init$ の実行により，$O(n)$ ラウンド以内に，S_I が MIS であるような状況になる．$Init$ では，小さい識別子を持つ頂点が優先して S_I に属すことになる．ある頂点 v は，S_I に属し v より識別子が小さいような隣接頂点を持たないならば，S_I に追加される．つまり $v.\overline{\texttt{mis}} \leftarrow \mathbf{true}$ を実行する．そうでないならば，つまり v が S_I に属し v より識別子が小さいような隣接頂点を持つならば，$v.\overline{\texttt{mis}} \leftarrow \mathbf{false}$ を実行する．

表 1: $Init$

[**Actions of process** v]			
I_1:	$v.\overline{\texttt{mis}}$	$\longleftarrow$	$(\forall w \in N_v : \neg w.\overline{\texttt{mis}} \vee (v.id < w.id))$

補題 4.2. 任意の状況から始まる $Init$ の全ての極大実行は，$\mathcal{L}_{\mathrm{input}}$ が成り立つような状況で $O(n)$ ラウンド以内で終了する．

Proof. まず，$Init$ の任意の最終状況で $\mathcal{L}_{\mathrm{input}}$ が成り立つことを示す．3 節の"$v.x \longleftarrow \chi(v)$"の定義より，$v.\overline{\texttt{mis}} \neq (\forall w \in N_v : \neg w.\overline{\texttt{mis}} \vee (v.id < w.id))$ であるとき，かつそのときのみ，頂点 v は実行可能である．したがって，実行可能が頂点が存在しないような最終状況 γ において，S_I に属す全ての頂点 v は，S_I に属すような隣接頂点を持たず，S_I に属さない全ての頂点 v は，S_I に属すような隣接頂点を少なくとも 1 つ持つ．以上より，任意の最終状況において，$\mathcal{L}_{\mathrm{input}}$ は成り立つ．

次に $Init$ の全ての極大実行は，$O(n)$ ラウンド以内に終了することを示す．$v_1, v_2, \ldots, v_n$ を，$v_1.id < v_2.id < \cdots < v_n.id$ であるような頂点の集合とする．アクション I_1 のガードは $v.\overline{mis}$ の真偽と，$w.id < v.id$ であるような $w.\overline{mis}$ の真偽でのみ決定される．したがって，$Init$ の任意の極大実行の第 1 ラウンドで $v_1.\overline{\texttt{mis}}$ は決定され，以後，実行可能となることはない．同様に，任意の $i \geq 2$ について，$v_i.\overline{\texttt{mis}}$ は，N_{v_i} に属す v_i より識別子が小さいような全ての頂点の $v_i.\overline{\texttt{mis}}$ が決定されたのち，1 ラウンド以内に決定され，以後実行可能となることはない．以上より，$Init$ の全ての極大実行は，$O(n)$ ラウンド以内に終了する． □

4.2 アルゴリズム Inc

基盤アルゴリズム Inc のアクションを表 2 に示し，表 2 で使用されている関数を表 3 に示す．Inc は S_I が MIS であることを仮定して動作する．このアルゴリズムの実行では，S_I が 1-MIS でないならば，$|S_O| \geq |S_I| + 1$ となるような極大独立集合 S_O を求め，S_I が 1-MIS ならば，$|S_O| = |S_I|$ であるような状況に到達する．図 2 は，アルゴリズム Inc の入力 S_I の 1 例であり，$S_I = \{5, 23, 71\}$ である．ここで，S_I は極大独立集合であるが，1-MIS ではない．

4.2.1 アルゴリズムの概要

この節では，S_I が 1-MIS でないならば，$|S_O| \geq |S_I| + 1$ となるような極大独立集合 S_O を求め，S_I が 1-MIS ならば，$|S_O| = |S_I|$ であるような状況に到達するようなアルゴリズムを与える．Inc のアクションを表 2 に示し，表 2 で使用される関数を表 3 に示す．

まず，頂点の親子関係を定義する．頂点 $v \in V \setminus S_I$ が 1 つの隣接頂点 $u \in S_I$ を持つとき，u は v の親であり，v は u の子であるとする．ここで，頂点 $v \in V \setminus S_I$ が 1 つの隣接頂点 $u \in S_I$ を持つとは，$N_v \wedge S_I = \{u\}$ が成り立つ

Table 2: Inc

[**Actions of process** v]			
M_1:	$v.\texttt{parent}$	$\longleftarrow$	$Parent(v)$
M_2:	$v.\texttt{numchild}$	$\longleftarrow$	$\lvert C_v \rvert$
M_3:	$v.\texttt{cand}$	$\longleftarrow$	$Cand(v)$
M_4:	$v.\texttt{qualifier}$	$\longleftarrow$	$Qualifier(v)$
M_5:	$v.\texttt{winner}$	$\longleftarrow$	$Winner(v)$
M_6:	$v.\texttt{mis}$	$\longleftarrow$	$InS_A(v) \vee InS_B(v) \vee InS_C(v)$

Table 3: Inc の関数

$$Parent(v) = \begin{cases} \min\{w.id \mid w.\overline{\texttt{mis}}\} & \textbf{if } |\{w \in N_v \mid w.\overline{\texttt{mis}}\}| = 1 \\ \bot & \textbf{otherwise} \end{cases}$$

$$C_v = \{w \in N_v \mid w.\texttt{parent} = v\}$$

$$Cand(v) \equiv v.\texttt{parent} \neq \bot \wedge (v.\texttt{parent}).\texttt{numchild} - |\{w \in N_v \mid v.\texttt{parent} = w.\texttt{parent}\}| \geq 2$$

$$Qualifier(v) \equiv v.\texttt{cand} \wedge (\forall w \in N_v : w.\texttt{cand} \Rightarrow v.\texttt{parent} \leq w.\texttt{parent})$$

$$Winner(v) \equiv v.\texttt{qualifier} \wedge (\forall w \in N_v : v.id < w.id \vee \neg w.\texttt{winner})$$

$$InS_A(v) \equiv v.\overline{\texttt{mis}} \wedge |\{w \in C_v \mid w.\texttt{winner}\}| \leq 1$$

$$InS_B(v) \equiv \neg v.\overline{\texttt{mis}} \wedge v.\texttt{winner} \wedge \neg(v.\texttt{parent}).\texttt{mis}$$

$$InS_C(v) \equiv \neg v.\overline{\texttt{mis}} \wedge \neg v.\texttt{winner} \wedge \begin{pmatrix} \forall w \in N_v \ s.t. \ w.\texttt{mis} : \\ \neg\left(w.\overline{\texttt{mis}} \vee w.\texttt{winner} \vee w.id < v.id\right) \end{pmatrix}$$

ことである．この定義により，S_I が 1-MIS でないならば，高さ 1 の木が，1 つ以上生成される．各頂点 v は，親が存在するならば，その親の識別子を $v.\texttt{parent}$ に保存する．また，親を持たないならば，$v.\texttt{parent}$ に $\bot$ を保存する．C_u を頂点 u の子の集合とする．よって $C_u = \{v \in N_u \mid v.\texttt{parent} = u\}$ となる．u と v を，u が v の親であるような任意の 2 つの頂点とする．$|C_u| - |C_v \setminus N_v| \geq 2$ が成り立つとき，v を**候補**と呼ぶ．言い換えると，v は，その親が存在し，その親が $V \setminus N_v$ に属す v 以外の子を持つならば，候補になる．図 4 を用いて具体的に説明する．頂点 31 の親 (頂点 5) は，頂点 31 の隣接頂点でない別の子 (頂点 61) を持つので，頂点 31 は候補である．一方で，頂点 13 は，頂点 13 の親 (頂点 5) の全ての子 (頂点 31 と頂点 62) に隣接しているので，候補でない．言い換えると，候補とは，その親が独立頂点でなくなることにより，その親の子である 2 つ以上の候補が独立頂点になることで，MIS の要素数を増やすことができるような頂点である．例えば，頂点 5 が独立頂点でなくなることで，その隣接する候補である頂点 31 と頂点 62 は独立頂点になることができるので，S_I より要素数が大きい MIS を得ることができる．しかしながら，候補同士が隣接する可能性があり，その隣接している候補の両方を独立頂点にすると，S_O が MIS でなくなるので，全ての候補を独立頂点にすることはできない場合がある．具体的には，頂点 53 と頂点 79 の両方，もしくは頂点 11 と頂点 81 の両方を独立頂点にすることはできない．

上記で述べた，候補が隣接する問題を解決するために 2 つのフィルタ (第 1 フィルタと第 2 フィルタ) を導入する．第 1 フィルタを通過する候補を**予選通過者**，第 2 フィルタを通過する候補を**勝者**とする．候補 v は，$w.\texttt{parent} < v.\texttt{parent}$ が成り立つような隣接する候補 w が存在しないならば，予選通過者になる．図 4 において，頂点 79 は，その親 (頂点 71) の識別子が 71 であり，その隣接頂点 (頂点 53) の親の識別子が 23 なので，予選通過者でない．図 4 の例で，別の親を持ち，隣接している候補が存在しないような候補は予選通過者となる．第 1 フィルタのみでは，同一の親を持ち，隣接している候補同士の問題を解決できていない．よって，第 2 フィルタで，アルゴリズム $Init$ と同じように，それら候補同士の識別子を比較することにより，勝者を決定する．$q_1, q_2, \ldots, q_s$ を，$q_1.id < q_2.id < \ldots q_s.id$ であるような予選通過者であるとする．そして，集合 W を次のように定義する．$q_1 \in W$ であり，各 $i \geq 2$ に対して，q_i に隣接する $j < i$ であるような q_j が存在しないとき，かつそのときのみ $q_i \in W$ である．図 4 の例では，頂点 11，31，37，53，62 が勝者となる．勝者について次の 2 つの補題が成り立つ．

補題 4.3. S_I が 1-MIS でないならば，子に 2 つ以上の勝者が存在するような頂点が少なくとも 1 つ存在する．

Proof. 子に候補が存在するような頂点のうち，最小の識別子である頂点を u をする．S_I が 1-MIS でないならば，そのような u が必ず存在する．c_1 を，C_u に属す候補のうち，最小の識別子である頂点とする．候補の定義より，C_u に属し，c_1 と隣接しないような候補が 1 つ以上存在する．それらの中で，最小の識別子である頂点を c_2 とする．c_1 と c_2 は，その親である u の識別子の最小性により，第 1 フィルタを通過し，c_1 と c_2 自身の識別子の最小性により，第 2 フィルタを通過する．以上より，u の子に少なくとも 2 つの勝者が存在する． □

補題 4.4. S_I が 1-MIS でないならば，勝者は存在しない．

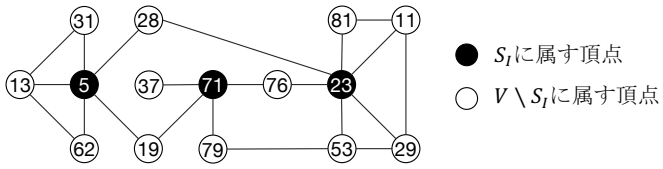


図 2: S_I の 1 例

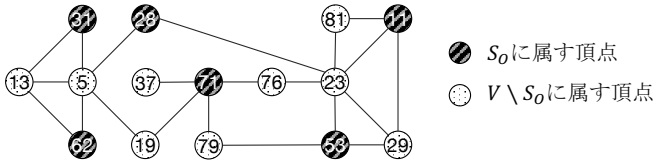


図 3: S_I が図 2 で示されるような Inc の 1 回の実行により導かれる S_O

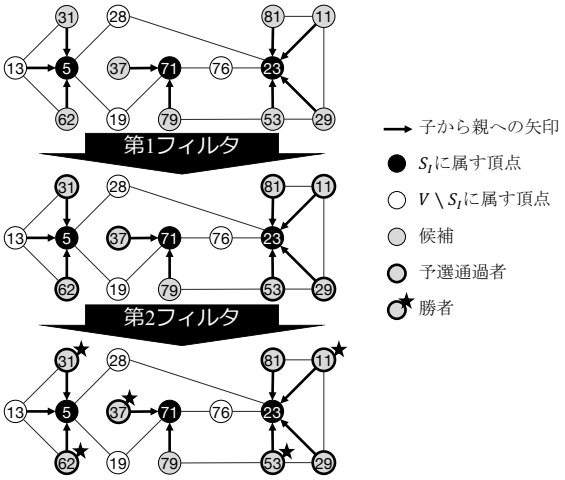


図 4: 第 1 フィルタと第 2 フィルタ

Proof. S_I が 1-MIS であり，勝者 $v \in V$ が存在すると仮定して，背理法で証明する．u を v の親とする．候補の定義より，u の子の中に，v に隣接しないような別の候補が少なくとも 1 つ存在する．そのような候補を w とする．v と w は候補なので，$S_I \cap (N_v \cup N_w) = \{u\}$ が成り立つ．したがって，$S_O = S_I \cup \{v, w\} \setminus \{u\}$ は独立集合であり，$|S_O| = |S_I| + 1$ が成り立つ．これは，S_I が 1-MIS であるという仮定に矛盾する． □

最終的な S_O の決定について述べる．$S_A(S_I)$ を，隣接する勝者が 2 つ未満であるような S_I に属す全ての頂点からなる集合とし，以降，単に S_A と表す．$S_B(S_I)$ を，$v.\texttt{parent} \notin S_A$ であるような全ての勝者 v からなる集合とし，以降，単に S_B と表す．これらの定義において，$S_A \cup S_B$ は独立集合であり，S_I が 1-MIS でないならば，$|S_A \cup S_B| \geq |S_I| + |S_B|/2 \geq |S_I| + 1$ が成り立つ．これは，次の (i)，(ii) が成り立つからである．(i) 補題 4.3 より，$S_B \neq \emptyset$ が成り立つ．(ii)$S_I \setminus S_A$ に属す各頂点は，その隣接頂点に少なくとも 2 つの勝者が存在する．したがって，$|S_B| \geq 2|S_I \setminus S_A|$ が成り立つ．一方で，S_I が 1-MIS ならば，補題 4.4 より，$S_A = S_I$ と $S_B = \emptyset$ が成り立つ．ここで，S_I が 1-MIS でないならば，$S_A \cup S_B$ は独立集合であるが，極大独立集合であることは保証されていない．図 4 の例では，$S_A \cup S_B = \{11, 31, 53, 62, 71\}$ であり，$S_A \cup S_B \cup \{28\}$ も独立集合となるため，$S_A \cup S_B$ は極大独立集合ではない．そこで，$S_C(S_I)$ を，次の 2 つを満たすような非勝者 $p_i \notin S_I$ の極大集合とし，以降，単に S_C と表す．

- p_i の隣接頂点に $S_A \cup S_B$ に属す頂点が存在しない．
- S_C に属し，p_i より識別子が小さいような隣接頂点が存在しない．

集合 $S_A \cup S_B \cup S_C$ は，$p_i \in S_C$ に属す各頂点が $S_A \cup S_B \cup S_C$ に属す隣接頂点を持たないため，独立集合である．さらに，$S_A \cup S_B \cup S_C$ が極大独立集合である．なぜならば，もし，$S_A \cup S_B \cup S_C$ が極大独立集合でないならば，$S_A \cup S_B \cup S_C$ に属す隣接頂点を持たないような，非勝者 $v \in V \setminus S_I$ が存在するはずである．しかし，そのような非勝者 v は，S_C の定義より，S_C に属すはずである．よって矛盾が生じるので，$S_A \cup S_B \cup S_C$ は極大独立集合である．以上より次の 3 つの補題が成り立つ．

補題 4.5. S_I が極大独立集合ならば，$S_A \cup S_B \cup S_C$ は極大独立集合である．

補題 4.6. S_I が極大独立集合であり，1-極大独立集合でないならば，$|S_A \cup S_B \cup S_C| \geq |S_I| + |S_B|/2 + |S_C| \geq |S_I| + 1$ が成り立つ．

補題 4.7. S_I が 1-極大独立集合ならば，$S_A \cup S_B \cup S_C = S_I$ と $S_B = S_C = \emptyset$ が成り立つ．

したがって，$S_O = S_A \cup S_B \cup S_C$ である．S_I が 1-極大独立集合でないならば，$|S_O| \geq |S_I| + 1$ が成り立ち，S_I が 1-極大独立集合であるならば，$S_O = S_I$ が成り立つ．図 4 において，$S_A = \{71\}$,，$S_B = \{11, 31, 53, 62\}$,，$S_C = \{28\}$

であり，$|S_O| = 6 > 3 = |S_I|$ が成り立つ．図 3 は，S_I が図 2 で示されるような Inc の 1 回の実行により導かれる S_O を表す．

証明は割愛するが，上記補題により，次の 3 つの補題が得られ，定理 4.11 が導ける．

補題 4.8 (Shiftable Convergence)．状況 $\mathcal{L}_{\text{input}}$ から始まる Inc の任意の極大実行 ϱ は，$\gamma^{\text{copy}} \in \mathcal{L}_{\text{input}}$ であるような状況 γ で終了する．

補題 4.9 (Loop Convergence)．$\varrho_0, \varrho_1 \ldots$ を Inc の極大実行の無限な系列とする．ここで，各 $i \geq 0$ に対して，$\varrho_i = \gamma_{i,0}, \gamma_{i,1}, \ldots, \gamma_{i,s_i}$，$\gamma_{0,0} \in \mathcal{L}_{\text{input}}$，$\gamma_{i+1,0} = \gamma_{i,s_i}^{\text{copy}}$ が成り立つとする．そのとき，ある $j \leq n$ に対して，$\gamma_{j,s_j} \in \mathcal{C}_{\text{goal}}(Inc, E_{\text{MIS}})$ と $R(\varrho_0) + R(\varrho_1) + \ldots R(\varrho_j) = O(n)$ が成り立つ．

補題 4.10 (Correctness)．任意の状況 $\gamma \in \mathcal{C}_{\text{goal}}(Inc, E_{\text{MIS}})$ は $\mathcal{L}_{\text{1MIS}}$ を満たす．

定理 4.11. アルゴリズム $\mathbf{Loop}(Inc, E_{\text{MIS}}, Init)$ は，述語 $\mathcal{L}_{\text{1MIS}}$ に対するサイレントな自己安定アルゴリズムである．任意の状況から始まる $\mathbf{Loop}(Inc, E_{\text{MIS}}, Init)$ の極大実行は $O(nD)$ ラウンド以内に終了する．アルゴリズム $\mathbf{Loop}(Inc, E_{\text{MIS}}, Init)$ の 1 頂点あたりの空間計算量は $O(\log n)$ ビットである．任意の状況 $\gamma \in \mathcal{C}_{\text{goal}}(Inc, E)$ は $\mathcal{L}$ を満たす．

5 結論

反復合成 [3] を用いて，1-極大独立集合問題を解く，サイレントな自己安定アルゴリズムを提案した．n をグラフの頂点数，D をグラフの直径とすると，収束時間は $O(nD)$ ラウンドであり，1 頂点あたりの空間計算量は $O(\log n)$ ビットである．

謝辞 本研究は，JSPS 科研費 17K19977, 18K18000, 19H04085,19K11826，および JST SICROP，JPMJSC1606 の支援を受けたものです．

参考文献

[1] Baruch Awerbuch, Michael Luby, Andrew V. Goldberg, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science*, pages 364–369. IEEE, 1989.

[2] Béla Bollobás, Ernest J. Cockayne, and Christina M. Mynhardt. On generalised minimal domination parameters for paths. In *Annals of Discrete Mathematics*, volume 48, pages 89–97. Elsevier, 1991.

[3] Ajoy K. Datta, Laurence L. Larmore, Toshimitsu Masuzawa, and Yuichi Sudo. A self-stabilizing minimal k-grouping algorithm. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, page 3. ACM, 2017.

[4] Ajoy K. Datta, Laurence L. Larmore, Toshimitsu Masuzawa, and Yuichi Sudo. A self-stabilizing minimal k-grouping algorithm. *arXiv preprint arXiv:1907.10803*, 2019.

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

[6] Shlomi Dolev. *Self-stabilization*. MIT press, 2000.

[7] Michiyo Ikeda, Sayaka Kamei, and Hirotsugu Kakugawa. A space-optimal self-stabilizing algorithm for the maximal independent set problem. In *the Third International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 70–74, 2002.

[8] Eijiro Namba. A hierachical self-stabilizing 1-MIS algorihtm, (in Japanese). Master's thesis, Osaka University, 2017.

[9] Zhengnan Shi, Wayne Goddard, and Stephen T. Hedetniemi. An anonymous self-stabilizing algorithm for 1-maximal independent set in trees. *Information Processing Letters*, 91(2):77–83, 2004.

[10] Sandeep K. Shukla, Daniel J. Rosenkrantz, S. Sekharipuram Ravi, et al. Observations on self-stabilizing graph algorithms for anonymous networks. In *Proceedings of the second workshop on self-stabilizing systems*, volume 7, page 15, 1995.

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

仮想グリッドネットワークにおける3点間の通信経路最適化分散アルゴリズムについて

On a Self-optimizing Three Nodes Routing Algorithm based on local information in Virtual Grid Networks

澤田裕介 [1]　金鎔煥 [1]　片山喜章 [1]
Yusuke Sawada　Yonghwan Kim　Yoshiaki Katayama

名古屋工業大学大学院工学研究科情報工学専攻 [1]
Nagoya Institute of Technology, Graduate School of Computer Science and Engineering

概 要

本論文では，仮想グリッドネットワーク上での経路最適化問題を扱う．仮想グリッドネットワークとは，無線通信端末で構成されている無線ネットワークを一定領域（以下，グリッドセルとする）ごとに仮想的に分割し，各グリッドセル毎に代表の無線通信端末（以下，ルータとする）を一つ決定し，隣接しているセル同士のルータを連結させた格子状の仮想ネットワークである．仮想グリッドネットワークにおける各無線通信端末間の通信は，複数のルータを経由することで実現され，送信元の無線通信端末 (以下, ソースとする) や送信先の無線通信端末 (以下, ターゲットとする) が移動する場合でも，移動先の隣接グリッドセルのルータを追加し経路を延長することで，通信経路を保持させることが可能である．しかし，このような経路の延長を繰り返す場合，通信経路が冗長になる恐れがある．そこで，仮想グリッドネットワーク上での経路を最適化する経路最適化問題が研究されている．仮想グリッドネットワーク上での経路最適化問題では，利用可能な局所情報と問題の可解性の関係を明らかにすることを主な目的として研究が進められており，ソース, ターゲットがそれぞれ 1 台であるグリッドネットワーク上の任意の通信経路が与えられたときに，各ルータが共通座標系を持たず，経路上 2 ホップ先までの情報を用いて局所的な経路の変更を繰り返すことで経路最適化ができることが証明されている．しかし，ターゲットが複数ある場合, それぞれの通信経路が既存研究の方法で最適化されたとしても通信経路全体で使用されるルータ数の観点で最適化されない場合がある．そこで，1 台のソースからデータを受け取り複数のターゲットへ分配する分配ルータ (以下, ディストリビュータとする) を設置することを考える．

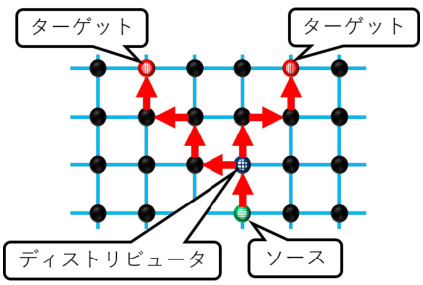


図 1: ソースが 1 台, ターゲットが 2 台の場合の冗長な経路

このとき図 1 と図 2 のようにディストリビュータの位置によって経路全体で必要となるルータ数が変わってしまう．図 1 ではルータの延べ数はソース, ターゲットも含めて 9 台, 図 2 ではルータの延べ数は 7 台となっている．本論文では，経路上 3 ホップ先までの情報を用いてディストリビュータの位置を最適化し，文献 [12] のアルゴリズムと組み合わせて経路全体を最適化するアルゴリズムを提案する．

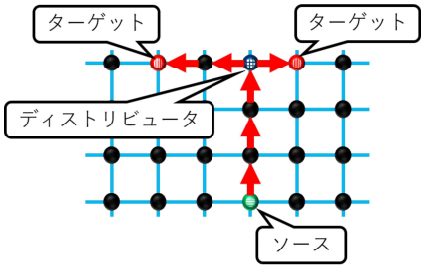


図 2: ソースが 1 台, ターゲットが 2 台の場合のルータの延べ数最小経路

1　はじめに

1.1　研究の背景

近年，無線通信端末の普及に伴い，無線ネットワークが注目されている [1–4] ．無線ネットワークの例として，無線通信端末間の通信に基地局や固定網を利用せず，無線通信端末のみでネットワークを構築するモバイルアドホックネットワーク（MANET）[1, 2] や，無線通信可能な多数のセンサを分散配置し，センサから得られた情報をセンサ同士の通信によって基地局へ伝送する無線センサネットワーク（WSNs）[1,5] などが挙げられる．各無線通信端末の通信範囲は限られており，この通信範囲内に存在する無線通信端末とのみ直接通信可能である．そのため，送信先の無線通信端末（以下，ターゲットとする）が送信元の無線通信端末（以下，ソースとする）の通信範囲外に存在する場合には，ほかの無線通信端末を中継して通信する，マルチホップ通信を行う．マルチホップ通信では，ターゲットまでの通信経路を構築する（以下，ルーティングとする）必要があり，これまでに多数のルーティングプロトコルが提案されている [1,5–11]．通常，ルーティングプロトコルでは，通信経路の評価尺度を定め，その評価尺度の下での最適な通信経路を構築する．通信経路の評価尺度として，ホップ数，通信遅延，消費エネルギー，故障耐性などがある [5]．中でも，

ホップ数は最もよく利用されている評価尺度の一つであり，ホップ数が少ない通信経路では通信遅延や消費エネルギーについても少ないことが期待できる．

無線ネットワークのモデルとして，仮想グリッドネットワークがある．このモデルでは，無線通信端末が存在する領域を，正方形状のグリッドセルによって仮想的に格子状に分割し，グリッドセルから構成される仮想グリッドを考える．そして，各セル毎に一つの代表の無線通信端末（以下，ルータとする）を決定し，4 方向に隣接しているグリッドセルのルータ同士を連結させ，格子状の仮想ネットワーク（以下，仮想グリッドネットワークと呼ぶ）を構築する．図 3 では仮想グリッドネットワークのイメージ図を表している. 白い円がルータとして選ばれなかった無線通信端末, 黒い円がルータを示している. また, 緑の点線はグリッドセルの境界線を表し, 水色の実線はルータが連結している様子を表す. 以下の説明ではグリッドセルの境界線とルータとして選ばれなかった無線通信端末を省略した, 図 3 の矢印の右側の図を用いる.

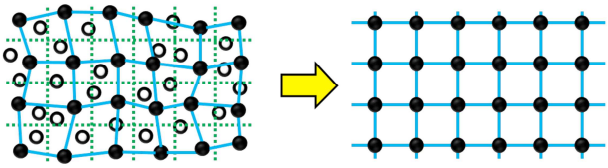


図 3: 仮想グリッドネットワーク

仮想グリッドネットワーク上の通信経路は，経由するグリッドセルを決定し，決定したグリッドセルのルータを通信の中継として利用することで構成される．このモデルは，ルータ以外の無線通信端末をスリープ状態とすることが可能なためシステム全体としてのエネルギー効率が良く，トポロジの単純化によってアルゴリズムの設計が容易となる等の利点から，幅広く利用されている [12–17]．本研究では，仮想グリッドネットワーク上の通信経路について扱う．

移動端末による無線ネットワークでは，無線通信端末が通信範囲外に移動し，既存の通信経路が利用できなくなる場合がある．このような場合には，利用可能な通信経路に更新する必要がある．仮想グリッドネットワークでは，ソースやターゲットが隣接グリッドセルに移動した場合，移動した隣接グリッドセルに経路を延長することで，通信経路を保持させることが可能である．しかし，図 4 のように，ソースやターゲットが移動を繰り返し，移動のたびに経路を延長する場合，冗長なホップ数の通信経路となってしまう．図 4 において, 赤い矢印はルータが通信を行い, 通信を送る方向を表す.

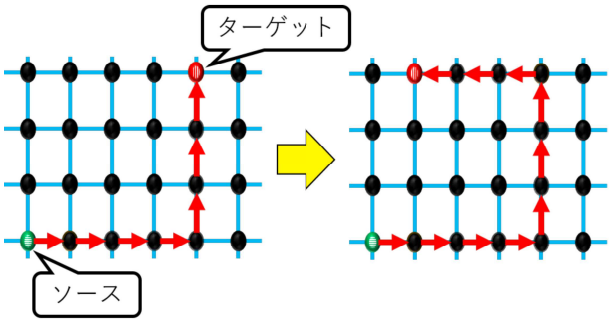


図 4: 冗長な通信経路

この問題を解決するには，ホップ数最小の通信経路を再構築すればよいが，そのためにシステム全体の状況（大域状況）を利用することは，多大な通信量によるエネルギー消費量の増大など，一般的な無線ネットワークにおいては望ましくない．そこで，通信経路の大域的な情報を用いず，局所的な情報のみに基づいて既存の通信経路を局所的に変更することでホップ数を改善し，この局所的な変更を繰り返すことで大域的にホップ数最小の通信経路を構築できることが望ましい．

このような，仮想グリッドネットワーク上の冗長な通信経路を最短化するアルゴリズムが提案されている [12–17]．S.Takatsu らは文献 [12] で，各無線通信端末が共通の座標系を持ち，通信経路に沿って前後 3 ホップ先までの経路情報がわかれば，ネットワークの規模に関係なく局所情報のみで最短経路を自律的に構築可能であるアルゴリズムを提案した．宮川らは文献 [13] で，文献 [12] で提案された Zigzag を改良し，前後 2 ホップ先までの経路情報のみで同じ問題を解決するアルゴリズムを提案した．風岡らは文献 [14] において, 各ルータが共通の座標系を持っていなくても文献 [13] と同様に経路上 2 ホップ先までの情報のみを用いて同じ問題を解決するアルゴリズムを提案した．さらに金らは文献 [15] で，各無線通信端末が共通の座標系を持ち，前後 3 ホップ先までの経路情報を用いて，2 つのターゲットまでの最短経路を構築するアルゴリズムを提案した．金らは文献 [16, 17] で，仮想グリッド上の通信経路を三次元に拡張し，各無線通信端末が共通の座標系を持ち，経路上で前後 3 ホップ先までの局所情報を用いて，最短経路を自律的に構築するアルゴリズムを提案した．

既存研究 [12–14, 16, 17] では，ターゲットが 1 つである経路の場合についての経路最適化アルゴリズムを提案しているが，本研究では，文献 [15] と同様にターゲットが 2 つある経路の場合のアルゴリズムを提案する．文献 [15] では, 仮想ホームと呼ばれる,3 点の経路をつなぐルータを配置し, その位置と周辺の経路を変更することによって経路全体のルータの延べ数を最小化するアルゴリズムを提案している. しかし, このアルゴリズムでは仮想ホームの初期配置がホームと呼ばれる送信ルータと同一の位置でなければ実行することができない．そこで，本論文では，3 点の経路をつなぐルータの初期配置をソース, ターゲットと同一の位置を含めた仮想グリッドネットワーク上の任意の場所に置き, その位置を変更するこ

とで経路を最適化することを考える．本論文では，1 台のソースと 2 台のターゲット間の経路に関して，これら 3 台のルータをつなぐルータ（ディストリビュータと呼ぶ）の位置を移動させることでこの経路を最適化するアルゴリズムを提案する．提案アルゴリズムは，共通座標系を仮定し，ディストリビュータとソースおよびターゲット間の経路最適化については文献 [12] のアルゴリズムを利用しており，経路上の前後 3 ホップの経路情報を利用している．

1.2 論文の構成

本論文の構成は次の通りである．第 2 章では関連研究を示す．第 3 章では本研究のモデルについて述べる．第 4 章では提案アルゴリズムについて説明する．第 5 章では提案アルゴリズムの正当性を証明する．最後に第 6 章でまとめを述べる．

2 関連研究

2.1 Zigzag アルゴリズム

S.Takatsu らは文献 [12] で，各ルータが共通の座標系を持ち，前後 3 ホップ先までの経路情報がわかれば，ネットワークの規模に関係なく局所情報のみで最短経路自律的に構築可能であることを示した．ここでは，文献 [12] で提案された，Zigzag アルゴリズムを紹介する．

文献 [12] では，図 5 のように，黒い円で表されているルータが仮想的にグリッド状に配置されていると考え，各ルータは自身と実線で 4 方向に連結しているルータとの間で通信を行うことが可能である. 緑の点線は仮想グリッドの境界線, 白い円はルータに選ばれなかった無線通信端末を表す. この仮想グリッドネットワーク上に存在するルータ p_i とルータ p_j，ルータ p_j とルータ p_k が連結しているとき，

$$|\overrightarrow{p_ip_j}| = |\overrightarrow{p_jp_k}| = 1$$

かつ,

$$\overrightarrow{p_ip_j} \cdot \overrightarrow{p_jp_k} = 0 \text{ または } |\overrightarrow{p_ip_j} \cdot \overrightarrow{p_jp_k}| = 1$$

が成り立つものとする. ここで, $\overrightarrow{p_ip_j}$, $\overrightarrow{p_jp_k}$ はそれぞれ p_i から p_j へ通信を行う方向, p_j から p_k へ通信を行う方向を表し, $|\overrightarrow{p_ip_j}|$, $|\overrightarrow{p_jp_k}|$ はそれぞれ p_i から p_j へ通信を行う距離, p_j から p_k へ通信を行う距離を表す.

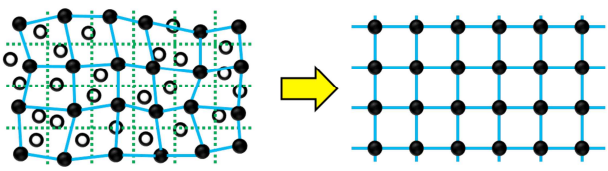


図 5: 仮想的にグリッド状に配置されたルータ

また，ルータを，経路に現れる順に $p_0, p_1, ..., p_n$ とし，経路を，$(p_0, p_1, ..., p_n)$ というように，ルータの順序集合で表す．

Zigazag アルゴリズムでは，仮想グリッドネットワーク上の任意の経路が与えられたときに，以下の 3 種類の局所変更を繰り返すことで，経路全体を最適化する．

shortcut1 $\overrightarrow{p_{i-1}p_i} \cdot \overrightarrow{p_ip_{i+1}} = -1$ ならば, $(..., p_{i-1}, p_i, p_{i+1}, ...)$ を $(..., p_{i-1}, ...)$ に変更する (図 6).

shortcut2 $\overrightarrow{p_{i-1}p_i} \cdot \overrightarrow{p_ip_{i+1}} = 0$ かつ, $\overrightarrow{p_{i-1}p_i} \cdot \overrightarrow{p_{i+1}p_{i+2}} = -1$ ならば,$(..., p_{i-1}, p_i, p_{i+1}, .p_{i+2}, ..)$ を $(..., p_{i-1}, .p_{i+2}, ..)$ に変更する (図 7).

zigzag $\overrightarrow{p_{i-1}p_i} \cdot \overrightarrow{p_ip_{i+1}} = 1$ かつ, $\overrightarrow{p_ip_{i+1}} \cdot \overrightarrow{p_{i+1}p_{i+2}} = 0$ ならば,$(..., p_{i-1}, p_i, p_{i+1}, .p_{i+2}, ..)$ を $(..., p_{i-1}, p_i, q, .p_{i+2}, ..)$ に変更する. ただし, q は $\overrightarrow{p_iq} = \overrightarrow{p_{i+1}p_{i+2}}$ を満たすルータとする (図 8).

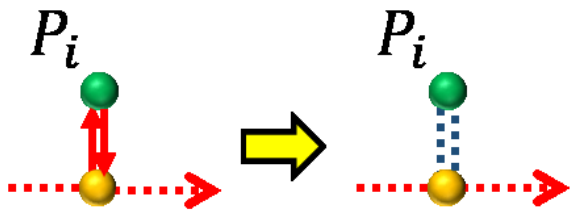


図 6: shortcut1

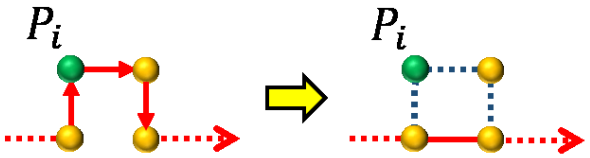


図 7: shortcut2

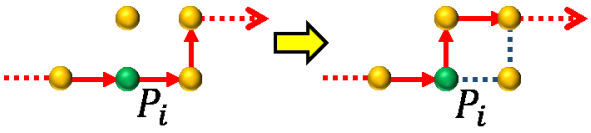


図 8: zigzag

Zigzag アルゴリズムでは，複数の適用可能な局所変更が同時に適用されると，経路が切断される場合がある．そこで，経路の切断を回避するために，局所変更に優先順位が設けられている．局所変更の優先順位を，表 1 に示す．

表 1: 衝突した場合の優先順位

p_i で適用可能な変更	shortcut1	shortcut2	zigzag
p_{i-2} で優先される局所変更	-	shortcut2	-
p_{i-1} で優先される局所変更	shortcut1	shortcut1,2	shortcut2,zigzag
p_{i+1} で優先される局所変更	-	shortcut1	shortcut1,2
p_{i+2} で優先される局所変更	-	shortcut1	shortcut1,2

表 1 では, p_i で適用可能な変更が存在する場合, その局所変更を実行できるかどうかを表している. 例えば, p_i で shortcut1 が適用可能な場合, p_{i-2}, p_{i+1}, p_{i+2} で適用可能な局所変更が存在し実行が行われた場合でも, 経路が切断されることがないため実行可能である. しかし, p_{i-1} で shortcut1 が適用可能な場合,p_i, p_{i-1} で shortcut1 の局所

変更を実行してしまうと経路が切断されてしまう．そこで，p_{i-1} で shortcut1 が適用可能な場合，p_i で shortcut1 の局所変更は実行しない．p_{i-1} で shortcut2, zigzag が適用可能な場合，経路が切断されることがないため，p_i で shortcut1 の局所変更を実行する．また，p_i から前後 3 ホップの経路情報を参照した場合，p_{i-2} から p_{i+2} において適用可能な局所変更が存在するか確認することが出来る．この範囲外での局所変更は p_i で適用可能な変更には直接影響しないため，p_{i-2} から p_{i+2} において経路が切断されないように優先順位を設定している．

2.2 経路結合アルゴリズム

金らは文献 [15] で，ソースが 1 つ，ターゲットが 2 つ与えられている仮想グリッドネットワーク上で，各ルータが共通の座標系を持ち，経路上前後 3 ホップの経路情報が参照可能なモデルで任意の経路を最短にするアルゴリズムを提案した．文献 [15] では，図 9 のように経路上の隣接するルータの位置を共通座標系に基づく方向ラベルに基づき経路情報として保持している．方向ラベルは L, U, R, D の 4 種類である．

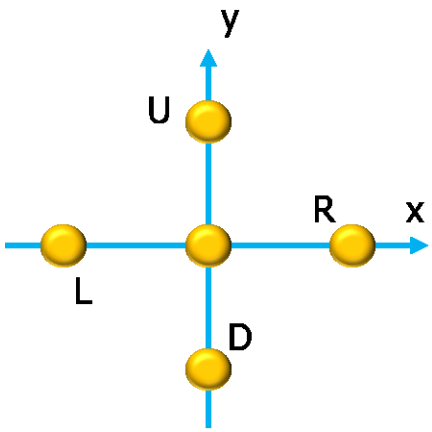


図 9: 各ルータの共通座標系に基づく方向ラベル

文献 [15] では，ホームと呼ばれる送信ルータから 2 ホップの経路の出力を表 2 のように修正した Zigzag アルゴリズム [15] の改良アルゴリズムと提案されているアルゴリズムを組み合わせて実行することでソースが 1 つ，ターゲットが 2 つ与えられている任意の経路を最短にする．

表 2: zigzag アルゴリズムの改良

ホームから 2 ホップの距離の方向	修正された距離の方向
(U, R)	(R, U)
(L, U)	(U, L)
(D, L)	(L, D)
(R, D)	(D, R)

ここでは，文献 [15] で提案されている経路結合アルゴリズムを紹介する．文献 [15] では，文献 [12] と同様に図 5 のように，黒い円で表されているルータが仮想的にグリッド状に配置されていると考え，各ルータは自身と実線で 4 方向に連結しているルータとの間で通信を行うことが可能である．緑の点線は仮想グリッドの境界線，白い円はルータに選ばれなかった無線通信端末を表す．この仮想グリッドネットワーク上に存在するルータ p_i とルータ p_j，ルータ p_j とルータ p_k が連結しているとき，

$$|\overrightarrow{p_ip_j}| = |\overrightarrow{p_jp_k}| = 1$$

かつ，

$$\overrightarrow{p_ip_j} \cdot \overrightarrow{p_jp_k} = 0 \text{ または } |\overrightarrow{p_ip_j} \cdot \overrightarrow{p_jp_k}| = 1$$

が成り立つものとする．ここで，$\overrightarrow{p_ip_j}$, $\overrightarrow{p_jp_k}$ はそれぞれ p_i から p_j へ通信を行う方向，p_j から p_k へ通信を行う方向を表し，$|\overrightarrow{p_ip_j}|$, $|\overrightarrow{p_jp_k}|$ はそれぞれ p_i から p_j へ通信を行う距離，p_j から p_k へ通信を行う距離を表す．

初期条件として，ホームから 2 つのターゲットまでを結ぶ経路が存在する．ホームの位置から仮想ホームと呼ばれる位置変更が可能なルータが 2 つのターゲットまでを結ぶ経路を変更しながら位置を変更することにより，経路全体の使用するルータの延べ数を最小化する．経路結合アルゴリズムでは，仮想グリッドネットワーク上の 3 点間の任意の経路が与えられたときに，図 10, 11, 12 のような局所変更を繰り返すことで，ルータの延べ数を最小化する．家の印がついた円は仮想ホーム，白い円は移動前の仮想ホームの位置を表す．また，点線は仮想グリッドネットワーク上のルータの連結部分を表し，縦の点線と横の点線の交点にルータが存在する．先端が円の矢印はホームから仮想ホームまでの経路，先端がひし形の矢印，先端が三角形の矢印は仮想ホームからターゲットまでの経路を示している．

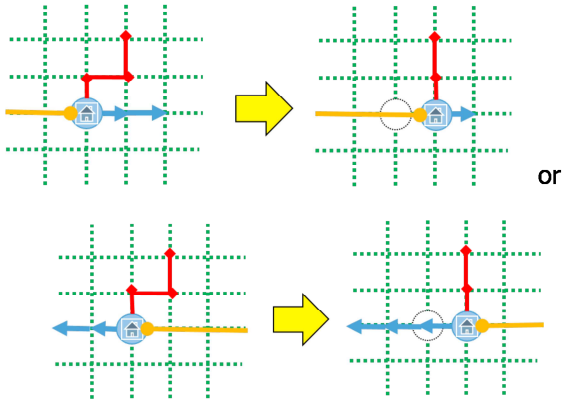


図 10: U-sharp

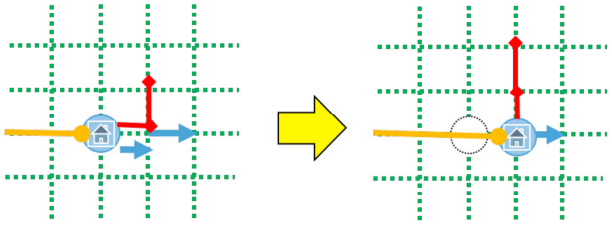


図 11: Common Path

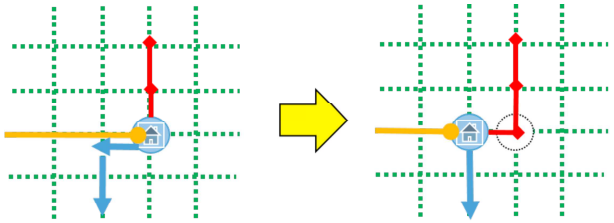


図 12: Redundant Path

2.3 その他の関連研究

先に紹介した二つの研究以外にも，仮想グリッドネットワーク上の冗長な通信経路の最適化について，様々な研究がなされている．宮川らは文献 [13] で, 文献 [12] で提案された Zigzag を改良し，より少ない前後 2 ホップ先までの経路情報のみで同じ問題を解決するアルゴリズムを提案した．風岡らは文献 [14] で, 各ルータが共通座標系を持っていなくても，経路上 2 ホップ先までの情報のみを用いて，局所的な経路の変更を繰り返すことで最短経路を構成するアルゴリズムを提案した．

金らは [16,17] で，仮想グリッド上の通信経路を三次元に拡張し，各ルータが共通の座標系を持ち，経路上で前後 3 ホップ先までの局所情報を用いて，最短経路を自律的に構築するアルゴリズムを提案した．

3 モデル

3.1 仮想グリッドネットワークのモデル

仮想グリッドネットワークは，無線ネットワークを仮想的に正方形状のグリッドセルに格子状に分割することにより得られる．このとき，各セルに 1 つ以上の無線通信端末が存在するように分割する．グリッドセルのサイズは，同じグリッドセル内の無線通信端末と，隣接するグリッドセル内の無線通信端末が直接通信できるように定められる．従って，無線通信端末の通信範囲を R とすると，グリッドセルの一辺の大きさは，$\frac{R}{\sqrt{5}}$ 以下となるように決定する．

仮想グリッドネットワークを，無向辺グラフ $N_g = (V_g, E_g)$ と表す．ここで，V_g はルータの集合，E_g は通信リンクの集合を表す．ルータ $p_i, p_j \in V_g$ それぞれを含むグリッドセルが辺を共有して隣接しているときのみ，通信リンク $(p_i, p_j) \in E_g$ が存在する．つまり，仮想グリッドネットワーク N_g は格子グラフとなる．また，ルータが仮想的にグリッド状に配置されているため，$(p_i, p_j), (p_j, p_k) \in E_g$ であるとき，図 13 のように，$\overrightarrow{p_ip_j} \cdot \overrightarrow{p_jp_k} = 0$ または,$|\overrightarrow{p_ip_j} \cdot \overrightarrow{p_jp_k}| = 1$ が成り立つものとする．黒い円がルータ, 緑の点線は仮想グリッドの境界線, 白い円はルータに選ばれなかった無線通信端末を表す．

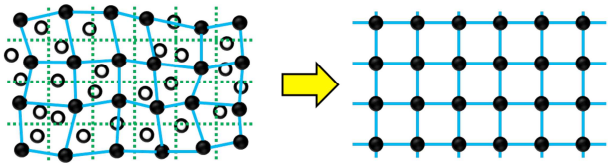


図 13: 仮想的にグリッド状に配置されたルータ

ルータ同士の通信は，完全同期通信である．各ルータは通信が可能な 4 方向に隣接するルータの位置のみ把握し, 直接通信できないルータから自身のルータまでの位置情報を持たないものとする．ルータは経路全体に関する情報を持っておらず, 経路上 3 ホップ先までの経路情報のみを持っている．経路上の各ルータは, 軸の方向と向き, 単位距離に合意を持ち, 共通の座標系をもつ. 図 14 のように，経路上の各ルータは，共通の座標系に基づく方向ラベル U(Up)，D(Down)，R(Right)，L(Left) を保存している．また経路上の各ルータは 4 方向に隣接するルータと通信を行い, お互いが所持している方向ラベルの情報を交換することで経路上の前後 3 ホップの経路情報を最新の情報に更新する．

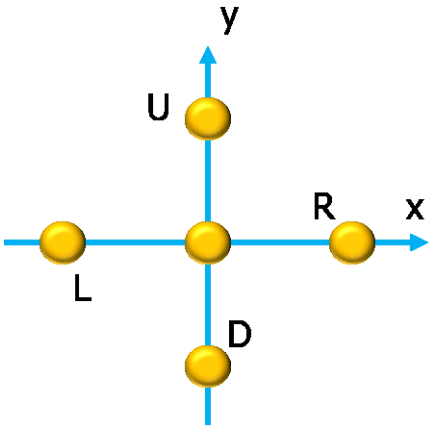


図 14: 各ルータの共通座標系に基づく方向ラベル

本研究では,1 台のソース s と 2 台のターゲット t_1 と t_2 を仮定し, さらに送信する情報の分配を担う 1 台の分配ルータ (以下, ディストリビュータとする)d を考える．ソース s はディストリビュータ d までの経路 P^s を持ち, この経路は s から d までのルータ列 P^s で次のように表現される：$P^s = (p_0^s, p_1^s, p_2^s, \cdots, p_i^s)$ (ただし,$0 \leq j < i$ の j に対して,p_j^s と $p_{(j+1)}^s$ は隣接ルータである)．同じく d は t_1 と t_2 までの異なる 2 つの経路 P^{t_1} と P^{t_2} を持つ．$\overline{P^s}$ は P^s の逆順列である：$\overline{P^s} = (p_i^s, p_{(i-1)}^s, \cdots, p_1^s, p_0^s)$.

3.2 仮想グリッドネットワーク上での通信経路最適化問題

仮想グリッドネットワークでは，ターゲットが現在のグリッドセルから隣接グリッドセルに移動した場合，移動した隣接グリッドセルに経路を延長することで，通信経路を保持させることが可能である．しかし，図 15 のようにターゲットが移動を繰り返し移動のたびに経路を延長する場合，冗長なホップ数の通信経路となってしまう．

図 15 において, 赤い矢印はルータが通信を行い, データを送る方向を表す.

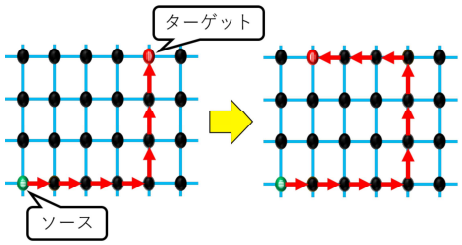


図 15: 冗長な通信経路

この問題は既存研究 [12–14] の方法を用いることで解決できる. ターゲットが 2 台の場合, それぞれのターゲットまでの経路が既存研究 [12–14] の方法を用いると, P^s, P^{t1}, P^{t2} の最短化は保証可能であるが, ディストリビュータの位置によって経路全体のルータの延べ数が増減してしまう場合がある. つまり, 経路全体のルータの延べ数の観点で見た場合, 既存研究 [12–14] を用いた各経路の最短化は必ずしも経路全体の延べ数を最小にするわけではない. 本論文では，このようにして生じた経路全体のルータの延べ数の観点で冗長な通信経路を，経路全体のルータの延べ数最小の通信経路に更新することを考える. 仮想グリッドネットワーク上での通信経路最適化問題は，次のように定義される.

定義 1. 仮想グリッドネットワーク上での経路最適化問題は, 予め決められたルータ s,d,t_1,t_2 及び, 経路 P^s, P^{t_1}, P^{t_2} が与えられたとき, 全ての経路の連結性を保持しつつ, 全経路上のルータの延べ数が最小の経路を構成する問題である.

4 提案アルゴリズム

4.1 局所的な変更

提案アルゴリズムは, 与えられた 3 つの経路 P^s, P^{t_1}, P^{t_2} に対してディストリビュータ d が局所的な変更を行う. 既存研究 [12] の局所変更のみでは, ルータ数を減少できるが, ルータの延べ数は最小にすることができない. ルータの延べ数が最小となる経路はディストリビュータ d を適切な位置に移動させ, それぞれの経路 P^s, P^{t_1}, P^{t_2} において最短経路を形成しているものである. 提案アルゴリズムの局所変更では, ディストリビュータ d が移動する際に経路を単純に延長することで経路の形状を変化させる. そこで, 提案アルゴリズムの局所変更と既存研究 [12] の仮想グリッドネットワークにおける 2 点間の経路最適化をそれぞれ実行することでルータの延べ数が最小の経路を形成する. 与えられた通信経路に対し, 提案アルゴリズムの局所変更と既存研究 [12] の局所変更を同時に行い, その局所変更が終了するまでの過程をラウンドと呼ぶ. このラウンドを繰り返すことでルータの延べ数を最小にする. このとき, 提案アルゴリズム, [12] のアルゴリズムの入力は,$\overline{P^s}$,P^{t_1},P^{t_2} とする（P^s は逆順列であることに注意）. よって本節では、d の移動アルゴリズムを紹介する. d の位置によって全体経路上のルータの数が変化するため,d を最適な位置に移動させることで通信経路最適化問題を解決する.

ディストリビュータ d は, 3 つの経路 $\overline{P^s}$,P^{t_1},P^{t_2} の最初の 3 ホップまでの経路情報が参照可能である（3 つの経路は全部 d から始まるため）. 提案アルゴリズムでは, 3 つの経路 $\overline{P^s}$,P^{t_1},P^{t_2} における p_1^s,$P_1^{t_1}$,$P_1^{t_2}$ で既存研究 [12] の局所変更が適用可能な場合は局所変更を行わない. 提案アルゴリズムでは, 3 つの経路 $\overline{P^s}$,P^{t_1},P^{t_2} のうち,2 つの経路を選択する. 選択する順序は P^{t_1},P^{t_2} の組み合わせ, $\overline{P^s}$,P^{t_1} の組み合わせ, $\overline{P^s}$,P^{t_2} の組み合わせの順である. 条件が当てはまる場合, 局所変更を行う. 選択された 2 つの経路を $P^x = (p_0^x, p_1^x, ..., p_i^x)$,$P^y = (p_0^y, p_1^y, ..., p_j^y)$(ただし, $P^x, P^y \in \{\overline{P^s}, P^{t_1}, P^{t_2}\}$) とする. 以下に 3 種類の局所変更を示す.

Rule1 $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = -1$ かつ, $\overrightarrow{p_1^x p_2^x} \cdot \overrightarrow{p_1^y p_2^y} = 1$ かつ, $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = 0$ ならば, $P^x = (p_0^x, p_1^x, ...)$ を $P^x = (q, p_0^x, p_1^x ...)$, $P^y = (p_0^y, p_1^y, ...)$ を $P^y = (q, p_0^y, p_1^y ...)$ に変更する. ただし, q は $\overrightarrow{qp_0^x} = \overrightarrow{qp_0^y} = -\overrightarrow{p_1^x p_2^x} = -\overrightarrow{p_1^y p_2^y}$ を満たすルータとする.

Rule2 $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = 0$ かつ, $\overrightarrow{p_1^x p_2^x} \cdot \overrightarrow{p_0^y p_1^y} = 1$, または $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = 0$ かつ, $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^y p_2^y} = 1$ ならば, 前者なら $P^x = (p_0^x, p_1^x, ...)$ を $P^x = (p_1^y, p_0^x, p_1^x ...)$, $P^y = (p_0^y, p_1^y, ...)$ を $P^y = (p_1^y, p_0^y, p_1^y ...)$ に変更し, 後者なら $P^x = (p_0^x, p_1^x, ...)$ を $P^x = (p_1^x, p_0^x, p_1^x ...)$, $P^y = (p_0^y, p_1^y, ...)$ を $P^y = (p_1^x, p_0^y, p_1^y ...)$ に変更する.

Rule3 $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = 1$ ならば, $P^x = (p_0^x, p_1^x, ...)$ を $P^x = (p_1^x, p_0^x, p_1^x ...)$, $P^y = (p_0^y, p_1^y, ...)$ を $P^y = (p_1^y, p_0^y, p_1^y ...)$ に変更する.

局所的な変更の様子を，図 16, 17, 18 に示す. 実線の矢印は P^x, 点線の矢印は P^y の経路を表す. 提案アルゴリズムでは選択する経路の組み合わせによって 2 つ以上の局所変更のルールが発見される場合がある. 複数の局所変更のルールを実行すると不必要に冗長な経路を生成し，適用する順序によっても実行結果が異なってしまう. そのため選択された 2 つの経路を基に発見された局所変更を $Rulei - \{P^x, P^y\}$ と表し，局所変更の衝突を避けるため, 優先順位は降順で $Rule1 - \{P^{t1}, P^{t2}\}$,$Rule1 - \{P^s, P^{t1}\}$,$Rule1 - \{P^s, P^{t2}\}$,$Rule2 - \{P^{t1}, P^{t2}\}$,$Rule2 - \{P^s, P^{t1}\}$,$Rule2 - \{P^s, P^{t2}\}$,$Rule3 - \{P^{t1}, P^{t2}\}$,$Rule3 - \{P^s, P^{t1}\}$,$Rule3 - \{P^s, P^{t2}\}$ とする.

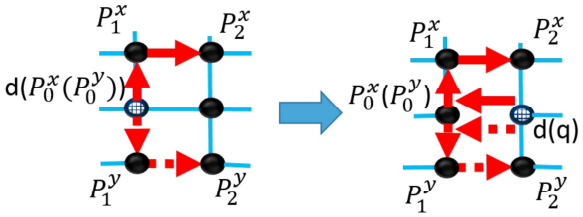


図 16: Rule1

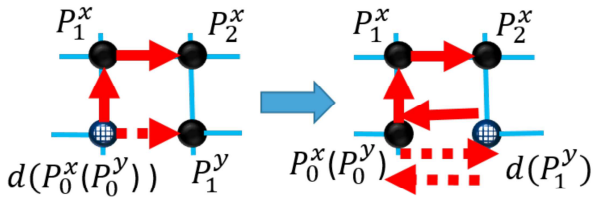


図 17: Rule2

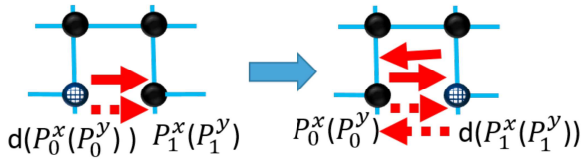


図 18: Rule3

4.2 提案アルゴリズムの実行の流れ

提案アルゴリズムは，与えられた仮想グリッドネットワーク上の経路を，既存研究 [12] の局所変更とそれぞれ独立して繰り返し適用することでルータの延べ数を最小にする．図 19, 20, 21,22, 23, 24 を使って提案アルゴリズムの実行の流れを説明する. 先端が三角形の矢印は P^s の経路, 先端が凹四角形の矢印は P^{t1} の経路, 先端がひし形の矢印は P^{t2} の経路を表す．図 19 の経路において P^{t1} と P^{t2} の組み合わせで Rule2 が発見され実行されると 図 20 のようになる．図 20 の経路において既存研究 [12] の局所変更が実行されると図 21 のようになる. 図 21 の経路において既存研究 [12] の局所変更の zigzag が発見されるためディストリビュータの移動は行わない. 図 21 の経路において既存研究 [12] の局所変更が実行されると図 22 のようになる．図 22 の経路において P^{t1} と P^{t2} の組み合わせで Rule1 が発見され実行されると図 23 のようになる．図 23 の経路において既存研究 [12] の局所変更が実行されると図 24 のようになる.

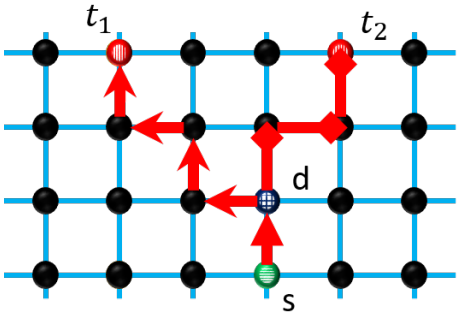


図 19: 冗長な経路

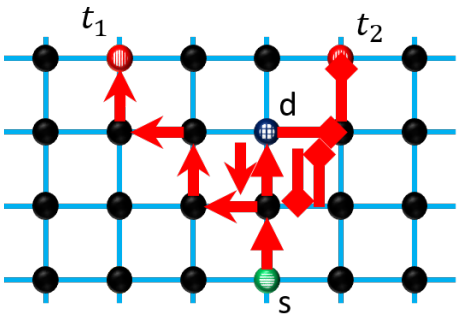


図 20: 1 ラウンド目終了後の経路

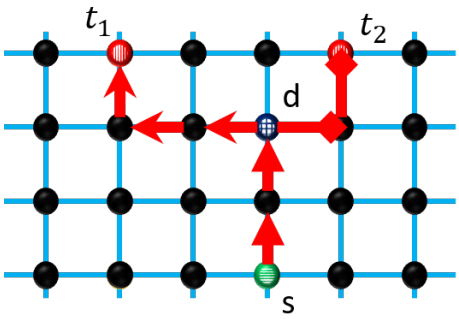


図 21: 2 ラウンド目終了後の経路

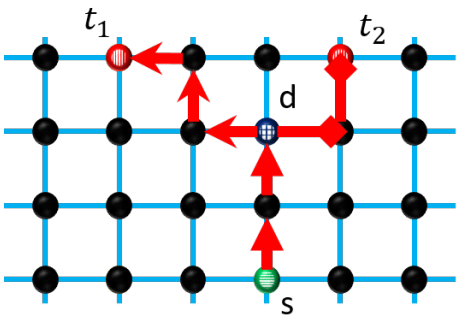


図 22: 3 ラウンド目終了後の経路

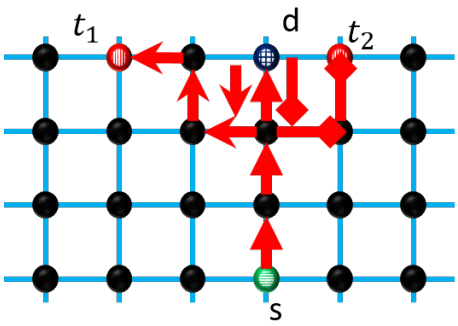


図 23: 4 ラウンド目終了後の経路

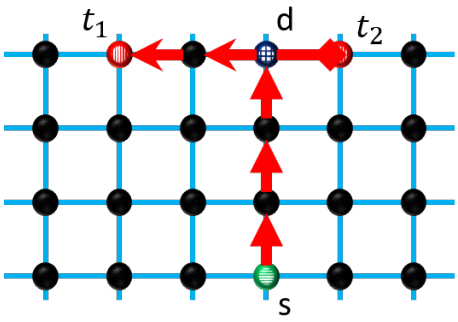


図 24: 5 ラウンド目終了後の経路

Algorithm 1 分配ノード d の移動アルゴリズム

var
  state ← null:d で局所変更が適用可能かどうかを格納する変数
  //1: Rule1, 2: Rule2, 3: Rule3, 4: d の局所変更を実行しない
  P^x ← null:経路を格納する変数
  P^y ← null:経路を格納する変数
  //P^s, P^{t1}, P^{t2}
function
  state ← null
  $P^x \leftarrow \overline{P^s}$
  ルーティング情報を確認する
if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = -1$
    state ← 4//shortcut1 が適用可能.
  else if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = 0$ **and**
      $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_2^x p_3^x} = -1$ **then**
    state ← 4//shortcut2 が適用可能.
  else if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = 1$ **and**
      $\overrightarrow{p_1^x p_2^x} \cdot \overrightarrow{p_2^x p_3^x} = 0$ **then**
    state ← 4//zigzag が適用可能
  $P^x \leftarrow P^{t1}$
  ルーティング情報を確認する
if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = -1$
    state ← 4//shortcut1 が適用可能.
  else if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = 0$ **and**
      $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_2^x p_3^x} = -1$ **then**
    state ← 4//shortcut2 が適用可能.
  else if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = 1$ **and**
      $\overrightarrow{p_1^x p_2^x} \cdot \overrightarrow{p_2^x p_3^x} = 0$ **then**
    state ← 4//zigzag が適用可能
  $P^x \leftarrow P^{t2}$
  ルーティング情報を確認する
if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = -1$
    state ← 4//shortcut1 が適用可能.
  else if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = 0$ **and**
      $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_2^x p_3^x} = -1$ **then**
    state ← 4//shortcut2 が適用可能.
  else if $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = 1$ **and**
      $\overrightarrow{p_1^x p_2^x} \cdot \overrightarrow{p_2^x p_3^x} = 0$ **then**
    state ← 4//zigzag が適用可能

d の移動アルゴリズムの詳細は以下の Algorithm 1 に示す.

1: //提案アルゴリズムの適用ができるか
2: **if** $state \neq 4$
3: $P^x \leftarrow P^{t1}$
4: $P^y \leftarrow P^{t2}$
5: **if** $(\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_0^y, p_1^y} = -1) \wedge (\overrightarrow{p_1^x, p_2^x} \cdot \overrightarrow{p_1^y, p_2^y} = 1) \wedge$
6: $(\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_1^x, p_2^x} = 0)$ **then** Apply *Rule 1*
7: $P^x \leftarrow \overline{P^s}$
8: $P^y \leftarrow P^{t1}$
9: **if** $(\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_0^y, p_1^y} = -1) \wedge (\overrightarrow{p_1^x, p_2^x} \cdot \overrightarrow{p_1^y, p_2^y} = 1) \wedge$
10: $(\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_1^x, p_2^x} = 0)$ **then** Apply *Rule 1*
11: $P^x \leftarrow \overline{P^s}$
12: $P^y \leftarrow P^{t2}$
13: **if** $(\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_0^y, p_1^y} = -1) \wedge (\overrightarrow{p_1^x, p_2^x} \cdot \overrightarrow{p_1^y, p_2^y} = 1) \wedge$
14: $(\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_1^x, p_2^x} = 0)$ **then** Apply *Rule 1*
15: $P^x \leftarrow P^{t1}$
16: $P^y \leftarrow P^{t2}$
17: **if** $((\overrightarrow{p_1^x, p_2^x} \cdot \overrightarrow{p_0^y, p_1^y} = 1) \vee (\overrightarrow{p_1^y, p_2^y} \cdot \overrightarrow{p_0^x, p_1^x} = 1))$
18: $\wedge (\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_0^y, p_1^y} = 0)$ **then** Apply *Rule 2*
19: $P^x \leftarrow \overline{P^s}$
20: $P^y \leftarrow P^{t1}$
21: **if** $((\overrightarrow{p_1^x, p_2^x} \cdot \overrightarrow{p_0^y, p_1^y} = 1) \vee (\overrightarrow{p_1^y, p_2^y} \cdot \overrightarrow{p_0^x, p_1^x} = 1))$
22: $\wedge (\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_0^y, p_1^y} = 0)$ **then** Apply *Rule 2*
23: $P^x \leftarrow \overline{P^s}$
24: $P^y \leftarrow P^{t2}$
25: **if** $((\overrightarrow{p_1^x, p_2^x} \cdot \overrightarrow{p_0^y, p_1^y} = 1) \vee (\overrightarrow{p_1^y, p_2^y} \cdot \overrightarrow{p_0^x, p_1^x} = 1))$
26: $\wedge (\overrightarrow{p_0^x, p_1^x} \cdot \overrightarrow{p_0^y, p_1^y} = 0)$ **then** Apply *Rule 2*
27: $P^x \leftarrow P^{t1}$
28: $P^y \leftarrow P^{t2}$
29: **if** $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = 1$ **then** Apply *Rule 3*
30: $P^x \leftarrow \overline{P^s}$
31: $P^y \leftarrow P^{t1}$
32: **if** $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = 1$ **then** Apply *Rule 3*
33: $P^x \leftarrow \overline{P^s}$
34: $P^y \leftarrow P^{t2}$
35: **if** $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = 1$ **then** Apply *Rule 3*

5 アルゴリズムの証明

文献 [12] では, 提案されているアルゴリズム (以下, Zigzag アルゴリズム [12] とする) が経路を切断することなく仮想グリッドネットワークにおける 2 点間の最短経路に収束することが示されている. そこで本章では, 提案アルゴリズムがルータ数の延べ数が最小となる経路に収束することを証明する. そのために, 提案アルゴリズムが以下の 3 つのことを満たすことを証明する.

1. 提案アルゴリズムの更新によって通信経路が切断されない.
2. 提案アルゴリズムが経路をルータの延べ数が最小となる通信経路を形成する.
3. 提案アルゴリズムの更新が経路をルータの延べ数が最小となる通信経路を形成したら, 以降は提案アルゴリズムの更新によって経路は変わらない.

補題 1. 提案アルゴリズムの実行により通信経路が切断されない.

Proof. 提案アルゴリズムの 3 種類の更新ルールはすべて経路を延長するだけなので経路が切断されることはない. 提案アルゴリズムは優先順位によって複数の更新が同時に起こることはなく, また 1 回の更新が実行されたら終了する. 経路が切断される場合は, [12] で提案されているアルゴリズムと本論文の提案アルゴリズムが同時に実行される場合だけである. しかし, 今回のモデルでは本論文の提案アルゴリズムは 3 つの経路 $\overline{P^s}$,P^{t_1},P^{t_2} における p_1^s,$P_1^{t_1}$,$P_1^{t_2}$ で既存研究 [12] の局所変更が適用可能な場合は局所変更を行わない. 従って, 提案アルゴリズムの局所変更に既存研究 [12] の局所変更が影響する場合は実行しないため, 経路の切断は行われない. よって, すべての場合において経路が切断されないため, 提案アルゴリズムの実行によって通信経路が切断されない. □

定義 2. 曲ルータ

$\overline{P^s} = (d = p_l^s, p_{l-1}^s, p_{l-2}^s, ..., p_0^s = s)$ となる経路を d-s 経路, $P^{t1} = (d = p_0^{t1}, p_1^{t1}, p_2^{t1}, ..., p_m^{t1} = t1)$ となる経路を d-t1 経路,$P^{t2} = (d = p_0^{t2}, p_1^{t2}, p_2^{t2}, ..., p_n^{t2} = t2)$ となる経路を d-t2 経路とする. 曲ルータは次の 3 つの条件のいずれかを満たすルータである. ルータが $\overline{P^s}$ に属し, s,d または $\overrightarrow{p_{i-1}^s p_i^s} \cdot \overrightarrow{p_i^s p_{i+1}^s} = 0$,$\overrightarrow{p_{i-1}^s p_i^s} \cdot \overrightarrow{p_i^s p_{i+1}^s} = -1$ となるような p_i^s であるなら曲ルータである. ルータが P^{t1} に属し, d, t1 または $\overrightarrow{p_{i-1}^{t1} p_j^{t1}} \cdot \overrightarrow{p_j^{t1} p_{j+1}^{t1}} = 0$,$\overrightarrow{p_{j-1}^{t1} p_j^{t1}} \cdot \overrightarrow{p_j^{t1} p_{j+1}^{t1}} = -1$ となるような p_j^{t1} であるならば, 曲ルータである. ルータが P^{t2} に属し, d, t2 または $\overrightarrow{p_{k-1}^{t2} p_k^{t2}} \cdot \overrightarrow{p_k^{t2} p_{k+1}^{t2}} = 0$,$\overrightarrow{p_{k-1}^{t2} p_k^{t2}} \cdot \overrightarrow{p_k^{t2} p_{k+1}^{t2}} = -1$ となるような p_k^{t2} であるならば曲ルータである. s,d,t1,t2 を曲ルータに含めるため, 定義より経路 P^s,P^{t1},P^{t2} において経路長が 1 以上なら曲ルータは 2 個以上存在する. 経路 P^s,P^{t1},P^{t2} において経路長が 0 であるならば, 曲ルータは 1 個存在する. 経路 P^s, P^{t1}, P^{t2} の中の曲ルーターの数をそれぞれ,$x+1, y+1, z+1$ 個とすると,$q_i^s (0 \leq i \leq x)$ は経路 P^s の中で i 番目の曲ルータ ,$q_j^{t1} (0 \leq j \leq y)$ は経路 P^{t1} の中で j 番目の曲ルータ ,$q_k^{t2} (0 \leq k \leq z)$ は経路 P^{t2} の中で k 番目の曲ルータである.

定義 3. 直線部分

$\overline{P^s} = (d = p_l^s, p_{l-1}^s, p_{l-2}^s, ..., p_0^s = s)$ となる経路を d-s 経路, $P^{t1} = (d = p_0^{t1}, p_1^{t1}, p_2^{t1}, ..., p_m^{t1} = t1)$ となる経路を d-t1 経路,$P^{t2} = (d = p_0^{t2}, p_1^{t2}, p_2^{t2}, ..., p_n^{t2} = t2)$ となる経路を d-t2 経路とする. $s_u^s (1 \leq u \leq x)$, $s_v^{t1} (1 \leq v \leq y)$, $s_w^{t2} (0 \leq w \leq z)$ を直線部分 $(q_{u-1}^s q_u^s)$, $(q_{v-1}^{t1} q_v^{t1})$, $(q_{w-1}^{t2} q_w^{t2})$ とする. 経路 P^s, P^{t1}, P^{t2} は直線部分の列 $(s_1^s, s_2^s, ..., s_x^s), (s_1^{t1}, s_2^{t1}, ..., s_y^{t1}), (s_1^{t2}, s_2^{t2}, ..., s_z^{t2})$ で表すことができ, これを経路 P^s, P^{t1}, P^{t2} の直線部分列とする. 経路 P^s, P^{t1}, P^{t2} の中の曲ルーターの数をそれぞれ,x+1,y+1,z+1 個とすると,$\overrightarrow{s_i^s}, \overrightarrow{s_j^{t1}}, \overrightarrow{s_k^{t2}}$ はルータのベクトル $\overrightarrow{q_{i-1}^s q_i^s} (0 \leq i \leq x)$, $\overrightarrow{q_{j-1}^{t1} q_j^{t1}} (0 \leq j \leq y)$, $\overrightarrow{q_{k-1}^{t2} q_k^{t2}} (0 \leq k \leq z)$ と表すことができる.

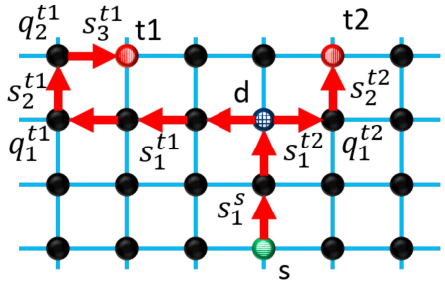


図 25: 直線部分

定義 4. ポテンシャル関数 $f(\overline{P^s}, P^{t1}, P^{t2})$

ポテンシャル関数 f は $f(\overline{P^s}, P^{t1}, P^{t2}) = (|\overline{P^s}| + |P^{t1}| + |P^{t2}|, |s_1^s| + |s_1^{t1}| + |s_1^{t2}|, |s_2^s| + |s_2^{t1}| + |s_2^{t2}|, ...)$ ($|P^s|, |P^{t1}|, |P^{t2}|$ はそれぞれ経路 P^s, P^{t1}, P^{t2} の全体の長さであり, $|s_i^s|(1 \leq i \leq x)$, $|s_j^{t1}|(1 \leq j \leq y)$, $|s_k^{t2}|(1 \leq k \leq z)$ を直線部分 $s_i^s, s_j^{t1}, s_k^{t2}$ の長さとする.$|P^s| = \sum_{i=1}^{x} |s_i^s|, |P^{t1}| = \sum_{j=1}^{y} |s_j^{t1}|, |P^{t2}| = \sum_{k=1}^{z} |s_k^{t2}|$ が成り立つ.) ポテンシャル関数の値は辞書順比較による全順序である.

定義 5. 中央点 Cp

s の座標を原点 (0, 0) とした場合, 一般的な直交座標系における t_1 と t_2 の座標が計算でき, それらを (t_{1x}, t_{1y}), (t_{1x}, t_{1y}) とする. 中央点 Cp の座標は ($Cp_x = M(0, t_{1x}, t_{2x})$, $Cp_y = M(0, t_{1y}, t_{2y})$)(M(a, b, c) は整数 a, b, c の中央値を返す関数) となる.

定義 6. ルータの延べ数最小経路

ルータの延べ数最小経路は次の条件を満たす. 1 つ目は, ディストリビュータ d は中央点 Cp の座標に存在する. 2 つ目は, 経路 P^s, P^{t1}, P^{t2} の直線部分列 $(s_1^s, s_2^s, ..., s_x^s)$,$(s_1^{t1}, s_2^{t1}, ..., s_y^{t1})$,$(s_1^{t2}, s_2^{t2}, ..., s_z^{t2})$ は, $(0 \leq i \leq x-1)$, $(0 \leq j \leq y-1)$, $(0 \leq k \leq z-1)$ において, $|P^s|, |P^{t1}|, |P^{t2}| \geq 0$ なら $|s_i^s| = |s_j^{t1}| = |s_k^{t2}| = 1$ を満たす. $|P^{t2}| = 0$ なら $|s_i^s| = |s_j^{t1}| = 1$ かつ $|s_k^{t2}| = 0$ を満たす. $|P^{t1}| = 0$ なら $|s_i^s| = |s_k^{t2}| = 1$ かつ $|s_j^{t1}| = 0$ を満たす. $|P^s| = 0$ なら $|s_j^{t1}| = |s_k^{t2}| = 1$ かつ $|s_i^s| = 0$ を満たす. この 2 つの条件を満たす経路をルータの延べ数最小経路と呼ぶ. 通常, ディストリビュータからソース, ターゲットまでの経路がそれぞれ 1 つの直線部分で構成されていなければ, 図 26, 27 のように 2 つ存在する. 1 つの直線部分で構成されている場合, ルータの延べ数最小経路は図 28 に示すようにただ 1 つ存在する. 図 29 のようにディストリビュータがソース, ターゲットと一致する場所に移動するときは一致したルータまでの経路長は 0 となる. ディストリビュータがソース, ターゲットと一致する場所に移動し, ディストリビュータからソース, ターゲットまでの経路がそれぞれ 1 つの直線部分で構成されていなければ, 図 30, 31, 32, 33 のように 4 つ存在する.

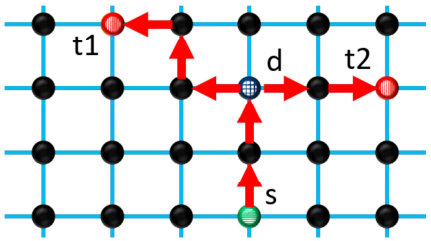


図 26: ルータの延べ数最小経路 1

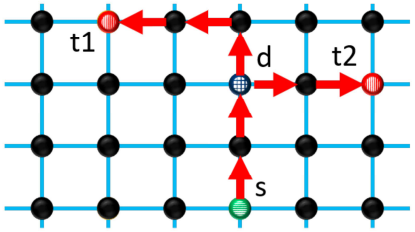


図 27: ルータの延べ数最小経路 2

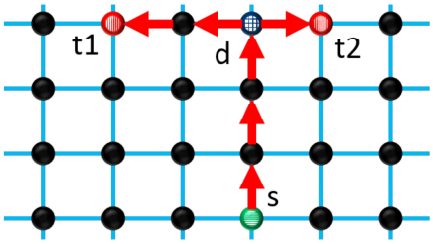


図 28: ルータの延べ数最小経路 3

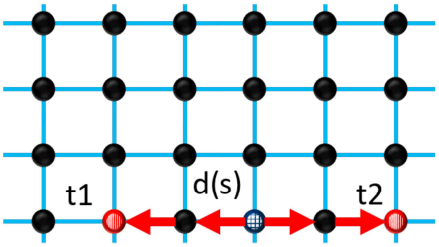


図 29: ルータの延べ数最小経路 4

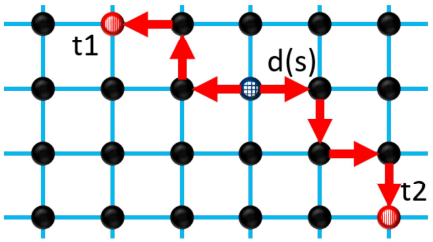


図 30: ルータの延べ数最小経路 5

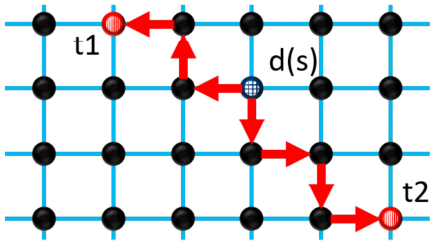


図 31: ルータの延べ数最小経路 6

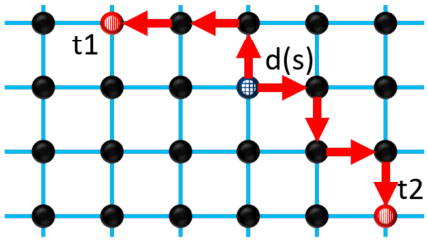


図 32: ルータの延べ数最小経路 7

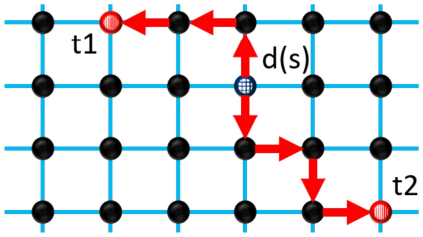


図 33: ルータの延べ数最小経路 8

補題 2. ディストリビュータ d の位置が中央点 Cp であり, *Zigzag* アルゴリズム *[12]* が収束しているのであれば, ルータの延べ数最小経路を形成する.

Proof. d の座標が中央点 Cp の座標 $(M(0, t_{1x}, t_{2x}, M(0, t_{1y}, t_{2y})$ であるとルータの延べ数最小経路となることを説明する. $0 \leq t_{1x} \leq t_{2x}$ として考え, d=($d'_x \neq M(0, t_{1x}, t_{2x}, M(0, t_{1y}, t_{2y})$ でルータの延べ数最小経路を形成すると仮定する. このとき $M(0, t_{1x}, t_{2x} = t_{1x}$ となる. $d'_x < 0$ とすると d'_x の値を 1 増加させることで $0, t_{1x}, t_{2x}$ との差がそれぞれ 1 減少し, 全体の経路長が減少する. よって仮定と矛盾する. $0 \leq d'_x < t_{1x}$ とすると d'_x の値を 1 増加させることで 0 との差は 1 増加するが, t_{1x}, t_{2x} との差がそれぞれ 1 減少し, 全体の経路長が 1 減少する. よって仮定と矛盾する. $t_{1x} < d'_x \leq t_{2x}$ とすると d'_x の値を 1 減少させることで t_{2x} との差は 1 増加するが, $0, t_{1x}$ との差がそれぞれ 1 減少し, 全体の経路長が 1 減少する. よって仮定と矛盾する. $t_{2x} < d'_x$ とすると d'_x の値を 1 減少させることで $0, t_{1x}, t_{2x}$ との差がそれぞれ 1 減少し, 全体の経路長が減少する. よって仮定と矛盾する. 従って, $d_x = M(0, t_{1x}, t_{2x} = t_{1x}$ のとき全体の経路長が最小となる. これは, $0, t_{1x}, t_{2x}$ の大小比較を変更した場合や $d'_y \neq M(0, t_{1y}, t_{2y}$ とした場合でも同様に示すことが出来る. 従って d の座標が中央点 Cp の座標 $(M(0, t_{1x}, t_{2x}, M(0, t_{1y}, t_{2y})$ であるときルータの延べ数最小経路となる. また, ルータの延べ数最小経路の 2 つ目の条件は Zigzag アルゴリズム [12] が収束した時, 保証される. 従って, ディストリビュータ d の位置が中央点と一致するならば, Zigzag アルゴリズム [12] が収束すればルータの延べ数最小経路を形成する. □

補題 3. 提案アルゴリズムは少なくとも *2* ラウンドあたりの経路変更によってポテンシャル関数の値が減少する.

Proof. 1 ラウンド目における経路変更前の経路, ポテンシャル関数をそれぞれ, $\overline{Ps}, P^{t1}, P^{t2}$, $f(\overline{Ps}, P^{t1}, P^{t2})$ とする. また,2 ラウンド目における経路変更後の経路, ポテンシャル関数をそれぞれ, $\overline{Ps}$', P^{t1}', P^{t2}', $f(\overline{Ps}, P^{t1}, P^{t2})$' とする. ディストリビュータが移動するラウンドを考える. まず, Rule1 について考える. $P^x = P^{t1}, P^y = P^{t2}$ とする. $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = -1, \overrightarrow{p_1^x p_2^x} \cdot \overrightarrow{p_1^y p_2^y} = 1, \overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = 0$ より $\overrightarrow{p_1^y p_2^y} \cdot \overrightarrow{p_1^y p_2^y} = 0$. よって $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_2^x} = \overrightarrow{p_0^x p_1^x} \cdot (-\overrightarrow{qp_0^x}) = 0, \overrightarrow{p_0^y p_1^y} \cdot \overrightarrow{p_1^y p_2^y} = \overrightarrow{p_0^y p_1^y} \cdot (-\overrightarrow{qp_0^y}) = 0$ となる. よって, 1 ラウンド目において, Rule1 を実行すると $P^x = P^{t1}, P^y = P^{t2}$ は曲ルータを生成し, $|\overline{Ps}|, |P^{t1}|, |P^{t2}|$ はそれぞれ 1 増加する. しかし, 2 ラウンド目で P^{t1}, P^{t2} において shortcut2 が 1 回ずつ実行され $|P^{t1}|, |P^{t2}|$ が 2 ずつ減少する. よって, ポテンシャル関数の最初の要素は, .$|\overline{Ps}|$' + $|P^{t1}|$' + $|P^{t2}|$' = ($|\overline{Ps}|$ + 1) + ($|P^{t1}|$ + 1 - 2) + ($|P^{t2}|$ + 1 -2) = ($|\overline{Ps}|$ + $|P^{t1}|$ + $|P^{t2}|$) -1 となり, ポテンシャル関数が減少する. これは, $P^x = P^s, P^y = P^{t1}$, $P^x = P^s, P^y = P^{t2}$ のときでも同様である. よって, Rule1 を実行するとポテンシャル関数の値が減少する.
Rule2 の場合を考える. $P^x = P^{t1}, P^y = P^{t2}$ とする. 1 ラウンド目で $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = 0$ かつ $\overrightarrow{p_1^x p_2^x} \cdot \overrightarrow{p_0^y p_1^y} = 1$ の場合は, $|\overline{Ps}|, |P^{t1}|, |P^{t2}|$ がそれぞれ 1 増加する. 2 ラウンド目において, P^{t1} では shortcut2 が実行可能で $|P^{t1}|$ が 2 減少し, P^{t2} では shortcut1 が実行可能で $|P^{t2}|$ が 2 減少する. 従ってポテンシャル関数の最初の要素は $|\overline{Ps}|$' + $|P^{t1}|$' + $|P^{t2}|$' = ($|\overline{Ps}|$ + 1) + ($|P^{t1}|$ + 1 - 2) + ($|P^{t2}|$ + 1 -2) = ($|\overline{Ps}|$ + $|P^{t1}|$ + $|P^{t2}|$) -1 となり, ポテンシャル関数が減少する.
1 ラウンド目で $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_0^y p_1^y} = 0$ かつ $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^y p_2^y} = 1$ の場合は, $|\overline{Ps}|, |P^{t1}|, |P^{t2}|$ がそれぞれ 1 増加する. 2 ラウンド目において P^{t1} では shortcut1 が実行可能で $|P^{t1}|$ が 2 減少し, P^{t2} では shortcut2 が実行可能で $|P^{t2}|$ が 2 減少する. 従ってポテンシャル関数の最初の要素は $|\overline{Ps}|$' + $|P^{t1}|$' + $|P^{t2}|$' = ($|\overline{Ps}|$ + 1) + ($|P^{t1}|$ + 1 - 2) + ($|P^{t2}|$ + 1 -2) = ($|\overline{Ps}|$ + $|P^{t1}|$ + $|P^{t2}|$) -1 となり, ポテンシャル関数が減少する.
これは, $P^x = P^s, P^y = P^{t1}$, $P^x = P^s, P^y = P^{t2}$ のときでも同様である. よって, Rule2 を実行するとポテンシャル関数の値が減少する.
Rule3 の場合を考える. $P^x = P^{t1}, P^y = P^{t2}$ とする. 1 ラウンド目において Rule3 を実行すると $|\overline{Ps}|, |P^{t1}|, |P^{t2}|$ はそれぞれ 1 増加する. 2 ラウンド目において, 経路変更された経路 P^x, P^y に対し $\overrightarrow{p_0^x p_1^x} \cdot \overrightarrow{p_1^x p_0^x} = -1, \overrightarrow{p_0^y p_1^y} \cdot \overrightarrow{p_1^y p_0^y} =$

-1 より shortcut1 がそれぞれ 1 回ずつ実行可能で $|P^{t1}|$, $|P^{t2}|$ がそれぞれ 2 減少する．従ってポテンシャル関数の最初の要素は $|\overline{Ps}|$' + $|P^{t1}|$' + $|P^{t2}|$' = ($|\overline{Ps}|$ + 1) + ($|P^{t1}|$ + 1 - 2) + ($|P^{t2}|$ + 1 -2) = ($|\overline{Ps}|$ + $|P^{t1}|$ + $|P^{t2}|$) -1 となり, ポテンシャル関数が減少する．
これは, $P^x = P^s, P^y = P^{t1}$, $P^x = P^s, P^y = P^{t2}$ のときでも同様である．よって, Rule3 を実行するとポテンシャル関数の値が減少する．
ディストリビュータが移動しないラウンドを考える. ディストリビュータが移動しない場合はディストリビュータの視野内で Zigzag アルゴリズムの局所変更を発見する場合, Zigzag アルゴリズムが収束していない場合である．
一番先頭で適用される更新が, shortcut1 の場合, 長さ自体が短くなるため, ポテンシャル関数の初めの要素が減少する．よって, 更新によりポテンシャル関数が減少する．
一番先頭で適用される更新が, shortcut2 の場合, 長さ自体が短くなるため, ポテンシャル関数の初めの要素が減少する．よって, 更新によりポテンシャル関数が減少する．
一番先頭で適用される更新が, zigzag の場合, $\overline{P^s} = (d = p_l^s, p_{l-1}^s, p_{l-2}^s, ..., p_0^s = s)$ となる経路を d-s 経路で更新が行われるとする.p_a^s を一番先頭の zigzag が適用されるルータの引数とする．$(s_1^s, s_2^s, ..., s_x^s)$ を P^s の直線部分列とし, s_b^s を p_a^s が含む直線部分とする, $\overline{P^s}$'=(d=p_l^s', p_{l-1}^s', p_{l-2}^s', ...,p_0^s'=s) とし, (s_1^s', s_2^s', ...,s_x^s') を P^s' の直線部分列とする．
$(0 \leq i \leq b-1)$ のとき, $|s_i^s$'$|$'=$|s_i^s|$'
$(i = b)$ のとき, $|s_i^s$'$|$'=$|s_i^s|$' - 1 となる．よって, 更新によりポテンシャル関数の値が減少する．よって少なくとも 2 ラウンドあたりの経路変更によってポテンシャル関数が必ず減少する． □

補題 4. ルータの延べ数最小経路ならば, ポテンシャル関数の値が最小 *(*あるいは *2* 番目 または *3* 番目 または *4* 番目に小さい値*)* になる．

Proof. 定義 4, 定義 6, 補題 2 より, ルータの延べ数最小経路の時, ポテンシャル関数の値も最小になる． □

補題 5. *Zigzag* アルゴリズム *[12]* が収束しルータの延べ数最小経路でなければ, 必ず有限な実行回数以内に提案アルゴリズムが実行される．

Proof. 現在のディストリビュータ d の座標を (d'_x, d'_y) とする. 現在のディストリビュータの位置からルータの延べ数最小経路となるときのディストリビュータへの座標 (d_x, d_y) に向かう方向を x 成分と y 成分に分けて $\overrightarrow{d'_x d_x}, \overrightarrow{d'_y d_y}$ と表す．また, s の座標を原点 $(s_x = 0, s_y = 0)$ として t_1, t_2 の座標をそれぞれ (t_{1x}, t_{1y}), (t_{1x}, t_{1y}) とする．現在のディストリビュータの位置から s, t_1, t_2 へ向かう方向をそれぞれ x 成分と y 成分に分けて, $\overrightarrow{d'_x s_x}, \overrightarrow{d'_y s_y}$, $\overrightarrow{d'_x t_{1x}}, \overrightarrow{d'_y t_{1y}}$, $\overrightarrow{d'_x t_{2x}}, \overrightarrow{d'_y t_{2y}}$ と表す．$s_x < d_x = t_{1x} < t_{2x}$ として考える.$d'_x < t_{1x}$ のとき, $\overrightarrow{d'_x t_{1x}} \cdot \overrightarrow{d'_y t_{2x}} > 0$ が成り立つ．Zigzag アルゴリズム [12] が収束したとき, 経路 P^{t1}, P^{t2} の 2 ホップの直線部分は次の 4 つのパターンのみである．$\overrightarrow{s_1^{t1}} \cdot \overrightarrow{s_1^{t2}} > 1$, $\overrightarrow{s_1^{t1}} \cdot \overrightarrow{s_2^{t2}} > 1$, $\overrightarrow{s_2^{t1}} \cdot \overrightarrow{s_1^{t2}} > 1$, $\overrightarrow{s_2^{t1}} \cdot \overrightarrow{s_2^{t2}} > 1$ の場合である．$\overrightarrow{s_1^{t1}} \cdot \overrightarrow{s_1^{t2}} > 1$ のとき Rule3 が実行可能である．$\overrightarrow{s_1^{t1}} \cdot \overrightarrow{s_2^{t2}} > 1$ または $\overrightarrow{s_2^{t1}} \cdot \overrightarrow{s_1^{t2}} > 1$ のとき Rule2 が実行可能である．$\overrightarrow{s_2^{t1}} \cdot \overrightarrow{s_2^{t2}} > 1$ のとき Rule1 が実行可能である．従って, 提案アルゴリズムの実行が行われる．$d'_x > t_{1x}$ のとき, $\overrightarrow{d'_x s_x} \cdot \overrightarrow{d'_y t_{1x}} > 0$ が成り立ち同様に考えることが出来る．s_x, t_{1x}, t_{2x} の大小比較を入れ替えても同様に示すことが出来る. 従って, x 成分の時ルータの延べ数最小経路でなければ提案アルゴリズムの実行が実行可能である．y 成分の時も同様にルータの延べ数最小経路でなければ提案アルゴリズムの実行が実行可能である．従って, ルータの延べ数最小経路でなければ, 必ず有限な実行回数以内に提案アルゴリズムが実行される． □

補題 6. ルータの延べ数最小経路が形成されたら更新は行われない．

Proof. Rule1, Rule2, Rule3 が適用できるルータの延べ数最小経路を考える. 補題 3 より, Rule1, Rule2, Rule3 のいずれかを実行しても 2 ラウンド単位で考えるとポテンシャル関数は減少する. しかし, 補題 4 よりルータの延べ数最小経路ならばポテンシャル関数は最小となることと矛盾する. これは Rule1, Rule2, Rule3 が適用できるルータの延べ数最小経路が存在すると仮定したためであるよって, Rule1, Rule2, Rule3 が適用できるルータの延べ数最小経路がしないため, 更新も行われない． □

定理 1. 提案アルゴリズムは通信経路を切断することなくルータの延べ数最小経路に収束させる．

Proof. 補題 1 は提案アルゴリズムの更新が通信経路が切断されないことを証明している．補題 3, 補題 4 は提案アルゴリズムの更新が通信経路をルータの延べ数が最小となる通信経路を形成することを証明している. 補題 6 は通信経路がルータの延べ数最小経路に形成したら以降は提案アルゴリズムの更新によって経路は変わらないことを証明している．よって, 提案アルゴリズムは通信経路を切断することなくルータの延べ数最小経路に収束する． □

6 まとめ

本論文では仮想グリッドネットワーク上でターゲットが 2 台ある場合のルータの延べ数を最小にする経路を生成するアルゴリズムを提案した. 提案アルゴリズムでは, ルータの延べ数を減少させることでルータの使用数を減少させることによるエネルギー効率の向上などのメリットがある．

今後の課題として，提案アルゴリズムの計算時間の解析, 別のアルゴリズムの条件では, ターゲットが 3 台以上ある場合などが挙げられる．

参考文献

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

[2] S. Giordano, "Mobile Ad-Hoc Networks" , Wiley, 2000.

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

[4] Y. Xiao, H. Chen, and F. H. Li, "Handbook on Sensor Networks" , World Scientific, 2010.

[5] Waltenegus Dargie and Christian Poellabauer , "Fundamentals of Wireless Sensor Networks:Theory and Practice" , page 187, John Wiley & Sons,Ltd, 2010.

[6] B. Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for wireless net- works," in Proceedings of the 6th Annual International Conference on Mobile Computing and Networking(MobiCom' 00) , 2000, pp.243-254.

[7] C. Perkins and E. Royer , "Ad hoc on demand distance vector routing," in Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications(WMCSA' 99) , 1999, pp.90-100.

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

[9] C. E. Perkins and P. Bhagwat , "Highly dynamic Destination-Sequenced Distance-Vector routing(DSDV) fo mobile computers," in Proceedings of Conference on Communications Architectures, Protocols and Applications(SIGCOMM' 94) , 1994, pp.234-244.

[10] D. Braginsky and D. Estrin , "Rumor routing algorithm for sensor networks," in Proceed- ings of the 1st ACM International Workshop on Wireless Sensor Networks and Applica- tions(WSNA' 02) , 2002, pp.22-31.

[11] T. H. Clausen, G. Hansen, L. Christensen, and G. Behrmann , "The optimized link state routing protocol evaluation through experiments and simulation," in Proceedings of IEEE Symposium on Wireless Personal Mobile Communications , 2001.

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

[13] 宮川歩，金鎔煥，片山喜章"仮想グリッドネットワークにおける経路最適化分散アルゴリズムの改良" 電子情報通信学会技術研究報告= IEICE technical report : 信学技報 116(116) ,pp. 49-56, 2016.

[14] 風岡弘樹，金鎔煥，片山喜章,“ 仮想グリッドネットワークにおける共通座標系を有しない経路最適化分散アルゴリズム ”電子情報通信学会大会講演論文集,D-1-3,2018.

[15] 金鎔煥，増澤利光，"仮想グリッドネットワークにおけるスタイナー木生成のための自律分散アルゴリズムに関する研究"，情報処理学会第 78 回全国大会， 3A-02， 2016.

[16] Yonghwan Kim and Yoshiaki Katayama, "A Self-optimizing Routing Algorithm using Lo- cal Information in a 3-dimensional Virtual Grid Network with Theoretical and Practical Analysis", International Journal of Networking and Computing, Vol. 7, No. 2, pp. 349-371, 2017.

[17] Yonghwan Kim and Yoshiaki Katayama, "A Self-optimizing Routing Algorithm in a 3- dimensional Virtual Grid Network", Proceedings of the Fourth International Symposium on Computer and Networking (CANDAR 2016), 2016.

長手数詰将棋構築の試み

*都勇志　名古屋大学　名古屋市千種区不老町　miyako.yuji@i.mbox.nagoya-u.ac.jp
木谷裕紀　名古屋大学　名古屋市千種区不老町　kiya.hironori@gmail.com
小野廣隆　名古屋大学　名古屋市千種区不老町　ono@nagoya-u.jp

1　はじめに

2015 年に勝利宣言が出された将棋 AI や 2017 年 DeepMind 社の作成した AlphaGo が世界トップクラスの棋士に勝利するなど二人零和有限確定完全情報ゲームの研究が近年進んでいる [1]. このような「強い」AI を作成する研究の一方で詰将棋や詰碁に関する研究が進められている. 例えば, 詰将棋作品の出来を定量的に評価する研究 [3], 詰将棋の自動創作に関する研究などがある [4]. また, 詰将棋を解くプログラムの研究 [2][5][6][7] のなかには非常に手数の長い詰将棋作品を解くためのアルゴリズムの研究もある [8]. これらの研究に対し, 本研究では詰将棋の作題を扱う.

そもそも詰将棋とは将棋が元となったパズルゲームであり, 将棋が今の形になったのと同時期に生まれたとされる. 詰将棋は将棋のルールに追加で固有のルールがある. 第一に, 詰将棋の問題を解く側のルールがあり, たとえば攻め手は毎回王手でなければならない, などがある. 第二に, 詰将棋の定義, 形式に関わる問題を製作する側のルールがある. 実は後者には公式のルールが存在しない, しかしおおよその決まりはあり, 多くの詰将棋作家が禁止としている要素, あるいは禁止とまではいかないが望ましくない要素が詰将棋の本や雑誌などで慣例の形で採用されている. このような慣例には, 攻め方が最短手数で玉を詰ませたとき, 攻め方の持ち駒があまらない, 攻め方の持ち駒を余らせずに複数の最短手順が存在してはいけない（余詰めの非存在性）などがある.

詰将棋にはさまざまな種類が存在するが, 詰将棋のなかには同じ手を繰り返すことで盤面全体にひとつだけ変化が現れ詰みに向かうことで特別長い手数になるものがある. 詰みまでの手順を式で表すと, (繰り返し部に入るまでの王手)+(連続して王手する部分）× (一組の手順でひとつだけ変化する部分)+(一般的な詰将棋) のような構成となっている. この性質を持つものは一般に長手数詰将棋といわれ, そのはじまりは 1775 年に伊藤看寿が作成した「寿」(611 手詰) である. ただし, 制作上の難度のためか長手数詰将棋は他の詰将棋には許されていない任意性 (余詰め, 手順前後）が慣例的に認められている. 2019 年 2 月現在, 最長手数の詰将棋は「ミクロコスモス」(原作 1986 年. 橋本孝治作）であり, これが確かに 1525 手で詰むことを脊尾らは示している [9] が, ミクロコスモスにもその任意性が含まれている. 現時点で知られている厳密な意味での最長の詰将棋は 2010 年摩利支天による「STARSHIP TROOPER」(625 手詰め) である.

これに対し本研究では, 625 手詰めよりも長い手数の厳密な意味での詰将棋の構築を試みる. 既存の余詰めのある長手数詰将棋「桃花源」(1990, 添川公司, 767 手詰め) を元に 767 手詰めの詰将棋を提案する. 本論文ではその設計の要点を説明する. これらまで知られていた最長詰将棋が 625 手詰めであったため, これが現時点での厳密な意味の最長詰将棋である.

本論文の以下の構成は以下の通りである. 2 節は基本的な将棋のルールと詰将棋のルールについて解説する. 3 節では長手数詰将棋とはなにか, これまで発表されてきた長手数詰将棋の紹介, 長手数となる仕組み, 抵触している禁止事項について説明する. 4 節にて本研究の成果である, 厳密な意味の最長詰将棋の構築について述べる.

2　準備

本節では本研究で採用する詰将棋のルールを説明する. なお, 詰将棋には公式のルールが存在しない. このためここでは一般に広く受け入れられていると思われるものを採用している.

詰将棋は相手の玉を詰ませる「攻め方」と「玉方」に分かれて思考が行われる. 攻め方はできる限り早く玉を詰ますことを目的とし, 玉方はできる限り詰まないように, 詰むとしてもできる限り長い手数になることを目的とする. 同手数で詰む分岐は攻め方の持ち駒が最終的になくなるように対応する.

詰将棋の基本的なルールや反則は将棋のルール通りであり, 詰将棋の将棋とは異なる点は, 攻め方は必ず王手をかけ, 玉方は必ず王手ではない状態にしなければならないこと (これができなくなったとき玉方は「詰み」となりそれまでの両プレイヤの合計手数が詰将棋の手数となる). 玉方は盤上と攻め方の持駒以外すべての駒

（ただし玉は除く）を持駒として使用できることと, 一方向に複数マス移動可能な駒で王手をした際に他の駒を置くことで王手でない状態にすることを合駒というが手数が 2 手伸びるがその後の手数と手順が変化しない合駒はできないことがあげられる.

以上が詰将棋を解く際のルールであるが, これに加えて詰将棋を作る際に考慮しなければならない点がいくつかある. 原則禁止事項とされているのは,

(1) 攻め方の持ち駒の余る詰将棋,

(2) 攻め方の持ち駒の余らない同手数の, 玉方の玉が詰んだ時の盤面を複数通り攻め方の選択によって決定できる詰将棋 (余詰め),

である. これらの要素を含むと基本的には不完全作品として扱われる. また禁止はされていないものの, 含まれていれば審美的観点から望ましくないとされる事項もいくつか存在する. (2) の中で禁止はされていないが望ましくないとされるものは, 攻め方が盤上の駒を移動させ王手しその駒が成れる時成るか否かにかかわらずそれ以降同じ進行で詰む場合と, 最終手の攻め方の手の変更による, 攻め方の持ち駒の余らない同手数の詰みが複数存在する場合がある. (1) の中で同様のものは, 玉方の $n-1$ 手目の対応次第で持ち駒の余らない n 手詰め, もしくは攻め方の持ち駒の余る $n+2$ 手詰めになる場合である. そのほかには玉方の手の変更による持ち駒の余らない同手数の詰みがある場合がある. ただし例外的に不問となる事項として (2) で, 攻め方が一方向に複数マス移動可能な駒で王手した時その遠近にかかわらずそれ以降同じ進行で詰む場合と, 玉方の手の変更により攻め方の持ち駒の余らない同手数の詰み手順で両プレイヤの手を数えて 3 手以内に合流する場合が挙げられる.

3 長手数詰将棋

これまで膨大な数の詰将棋が考案されているがその手数はさまざまである. その中には非常に長い手数のものがあり, これまで知られている作品の中で最長の「ミクロコスモス」は 1525 手にもなる. ミクロコスモスを始めとする長手数の詰将棋は, ひと組の手順によって盤面全体でひとつだけ変化が起きる仕組みが繰り返されることで手数が伸びている. 表 1 にこれまで発表されている代表的な長手数詰将棋を挙げる. 長手数詰将棋を式で表せば, (繰り返し部に入るまでの王手)+(連続して王手する部分)× (一組の手順でひとつだけ変化する部分)+(一般的な詰将棋) となる. しかし, 慣例的に長手数詰将棋においては通常は禁止されている事項がしばしば許される. 実際, 表 1 で挙げた詰将棋もほとんどが何らかの禁止事項 (2 節 (2)) に抵触している. これらを対応する形でまとめたのが表 2 である.

表 2 において駒の前についている二桁の数はそれぞれアラビア数字が攻め方側の右から数えて, 漢数字が上から数えて何番目の行, 列にあるかをあらわしている. 「同」は一手前に移動してきた駒と同じマスに移動し, 取ったことを意味する.

表 1: 長手数順 長手数詰将棋

順位	題名	作者	手数
1	ミクロコスモス	橋本孝治	1525
2	新桃花源	添川公司	1205
3	アトランティス	中村, 芝	951
4	メタ新世界	山本昭一	941
5	新扇詰	奥薗幸雄	873
6	涛龍	井上徹也	849
7	（無題）	田島秀男	833
8	内燃機関	添川公司	805
9	イオニゼーション	橋本孝治	789
10	桃花源	添川公司	767
11	明日香	添川公司	703
12	ゴゴノソラ	今村修	651
13	阿吽	添川公司	647
14	無銭旅行	佐々木恭閑	645
15	アルカナ	橋本孝治	639
16	小涌谷	添川公司	635
17	STARSHIP TROOPER	摩利支天	625

4 余詰め無し最長詰将棋の作成

表にある通り厳密な意味での最長手数詰将棋は 625 手であり, 本研究では 767 手の桃花源を元に余詰めの無い現段階の最長手数の詰将棋を作成する. 図 1 がこの桃花源の初期盤面である.

余詰めの消去を考察するにあたって, 改変を試みる必要があるが, 改変の幅が大きくなればなるほど難易度が高くなる, ここで, 攻め方の分岐が (連続して王手する部分) × (一組の手順でひとつだけ変化する部分) によって発生する場合, 改変の幅は大きくなることがわかった. 「長手数順 禁止事項リスト」より, (繰り返し部に入るまでの王手), (一般的な詰将棋) に分岐が存在する作品のうち, 本研究では, 詰将棋のルールに沿った最長手数の作品を 767 手の「桃花源」を元に禁則を含まない長手数詰将棋を構築する. 桃花源は表の通り 757 手目で禁止事項 (2) に触れている. 8 三のと金は 757 手

表 2: 長手数順 禁止事項リスト

順位	禁止事項
1	1499 手目 8 二龍同金 ⇔ 1493・95・97・1501 手目
2	1153 手目の 8 六と 7 七玉⇔ 7 七と同玉
3	269 手目など 5 一龍のところすぐ同香
4	11 手目 6 一飛 6 二香合 ⇔ 13 手目 5 二銀同玉 5 一角成 6 三玉
5	2 一歩をとるのが往路 (75 手目) でも 復路 (105 手目) でもよい
6	248 手目 8 八銀⇔ 346 手目 7 七銀
7	盤上の玉方の駒を取る順は 5 二成桂が最後, 7 一とがその前, それ以外は順不同
8	133 手目 7 七飛⇔ 9 七飛
9	763 手目同金の変化で 769 手目 4 四金⇔ 3 五金
10	757 手目 5 四銀⇔ 5 六銀
11	1)63 手目 1 三香 1 金合 ⇔ 65 手目 6 六馬 5 五と および以下同様の形になる箇所
	2)111 手目 1 四香 1 三金合 ⇔ 113 手目 6 七馬 4 五と寄 および以下同様の形になる箇所
	3) 267 手目 6 二馬 4 四と と 269 手目 8 七龍 5 七銀成
	4) 619 手目 1 九香 1 八香 と 621 手目 5 七龍 同金
12	391 手目 4 三銀からの 6 手 ⇔ 397 手目 5 三龍からの 24 手
13	53 手目 7 七龍⇔ 6 六龍 55 手目 6 八龍 ⇔ 5 七龍 65 手目など 6 八龍⇔ 5 七龍
14	549 手目 7 二龍 8 四玉 8 四香 8 五玉 ⇔ 8 三香 7 四玉 7 二龍 8 四玉
15	171 手目 4 八角 2 九玉 6 五馬同と ⇔ 6 五馬同と 4 八角 2 九玉 225 手目 297 手目も同様
	635 手目 9 九香最終手 9 九金 ⇔ 9 九金 9 九香
16	22 手目 8 八歩⇔ 82 手目 7 七歩
17	なし (554 手目 6 四玉と 6 六玉の 同手数詰のそれぞれ駒の余らない変化)

目まで詰みに役立つわけでもなければ妨害もしない. また, 757 手目以降の影響は, 攻め方から順に, 5 四銀 3 四玉 (もしくは 4 四玉. ここで 4 六や 3 六に玉が移動すると 3 七金で詰み)4 五銀となれば同玉とする以外に最長手順になる逃げ方が存在しない一方で 5 四銀ではなく 5 六銀としたときに 5 四玉とされると 4 五銀に対して同玉でなければ早く詰むようになる働きを盤面左側でしている. 具体的には 6 三玉なら 7 三と 6 四玉 7 五と, 6 四玉なら 7 五と 6 三玉 7 三と, となる. この働きがあるために 5 四銀と 5 六銀の選択肢が生まれてしまっている. そこで 757 手目まで影響が無い 8 三とを作品全体を通してまったく影響しない位置 (1 一) に移すことで 5 六銀では詰むまでの手数が伸びてしまうようになり, 攻め方の選択肢がなくなる. 以上により我々の作成した初期盤面が図 2 である. またこの初期盤面から始めて, 攻め方の分岐がなくなった場面を表したのが図 3 である.

以上により, 得られた詰将棋は禁則を含まない 767 手詰めの長手数詰将棋となる. これは現時点で最長の禁則を含まない詰将棋である.

5 まとめ

長手数詰将棋は, 著者らの知る限りいずれも一組の同じ手順を繰り返すことで盤面全体にひとつだけ変化が現れ詰みに向かうことで特別長い手数を実現している. しかし多くの長手数詰将棋は通常の詰将棋には許されていない任意性 (余詰め, 手順前後) を含むため厳密な意味での詰将棋ではない. 本研究では, このような任意性を含まない, 厳密な意味でこれまで知られている最長の詰将棋の構築に成功した.

参考文献

[1] Gibney, Elizabeth.“Google AI algorithm masters ancient game of Go.” Nature News 529.7587 (2016): 445.

[2] 脊尾昌宏.“詰将棋を解くアルゴリズムにおける優越関係の効率的な利用について.” ゲームプログラミングワークショップ 1999 論文集 1999.14 (1999): 129–136.

[3] 小山謙二, 河野泰人. “名作詰将棋における感性の定量的評価.” 情報処理学会論文誌 35.11 (1994): 2338–2346.

[4] 広瀬正幸, 伊藤琢巳, 松原仁. “逆算法による詰め将棋の自動創作.” 人工知能学会誌 13.3 (1998): 452–460.

図 1: 桃花源 初期局面

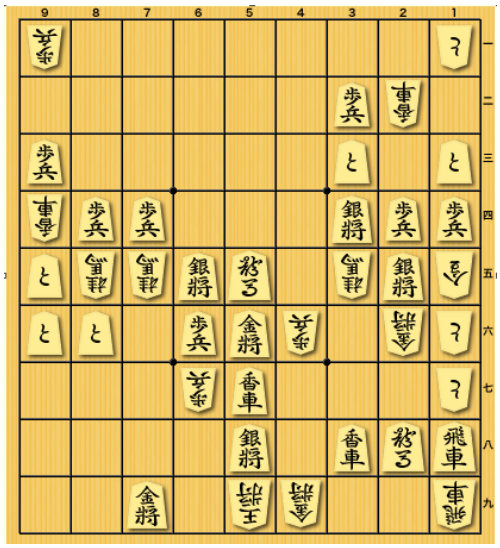


図 2: 本研究 初期局面

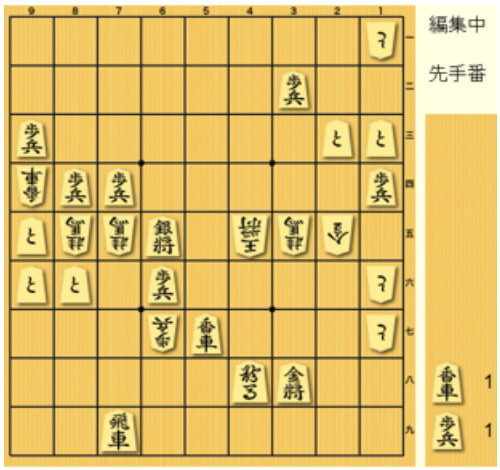


図 3: 本研究 分岐が無くなった局面

[5] 長井歩, 今井浩. “df-pn アルゴリズムの詰将棋を解くプログラムへの応用.” 情報処理学会論文誌 43.6 (2002): 1769–1777.

[6] 伊藤琢巳, 野下浩平. “詰将棋を速く解く 2 つのプログラムとその評価.” 情報処理学会論文誌 35.8 (1994): 1531–1539.

[7] 伊藤琢巳, 河野泰人, 脊尾昌宏, 野下浩平. (1995). 詰将棋を解くプログラムの進歩 (< 小特集 >「ゲームプログラミング」). 人工知能学会誌, 10(6), 853-859.

[8] 伊藤琢巳, 河野泰人, 野下浩平. “非常に手数の長い詰将棋問題を解くアルゴリズムについて.” 情報処理学会論文誌 36.12 (1995): 2793–2799.

[9] 脊尾昌宏. “C* アルゴリズムによる and/or 木の探索および詰将棋プログラムへの応用.” 情報処理学会研究報告知能と複雑系 (ICS) 1995.23 (1994-ICS-099) (1995): 103–110.

のりのり, 変形版へやわけの ゼロ知識証明に対する物理プロトコル

迫田 賢宜 * 小野 廣隆 †

概 要

ゼロ知識証明とは, 証明者が「ある命題が真であることを知っている」と伝えるのに, それ以外の知識を漏らすことなく, 検証者に証明できるようなやりとりの手法である. これは情報セキュリティに大きく関わる手法であるが, カードなどの物理的な道具を用いることによっても実装可能であることが知られている. これらのプロトコルは, 人の手によって道具を用いて実行されるため, 高校生といった非専門家にもプロトコルの安全性や実行の過程などを理解しやすいという利点をもつ.

2009 年に数独に対する物理的ゼロ知識証明プロトコルが考案されて以降, さまざまなペンシルパズルに対する物理プロトコルが考案されている. 本研究では NP 完全問題として知られる, のりのりと変形版へやわけの 2 つのペンシルパズルに対する物理プロトコルを考案する.

1 はじめに

暗号理論の中に, ゼロ知識証明といわれるものがある. これは高い計算能力をもつ証明者 (Prover) と多項式時間の計算能力しかもたない検証者 (Verifier) によっておこなわれる対話型証明の一種である. ある命題が真であることを知っている証明者 P は, それ以外の情報を漏らすことなく検証者 V にそのことを納得させなければならない. さらに, 証明者 P が命題が真であることを知らない場合, 検証者 V は高い確率でそのことを見抜くことができる必要がある. 以降, 証明者 P, 検証者 V をそれぞれ P, V と呼ぶ.

ゼロ知識証明は公開鍵暗号やデジタル署名など, 実生活にさまざまな応用をもつ. しかし, これらのプロトコルはコンピュータ上で処理されるため, 非専門家にはその仕組みや安全性などが理解しにくいという側面がある.

2009 年には, トランプのようなカードを用いた, 数独に対する物理的ゼロ知識証明が考案された. これらのプロトコルは物理的なアイテムを用いて人の手によりプロトコルがおこなわれるため, 非専門家にもゼロ知識証明がどのようなものであるのかがわかりやすい, という教育上の利点をもっている.

*名古屋大学大学院 sakoda.genki@i.mbox.nagoya-u.ac.jp
†名古屋大学大学院 ono@i.nagoya-u.ac.jp

この数独に対する物理プロトコルが考案されて以降, 時間ドロボー [7] といったパズルや美術館, カックロ, ケンケン [1] など, さまざまなペンシルパズルに対する物理プロトコルが考案されている. また, 「非専門家にもわかりやすくゼロ知識証明の概念を伝える」ため, 問題サイズに対して用いられるアイテムの数や手順がより少ないプロトコルも考案されている [3][9].

本論文では, NP 完全問題として知られる「のりのり」と「変形版へやわけ」という 2 つのペンシルパズルに対して, 物理プロトコルを提案する. さらに Dumas らによって考案されたのりのりに対するゼロ知識証明物理プロトコル [6] と比較し, 本考案プロトコルの優位性を示す.

2 ペンシルパズルの物理的ゼロ知識証明

ゼロ知識証明は, 次の 3 つの性質を満たす.

(完全性) P が「ある命題が真である」と知っていれば, V は必ず P を受理する

(健全性) P が「ある命題が真である」と知らなければ, V は十分高い確率で P を拒否する

(ゼロ知識性) V はプロトコルにおいてどのように振る舞っても「ある命題が真である」以外の情報を得ることができない

本論文で考えるのは, 以下のようなシチュエーションである. まず与えられたパズルの解盤面を, 高い計算能力をもつ P は知っているが, 多項式時間の計算能力しか持たない V は自力で求めることが難しい. そして P は自分が解盤面を知っていることを V に伝えたいが, 解盤面を V に明らかにすると V がパズルを解く楽しみがなくなってしまうので, それはおこなわない. 同様に解盤面に関する情報は一切漏らさない. これらの条件のもとで, P が正しい解盤面を知っていると V に納得させる手法を, ペンシルパズルのゼロ知識証明とよぶ.

3 数独に対する物理プロトコル

ここでは, 先行研究で考案された数独に対する物理的ゼロ知識証明プロトコルを紹介する [3]. まず, 数独とは次のルールに従い, 各セルに数字を埋めるペンシルパズルである. これは NP 完全であることが示されている [2].

問題 (数独) $n \times n$ のグリッドが $m \times m$ のサブグリッドに分割されており *($n = m^2$),* いくつかのセルに数字が書かれている. このとき, 各行, 列, サブグリッドに 1 から n までの数字がちょうど一回ずつ現れるよう, セルを埋めることができるか.

この問題に対しては次のプロトコルが考案されている. なお, ここで使用されるアイテムは表が 1 から n, 裏が同一のカードを $3n$ セット, つまり $3n^2$ 枚のカードである.

数独に対する物理的ゼロ知識証明プロトコル

- P は各セルに, 解盤面と一致するカードを 3 枚ずつ裏返して置く. あらかじめ数字が書かれていたセルにはその値と一致するカードを表にして置く
- V は各行, 列, サブグリッドごとに 3 枚のうちからランダムにカードを 1 枚選ぶ
- P は選ばれたカードを各行, 列, サブグリッドごとにまとめ, パケットを作る. パケットごとにカードをシャッフルし, V に返す
- V は返されたパケットの中身を確認し, すべてのパケットにおいて 1 から n の数字がちょうど一回ずつ現れていれば受理. そうでなければ拒否

このプロトコルにおいて, 解盤面に関する情報は一切漏れていない. このプロトコルは 2009 年に考案されたものであるが, その後 Sasaki らによって改良されたプロトコルが登場し, 必要なカードの枚数が $3n^2$ から $2n^2 + n$ に削減されている [9].

4 のりのり, 変形版へやわけに対する物理プロトコルとその解析

4.1 カード列の id づけによる復元

本小節では考案プロトコルに用いる, カード列の復元のための操作の説明を与える. id づけとは, 裏のままのカード列に対し, 同枚数の数字が描かれたカードを使用することでカード列の表を確認することなく初期順列を復元するプロトコルである. id づけには, 橋本らによって考案されたパイルスクランブルシャッフルを利用する [4].

以下が, id づけの具体的な手順である.

(1) id づけを施したいカード列 v を置き, その下に 1 から n の数字カードを順に並べる

?	?	?	…	?
1	2	3	…	n

(2) 下段の数字カード列を裏にし, 上下を組にしたままガード列をランダムシャッフルする

?	?	?	...	?
?	?	?	...	?

(3) プロトコルの必要に応じて, 上段のカード列を表にして確認する.
このとき, 上段のカード列の初期順列に関する情報は全く漏れていない

(4) 確認後, 上段のカード列を裏にして再びランダムシャッフルする

(5) 下段のカードを表にして, 上下の組を保ったまま 1 から昇順に並べ替える

?	?	?	...	?
1	2	3	...	n

以上の操作をおこなうことで, カード列の初期順列に関する情報を明らかにすることなく, そこに含まれるカード列の内容を知ることができる. さらにカード列の初期順列を復元することも可能である.

4.2 のりのりに対する物理的ゼロ知識証明

4.2.1 考案プロトコル

まず, のりのりの問題を紹介したあと, それに対する物理的ゼロ知識証明を提案する.

問題 (のりのり) $m \times n$ のグリッドが適当に分割されている. このとき, 次の条件をすべて満たすようにセルを黒く塗ることができるか.

- 各分割されたエリア (以降, 部屋と呼ぶ) にはちょうど 2 つの黒マスが入る
- 黒マスはちょうど 1 つの黒マスと隣接する

この 2 つがペンシルパズルのりのりの条件であるが, その初期盤面と解盤面を図 1 に示す.

これに対するプロトコルに用いるのは表が黒または白で, 裏が同一のカードである.

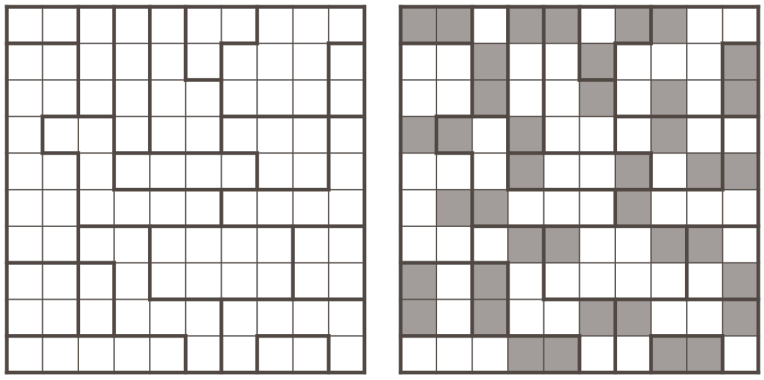


図 1: のりのりの初期盤面 (左) と解盤面 (右)

のりのりに対する物理的ゼロ知識証明プロトコル

- P は盤面の各セルに一致する黒または白のカードを 6 枚ずつ裏返して置く. さらに盤面の上下左右の外部にも白カードを 1 枚ずつ置く
- V は (a)(b) の検証のどちらかをランダムにおこなう
  - (a) P は各セルをパケットにし, 袋に入れる. V は袋からランダムにパケットを取り出し, 表にする. すべてのパケットでカードの色がそろっていれば受理
  - (b) V は以下の検証をおこなう
    - V は部屋を指定し, P は各セルからカードを 1 枚ずつ取ってきて, カード組をシャッフルする. カードを表にして, 黒カードの数がちょうど 2 枚であれば受理. これをすべての部屋に対しておこなう
    - V はセルを指定する. P は指定されたセルに置かれたカードを最低面, それに隣接する 4 枚のカードをその上に重ね, パケットを作る. これをすべてのセルに対しておこなう. V は m × n 個のパケットからランダムに 1 つ取り出し, 最低面を確認. 最低面が白であればそのパケットを破棄, 黒であれば展開し, パケット内にちょうど 2 枚の黒カードがあれば受理. これをすべてのパケットに対しておこなう

図 2 にあるように, (b) の 2 つめの検証をおこなった場合, 最低面が黒であれば必ずそのパケットには黒カードがちょうど 2 枚含まれる.

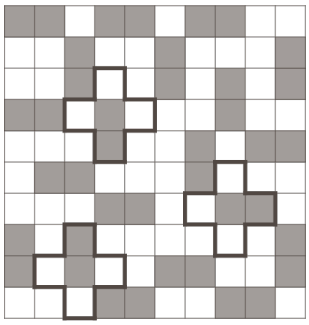


図 2: のりのり 第 2 条件の検証

4.2.2　健全性誤り確率の解析

解盤面を知らない (悪意のある)P は, V によってランダムでおこなわれる 2 つの検証 (a)(b) のどちらも満たすようなカード配置はできない. よってこのプロトコルを l 回繰り返すことによって, この P が拒否される確率は $1-\left(\frac{1}{2}\right)^l$ である.

4.2.3　既存プロトコルとの比較

こののりのりに対して, 2019 年に Dumas らによって物理的ゼロ知識証明プロトコルが考案された [6]. Dumas らのプロトコルではのりのりの第 2 条件の検証の際に, 1 つの黒いセルに対してチェックするごとに, シャッフル操作により解盤面を復元する, というステップをおこなっている. そのため, セルに置かれたカードを繰り返し用いることができ, カードの総枚数は少ないものとなっている. 一方で, 第 2 条件の検証のためだけでも復元操作を黒のセル数分おこなうため, カードのシャッフルや並べ替えといった操作数が多くなっている. 加えて解盤面を復元するためのマーカー用カードや数字カードなど, 用いられるカードの種類が我々が考案したプロトコルより 3 種類多い.

以上のことから, プロトコルの実行しやすさ, わかりやすさという点において, 我々のプロトコルが優れているといえる.

4.3　変形版へやわけに対する物理的ゼロ知識証明

本小節では変形版へやわけに対する物理的ゼロ知識証明を提案する. 以下が変形版へやわけの問題である.

問題　(変形版へやわけ) $m \times n$ のグリッドが長方形に分割されていて, いくつかの長方形の内側に数字が埋められている. このとき, 次の条件をすべて満たすようにセルを黒く塗ることができるか.

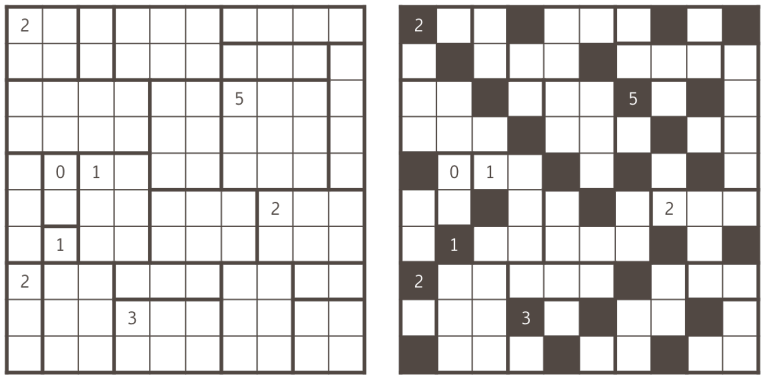


図 3: 原型へやわけの初期盤面 (左) と解盤面 (右)

- 黒マスは隣接しない
- 数字はその長方形内に塗られる黒マスの数
- 白マスは縦, 横方向に 3 つ以上連続して部屋をまたがない

へやわけ問題の原形では上に加えて,

- 黒マスで盤面を分断しない

という条件が含まれる. このへやわけ問題の原形は NP 完全, さらに変形版へやわけも NP 完全であることが知られている [5].

この問題に対する提案プロトコルを以下に記す. 用いるアイテムは 2 種類のカードであり, 表の柄が異なる. 1 つは黒または白の色カード, もう 1 つは 1 から n の数字 (n は盤面内の長方形の最大サイズ) が書かれた数字カードである. どちらも裏にすると区別のつかない同一の柄であるとする.

変形版へやわけに対する物理的ゼロ知識証明プロトコル

- P は盤面の各セルに一致する黒または白のカードを裏返して置く
- V は順に P の置いたカードが, このパズルの条件を満たしているか確認する.
  以降このプロトコルにおける各検証では, 前述の id づけによるカード列の復元をおこない, 盤面を復元するものとする
  - V はランダムに盤面から隣接する 2 つのセルを選び, P はカードを取ってくる. P は選ばれたカードを確認し, (黒, 白) であれば白を, (白, 白) であれば黒のカードを追加し, シャッフルして V に渡

す. V は渡されたカード組が (黒, 白, 白) であれば受理. これをすべての隣接するセルに対しておこなう.

- 各部屋に対する黒マスの数は, のりのりと同様の手法で確認する
- V はランダムに 1 部屋を縦断または横断するセルと, それに同方向に隣接するセルを 1 つずつ選ぶ. P は選ばれた範囲のカードを取ってくる. P はカードを確認し, そこに含まれる黒のカードの数が「含まれうる黒カードの最大数 (つまり範囲のサイズを L とすると, $\lceil \frac{L}{2} \rceil$)」に達していなければ, 黒カードをその枚数になるまで追加する. そして追加した黒カードとの合計が $\lfloor \frac{L-1}{2} \rfloor$ になるように白カードも追加する. シャッフルして返却されたカード組に, 黒いカードが「そこに含まれうる黒カードの最大数」含まれれば V は受理. これをすべての 3 長方形の境界を含む範囲に対しておこなう.

プロトコルの最後の検証は, 指定した範囲がすべて白マスであった場合のみ, 拒否される仕組みになっている. もし, 指定した範囲がすべて白マスであった場合, プロトコルにしたがってカードを追加しても「そこに含まれうる黒マスの最大数」には達しないことがわかる. 図 4 を例にして示すと, ここでは 2 境界をまたぐ連続した 5 つのセルが選択されている. このとき黒カードは 1 枚置かれているので 2 枚の黒カードを追加する. もし黒カードが 2 枚置かれていれば追加するのは黒カード 1 枚と白カード 1 枚である. 選択された範囲がすべて白カードであれば, 2 枚の黒カードを追加してもこの範囲に含まれうる最大枚数である 3 枚には達しない.

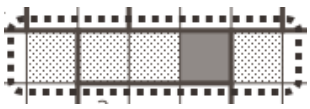


図 4: 変形版へやわけ 第 3 条件の検証

5 おわりに

本論文では, NP 完全問題である 2 つのペンシルパズル, のりのりと変形版へやわけに対する物理的ゼロ知識証明プロトコルを考案し, のりのりに関しては既存のプロトコルとの比較をおこなった. 今後の課題としては, 変形版へやわけに対するプロトコルの健全性誤りの解析や, どちらのプロトコルに対しても必要なアイテム数や手順の削減が挙げられる.

参考文献

[1] Bultel X., Dreier J., Dumas J., Lafourcade P.: Physical Zero-knowledge Proofs for Akari Takuzu, Kakkuro and Kenken - Fun with Algorithms, LIPIcs, vol. 49, pp 8:1–8:20 (2016).

[2] Colbourn, c.: The complexity of copleting partial latin squares, Discrete Applied Mathematics, vol. 8, pp. 25–30 (1984).

[3] Gradwohl, R., Naor, M., Pinkas, B., Rothblum, G.: Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles, Theory of Computing Systems. 44(2), pp. 245–268 (2009).

[4] Hashimoto, Y., Shinagawa, K., Nuida, K., Inamura, M., Hanaoka, G.: Secure grouping protocol using a deck of cards, ICITS 2017. LNCS, vol. 10681, pp. 135–152. Springer (2017).

[5] Holzer M., Ruepp O.: The Troubles of Interior Design - A Complexity Analysis of the Game Heyawake - 4th International Conference on Fun with Algorithms, Lecture Notes in Computer Science, vol.4475, pp.198–212 (2007).

[6] Dumas J., Lafourcade P., Miyahara D., Mizuki T., Sasaki T., Sone H.: Interactive Physical Zero-Knowledge Proof for Norinori, COCOON, Lecture Notes in Computer Science, vol.11653 pp. 166–177 (2019).

[7] Ueda, K., Nishimura, H.: Physical Zero-Knowledge Proof Systems for Instant Insanity, Publications of the Research Institute for Mathematical Sciences, vol.1849, pp. 120–126 (2013).

[8] Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Securely computing three-input functions with eight cards, IEICE Transa. Fundamentals, vol.E98-A, no.6, pp.1145–1152 (2015).

[9] Sasaki, T., Mizuki, T., Sone, H.: Card-Based Zero-Knowledge Proof for Sudoku, LIPIcs-Leibniz International Proceedings in Informatics, vol. 100, pp. 29:1–29:10 (2018).

空間計算量と局所的時間計算量を削減したアトミッククロスチェーンスワップ

井本　宗一郎 †, 首藤　裕一 †, 角川　裕次 ‡, 増澤　利光 †

† 大阪大学　大学院情報科学研究科
‡ 龍谷大学　理工学部

アトミッククロスチェーンスワッププロトコルは Herlihy によって 2018 年に提案された分散プロトコルで, 複数の取引者が異なるブロックチェーンにまたがった資産の交換を実現する [1]. この資産の交換のことをスワップと呼ぶ. このプロトコルは以下の 3 つの条件を保証しており, 取引者達はお互いのことを信頼できない場合においても安全に取引をすることができる.

1. すべての取引者がプロトコルに準拠した処理を実行する場合, すべての資産が取引者間で交換される.
2. 一部の取引者またはその連合がプロトコルから逸脱したとしても, プロトコルに準拠する者は損失が発生しない.
3. プロトコルから逸脱しても利益は得られない.

各取引者にはスワップ開始前に, どの取引者へどれだけの資産を移送し, その対価としてどの取引者からどれだけの資産が移送されるかが知らされ, その上で資産の交換を行う. そのために各取引者には, 取引者を頂点, それらの資産の移送を有向辺で示した図 1 のような有向グラフ $D = (V, A)$ が与えられる. ここで V は頂点の集合,A は有向辺の集合である. また全取引者はグラフ D における, あるフィードバック頂点集合 L を計算する. グラフ D におけるフィードバック頂点集合とは, その集合を D から取り除いたならば,D がサイクルを 1 つも含まないグラフになるような頂点の集合である.

多くのブロックチェーンでは取引者間の資産の交換にスマートコントラクトが使用される. スマートコントラクトとはブロックチェーン上で動作するプログラムで, まず契約内容となる事前条件を設定し, それを満たすイベントが行われたならば, 契約に指定された資産の移送が正しく実行されるというものである. Herlihy[1] はスマートコントラクトを用いて上記の 3 条件を満たすプロトコル，すなわちアトミッククロスチェーンスワッププロトコルを提案した. 上記の 3 つの条件を満たすためには, 各取引者 $v \in V$ は，有向グラフ

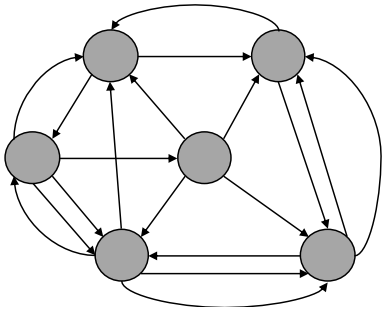


図 1: 複数の取引者によるスワップをグラフ化した例

D における v から他者への外向辺に対応する資産の移動がひとつでも発生したならば,v のすべての内向辺に対応する他者から自身への資産の移動が遂行されるということを保証しなければならない. そのために, 既存のプロトコル [1] では, 任意の取引者 v において, 他者から v への資産の移送に対応する全てのスマートコントラクトの有効期限が,v から他者への資産の移送に対応する全てのスマートコントラクトの期限よりも十分に大きくなるように設定され,v から他者へ資産が移送されたならば移送条件を満たす鍵を知ることができるように設定している. しかし取引者が $L \leq 1$ であるようなフィードバック頂点集合 L を発見できなかった場合は, そのような有効期限を設定することはできない. Herlihy のプロトコルはスワップのグラフトポロジと取引者の電子署名を各スマートコントラクトに格納し, それらを用いた巧妙な移送条件と有効期限を設定することで, アトミッククロスチェーンスワッププロトコルを実現している.

本稿では, 移送条件と有効期限を, 各取引者がこれまでに取得した他の取引者の署名の数に応じた単純なものに変更することで,Herlihy のプロトコルに比べ時間複雑度・空間複雑度を改善したプロトコルを提案する. 具体的には, 既存プロトコルの空間複雑度（全ブロックチェーンに書き込まれたコントラクトに必要なビットの総数）は $O(|A|^2)$ であるのに対して, 提案プロトコルの空間複雑度は $O(|A||V|)$ に削減されている. また既存研究の局所的な計算複雑度（スワップ内のコントラクトの資産を移送するために必要な計算の最悪時実行時間）は $O(|V||L|)$ であるが, 提案プロトコルの局所的計算複雑度は $O(|V|)$ となる. したがって既存手法と比較して提案プロトコルは，与えられた有向グラフ D が密であるときに空間複雑度が著しく改善され，フィードバック頂点集合 L の要素数が大きいときに局所計算複雑度が著しく改善される. 本研究の詳細はフルペーパー [2] を参照されたい.

参考文献

[1] M. Herlihy. Atomic cross-chain swaps. Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing. ACM, 2018.

[2] https://arxiv.org/abs/1905.09985

ビザンチン環境を考慮した距離 k 極大独立集合問題を解く自己安定乱択アルゴリズム

土田 将司, 大下 福仁, 井上 美智子

奈良先端科学技術大学院大学

2019 年 9 月 6 日

1 はじめに

近年，多数のコンピュータ (以下，ノード) で構成された分散システムは様々な機能を実現するため日々進歩している．大小様々な分散システムが構築され利用されているが，システムが大規模になるに連れて各ノード間の通信が複雑化するという問題が生じる．各ノード間の通信が複雑化するとデータの移動に遅延が生じ，システム全体における処理速度自体が低下する要因にもなりうる．そこで本稿では大規模な分散システムであっても小規模な分散システム同様の通信複雑度を実現することができれば，大規模なシステムであってもシステム全体の処理速度を下げることのない通信が可能になるのではないかと考える．大規模分散システムにおける通信複雑度を下げるため，システム内のノードをクラスタリングし階層構造の作成を目指す．そのため本稿では分散システムを連結無向グラフと考え，各頂点を各ノードと見立てた時に距離 k での極大独立集合を求める．独立集合に含まれるノードを $head$ ノード，含まれていないノードを $child$ ノードとしたとき，$child$ ノードは最寄りの $head$ ノードと通信を行い，$head$ ノードは $head$ ノードだけで仮想的なグラフを作ることでクラスタ環境を実現できると考えている．このときの $head$ ノード間の距離 k を変更することで任意のサイズにノードをクラスタリングすることができる．

また，ノードは物理エンティティであるため故障する恐れがある．ノードに故障が発生した場合，システム全体でクラスタリングし直す必要が生まれる可能性があるが，システム自体が大規模であるときに全システムを一旦初期化して再構築するのは困難となる．そこで本研究では大規模なシステムを考慮してシステムが動作中であっても故障したノードを除いて自動で再構成するようなアルゴリズムの提案を行う．すなわち，任意の状況からでも安定した状態に到達可能なアルゴリズムである自己安定アルゴリズムの提案を行う．また，通常の自己安定アルゴリズムが想定するモデルは，システムに一時的に故障が生じ，その後故障が生じないことを仮定している．本研究では故障により完全に停止せず，おかしな状態に遷移し続けるような故障も考慮するため，永続的な故障が発生する場合についても動作可能な自己安定アルゴリズムを考える．任意の動作を行う永続的故障を想定するため，本稿ではこのような故障をビザンチン故障としてモデル化する．自己安定アルゴリズムでビザンチン故障を考慮した研究はいくつか存在する [1, 2, 3]．本研究ではこのビザンチン故障が存在する分散システム上での距離 k 極大独立集合に注目する．

2 諸定義

2.1 分散システム

分散システムをノード集合を $V = \{v_1, v_2, \cdots, v_n\}$，辺集合を E とした無向連結グラフ $G = (V, E)$ として表す．グラフ中のノード数を $n = |V|$ とする．ノード $v_i, v_j \in V$ 間に辺 $(v_i, v_j) \in E$ が存在するとき，v_i と v_j は隣接していると呼ぶ．ノード v_i の隣接ノードの集合を $N_i = \{v_j | (v_i, v_j) \in E\}$ で表し，v_i の次数を $\Delta_i (= |N_i|)$ と表す．グラフ $G = (V, E)$ の最大次数を $\Delta = max(\{\Delta_i | v_i \in V\})$ と定義する．ノード v_i と v_j が隣接であるとき，v_i は v_j の全ての変数の値を直接参照することができる．各ノードは固有の ID を持ち，ノード v_i の ID を id_i と表す．

各ノード v は各隣接ノードと直接情報交換が行える状態機械 (S, δ) としてモデル化する．S はノードの状態集合を表し，各ノードの状態はノードメモリ内の変数の値によって決定される．状態遷移関数 δ は現在のノード v の状態及び各隣接ノードの状態を入力値とした関数であり，出力は v の次の状態である．

分散システムの全体の状態を状況と呼び，全てのノードの状態の組によって表される．分散システムでとり得る全ての状況の集合を $\mathcal{C}$ と定義する．初期状況 $C_0 \in \mathcal{C}$ では全てのノードは任意の状態で存在する．

ノード集合の無限のシーケンスをスケジュール $(schedule)$ とし $X = \rho_0, \rho_1 \cdots$(各 $x \geq 0$ に対して $\rho_x \subseteq V$) と表す．また状況 C_x において全ての $v \in \rho_x$ が動作を行い状況 C_{x+1} に遷移したとき，この遷移を $C_x \overset{\rho_x}{\mapsto} C_{x+1}$ と表す．各 x において $C_x \overset{\rho_x}{\mapsto} C_{x+1}$ が成り立つとき，初期状況から始まる状況の無限のシーケンスを実行 $(execution)$ とし，$E = C_0, C_1, \cdots$ と表す．このとき，2 つの状況 C_x と C_{x+1} では ρ_x に含まれるノードのみ状態が異なることが許され，ρ_x に含まれていないノードは C_x と C_{x+1} において同一の状態であることを要求する．また，ρ_x に含まれるノード v_i の動作は状況 C_x での v_i の状態及び v_i の隣接ノードの状態にのみ依存し，動作の結果は C_{x+1} での v_i の状態である．非同期環境での弱公平な分散デーモン (Distributed daemon) を考えるため，スケジュールに対しては全てのノードが無限にしばしば現れることのみを仮定する．

本稿ではノードの故障モデルとしてビザンチン故障を考える．ビザンチン故障とはアルゴリズムに従わず任意の動作を行える故障である．ビザンチン故障を起こしたノードをビザンチンノードと呼び，故障を起こしていないノードは正常ノードと呼ぶ．ノード v_f がビザンチンノードであるとき，v_f はノードメモリ内変数を自由に決定することができる．ただし，v_f は正常ノード同様他のノードの変数の値を変更することはできない．

非同期分散システムを考えているため，非同期ラウンドを定義する．初期状況 C_0 から全ての正常ノードが ρ_i に少なくとも 1 度現れるまでの状況のうち，最小の添字を持つ状況 C_s までのシーケンス $\varrho_1 = C_0, \cdots, C_s$ を 1 ラウンド目とする．次の非同期ラウンドは C_s を初期状況として同様に ϱ_2 を定義する．

2.2 距離 k 極大独立集合問題

アルゴリズムが上位のアプリケーションに対して出力する計算結果は各ノードが持つある変数の集合とする．この出力される変数の集合を出力変数とし，全ノードの出力変数のベクトルを Out とする．各ノード v_i に対して問題 P における v_i の条件 $Spec_i(Out)$ を定義する．$Out(C)$ を状況 C における全ノードの出力変数のベクトルとしたとき，全てのノード v_i について $Spec_i(Out(C))$ が満たされるのであればアルゴリズム A は状況 C にて問題 P の仕様を満たしたとする．

アルゴリズム A のある実行 $E = C_0, C_1, \cdots$ を考え，ある状況 C_i で問題 P の仕様を満たし，任意の $j \geq i$ において状況 C_j でも仕様を満たすとき，アルゴリズム A は実行 E において問題 P を解いたとする．また，

このときアルゴリズム A の実行により到達すべき状況を判定する条件式，すなわち以下の性質を満たすような，実行 E の状況に対する条件式 I を考える．

- ある状況 C_i で I を満たすとき，任意の $j \geq i$ において，状況 C_j が P の仕様を満たす．

故障の存在する問題の仕様は正常なノードにのみ定義する．$Out(C')$ を状況 C' における全ノードの出力変数のベクトルとしたとき，故障ノードから距離 $d+1$ 以上離れた全てのノード v_i について $Spec_i(Out(C'))$ が満たされるのであればアルゴリズム A は状況 C' にて問題 P の仕様を満たしたとする．アルゴリズム A のある実行 $E' = C'_0, C'_1, \cdots$ を考え，ある状況 C'_i で故障ノードから $d+1$ 以上離れた全てのノードが仕様を満たし，任意の $j \geq i$ において状況 C'_j でも故障ノードから $d+1$ 以上離れた全てのノードが仕様を満たすとき，アルゴリズム A は実行 E' において問題 P を解いたとする．また，このときアルゴリズム A の実行により到達すべき状況を判定する条件式，すなわち以下の性質を満たすような，実行 E' の状況に対する条件式 I_d を考える．

- ある状況 C'_i で I_d を満たすとき，任意の $j \geq i$ において，状況 C'_j が P の仕様を満たす．

もし任意の状況から始まる全ての A の実行が条件式 I_d を満たす状況を含むとき，アルゴリズム A を問題 P を解く d-安定アルゴリズムと呼ぶ．また，A が I_d を満たして問題 P を解くとき，故障ノードが存在しなければ A は任意の状況から正常な状況，すなわち問題 P の仕様を満たした状況に収束し維持するため，A は自己安定アルゴリズムであると言える．

本稿では d-安定アルゴリズムを考えるため，収束性と閉包性について定義する．

収束性：P の仕様を満たさない任意の状況からアルゴリズムを開始したとしても，システムはいつかは I_d を満たす

閉包性：収束性を満たした後，システムは常に P の仕様を満たす

ここで本稿における問題 P，すなわち距離 k 極大独立集合問題の仕様を定義する．各ノード v_i は出力変数として変数 $v_i.state$ を持ち，2 つの値 $head$ と $child$ のどちらかが格納されているとする．$v_i.state = head$ のノードを $head$ ノードとする ($v_i.state = child$ のノードを $child$ ノードとする)．また，v_i から距離 k 以内のノードの集合を N_{ik} とする．

問題 P：距離 k 極大独立集合問題

問題 P の仕様としてノード v_i の $Spec_i()$ とする．以下の式のどちらかを C で満たすとき，$Spec_i(Out(C)) == true$ となる．

- $v_i.state == head \wedge (\nexists v_j \in N_{ik} | v_j.state == head)$
- $v_i.state == child \wedge (\exists v_j \in N_{ik} | v_j.state == head)$

3　アルゴリズム

諸定義を踏まえ，本稿では距離 k 極大独立集合問題を解くアルゴリズムを考える．

3.1　アルゴリズムの概要

距離 k 極大独立集合問題を解くため，アルゴリズムでは *head* ノードは距離 k 以内に *head* ノードが存在しないかどうかを確認し，*child* ノードは距離 k 以内に *head* ノードが存在することを確認する．そこで本アルゴリズムでは *head* ノードは自身が *head* であることを示す情報として，自身 ID に距離情報として発信者を表す 0 を付与した項組を作成し *head* ノード情報とする．全てのノードはこの *head* ノード情報を距離情報が k を超えるまで隣接ノードから距離情報をインクリメントしながらコピーする．この情報により *head* ノードは距離 k 以内に競合している *head* ノードが存在するかどうかを判別でき，*child* ノードは距離 k 以内に *head* ノードが存在するかどうかを判別することができる．

head ノードが競合している場合，どちらかのノードは *head* ノードをやめ，*child* ノードになる．このとき，競合中の *head* ノードの中から少なくとも 1 つのノードが *head* ノードとして残るようにする．具体的には各 *head* ノード v_i 及び v_j は 1bit の乱数値 (0 もしくは 1) を生成し，お互いに送り合う．その後，v_i の乱数値が v_j の乱数値より大きいとき，v_j は *child* ノードになる．すなわち，*head* ノードが *child* ノードになるとき，少なくとも 1 つの *head* は乱数値として 1 を生成しており，*head* ノードのままとなる．

また，*head* ノード間に距離が存在するため情報が伝達されるのに遅延が生じる．この遅延中に同一の情報を何度も参照する場合があるため，本稿では競合している任意の 2 ノード v_i と v_j は交互に乱数値の比較動作を行うための工夫を実装する．各ノードは乱数値とは別に 1bit のカウンタを持ち，競合相手の ID より大きい場合にはカウンタが等しいとき，そうでないときはカウンタが異なるときに比較動作を行う．v_i が比較動作を行った場合はこの 1bit カウンタを反転させ相手に送る．v_i が比較動作を行わなかった場合，現在存在する情報では競合相手 v_j がスケジューラで指定されたときに比較動作を行うような値になっているため，v_j が比較動作を行い新しい情報が v_i に到達するのを待つ．よって，*head* ノードとして競合している相手に伝達させる情報としては自身の ID，競合相手の ID，乱数値，1bit カウンタの値の 4 つで，そこに距離情報を足した 5 項組をやり取りする．

ノード v_i は受け取った全ての情報に対して処理を行った後，*head* ノードであるなら乱数値を更新し，発信する情報の更新を行う．

ノード v_i が *child* ノードであり，距離 k 以内に *head* ノードが存在するという情報を 1 つも持っていない場合，v_i は *head* ノードとなり *head* ノードであるという情報を発信する．

本稿ではビザンチンノードの存在する環境を想定している．アルゴリズムとして距離 k までしか情報を伝播させないため，ビザンチンノードからの情報は高々距離 k までの範囲にしか届かない．そのためビザンチンノードから距離 k までの *head* ノード v_a は変動する可能性がある．また，v_a からの情報も届かない距離 $2k+1$ 以上離れた *head* ノード v_b は，I_d を満たした状況以降，$v_b.state$ の値を変化させることはない．これについては閉包性の証明にて明らかにする．

3.2 アルゴリズムの詳細

本アルゴリズムの疑似コードを Algorithm 1 に示す．変数 $v_i.state$ は $head$ もしくは $child$ を保存する変数である．変数 $v_i.Info$ は (差出人 ID，差出人からの距離) の $head$ ノードの情報を表す 2 項組を保存する集合であり，この 2 項組を $head$ ノード情報と呼ぶ．変数 $v_i.Conflict$ は (差出人 ID，宛先 ID，乱数値，カウンタ値，差出人からの距離) の 5 項組を保存する集合であり，この 5 項組を競合情報と呼ぶ．変数 $v_i.LocalInfo$ は (相手 $headID$，カウンタ値) の 2 項組を保存する集合である．変数 $v_i.rand$ は現在の乱数値を保存する変数である．

ノード v_i がアクティブになったとき，初めに v_i の隣接ノードからの情報を元に $v_i.Info$ と $v_i.Conflict$ を更新する (1 行目から 4 行目)．1 行目及び 2 行目では全ての隣接の $v_i.Info$ と $v_i.Conflict$ それぞれの和集合を取得する．その後 3 行目及び 4 行目で同一差出人 ID を持つ情報のうち距離情報が最小の情報以外を削除する．この結果 $v_i.Info$ には同一ノードから発信された距離情報が最小の $head$ ノード情報だけが残り，$v_i.Conflict$ には同一ノードから発信された距離情報が最小の競合情報だけが残る．

5 行目から 24 行目は v_i が $head$ ノードであるときの動作であり，25 行目から 29 行目は v_i が $child$ ノードのときの動作である．v_i が $head$ ノードであるとき，はじめに $head$ ノード情報から $v_i.LocalInfo$ に含まれていない ID の情報があれば，その ID の情報を作成する．この $v_i.LocalInfo$ に含まれている ID は競合中であることを示す．また，同時に初期値 0 としたカウンタ値を保存する．その後，$v_i.Conflict$ 内に比較すべき $head$ ノード v_j からの情報が入っている場合に乱数の比較動作を行う (10 行目から 17 行目)．比較すべき情報とは $v_i.Conflict$ 内の id_j の情報のカウンタ値と $v_i.LocalInfo$ 内の id_j の情報のカウンタ値が等しく $id_i > id_j$ である，もしくはカウンタ値が異なっており $id_i < id_j$ であるかのどちらかを満たす情報である．カウンタ値は 1bit であり，各ノードは固有の ID を持つため必ず v_i か v_j のどちらか 1 つのノードが条件に当てはまる．v_i が条件に当てはまったとき，$v_i.LocalInfo$ 内の id_j のカウンタ値を反転させ，乱数値の比較を行う．$v_i.rand$ の値が $v_i.Conflict$ 内の id_j の情報の乱数値より小さいときに $child$ ノードとなる．この動作を $v_i.Conflict$ 内に含まれる条件の当てはまる情報全てに対して行う．その後，v_i が $head$ ノードであれば乱数値を再生成し，$v_i.LocalInfo$ から $v_i.Info$ に存在しない ID の情報を削除し，$v_i.Info$ と $v_i.Conflict$ を更新する．$v_i.Info$ には $head$ ノード情報を，$v_i.Conflict$ には $v_i.LocalInfo$ 内の情報に $v_i.id$ と $v_i.rand$，距離情報を加えて作成する．作成された情報は次に隣接ノードがアクティブになった際，1 から 4 行目で伝播していく．

26 行目からは v_i が $child$ ノードのときの動作である．$v_i.Info$ が $NULL$ のとき，すなわち距離 k 以内に $head$ ノードが存在するという情報が無いとき，v_i は $head$ ノードとなり，$head$ ノード情報を作成する．この情報に関しても次に隣接ノードがアクティブになった際に伝播する．

3.3 閉包性の証明

本稿において本来であれば収束性の証明と閉包性の証明が必要となるが，収束性の証明は現在未完了であるため閉包性の証明のみを示す．収束性に関してはシミュレーションによる結果を示す．

閉包性の証明として，I_d を満たした状況 C_a において，任意 $b \geq a$ において C_b では問題 P の仕様を満たすことを補題にて示す．ここで，本稿における条件式 I_d を定義する．条件式 I_d を以下のように定義し，I_d を満たした状況 C_a において，任意 $b \geq a$ において C_b では距離 k 極大独立集合問題が解けていることを示す．

Algorithm 1 Code of node v_i

1: $v_i.Info = \bigcup_{v_j \in N_i} \{(j, a+1) \mid (j,a) \in v_j.Info \wedge a < k \wedge j \neq i\}$
2: $v_i.Conflict = \bigcup_{v_j \in N_i} \{(j, y, r, c, a+1) \mid (j,y,r,c,a) \in v_j.Conflict \wedge a < k \wedge j \neq i\}$
3: $v_i.Info = v_i.Info \setminus \{(x,a) \mid (x,a),(x,b) \in v_i.list \wedge a > b\}$
4: $v_i.Conflict = v_i.Info \setminus \{(x,y,r_1,c_1,a) \mid (x,y,r_1,c_1,a),(x,y,r_1,c_1,b) \in v_i.list \wedge a > b\}$
5: **if** $v_i.state == head$ **then**
6: **for all** $(id_j, -) \in v_i.Info$ **do**
7: **if** $(id_j, -) \notin v_i.LocalInfo$ **then**
8: $v_i.LocalInfo = v_i.LocalInfo \cup \{(id_j, 0)\}$
9: **end if**
10: **if** $(id_j, v_i.id, rand_j, c_j, -) \in v_i.Conflict$ **then**
11: **if** $\exists (id_j, lc_j) \in v_i.LocalInfo \mid (v_i.id > id_j \;\&\&\; c_j == lc_j) \vee (v_i.id < id_j \;\&\&\; c_j \neq lc_j)$ **then**
12: $v_i.LocalInfo = v_i.LocalInfo \setminus \{(id_j, lc_j)\} \cup \{(id_j, (lc_j + 1) \bmod 2)\}$
13: **if** $v_i.rand < rand_j$ **then**
14: $v_i.state = child$
15: **end if**
16: **end if**
17: **end if**
18: **end for**
19: **if** $v_i.state == head$ **then**
20: $v_i.rand = random(0,1)$
21: $v_i.LocalInfo = v_i.LocalInfo \setminus \{(id_j, -) \mid (id_j, -) \notin v_i.Info\}$
22: $v_i.Info = \{(v_i.id, 0)\}$
23: $v_i.Conflict = \bigcup_{(id_j, lc_j) \in v_i.LocalInfo} \{(v_i.id, id_j, v_i.rand, lc_j, 0)\}$
24: **end if**
25: **else**
26: **if** $v_i.Info == NULL$ **then**
27: $v_i.state = head$
28: $v_i.Info = \{(v_i.id, 0)\}$
29: **end if**
30: **end if**

条件式 I_d

1 状況 C_a において故障ノードから距離 $\frac{d}{2}+1$ 以上離れた全てのノード v_i が $Spec_i(Out(C_a))$ を満たす
2 状況 C_a において故障ノードから距離 $\frac{d}{2}+1$ 以上離れた任意の正常ノード v_i が発信した $head$ ノード情報を v_i から距離 k 以内の全てのノードが保持している
3 状況 C_a において正常 $child$ ノードの $head$ ノード情報を保持する正常ノードが存在しない

ここで「ノード v_i の $head$ ノード情報を v_j が保持する」とは，$(v_i.id, -) \in v_j.Info$ が成り立つことである．

補題 1. 任意の状況 C_a においてアルゴリズムが条件式 I_{2k} を満たしたとき，任意の状況 $C_b(a \leq b)$ ではビザンチンノードから距離 $2k+1$ 以上離れた任意のノード v_i は $v_i.state$ を変更しない．

Proof. I_{2k} を満たした状況を C_a とし，任意の状況 $C_b(a \leq b)$ においてビザンチンノードから距離 $2k+1$ 以上の場所に存在する任意の正常エージェントを v_i とし，以下の 3 つの事象について考察する．

(1) C_a で v_i が *head* ノードであるとき，C_b では v_i は *child* ノードになることはない
(2) C_a で v_i が *child* ノードであり，v_i から距離 k 以内に存在する *head* ノード v_j がビザンチンノードから $k+1$ から $2k$ の場所に存在するとき，C_b で v_i は *head* ノードになることはない
(3) C_a で v_i が *child* ノードであり，v_i から距離 k 以内に存在する *head* ノード v_j がビザンチンノードから $2k+1$ 以上の場所に存在するとき，C_b で v_i は *head* ノードになることはない

(1) について考える．仮定より C_a にて I_{2k} が成り立つため，v_i の *head* ノード情報は v_i から距離 k 以内の任意のノード v_j が保持しており，また v_j が出した *head* ノード情報はグラフ上に存在しない．v_j は v_i の *head* ノード情報を保持しているため，v_j は *head* ノードになることはない．よって C_b において v_i から k 以内で新たに *head* ノードとなるノードは存在せず，v_i に到達する *head* ノード情報は存在しないため v_i は *child* ノードになることはなく，(1) は成り立つ．

(2) について考える．仮定より C_a にて I_{2k} が成り立つため，v_j の *head* ノード情報は v_j から距離 k 以内の任意のノード v_l が保持しており，また v_l が出した *head* ノード情報はグラフ上に存在しない．また，v_l は *head* ノード情報を保持しているため *head* ノードになることはなく，新たに *head* ノード情報を作成することもない．加えて，v_j はビザンチンノードから少なくとも $k+1$ 離れており，ビザンチンノードからの虚偽の情報も到達しない．そのため v_j は他ノードの *head* ノード情報を保持することはないため *child* ノードになることはなく，v_i は常に k 内に *head* ノードを有することになる．よって (2) は成り立つ．

(3) について考える．仮定より C_a にて I_{2k} が成り立つため，v_j の *head* ノード情報は v_j から距離 k 以内の任意のノード v_l が保持しており，また v_l が出した *head* ノード情報はグラフ上に存在しない．また，v_l は *head* ノード情報を保持しているため *head* ノードになることはなく，新たに *head* ノード情報を作成することもない．そのため v_j は他ノードの *head* ノード情報を保持することはないため，*child* ノードになることはない．よって，v_i は常に k 内に *head* ノードを有することになり，(3) は成り立つ．

以上 3 つの事象により，状況 C_a において I_{2k} を満たすとき，任意の状況 $C_b(a \leq b)$ ではビザンチンノードから $2k+1$ 以上離れたノードは出力変数 $(v_i.state)$ を変更することはない．よって題意は成り立つ． □

補題 1 より，状況 C_a で I_{2k} を満たすことができれば距離 k 極大独立集合問題で与えられる仕様を故障ノードから $2k+1$ 以上離れた全てのノードが任意の $j \leq i$ において状況 C_j でも満たせることを示した．

3.4 シミュレーションによる結果

ビザンチンノードが存在しないとき，任意の状況から I_{2k} を満たすまでの非同期ラウンド数を計測するため，本稿では k の値と n の値をパラメータとしシミュレーションを行った．シミュレーションの結果を図 1 に示す．縦軸が I_{2k} を満たすまでにかかった非同期ラウンド数の平均値，横軸がノード数である．また，青線が $k=1$ のとき，オレンジが $k=2$ のとき，グレーが $k=3$ のとき，黄色が $k=4$ のときである．シミュレーションの条件はノード数 n で任意のグラフを作成し，そのグラフにおいてアルゴリズムを実行する．ビザンチンノードは存在しないものとし，全ノードが正常ノードとしてシミュレーションを行った．作成されるグラフ

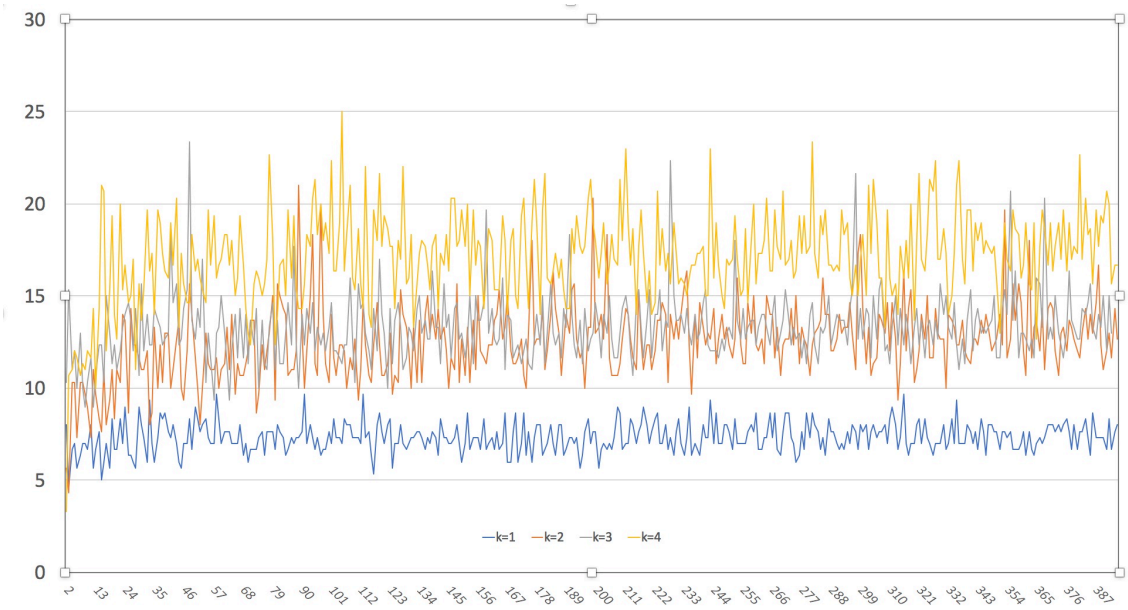


図 1：$k = 1$ から $k = 4$ までのシミュレーション結果

ではシミュレーションの処理速度の関係上，グラフの最大次数を 5 としている．各 n の値で 3 度シミュレーションを行い，各非同期ラウンド数の平均値を計算している．

結果として I_{2k} を満たすまでの非同期ラウンド数は n の値に影響されず，k の値に大きく影響されることが分かった．各 k の値における非同期ラウンド数の平均は $k = 1$ のときで 7.36 ラウンド，$k = 2$ で 13.0 ラウンド，$k = 3$ で 13.4 ラウンド，$k = 4$ で 17.1 ラウンドであった．ビザンチンノードが存在しないとき，シミュレーションの結果より収束性は満たされていると考えられる．

4　まとめ

本稿では距離 k 極大独立集合問題を解く自己安定乱択アルゴリズムを提案した．アルゴリズムでは乱数を用いており，ビザンチンノードが存在する分散システム上で距離 k 極大独立集合を求めることができる．シミュレーションにてビザンチンノードが存在しない分散システムでアルゴリズムが安定するまで，すなわち条件式 I_{2k} を満たすまでに必要な非同期ラウンド数を求め，証明にて I_{2k} を満たした後はビザンチンノードから $2k + 1$ 以上離れたノードは距離 k 極大独立集合問題を解けていることを示した．

今後の課題としては任意の状況から I_{2k} を満たすことのできる確率及び非同期ラウンド数の証明を行うことである．シミュレーション結果から k の値に依存した非同期ラウンド数で可能であると考えているが，ビザンチンノードの影響を考える必要がある．

参考文献

[1] Dubois, S., Tixeuil, S., Zhu, N.: The byzantine brides problem. In: International Conference on Fun with Algorithms. pp. 107–118. Springer (2012)

[2] Masuzawa, T., Tixeuil, S.: A self-stabilizing link-coloring protocol resilient to unbounded byzantine

faults in arbitrary networks. In: International Conference On Principles Of Distributed Systems. pp. 118–129. Springer (2005)

[3] Maurer, A., Tixeuil, S.: Self-stabilizing byzantine broadcast. In: 2014 IEEE 33rd International Symposium on Reliable Distributed Systems. pp. 152–160. IEEE (2014)

不完全情報二人単貧民

木谷 裕紀[1,a)]　大渡 勝己[,b)]　小野 廣隆[1,c)]

Deciding the winning player in two-player TANHINMIN and its variant

KIYA Hironori[1,a)]　OHTO Katsuki[,b)]　ONO Hirotaka[1,c)]

1. はじめに

大貧民はトランプカードゲームの中でも全国的に認知度, 人気が高い遊びであり, 大富豪とも呼ばれる. このゲームは不完全情報多人数ゲームであり, 一般には最適戦略が存在しない, あるいは求めることが困難である. 近年, 将棋やオセロなどの完全情報ゲームに関する研究と共に, 大貧民のような不完全情報ゲームの研究も進んでいる. 2006 年度から毎年コンピューター大貧民大会が電気通信大学で開催され, 「強い」大貧民 AI の開発を競う場となっている. 2015 年, プロ棋士に対する勝利宣言が出された将棋コンテスト同様, 日々コンピューター大貧民 AI の実力も向上しているがまだまだ成長の余地がある [1], [2], [5].

このような大貧民研究の一環として電気通信大学の西野は単貧民というゲームを定義した [6]. 単貧民は大貧民に

- 特殊ルールが一切存在しない,
- 1 枚出しのみを認める,
- 手札は公開で行われる,

という制約を課したものであり, 大貧民を単純化し, かつ完全情報化したものと言える. 二人単貧民は完全情報型の 2 人ゲームとなるため, いずれかのプレイヤーに必勝戦略が常に存在する. しかし, 将棋や囲碁などを考えると明らかなように, 完全情報型の 2 人ゲームにおける必勝戦略の存在性とそれを具体的に知ることには大きな隔たりがある. これに対し著者らはこれまで特殊ルールまで拡張した単貧民においての具体的な必勝判定及び必勝戦略を与える研究を行ってきた [4].

本研究では, 単貧民をより通常の大貧民に近づけるために単貧民ゲームに**不完全情報性**を導入した**不完全情報単貧民**, つまり基本的にはお互いの手札が全く分からない単貧民の必勝判定を考える. 不完全情報二人ゲームに関する解析はいくつか知られているが [3], 確立した解析手法はまだ知られていない. まず, このゲームにおいて常には最善手が分からないことを示す. また, 単貧民において, 完全情報性がどのくらいゲームを簡単にしているかを確かめるためにオラクルを導入する. オラクルが与えられたとき, 各プレイヤは相手の手札を部分的な情報をオラクルによって得られるとする (例えば, 全ての相手の手札情報がオラクルによって与えられる単貧民が完全情報単貧民である).

本研究の目的はどのような情報を与えるオラクルがあれば必勝判定が可能かあるいは完全情報単貧民の必勝判定における本質となるものは何かを考えるものである.

2. 問題

2.1 単貧民のルール

まず本研究の基礎となる二人単貧民を以下のようにモデル化する: 各プレイヤーの**手札集合**を $[N] = \{1, 2, \ldots, n\}$ の部分集合の形で与える. プレイヤーに配布された札 (**手札**と呼ぶ) の札数は必ずしも等しくなくてよい. 札の番号は強さを表し, 大きいほど強いものとする. この設定の下, 以下の形でゲームを進める.

[1] 名古屋大学大学院情報学研究科 数理情報学専攻
Department of Mathematical Informatics, Graduate School of Informatics, Nagoya University
[a)] kiya.hironori@gmail.com
[b)] katsuki.ohto@gmail.com
[c)] ono@i.nagoya-u.ac.jp

- 先手後手を決め, **先手プレイヤー**, **後手プレイヤー**の順に交代で, 手札から**場**に 1 枚ずつ札を出していく.
- 場は最初, 空である (0 の札がおかれていると考える).
- 順番のプレイヤーは, 手札の中から場の札の値よりも大きい値の札を 1 枚出すことができる. 出した札はそれまで出ていた札の上に置かれる (場に出ている札は今出した札に代わる). このとき, 順番はもう一人のプレイヤーに移る.
- 順番のプレイヤーは, 手札を出さずに順番をもう一人のプレイヤーに譲ることができる (**パス**する, という). このとき場に出ている札は空になる (改めて, 0 の札がおかれる).
- いずれかのプレイヤーの手札がなくなった時点で終了であり, このとき手札を 0 枚にしたほうが勝ちである.

ゲームを通して順番は交互に移るが, それぞれの手札を出すタイミングを**手番**, ある時点で順番が来ているプレイヤーを**手番プレイヤー**, もう一人のプレイヤーを**相手プレイヤー**と呼ぶ. また場が空の状態からいずれかのプレイヤーがパスをするまでを**巡**と呼ぶ.

2.2 諸定義

P_0 を初期手番プレイヤー, P_1 を初期非手番プレイヤーとする. 初期手番プレイヤー (以下では単に先手と呼ぶ) の手札集合を $X_0 \subseteq [N]$, 初期非手番プレイヤーの手札集合を $X_1 \subseteq [N]$ とする. また場に最後に出された札を r とする. まだ札が出されていない空場では仮想的に 0 の札が置かれているものとする. このとき, 任意の手番に対して, 以下のような二部グラフを定義する: $G_0 = (X_0, X_1 \cup \{r\}, E_0)$, ただし, $E_0 = \{\{i,j\} \mid i \in X_0, j \in X_1 \cup \{r\}, i > j\}$. つまり, G_0 は, プレイヤー P_0 の全ての手札と場の札の各札を点としたグラフであり, プレイヤー P_0 の各手札から, その札が勝てるプレイヤー P_1 の手札へ辺を引く形で定義されている.

このように定義したグラフ G_0 においてマッチング辺は, プレイヤー P_1 が出した札に対してプレイヤー P_0 が札を出す関係を表しており, ゲームが進むことはそれぞれのグラフにおいてマッチング辺が抜かれていく状況を表している. 本研究で提案する判定法ではこれを利用するため, G_0 の最大マッチングのサイズ $\mu_{(G_0)}$ を定義する. また, 後に示す定理において単貧民における必勝戦略保持者は P_1 の最弱札を除いたグラフにおける最大マッチングのサイズにも影響されることを示していく. 従って P_1 の最弱札を除いたグラフにおける最大マッチングのサイズを $\mu(G_0 \setminus \{\min X_1)\}$ と表記する. ただし, $\{\min X_1)\}$ は $|P_1|$ の手札集合で最も弱い札 (そのような札が複数枚あればその中から一枚) である.

また同様に以下のような二部グラフを定義する: $G_1 = (X_1, X_0, E_1)$, ただし, $E_1 = \{\{i,j\} \mid i \in X_1, j \in X_0, i > j\}$. つまり, G_1 は, プレイヤー P_1 の全ての手札と P_0 の最弱札を除く全ての札と場の札の各札を点としたグラフであり, プレイヤー P_0 の各手札から, その札が勝てるプレイヤー P_1 の手札へ辺を引く形で定義されている. 同様に G_1 の最大マッチングのサイズ $\mu_{(G_1)}$, $\mu(G_1 \setminus \{\min X_0)\}$ を定義する. ただし, $\{\min X_0)\}$ は $|P_0|$ の手札集合で最も弱い札 (そのような札が複数枚あればその中から一枚) である.

本稿での証明には, この単貧民ゲームにおいて以下の補題が成立することを利用する.

補題 1. 単貧民ゲームにおいて各プレイヤのソートされた手札集合が入力として与えられたとき, 最大マッチング $\mu_{(G_0)}$, $\mu_{(G_1)}$ は線形時間で計算可能.

この補題は以下の貪欲アルゴリズムが $\mu_{(G_0)}$ と同値の値を出力することを保証することから示すことができる. 本稿では詳細な証明は省略する.

Algorithm 1 貪欲アルゴリズム

1: $G_0 = (X_0, X_1, E_0)$ とする. ただし $X_0 = \{x_1, x_2, \ldots, x_{n_0}\}$, $x_1 < x_2 < \cdots < x_{n_0}$, $X_1 = \{y_1, y_2, \ldots, y_{n_1}\}$, $y_1 < y_2 < \cdots < y_{n_1}$ とする. 初期値として $M := \emptyset$, $i := 1, j := 1$ とする.
2: $i > n_0$ または $j > n_1$ が成立しているならば M を出力しアルゴリズムを停止する. $i > n_0$ と $j > n_1$ のどちらもが成立していないとき, $x_i > y_j$ が成立しているか確かめる. 成立している場合ステップ 3 へ移行する. 成立していない場合, $i := i + 1$ へ値を更新し, ステップ 2 の最初へ戻る.
3: $M := M \cup \{(x_i, y_j)\}$, $i := i + 1$, $j := j + 1$ へ値を更新する. ステップ 2 へ戻る.

また以下の補題も成立する.

補題 2. $x_i > x_j$, $x_i \in X_0$, $x_j \in X_1$ を満たす x_i, x_j が存在するとき, 任意の x_j に対して, $x_i > x_j$, $x_i \in X_0$, $x_j \in X_1$ を満たす最小の ${x_i}^*$ をとることができ, $G_0 = (X_0, X_1, E_0)$ において, 辺 $({x_i}^*, x_j)$ を含む最大マッチングが存在する.

この補題から相手の出した札に対して出すことのできる最小の札を場に出すことによるグラフの最大マッチング数の減少は高々 1 であることが導かれる. この証明も本稿では省略する.

また, 本研究ではオラクルをプレイヤが受け取ることにより必勝判定可能な局面を示す. $\mu_{(G_0 \setminus \{\min X_1\})}$ を受け取ることができるオラクルを μ_{G_0} オラクル, $\mu_{(G_1 \setminus \{\min X_0\})}$ を受け取ることができるオラクルを μ_{G_1} オラクル, プレイヤの手札集合のサイズを受け取ることができるオラクルを $|X_i|(i = 0, 1)$ オラクルとする.

3. 不完全情報単貧民

不完全情報で行う単貧民に対して, まず, 以下の定理が成立する.

定理 1. P_0 の手番において, $\mu_{(G_0 \setminus \{\min X_1\})} > \mu_{(G_1)}$ が成立するとき, P_0 は必勝戦略を持つ (1). $\mu_{(G_0)} \leq \mu_{(G_1 \setminus \{\min X_0\})}$ が成立するとき, P_1 は必勝戦略を持つ (2).

証明. 常に場の札に勝つことができる最小の札を出し続けるという戦略が上記の条件を満たすときの必勝戦略であることを $\mu_{(G_0\setminus\{\min X_1\})}, \mu_{(G_1\setminus\{\min X_0\})}$ に関する帰納法を用いて示す.

基礎ステップ

(1)P_0 の手番において $\mu_{(G_0\setminus\{\min X_1\})} > \mu_{(G_1)} = 0$ が成立するとき, P_0 が場の札に対して勝つ札があるかどうかで場合分けを行う. なお, このとき $\mu_{(G_0\setminus\{\min X_1\})} > 0$ より, $X_1 \setminus \{\min X_1\} > 0$ が成立するため, $|X_1| > 1$ が成立することに留意したい.

(1-1)P_0 が場の札に対して勝つ札があるとき, P_0 は勝てる札の中から強さ最小の札をだすことができる. $\mu_{(G_1)} = 0$ が成立しているため, 札を出した直後の P_1 の手番において, P_1 は場の札に対して勝てる札を持っていない. 従って, P_1 はパスを選択し, 空場において再び P_0 の手番となる. P_1 の手札内には P_0 の手札に対して勝てる札が一枚もないため, P_0 はゲーム終了まで最小の札を出し続け, 勝つことができる. 従ってこの場合, P_0 が必勝戦略を持つ.

(1-2)P_0 が場の札に対して勝つ札がないとき, P_0 はパスを選択し, 場の札は空になり P_1 の手番になる. $|X_1| > 1$ が成立するため, P_1 は 2 枚以上の手札から選んで手札を出すことになる. 一方で $\mu_{(G_0\setminus\{\min X_1\})}$ が成立しており且つ P_0 が場の札に対して勝つ札がなかったことから, P_1 の手札内には少なくとも 2 枚は P_0 の強さ最大の札よりも弱い札が存在する. これらから, P_1 が空場に出す札に対し, 出せない札であればパスを出せる札になれば札を出すことによって P_1 が勝利条件を満たす前に P_0 は札を出すことができる. また, このとき, $\mu_{(G_1)} = 0$ が成立しているため, 札を出した直後の P_1 の手番において, P_1 は場の札に対して勝てる札を持っていない. 従って, P_1 はパスを選択し, 空場において再び P_0 の手番となる. P_1 の手札内には P_0 の手札に対して勝てる札が一枚もないため, P_0 はゲーム終了まで最小の札を出し続け, 勝つことができる. 従ってこの場合も, P_0 が必勝戦略を持つ.

従ってこの局面は P_0 が必勝戦略を持つ.

(2)P_0 の手番において, $\mu_{(G_0)} \leq \mu_{(G_1\setminus\{\min X_0\})} = 0$ が成立するとき, $\mu_{(G_0)} = 0$ が成立する. したがって初期の場の札及び任意の P_1 の手札に対して P_0 の手札の中にはそれより真に強い札はない。従って P_1 が場の札に勝つことができる最小の札を出し続ける戦略を続けることによって P_0 は札を一枚も場に出すことができない. 従って $\mu_{(G_0)} \leq \mu_{(G_1\setminus\{\min X_0\})} = 0$ が成立する局面は P_1 が必勝戦略を持つ.

帰納ステップ

P_0 の手番において, $k-1 \geq \mu_{(G_0\setminus\{\min X_1\})} > \mu_{(G_1)}$ が成立するとき (k は自然数), P_0 は必勝戦略を持ち, $\mu_{(G_0)} \leq \mu_{(G_1\setminus\{\min X_0\})} \leq k-1$ が成立するとき, 必勝戦略を持つと仮定する.

(1)P_0 の手番において $k = \mu_{(G_0\setminus\{\min X_1\})} > \mu_{(G_1)}$ が成立するとき, 場の札よりも強い札が手札にある場合とない場合で場合分けを行う.

(1-1)P_0 に手札に場の札よりも強い札がある場合, P_0 は出すことができる最も弱い札を P_0 は場に出す. このとき, 手番は P_1 に移り, 補題 2 より $\mu_{(G_0\setminus\{\min X_1\})}$ は 1 だけ減少するため, $\mu_{(G_1)} \leq \mu_{(G_0\setminus\{\min X_1\})} \leq k-1$ が成立する盤面になる. 帰納法の仮定よりこの局面は P_0 が必勝戦略を持つ局面である.

(1-2) 手札に場の札よりも強い札が手札にない場合. P_0 はパスを選択せざるを得ない. パスした直後の空場の盤面に対し, P_1 は何かしらの札を出すことになる. このとき, P_1 の出した札が P_0 のすべての手札よりも強い (P_0 のどの手札も勝つことができない札) かどうかでさらに場合分けを行う.

(1-2-a)P_0 のすべての手札よりも強い (P_0 のどの手札も勝つことができない札) が場に出た場合, P_0 はパスを選択するため, $\mu_{(G_0\setminus\{\min X_1\})}$ は減少せず, $\mu_{(G_0\setminus\{\min X_1\})}$=k で再び P_1 が空場で手番となる. このとき $\mu_{(G_0\setminus\{\min X_1\})}$=k が成立しているため, このループは高々 (プレイヤ 1 の手札の総数-k) 回しかおきない.

(1-2-b)P_0 がその札よりも強い札を保持している場合, P_0 は出すことができる最も弱い札を P_0 は場に出す. このとき, 手番は P_1 に移り, 補題 2 より $\mu_{(G_0\setminus\{\min X_1\})}$ は 1 だけ減少するため, $\mu_{(G_1)} \leq \mu_{(G_0\setminus\{\min X_1\})} \leq k-1$ が成立する盤面になる. 帰納法の仮定よりこの局面は P_0 が必勝戦略を持つ局面である.

従っていずれの場合も P_0 が必勝戦略を持つ.

(2)P_0 の手番において $\mu_{(G_0)} \leq \mu_{(G_1\setminus\{\min X_0\})} = k$ が成立しているとき, P_0 がパスを選択するか否かで場合分けを行う.

(2-1)P_1 がパスを選択しない場合, 直後の盤面は, P_1 の手番となり, $k = \mu_{(G_1\setminus\{\min X_0\})} > \mu_{(G_0)}$ が (補題 2 より $\mu_{(G_0)}$ が 1 減少するため) 成立する. 従ってこの局面は P_1 が必勝戦略を持つ.

(2-2)P_0 がパスを選択する場合, パスを選択した直後の盤面は, P_1 の手番となる. パスを選択した直後の P_1 の手番において, P_1 の手札の枚数 $|X_1|$ と $\mu_{(G_1\setminus\{\min X_0\})}$ が一致しているか否かでさらに場合分けを行う.

(2-2-a)P_1 の手札の枚数 $|X_1|$ と $\mu_{(G_1\setminus\{\min X_0\})}$ が一致しているとき, P_1 の手札の枚数 $|X_1|$ と $\mu_{(G_1\setminus\{\min X_0\})}$ が一致しているため, $\mu_{(G_0)}$ は高々 $|X_1| = \mu_{(G_1\setminus\{\min X_0\})}$ である. このとき, $|P_1|$ は最小札を出すことによって再び $\mu_{(G_0)} \leq \mu_{(G_1\setminus\{\min X_0\})} \leq k$ が成立する局面となる. このとき, さらに場合分けが生じる.

(2-2-a') 再び P_0 がパスを選択する場合, 再び $|P_1|$ は最小札を出すことによって, $\mu_{(G_0)} \leq \mu_{(G_1\setminus\{\min X_0\})} \leq k-1$ が成立するため, 後手必勝である.

(2-2-a") また, 再び P_0 が場に何かしらの札を出す場合もその札に勝てる最小の札を出す (できないときはパスをする) ことによって $\mu_{(G_0)} \leq \mu_{(G_1 \setminus \{\min X_0\})} \leq k-1$ が成立する盤面になるため, この局面は P_1 が必勝戦略を持つ.

したがって P_0 がパスを選択する場合 P_1 が必勝戦略を持つ.

(2-2-b)P_1 の手札の枚数 $|X_1|$ と $\mu_{(G_1 \setminus \{\min X_0\})}$ が一致していないとき, 空場に対して強さ最小の札を出すと, $\mu_{(G_0)}$ の値は増加せず, $\mu_{(G_1 \setminus \{\min X_0\})}$ の値も減少しない. 従って, 再び, P_0 の手番となり ,$\mu_{(G_0)} \leq \mu_{(G_1 \setminus \{\min X_0\})} \leq k$ が成立する盤面となる. この局面は高々 P_1 の手札の総数-$\mu_{(G_0)}$ 回しか起きず, (2-1), あるいは (2-2-a) の状況へ移行することになる. 従ってこの局面も P_1 が必勝戦略を持つ.

以上より帰納的に題意は示された. □

定理 2. *$\mu_{(G_0 \setminus \{\min X_1\})} \leq \mu_{(G_1)}$ 且つ, $\mu_{(G_0)} > \mu_{G_1 \setminus \{\min X_0\}}$ のとき, P_0, P_1 は必勝戦略を持たない局面が存在する.*

証明. 具体的な局面を記述する. 場を空場とし, P_0 の手番とする. P_0 の初期手札集合を {2, 3} とする. このとき $\mu_{(G_0 \setminus \{\min X_1\})} \leq \mu_{(G_1)}$ 且つ, $\mu_{(G_0)} > \mu_{G_1 \setminus \{\min X_0\}}$ を満たす局面として, P_1 の初期手札集合 $X_1 = \{1,2,5\}$, $\{2,2,3\}$ が考えられる.

$X_1 = \{1,2,5\}$ のとき, P_0 は 2 を出すことによって, P_1 の対応手に関わらず勝つことができる一方, 3 を出した場合その対応手として, P_1 が 5 を出すと P_0 はパスをせざるを得ない局面になるが, その後 P_1 が 2 を出すことによって再び P_0 はパスをせざるを得ないため, その後 P_1 が 1 を出して P_1 の勝ちとなる. 従ってこの場合, P_0 にとって, 2 を出すことが最善手である.

$X_1 = \{2,3,3\}$ のとき, P_0 は 3 を出すことによって, P_1 の対応手に関わらず勝つことができる一方, 2 を出した場合その対応手として, P_1 が 3 を出すと P_0 はパスをせざるを得ない局面になるが, その後 P_1 が 3 を出すことによって再び P_0 はパスをせざるを得ないため, その後 P_1 が 2 を出して P_1 の勝ちとなる. 従ってこの場合, P_0 にとって, 3 を出すことが最善手である.

上記 2 局面は区別不可能であるため, この局面は P_0 は必勝戦略を持たない. 一方で P_1 が正しい応手をすることによって P_1 の応手関係なく負けてしまうため. P_1 も必勝戦略を持たない.

以上より題意は示された. □

この定理は, 完全情報で行う場合は最善手がわかる局面においても [4], 不完全情報で行う場合はその不完全情報性からそのような戦略が分からないため, 必勝でない局面が現れることを意味している.

また, オラクルが与えられる不完全情報単貧民において以下の定理が成立する.

定理 3. *不完全情報単貧民において, ゲーム開始時に P_0 が μ_1 オラクルを得るならば, ゲーム開始時に $\mu_{(G_0 \setminus \{\min X_1\})} > \mu_{(G_1 \setminus \{\min X_0\})}$ のとき, P_0 は必勝戦略を持つ. ゲーム開始時に P_1 が μ_0 オラクルを得るならば, ゲーム開始時に $\mu_{(G_0 \setminus \{\min X_1\})} \leq \mu_{(G_1 \setminus \{\min X_0\})}$ のとき, P_1 は必勝戦略を持つ.*

証明. 得られたオラクルの値をゲームの進行に従って計算した値が 0 のとき, 手札の最も弱い札をださないように札を出し, それ以外の局面においては, 最小の札を出す戦略が必勝戦略になっていることを示す. P_0 が札を出す毎に $\mu_{(G_0 \setminus \{\min X_1\})}$ が減少することを利用して μ_0 オラクル, μ_1 オラクルに関する帰納法を行う.

基礎ステップ

(1)P_0 の手番において $\mu_{(G_0 \setminus \{\min X_1\})} > \mu_{(G_1 \setminus \{\min X_0\})} = 0$ が成立するとき, P_0 が場の札に対して勝つを手札に保持しているか否かによってさらに場合分けを行う.

(1-1)P_0 が場の札に対して勝つ札があるとき, 手札が残り 1 枚であるかどうかによってさらに場合分けを行う.

(1-1-a) 手札が残り一枚であるとき,

その札を出すことによって勝利することができるのでこの局面は P_0 が必勝戦略を持つ.

(1-1-b) 手札が残り 2 枚であるとき,

強さ最小の札を除く札から P_0 は勝てる札の中から (強さ最小の札が勝てる場の札に対してはほかの札も勝てるため) だすことができる. 札を出した直後の P_1 の手番において, $\mu_{(G_1 \setminus \{\min X_0\})} = 0$ が成立するので, P_1 は場の札に対して勝てる札を持っていない. 従って, P_1 はパスを選択し, 空場において再び P_0 の手番となる. P_1 の手札内には P_0 の強さ最小の札以外の手札に対して勝てる札が一枚もないため, P_0 はゲーム終了まで最小の札を出し続け, 勝つことができる. 従ってこの場合も, P_0 が必勝戦略を持つ. 最後に手札残り 1 枚になったときその札を出すことにより P_0 が勝つことができる. 従ってこの局面は P_0 が必勝戦略を持つ.

(1-2)P_0 が場の札に対して勝つ札がないとき, P_0 はパスを選択し, 場の札は空になり P_1 の手番になる. P_0 が場の札に対して勝つ札がないとき, P_0 はパスを選択し, 場の札は空になり P_1 の手番になる. $|X_1| > 1$ が成立するため, P_1 は 2 枚以上の手札から選んで手札を出すことになる. 一方で $\mu_{(G_0 \setminus \{\min X_1\})}$ が成立しており且つ P_0 が場の札に対して勝つ札がなかったことから, P_1 の手札内には少なくとも 2 枚は P_0 の強さ最大の札よりも弱い札が存在する. これらから, P_1 が空場に出す札に対し, 出せない札であればパスを出せる札になれば札を出すことによって P_1 が勝利条件を満たす前に P_0 は札を出すことができる. また, このとき, $\mu_{(G_1)} = 0$ が成立しているため, 札を出した直後の P_1 の手番において, P_1 は場の札に対して勝てる札を持っていない. 従って, P_1 はパスを選択し, 空場において再び P_0 の

手番となる. P_1 の手札内には P_0 の手札に対して強さ最小の札を除くと勝てる札が一枚もないため, P_0 はゲーム終了まで強さ最小から 2 番目の札を出し続け, 勝つことができる. 従ってこの場合, P_0 が必勝戦略を持つ.

(2)P_0 の手番において, $\mu_{(G_0\setminus\{\min X_1\})} \leq \mu_{(G_1\setminus\{\min X_0\})} = 0$ が成立するとき, $\mu_{(G_0)} = 0$ が成立する. したがって初期の場の札及び強さ最小札を除く P_1 の手札に対して P_0 の手札の中にはそれより真に強い札はない. 従って P_1 は強さ最小の札以外を出し続けることによって P_0 は札を一枚も場に出すことができない. 従って $\mu_{(G_0\setminus\{\min X_1\})} \leq \mu_{(G_1\setminus\{\min X_0\})} = 0$ が成立する局面も P_1 が必勝戦略を持つ.

帰納ステップ

P_0 の手番において, $k-1 \geq \mu_{(G_0\setminus\{\min X_1\})} > \mu_{(G_1\setminus\{\min X_0\})}$ が成立するとき (k は自然数), P_0 は必勝戦略を持ち, $\mu_{(G_0\setminus\{\min X_1\})} \leq \mu_{(G_1\setminus\{\min X_0\})} \leq k-1$ が成立するとき, 必勝戦略を持つと仮定する.

(1)P_0 の手番において $\mu_{(G_0\setminus\{\min X_1\})} > \mu_{(G_1)} = k$ が成立するとき, 場の札よりも強い札が手札にある場合とない場合で場合分けを行う.

(1-1)P_0 に手札に場の札よりも強い札がある場合, P_0 は出すことができる最も弱い札を P_0 は場に出す. このとき, 手番は P_1 に移り, 補題 2 より $\mu_{(G_0\setminus\{\min X_1\})}$ は 1 だけ減少するため, $\mu_{(G_0\setminus\{\min X_1\})} \leq \mu_{(G_1\setminus\{\min X_0\})} \leq k-1$ が成立する盤面になる. 帰納法の仮定よりこの局面は P_0 が必勝戦略を持つ局面である.

(1-2) 手札に場の札よりも強い札が手札にない場合. P_0 はパスを選択せざるを得ない. パスした直後の空場の盤面に対し, P_1 は何かしらの札を出すことになる. このとき, P_1 の出した札が P_0 のすべての手札よりも強い (P_0 のどの手札も勝つことができない札) かどうかでさらに場合分けを行う.

(1-2-a)P_0 のすべての手札よりも強い (P_0 のどの手札も勝つことができない札) が場に出た場合, P_0 はパスを選択するため, $\mu_{(G_0\setminus\{\min X_1\})}$ は減少せず, $\mu_{(G_0\setminus\{\min X_1\})}$=k で再び P_1 が空場で手番となる. このとき $\mu_{(G_0\setminus\{\min X_1\})}$=k が成立しているため, このループは高々 (プレイヤ 1 の手札の総数-k) 回しかおきない.

(1-2-b)P_0 がその札よりも強い札を保持している場合, P_0 は出すことができる最も弱い札を P_0 は場に出す. このとき, 手番は P_1 に移り, 補題 2 より $\mu_{(G_0\setminus\{\min X_1\})}$ は 1 だけ減少するため, $\mu_{(G_1\setminus\{\min X_0\})} \leq \mu_{(G_0\setminus\{\min X_1\})} \leq k-1$ が成立する盤面になる. 帰納法の仮定よりこの局面は P_0 が必勝戦略を持つ局面である.

従っていずれの場合も P_0 が必勝戦略を持つ.

(2)P_0 の手番において $\mu_{(G_0\setminus\{\min X_1\})} \leq \mu_{(G_1\setminus\{\min X_0\})} \leq k$ が成立するとき, P_0 がパスを選択するか否かで場合分けを行う.

(2-1)P_1 がパスを選択しない場合, 直後の盤面は, P_1 の手番となり, $k \geq \mu_{(G_1\setminus\{\min X_0\})} > \mu_{(G_0\setminus\{\min X_1\})}$ が成立する. 従ってこの局面は P_1 が必勝戦略を持つ.

(2-2)P_0 がパスを選択する場合, パスを選択した直後の盤面は, P_1 の手番となる. このとき, P_1 の手札の枚数 $|X_1|$ と $\mu_{(G_1\setminus\{\min X_0\})}$ が一致しているか否かでさらに場合分けを行う.

(2-2-a)P_1 の手札の枚数 $|X_1|$ と $\mu_{(G_1\setminus\{\min X_0\})}$ が一致しているとき, P_1 の手札の枚数 $|X_1|$ と $\mu_{(G_1\setminus\{\min X_0\})}$ が一致しているため, $\mu_{(G_0\setminus\{\min X_1\})}$ は高々 $|X_1|-1 = \mu_{(G_1\setminus\{\min X_0\})} - 1$ である. $k \geq \mu_{(G_1\setminus\{\min X_0\})} > \mu_{(G_0\setminus\{\min X_1\})}$ が成立する. 従ってこの局面は P_1 が必勝戦略を持つ.

(2-2-b)P_1 の手札の枚数 $|X_1|$ と $\mu_{(G_1\setminus\{\min X_0\})}$ が一致していないとき , 空場に対して強さ最小の札を出すと, $\mu_{(G_0\setminus\{\min X_1\})}$ の値は増加せず, $\mu_{(G_1\setminus\{\min X_0\})}$ の値も減少しない. 従って, 再び, P_0 の手番となり, $\mu_{(G_0\setminus\{\min X_1\})} \leq \mu_{(G_1\setminus\{\min X_0\})} \leq k$ が成立する盤面となる. この局面は高々 P_1 の手札の総数-$\mu_{(G_0\setminus\{\min X_1\})}$ 回しか起きず, (2-1), あるいは (2-2-a) の状況へ移行することになる. 従ってこの局面も P_1 が必勝戦略を持つ.

従っていずれの場合も P_1 が必勝戦略を持つ.

以上より題意は示された. □

定理 4. 不完全情報単貧民において, ゲーム開始時に各プレイヤが $\mu_i(i=0,1)$ オラクル, $|X_i|$ オラクルの両方を得ても, 各プレイヤは最善手を一意に決定できない.

証明. 具体的な区別できない局面を示す. 初期盤面を $X_1=\{2,3\}$ 場札を空とする. また, このとき与えられるオラクルを $\mu_{(G_0\setminus\{\min X_1\})}=2$, $\mu_{(G_1\setminus\{\min X_0\})}=1$, $|X_0|=5$ とする. このゲームにおいて以下の進行が与えられたとする.

1. P_0 が場に 4 を出す.
2. P_1 が出す札がないため, パスを選択する.
3. P_1 が 1 を出す.

この局面 (P_1 が 1 を出した局面) において相手の初期手札集合として考えられる集合として, $X_0=\{1,2,3,3,4\}$ という集合が考えられる. このとき, P_1 が 1 を出した局面において X_0 の手札は $X_0=\{2,3,3\}$ となり, P_0 は 3 を出すことによってのみ勝つことができる. 一方で, この局面において相手の初期手札集合として考えられる集合として, $X_0=\{1,2,2,4,5\}$ という集合も考えられる. このとき, P_0 は 2 を出すことによってのみ勝つことができる.

このように P_0 は相手の手札集合によって出す札を適切に選択することで勝利することができるがどちらの手が最善手であるかをオラクル情報だけでは区別不可能である. 従って題意は示された.

□

この定理 3 は相手側から見たマッチングと自分の手札の初期状況のみで初期局面が必勝戦略を持つならばその最善手を遂行可能であることを意味している. 一方で定理 4 は初期局面に必勝戦略を持っているプレイヤがミスをすることにより, 最善手を指せば逆転できるような局面において, そのような最善手を一意に発見することはできないことを示している. また, 定理 3 から容易に次の系を示すことができる.

系 1. 毎手番開始時に μ オラクルが与えられる不完全情報単貧民において, $\mu_{(G_0)} > \mu_{(G_1)}$ のとき, P_0 は必勝戦略を持つ. $\mu_{(G_0)} \leq \mu_{(G_1)}$ のとき, P_1 は必勝戦略を持つ.

この系は毎手番開始時にオラクルを受け取ることで相手プレイヤのゲーム中のミスにつけ込むことができることを意味している.

参考文献

[1] K. Ayabe, S. Okubo, T. Nishino: Cluster analysis using N-gram statistics for daihinmin programs and performance evaluations. International Journal of Software Innovation, 4(2), 33–57, 2016

[2] M. Konishi, S. Okubo, T. Nishino, M. Wakatsuki: A Decision Tree Analysis of a Multi-Player Card Game With Imperfect Information. International Journal of Software Innovation (IJSI), 6(3), 1-17.

[3] 川上直人, 橋本剛: 完全情報ゲームの探索を用いたガイスター AI の研究. ゲームプログラミングワークショップ 2018 論文集, 2018, 35-42.

[4] 木谷裕紀, 大渡勝己, 小野廣隆: ”8 切りルールを含む二人単貧民の必勝判定問題.” 研究報告ゲーム情報学 (GI) 2018.3 (2018): 1-5.

[5] 土橋康希, 大久保誠也, 若月光夫, 西野哲朗: 大貧民におけるモンテカルロ法の報酬値に関する研究. 研究報告ゲーム情報学 (GI), 2019(18), 1-8.

[6] 西野順二: 単貧民における多人数完全情報展開型ゲームの考察 (“An analysis on TANHINMIN game”), ゲームプログラミングワークショップ 2007 論文集, pp. 66 – 73, 2007.

モバイルエージェントによる自己安定グラフ探索

原悠樹 [1] 首藤裕一 [1] 角川裕次 [2] 増澤利光 [1]
[1] 大阪大学大学院 情報科学研究科
[2] 龍谷大学理工学部 数理情報学科

概要 ロータールーターはモバイルエージェントによるグラフの永続探索を実現する自己安定アルゴリズムである．すなわち，どのような初期状況から実行を開始しても，いずれモバイルエージェントはあるオイラー閉路に沿った巡回を繰り返す．本稿は各ノードごとにエージェントが訪問したポート番号の順序を記憶するようにロータールーターを拡張することによって，グラフの辺が削除された際に，新たなオイラー閉路を再構築するまでの時間の改善する手法を提案し，その性能の評価を行う．

1 はじめに

モバイルエージェント（以下，**エージェント**）とは，ネットワーク中の計算機を移動しながら，自律的に動作するソフトウェアである．グラフ探索問題（以下，**探索問題**）とは，エージェントシステムにおける最も基本的な問題で，エージェントにネットワーク中の全てのノードとリンクを訪問させることである．グラフ探索により，ネットワーク・トポロジの認識やネットワーク中でのデータ検索が可能である．また，ノードやリンクの故障の検知が可能である．[1, 2, 3, 4, 5]

ロータールータは，有向グラフ上を移動するエージェントがグラフ探索を繰り返し行うためのアルゴリズムである．ノード v の出次数を $\delta(v)$ とすると，ノード v に存在するエージェントは，v の外向辺をポート番号 $\rho_v : \{0, 1, \cdots, \delta(v) - 1\}$ で識別可能である. 各ノード v はエージェントが次に v を訪問したときに，v から次ノードに移動するための辺のポート番号をノードの変数 π_v（以降，ポインタとよぶ）で保持する. エージェントがノード v に訪れると，ポインタ π_v が指定する辺にエージェントを送り出す. その後，π_v の値を次のポート番号に変更する ($\pi_v \leftarrow (\pi_v + 1) \bmod \delta(v)$)．ロータールータでは，ある状況において，その後のエージェントの移動経路は，エージェントの位置と各ノードのポインタの値によって一意に決定される．以降では，エージェントの現在地と全ノードのポインタの値の組をグラフ状況と呼ぶ．あるグラフ状況において，その後のエージェントの移動経路がグラフのあるオイラー閉路の無限回の周回となるとき，グラフ上に**オイラー閉路が生成された**という．オイラー閉路とは，グラフ上のすべての辺を一度だけ通る閉路のことである．

任意の双方向有向グラフ上で，ロータールータは，探索開始から $O(mD)$ 回のエージェントの移動でオイラー閉路が生成することが証明されている [6]. ここで，双方向グラフとは，有向辺 (u, v) がある場合に必ず逆向きの有向辺 (v, u) が存在する有向グラフのことであり，m はグラフ中の有向辺の数，D はグラフの直径である．一度オイラー閉路が生成されると，エージェントは同じオイラー閉路に沿ってグラフ探索を繰り返し行う. また，オイラー閉路が生成されている状況でグラフ上の 1 組の隣接頂点 u, v 間の有向辺 (u, v) および (v, u) が削除された場合，その後の $O(\gamma m)$ 回のエージェントの移動で新たなオイラー閉路が生成されることが証明されている [7]. ここで γ は，辺の削除以前のグラフにおける，削除した辺を一度だけ通る閉路の長さの最小値である．

本稿では，ロータールータを改良することによって，辺の削除からの復旧に要する時間を改善する手法を提案する．提案手法は，各ノードがエージェントが訪れたポート番号の順序を記憶することによって，復旧時間の短縮を実現する．

従来のロータールータと，本稿で提案するアルゴリズムとの比較を表 1 に示す．提案アルゴリズムでは，

表 1: 既存研究と提案手法の比較

	既存研究 [1]	提案手法
任意の初期状況から正当な状況	$O(mD)$	$O(mD)$
1 辺削除から正当な状況まで	$O(\gamma m)$	$O(m)$
複数辺削除から正当な状況まで	$O(mD)$	$O(md)$
エージェントのメモリ	0	$O(1)$
ノードの白板記憶領域	$O(\log\delta(v))$	$O(\delta(v)\log\delta(v))$
根ノードの存在	なし	あり

エージェントとノードの記憶容量を既存手法より多く使用することによって 1 リンクの削除からの復旧時間を $O(\gamma m)$ から $O(m)$ に改善する．また，複数リンクの削除からの復旧時間を $O(mD)$ から $O(md)$ に改善する．(d は複数辺削除後のグラフでの，閉路をノードと見立て、隣接する閉路は辺で結んだグラフの直径を表す.）具体的には，既存手法ではエージェント記憶領域とノード記憶領域の空間計算量はそれぞれ 0, $O(\log\delta(v))$ であるが，提案手法ではそれぞれ $O(1)$, $O(\delta(v)\log\delta(v))$ となる．また，従来のローターループとは異なり，提案アルゴリズムは根ノードの存在を仮定している．

本稿の構成は以下の通りである．2 節では，本稿で用いる用語の定義を行う．3 節では，辺削除からの復旧時間を改善したアルゴリズムを提案する．4 節では，本稿の結果をまとめる．

2 諸定義

2.1 ネットワークモデル

任意の単純連結無向グラフ $G = (V, E)$ を考える. ここで V は n 個のノードの集合であり, E は m 本の無向辺の集合である.

双方向有向グラフ $\overrightarrow{G} = (V, \overrightarrow{E})$ は，G の各辺を向きの異なる 2 本の有向辺に置き換えることによって生成されたグラフである．すなわち，$\overrightarrow{E}$ は弧の集合であり，$\overrightarrow{E} = \{(u,v),(v,u) : \{u,v\} \in E\}$ である．本稿では任意の双方向有向グラフ $\overrightarrow{G}$ に対する単一エージェントによるグラフの探索を考える．

G において各ノード v に接続する無向辺の集合 $\{\{u,v\} \in E | u \in V\}$ を I_v で表す．同様に，$\overrightarrow{G}$ において各ノード v に接続する内向辺の集合 $\{(u,v)|(u,v) \in \overrightarrow{E}\}$ および外向辺の集合 $\{(v,u)|(v,u) \in \overrightarrow{E}\}$ をそれぞれ I_v^- および I_v^+ で表す．また，各ノードの次数 $|I_v|$ を $\delta(v)$ で表す．ノード v において I_v^- および I_v^+ の各辺にはポート番号と呼ばれる局所的なラベル $p_v^- : I_v^- \mapsto \{1,2,...,\delta(v)\}$ および $p_v^+ : I_v^+ \mapsto \{1,2,...,\delta(v)\}$ が割り当てられている．異なる内向辺，外向辺は v においてそれぞれ異なるポート番号を持つ．また，同一の無向辺 $\{u,v\}$ に対応する内向辺 (u,v) および外向辺 (v,u) はノード v において同一のポート番号が割り当てられる．すなわち，任意の $\{u,v\} \in E$ について $p_v^-((u,v)) = p_v^+((u,v))$ である．以下では，$p_v^-((u,v))(= p_v^+((v,u)))$ を $p_v(u)$ と表す．

2.2 エージェントモデル

本稿では単一のエージェントによる任意の双方向有向グラフ $\overrightarrow{G}$ の永続探索（グラフ探索を繰り返し行うこと）を考える．エージェントは有限状態機械である．エージェントは常にいずれかのノードに存在（つまり，エージェントが辺に存在する時点は考えない）し，ステップごとにグラフ上の辺をひとつ通って移動する．また，各ノードは**白板**と呼ばれるメモリを保有する．アルゴリズムはエージェントの状態の集合，白板の状態の集合，各ステップにおいてエージェントが行う動作を規定する．ステップ $t(\geq 0)$ においてエージェントが存在するノードを v^t とする．各ステップ $t(\geq 0)$ においてエージェントは，

- 現在のエージェントの状態

- ノード v^t の白板の内容
- ノード v^t の次数 $\delta(v^t)$
- 前ステップにおいて v^t に移動するために用いた辺の v_t におけるポート番号 $p_{in} = p_{v^t}(v^{t-1})$

を入力として，与えられたアルゴリズムに従って，以下の 4 処理を実行する．

- エージェントの状態を更新する．
- ノード v^t の白板を書き換える．
- 移動先ポート番号 $p_{out} \in \{1, 2, ..., \delta(v^t)\}$ を決定する．
- p_{out} に対応する外向辺を経由して隣接ノードに移動する．

なお，最初のステップ（ステップ 0）における p_{in} は，$\{1, 2, ..., \delta(v^t)\}$ 中の任意の値であるとする．また，本稿では V 中にただひとつの根ノード r が存在することを仮定する．アルゴリズムは根 r 以外の通常ノードと根 r で，エージェントに異なる動作をさせることを可能とするために，根ノードと通常ノードの白板に異なる状態空間を与えてもよい．

2.3 永続探索問題

グラフのノード数が n であるとき，各ノードを $(v_0, v_1, ..., v_{n-1})$ と表す．グラフのステップ t における状況 C は，

- エージェントの状態 *state*
- 各ノードの白板の状態 $(w_0, w_1, ..., w_{n-1})$
- エージェントの位置 $v^t \in V$（エージェントが存在するノード v^t）
- v^t への移動の際に用いた内向辺のポート番号 p_{in}

の 4 つで構成される．アルゴリズムが与えられると状況の集合は一意に決まる．アルゴリズム A のすべての状況の集合を C_A で表す．状況 $C \in C_A$ においてエージェントが 1 ステップの動作を行うことでグラフの状況が C' に遷移するとき，$C \to C'$ と表す．初期状況 $C_o \in C_A$ が与えられたときのアルゴリズム A の実行 $\xi_A(C_0)$ を次式で定義される状況の無限系列とする．

$\xi_A(C_0) = C_0, C_1, C_2, ..., s.t. \forall i \geq 0; C_i \to C_{i+1}$

$\xi_A(C)$ におけるエージェントの移動経路（位置の系列）がグラフ $\overrightarrow{G}$ 上のあるオイラー閉路の無限回の周回となっているとき，状況 C において**オイラー閉路が生成されている**という．また，オイラー閉路が生成されている状況のことを**正当な状況**と呼ぶ．

[定義 1]

下記を満たす状況の集合 $C \subseteq C_A$ が存在するとき，アルゴリズム A は永続探索問題を解く自己安定アルゴリズムであるという．

- 任意の状況 $C_0 \in C_A$ についてアルゴリズム A の実行 $\xi_A(C_0) = C_0, C_1, ...$ が C に属す状況を含む．すなわち，ある $i \geq 0$ について $C_i \in C$
- 任意の状況 $C \in C$ においてオイラー閉路が生成されている．

[定義 2]

永続探索問題を解く自己安定アルゴリズム A について，任意の初期状態から開始する A の実行が i ステップ以内に正当な状況に到達するならば，A の収束時間は高々 i であるという．

2.4 1 組の辺の削除からの復旧時間

本稿では，永続探索アルゴリズムの（任意の状況からの）収束時間に加え，正当な状況において任意の 1 組の辺 $(u,v),(v,u)$ を削除した際，再びオイラー閉路を生成するのに要する復旧時間を考える．

いま，グラフ $\overrightarrow{G}$ から辺 (u,v) および (v,u) が削除して得られるグラフを $\overrightarrow{G'}$ とする．ただし，辺 $(u,v),(v,u)$ の削除によってグラフ $\overrightarrow{G}$ の連結性は失われないものとする．以降では，辺削除に伴うグラフの変化により，元の状況 C がどのように変化するかを定義する．辺の削除に伴い，ノード u および v においてポート番号の再割り当ておよび白板の内容変更がアルゴリズムによって即座に行われる．ノード u,v のポート番号の再割り当てはそれぞれ全単射関数 $\alpha_u : \{1,2,...,\delta(u)\} \setminus \{p_u(v)\} \rightarrow \{1,2,...,\delta(u)-1\}$ および $\alpha_v : \{1,2,...,\delta(v)\} \setminus \{p_v(u)\} \rightarrow \{1,2,...,\delta(v)-1\}$ によって行われるものとする．すなわち，ノード u(resp.v) の各接続辺 $(u,w),(w,u)$(resp.$(v,w'),(w',v)$) に対し，グラフ $\overrightarrow{G'}$ においてはラベル $\alpha_u(p_u(w))$(resp.$\alpha_v(p_v(w'))$) が割り当てられるものとする．2.2 節でアルゴリズムはエージェントの状態空間，白板の状態空間，各ステップのエージェントの動作規則を与えるものと定義したが，アルゴリズムはさらに，グラフから辺 (u,v) および (v,u) が削除されたときに，消失辺 $(u,v),(v,u)$ および α_u,α_v の情報をもとにノード u およびノード v の白板の内容を書き換える規則を与えるものとする．

削除される 1 組の辺 $(u,v),(v,u)$ および α_u,α_v，アルゴリズム A，グラフの状況 C が与えられると，$(u,v),(v,u)$ を削除した結果生じる新たな状況 C' が一意に決まる．このような C' を $R_{u,v,\alpha_u,\alpha_v,A}(C)$ と表す．アルゴリズムの 1 組の辺の削除からの復旧時間を以下で定義する．

[定義 3]

アルゴリズム A を，永続探索問題を解く任意の自己安定アルゴリズムとする．また，u,v をグラフ G において隣接する任意の 2 ノードとし，α_u,α_v をそれぞれ辺 $(u,v),(v,u)$ を削除したときのノード u,v における任意のポート再割り当て関数 $\alpha_u : \{1,2,...,\delta(u)\} \setminus \{p_u(v)\} \rightarrow \{1,2,...,\delta(u)-1\}, \alpha_v : \{1,2,...,\delta(v)\} \setminus \{p_v(u)\} \rightarrow \{1,2,...,\delta(v)-1\}$ とする．グラフ G におけるアルゴリズム A の任意の正当な状況 C について，実行 $\xi_A(R_{u,v,\alpha_u,\alpha_v,A}(C)) = C_0, C_1, ...$ が i ステップ以内に正当な状況に到達するとき，1 組の辺削除に対するアルゴリズム A の復旧時間は高々 i ステップであるという．

3 提案アルゴリズム

3.1 アルゴリズムの戦略

本節では辺削除に対して復旧時間を短縮した自己安定永続探索アルゴリズムを提案する．1 節で述べた通り，従来のロータルータアルゴリズムでは 1 組の辺削除からのオイラー閉路の復旧に $O(\gamma m)$ ステップを要する．ここで γ は，辺の削除以前のグラフ上における，削除した辺を一度だけ通る閉路の最小値である．γ は最悪時 $O(D)$ となるので，従来手法の 1 組の辺削除に対する復旧時間は最悪時 $O(mD)$ となり，任意の初期状況からの収束時間と漸近的に等しいステップ数を要することになる．従来手法が 1 組の辺 $(u,v),(v,u)$ の削除から $O(\gamma m)$ ステップの復旧時間を要するのは，ノード u,v の近傍を中心にオイラー閉路を大幅に修正することが起こり得るからである．提案アルゴリズムは，辺削除前の正当な状況で生成されていたオイラー閉路を最大限活用することで $O(m)$ ステップでの復旧を実現する．また，複数の辺が同時に削除された場合，1 組の辺の削除の場合のアルゴリズムを応用して復旧を行う．

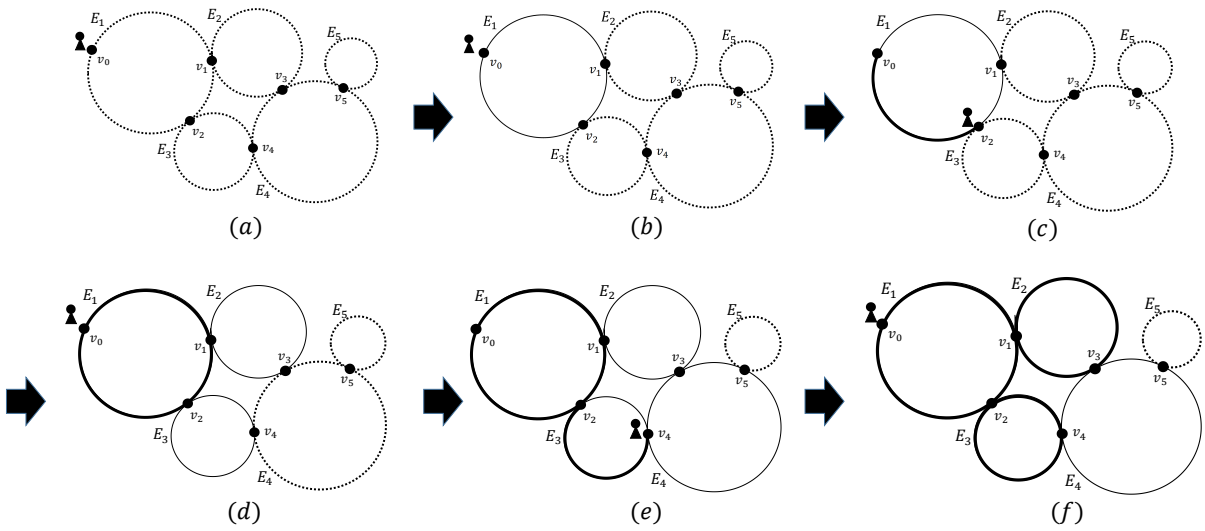


図 1: 復旧段階におけるエージェントの通過辺を段階的に表している

3.2 アルゴリズムの全体像

提案アルゴリズムは 6 つのモードによって動作する．各モードの役割は以下である．

- *normal* … 正当な状況.
- *recover* … 復旧開始時のみ行うモード.
- *search* … 結合点の検知.
- *trace* … 復旧段階での未探索辺の通過.
- *search*′ … 結合点の検知（*trace* モードの後）.
- *cleaning* … 初期化，自己安定.

各モードの詳細を説明する．*recover* モードは，復旧開始時に行うモードである．復旧開始点から最初に通過する辺を含む閉路を通過するモードである．復旧開始点に戻り，*search* モードに遷移する．

search モードは，結合点を発見するモードである．結合点とは，復旧開始から通過していない外向辺含むノードのことである．結合点を発見すると，*trace* モードへ遷移する．

trace モードは，未通過の閉路に沿って移動するモードである．*search* モードで発見した結合点から開始し，閉路を 1 周して結合点に戻る．その後，*search*′ モードに遷移する．

search′ モードは，*search* モードと同様に，結合点を発見するモードである．*search* モードとの違いは結合点を発見したかどうかである．*search* モードは，後に説明するフェーズ間でまだ結合点を発見していないモードであることを示し，*search*′ モードは既に結合点を発見したモードであることを表す.

cleaning モードは，正当な状況にするモードである．具体的には，白板の初期化，オイラー閉路の確認である．

正当な状況からの複数辺削除に対して，復旧時の各モードの動作について説明する．

図 1 は復旧段階でのエージェントの通過手順を図示したものである．E_1 から E_5 は分割された閉路を簡易的に表したものである．v_0 は復旧開始時のノード，v_1 から v_5 は 2 つの閉路を通過するノードである．例えば，v_1 は E_1 と E_2 が通過するノードである．図 1 では，v_1 から v_5 以外では閉路は連結しないとする．図

1 では，線の状態によってエージェントの通過状況を表している．点線は未通過辺，細線は 1 度通過辺，太線は 2 回以上通過した辺を表す．エージェントの移動は閉路を反時計回りに行う．

図 1 の各グラフの状況について説明する．

(a) は復旧開始時である．エージェントは復旧開始点 v_0 に存在し，これから閉路 E_1 に沿って移動を行う．ここでフェーズを定義する．フェーズとは，復旧開始点を基準とした時間の区切りである．フェーズは復旧開始点 v_0 から開始し，閉路 E_1 に含まれる辺をすべて通過し，v_0 に戻ってきたときにフェーズを終了する．復旧開始時はフェーズ 0 の開始とする．

(b) はフェーズ 0 の最後である．フェーズ 0 中は常に recover モードであり，フェーズ 0 で通過した辺は E_0 のみである．search モードに遷移し，フェーズ 1 が開始する．

(c) はフェーズ 1 の search モードで結合点 v_2 を発見した時の状況である．ここから trace モードに遷移し，E_3 の辺を移動する．E_3 を全辺移動後は，search' モードに遷移し，v_2 から E_1 に戻る．

(d) はフェーズ 1 が終わり，v_0 に来たところである．フェーズ 1 では，フェーズ 0 で通過した E_1 に加えて，E_1 に隣接する E_2 と E_3 を通過した．search' モードから search モードに遷移し，フェーズ 2 が開始する．

(e) はフェーズ 2 で初めて結合点 v_4 を発見し，search モードから trace モードに遷移する状況である．

(f) はフェーズ 2 を終えた時である．フェーズ 2 では，E_1, E_2, E_3, E_4 を通過し，E_4 はフェーズ 2 で初めて通過した．search' モードから search モードに遷移し，フェーズ 3 が開始する．以上が図 1 における復旧の手順である．

3.3 エージェントの保持する変数

エージェントはモードを表す変数 $\mathtt{mode} \in \{normal, recover, trace, search, search', cleaning\}$ を持つ．また，オイラー閉路復旧時に，通過した辺を識別するために，$\mathtt{f\_check} \in \{1, 2, 3\}$ を持つ．

3.4 白板の保持する変数

各ノード s の白板は，5 つの変数 $s.\mathtt{in}, s.\mathtt{out}, s.\mathtt{check}, s.\mathtt{d}, s.\mathtt{r}$ を保持する．変数 $s.\mathtt{in}, s.\mathtt{out}$ は，それぞれ $\delta(s)$ 個の相違なるポート番号の並びである．

$s.\mathtt{in}$ は，ノード s の内向辺について，その最新の訪問時刻が最も古いものから順に $\delta(s)$ 個のポートを並べたものである．$s.\mathtt{in}$ の i 番目の要素を $s.\mathtt{in}_{i-1}$ で表す．すなわち，$s.\mathtt{in} = s.\mathtt{in}_0, s.\mathtt{in}_1, ..., s.\mathtt{in}_{\delta(s)-1}$ である．$s.\mathtt{in}_0$ は，ノード s の内向辺のなかで最も長い間エージェントの移動に利用されなかった内向辺のポート番号である．

$s.\mathtt{out}$ は，ノード s の外向辺について，その最新の訪問時刻が最も古いものから順に $\delta(s)$ 個のポートを並べたものである．$s.\mathtt{out}$ の i 番目の要素を $s.\mathtt{out}_{i-1}$ で表す．すなわち，$s.\mathtt{out} = s.\mathtt{out}_0, s.\mathtt{out}_1, ..., s.\mathtt{out}_{\delta(s)-1}$ である．$s.\mathtt{out}_0$ は，ノード s の外向辺のなかで最も長い間エージェントの移動に利用されなかった外向辺のポート番号である．

$s.\mathtt{in}, s.\mathtt{out}$ はそれぞれ，各要素の値がすべて異なる．つまり，$\forall i, j \in \{0, .., \delta(s)-1\}(i \neq j), s.\mathtt{in}_i \neq s.\mathtt{in}_j, s.\mathtt{out}_i \neq s.\mathtt{out}_j$ となる．

$s.\mathtt{check}$ は $\delta(s)$ 個の要素を持つ配列で，各要素は 0 から 7 のいずれかの値を持つ．$s.\mathtt{check}$ の $\delta(s)$ 個の要素を $s.\mathtt{check}_0, \cdots, s.\mathtt{check}_{\delta(s)-1}$ で表し，$s.\mathtt{check}_k$ は，ノード s におけるポート番号 k に対応する外向辺を復旧段階で 1 度でも移動したことを記録する（復旧段階とは，$\mathtt{mode} \in \{recover, trace, search, search'\}$ の時である）．また，結合点での，e_2 から e_1 への移行の際の目印の役割も担う．

$s.\mathtt{d}$ はノード s が削除された辺に接続するノードかどうかを示す（$s.\mathtt{d} \in \{0, 1\}$）．通常，$s.\mathtt{d} = 0$ である．辺 $(u, v), (v, u)$ が削除されると，辺消失点 u, v においてそれぞれ $u.\mathtt{d}, v.\mathtt{d}$ の値が 1 に変更される．

$s.\mathtt{r}$ はノード s が根ノードかどうかを示す（$s.\mathtt{r} \in \{0, 1\}$）．2 節で定義した通り，根ノードは有向グラフ中でただひとつ存在するノードである．$s.\mathtt{r} = 1$ のとき，ノード s は根ノードであることを，$s.\mathtt{r} = 0$ のとき，s は根ノードでないことを示す．$s.\mathtt{r}$ の値は固定である．すなわち，アルゴリズムの実行中，$s.\mathtt{r}$ の値が変化することはない．

3.5 提案アルゴリズムの詳細

Algorithm1 は提案アルゴリズムのメイン関数の疑似コードを示す．Algorithm1 はエージェントがノード s に移動したときに実行される．p_{in} は，ノード s に入ってきたポート番号を表す．Algorithm1 の行番号ごとに，主な動作について以下に示す．

- 1-4 行目 … 異常検知.
- 5,6 行目 … 諸定義.
- 7-9 行目 … 削除を検知，復旧開始（*recover* モードへ遷移）.
- 10-12 行目 … *trace* モードの終了.
- 13-21 行目 … 現在のフェーズの終了, 復旧終了か判定.
- 22-26 行目 … 結合点の発見、*trace* モードへ遷移.
- 27-29 行目 … 通過した辺を記録する.
- 30-46 行目 … *cleaning* モードの動作を行う.
- 47-49 行目 … 白板の書き換え，別のノードに移動する.

Algorithm1 の各動作について以下に詳しく述べる．

1-4 行目では，不整合が発生した場合に *cleaning* モードに移行する．不整合は初期状況が正当な状況でない場合に発生する．

5 行目で，k を定義する．k は他のノードからノード s に来た時に通過した内向辺のポート番号と同じ値となる $s.\mathtt{in}_k$ の事である．正当な状況の場合，$k = 0$ となることが保証される．

6 行目で，p と定義する．p は s の白板において p_{in} に対応している外向辺のポート番号である．

7-9 行目では，辺の削除を初めて検知した場合，*normal* モードから *recover* モードに移行する．エージェントが辺の削除を検知するのは，$s.\mathtt{d} = 1$，つまり辺消失点にエージェントが来たときである．*recover* モードに移行することでオイラー閉路の復旧を開始する．

10-12 行目で、*trace* モードから *search′* モードに遷移する．$s.\mathtt{check}_p$ が 4 以上であるのは trace モードに遷移したときに立てたフラグ（24 行目）を表す．12 行目でフラグを回収する．

13-21 行目は、開始ノード v_0 に戻ってきた時の動作を行う．v_0 に戻ってきたことをエージェントが識別するために，次に通過する外向辺に対応する check の値である $s.\mathtt{check}_p$ の値と f_check の値を比較している．f_check は通過した辺に対応する $s.\mathtt{check}$ に代入する値を表している．$s.\mathtt{check}_p$ = f_check となるのは v_0 を基準として 1 周し v_0 に戻ってきたときになる．1 周すると f_check の値を変更する．

エージェントの現在のモードによって遷移先が 2 通りに分岐する．

recover モードまたは *search′* モードのとき，オイラー閉路が生成されていないため，1 周する．*search* モードに移行し，f_check の値を変更する．

search モードのとき，オイラー閉路は生成されたので，復旧段階を終了する．*cleaning* モードに移行する．

22-26 行目で，エージェントが結合点を発見した場合の動作を行う．結合点は *search* モードまたは *search′* モードの時に，$s.\mathtt{check}_{s.\mathtt{out}_0} = 0$ であるノードが結合点である．白板の外向辺の先頭の要素が復旧段階において未通過辺であることを確認し，その外向辺を通過している．結合点を発見したエージェントは，*trace* モードに移行する．24 行目で $s.\mathtt{check}_p$ にフラグを立てておくことで，本来次に通過する辺を白板に記憶しておく．*trace* モードへ移行時，$s.\mathtt{in}_1$ と $s.\mathtt{in}_k$ を入れ替える．この処理により，辺削除により分断された 2 つの閉路が結合される．

27-29 行目では，復旧段階での共通の動作を行う．4 モードでは，p_{in} の値によってエージェントの次の移動先を決定する．具体的には，$(s.\mathtt{in}_1, s.\mathtt{out}_1)$ を $(s.\mathtt{in}_k, s.\mathtt{out}_k)$ と入れ替え，$s.\mathtt{out}_1$ に対応する外向辺を移動することとする．この移動方法は，s が管理する移動履歴 $s.\mathtt{in}$ および $s.\mathtt{out}$ において，s に移動してきた

Algorithm 1 提案アルゴリズム（ノード s での動作）

Main routine:

if ($\mathtt{mode} = \mathit{normal} \wedge (s.\mathtt{in}_0 \neq p_{in} \vee s.\mathtt{check}_{s.out_0} \neq 0)$) **or** ($\mathtt{mode} = \mathit{recover} \wedge s.\mathtt{check}_p \in \{2,3,4,5,6\}$)
  or ($\mathtt{mode} = \mathit{trace} \wedge s.\mathtt{check}_p \in \{1,2,3\}$)
  or ($\mathtt{mode} \in \{\mathit{search}, \mathit{search}'\} \wedge (s.\mathtt{check}_p \in \{0, (\mathtt{f\_check} \bmod 3) + 1, 4, 5, 6\}$)
  or ($s.\mathtt{check}_{s.out_0} = (\mathtt{f\_check} \bmod 3) + 1$)) **then**
  $\mathtt{f\_check} \leftarrow 1$
end if

Let k be the interger such that $s.\mathtt{in}_k = p_{in}$.
Let p be the interger such that $p = s.\mathtt{out}_k$.
if $\mathtt{mode} = \mathit{normal}$ **and** $s.\mathtt{d} = 1$ **then**
  $\mathtt{mode} \leftarrow \mathit{recover}$
  $\mathtt{f\_check} \leftarrow 1$
else if $\mathtt{mode} = \mathit{trace}$ **and** $s.\mathtt{check}_p \geq 4$ **then**
  $\mathtt{mode} \leftarrow \mathit{search}'$
  $s.\mathtt{check}_p \leftarrow s.\mathtt{check}_p - 3$
else if $\mathtt{mode} \in \{\mathit{recover}, \mathit{search}, \mathit{search}'\}$ **and** $s.\mathtt{check}_p = \mathtt{f\_check}$ **then**
  if $\mathtt{mode} \in \{\mathit{recover}, \mathit{search}'\}$ **then**
    $\mathtt{mode} \leftarrow \mathit{search}$
    $\mathtt{f\_check} \leftarrow (\mathtt{f\_check} \bmod 3) + 1$
  else if $\mathtt{mode} = \mathit{search}$ **then**
    $\mathtt{mode} \leftarrow \mathit{cleaning}$
    $\mathtt{f\_check} \leftarrow 1$
  end if
end if
if $\mathtt{mode} \in \{\mathit{search}, \mathit{search}'\}$ **and** $s.\mathtt{check}_{s.out_0} = 0$ **then**
  $\mathtt{mode} \leftarrow \mathit{trace}$
  $s.\mathtt{check}_p \leftarrow s.\mathtt{check}_p + 3$
  $s.\mathtt{in}_k \leftrightarrow s.\mathtt{in}_0$
  $s.\mathtt{check}_{s.\mathtt{out}_0} \leftarrow \mathtt{f\_check}$
else if $\mathtt{mode} \in \{\mathit{recover}, \mathit{search}, \mathit{search}', \mathit{trace}\}$ **then**
  $(s.\mathtt{in}_k, s.\mathtt{out}_k) \leftrightarrow (s.\mathtt{in}_0, s.\mathtt{out}_0)$
  $s.\mathtt{check}_{s.\mathtt{out}_0} \leftarrow \mathtt{f\_check}$
else if $\mathtt{mode} = \mathit{cleaning}$ **then**
  $s.\mathtt{check}_q \leftarrow 0$ for all $q = 0, 1, \ldots, \delta(s) - 1$
  $s.\mathtt{d} \leftarrow 0$
  if $s.\mathtt{in}_k \neq s.\mathtt{in}_0$ **then**
    if $\mathtt{f\_check} = 3$ **then**
      $\mathtt{f\_check} \leftarrow 2$
    end if
    $s.\mathtt{in}_k \leftrightarrow s.\mathtt{in}_0$
  end if
  if $s.\mathtt{r} = 1$ **and** $s.\mathtt{out}_0 = 0$ **then**
    if $\mathtt{f\_check} \in \{1,2\}$ **then**
      $\mathtt{f\_check} \leftarrow \mathtt{f\_check} + 1$
    else if $\mathtt{f\_check} = 3$ **then**
      $\mathtt{mode} \leftarrow \mathit{normal}$
    end if
  end if
end if

$p_{out} \leftarrow s.\mathtt{out}_0$
Rotate()
Move throuth port p_{out}

際に用いた内向辺に対応する外向辺を用いて移動することを意味する．したがって，この移動方法によって，エージェントは辺削除により分断された閉路に沿って巡回することができる．

30-45 行目では，*cleaning* モードでの動作を行う．*cleaning* モードでは 2 つの役割が存在する．

1 つ目は，オイラー閉路の再構築のために用いた白板の変数の値を初期化することである．具体的には 31,32 行目でノードの白板の変数 $s.\mathtt{check}$ と $\mathtt{d}$ の値を 0 にしている．

2 つ目は，初期状況でオイラー閉路が形成されていない場合，もしくは切断以外の何かしらの理由によってオイラー閉路が崩れた場合に，オイラー閉路を復旧することである．

33-45 行目では，オイラー閉路が生成されたかを確認する．根ノード $\mathtt{r}$ のポート番号 0 に対応する外向辺を移動してから，ノード s 他の外向辺をすべて通過し，再びポート番号 0 の外向辺を通過しようとする間の経路がオイラー閉路であるか確認している．オイラー閉路が生成されていることをどのようにして確認しているか説明する．根ノードのポート番号 0 の外向辺を基準として 1 周する間に，白板の対応の入れ替えが発生したかを見る．具体的には，33 行目の $s.\mathtt{in}_k \neq s.\mathtt{in}_0$ で白板の書き換えを確認している．$s.\mathtt{in}_k$ と $s.\mathtt{in}_0$ が異なる場合，$\mathtt{f\_check}$ の値を変更することで白板の書き換えが発生したことを記憶する．その後，白板の書き換えを行う（37 行目）．白板の書き換えが発生した場合，前回の 1 周で通過した経路とは異なる経路を通過しているため，オイラー閉路ではないとわかる．

39-45 行目で，オイラー閉路が生成されたかのチェックを行う．$\mathtt{f\_check} = 3$ ならば，オイラー閉路が生成され，*normal* モードへと遷移できる．

47-49 行目で白板の変数を書き換えてエージェントが移動を行う．46 行目でエージェントが次に移動する外向辺のポート番号を p_{out} に記憶しておく．その後，関数 Rotate() によって，ノード s の白板の内容を書き換える．関数 Rotate() は $s.\mathtt{in}_1$ と $s.\mathtt{out}_1$ をキューの後ろに回すことで通過した順にポート番号を保持する．詳しくは Algorithm2 に示す．その後，p_{out} のポート番号が示す外向辺を通過してエージェントが移動する．

Algorithm 2 提案アルゴリズム

Rotate():

1: $(s.\mathtt{in}_1, s.\mathtt{in}_2, ..., s.\mathtt{in}_{\delta(s)}) \leftarrow (s.\mathtt{in}_2, s.\mathtt{in}_3, ..., s.\mathtt{in}_{\delta(s)}, s.\mathtt{in}_1)$

2: $(s.\mathtt{out}_1, s.\mathtt{out}_2, ..., s.\mathtt{out}_{\delta(s)}) \leftarrow (s.\mathtt{out}_2, s.\mathtt{out}_3, ..., s.\mathtt{out}_{\delta(s)}, s.\mathtt{out}_1)$

3.6 削除された辺に接続するノード（辺消失点）の処理

辺消失点の処理アルゴリズムを Algorithm3 に示す．ノード u に接続した辺 (u, v) が削除された場合を考える．ポート番号の再割り当てが必要であり，ポート再割り当て関数 α_u を用いて $u.\mathtt{in}, u.\mathtt{out}$ の値を変更する．

Algorithm 3 提案アルゴリズム（辺削除発生時）

1: Delete $p_u(v)$ from $u.\mathtt{in}, u.\mathtt{out}$ and $u.\mathtt{check}$

2: **for** $i = 1, 2, ..., \delta(u) - 1$ **do**

3: $\quad u.\mathtt{in}_i \leftarrow \alpha_u(u.\mathtt{in}_i)$

4: $\quad u.\mathtt{out}_i \leftarrow \alpha_u(u.\mathtt{out}_i)$

5: **end for**

6: $u.\mathtt{d} \leftarrow 1$

辺の削除によって，ノード u に接続されている内向辺と外向辺の本数はどちらも 1 減る．そのため，最初に，$u.\mathtt{in}, u.\mathtt{out}, u.\mathtt{check}$ から，辺 (u, v) を削除する．

4 まとめ

本稿では双方向有向グラフにおいてグラフ探索を実現するロータールータについて，1 辺、または複数辺の削除からのオイラー閉路の復旧までの時間を，$O(\gamma m)$ から改善する方法を示した. 一方，このために，エージェントおよびノードが要する記憶容量が増加している．

今後の課題は，根ノードが存在しない状況での同様の改善を考えたい．

参考文献

[1] Y. Sudo, D. Baba, J. Nakamura, F. Ooshita, H. kakugawa, and T. Masuzawa,“A Single Agent Exploration in Unknown Undirected Graphs with Whiteboards,”IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E98, no.10, p. 2117-2128, 2015.

[2] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro,“Map construction of unknown graphs by multiple agents,”Theor. Comput. Sci., vol.385, no1-3, pp.34-48, 2007.

[3] S. Albers and M.R. Henzinger,“Exloring an unknown environments,”Proc. Twenty-Ninth Annual ACM Symposium on Theory of Computing - STOC’97, pp416-425, 1997.

[4] X. Deng andC.H. Papadimitriou, "Exploring an unknown graph," J. Graph. Theor, vol.32, no.3, pp.265-297, 1999.

[5] R. Fleischer and G. T rippen,“Exploring an unknown graph efficiently,”Algorithms ESA 2005, Lecture Notes in Computer Science, vol.3669, pp.11-22, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[6] Vladimir Yanovski, Israel A. Wagner, Alfred M. Bruckstein,“A Distributed Ant Algorithm for Efficiently Patrolling a Network,”Algorithmica in Springer-Verlag, vol.37, no.2, pp.165-186, 2003.

[7] Evangelos Bampas, Leszek Gasieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, Adrian Kosowski, Tomasz Radzik,“Robustness of the Rotor-Router Mechanism,”Algorithmica in Computer Science, vol 5923, pp 345-358, 2017.

自律移動ペアロボットモデルによる正三角形から直線への形状形成アルゴリズムについて

A Study on Line Formation Algorithms from a Triangle Form in Autonomous Mobile Pair-Robot System

高橋一生 [1] 金鎔煥 [1] 片山喜章 [1] 和田幸一 [2]
Kazuki Takahashi Yonghwan Kim Yoshiaki Katayama Kouichi Wada

名古屋工業大学大学院工学研究科情報工学専攻 [1]
Nagoya Institute of Technology, Graduate School of Computer Science and Engineering
法政大学理工学部応用情報工学科 [2]
Hosei University, Faculty of Science and Engineering Department of Applied Informatics

概 要

Programmable matter とは, 複数の小さな自己組織化ロボットが互いに連携し, 全体として機能を発揮する機械であり, 近年注目を浴び, 幅広く研究されている. 自律分散ロボットシステムは低機能で移動可能な複数の端末によって構成されるシステムであり,Programmable matter のモデルのひとつとして捉えることができる. 本論文では,Programmable matter の一つであるペアロボットモデルに着目する. ペアロボットシステムモデルとは, 予め決められた 2 台の自律分散移動ロボットが一つのペアとなって, 二次元離散平面上で協調して動作するモデルである. 本論文では, ペアロボットモデルにおいて座標系に関して 1 軸の向きと方向を共通知識とするモデル上で, 正三角形を形成しているペアロボット群を一直線上に整列させる問題を解くアルゴリズムを提案する.

1 はじめに

Programmable matter[1]〜[4] とは, 複数の小さな自己組織化ロボットが互いに連携し, 全体として機能を発揮する機械のことで, ロボティクス, 計算機科学など多くの分野で理論的あるいは実践的な研究が行われれている. その応用は広く, ナノロボットによる外科治療や自由に変形可能な移動ロボットの実現など, さまざまな分野でその成果が期待されている. 計算機科学分野に注目すると,Programmable matter を, 計算能力を有する低機能端末 (ロボット) によって構成されるシステムとしてモデル化し, それに関する理論を中心とした研究が活発に行われている. 特にシステムモデルの違いによる問題解決能力の解明や問題解決手法 (アルゴリズム) の開発が主な目的となっている.

自律分散ロボットシステムは Programmable matter のモデルのひとつとして捉えることができる. 自律分散ロボットシステムとは, 低機能で移動可能な複数の端末によって構成されるシステムである. 自律分散ロボットは, 自律的に動作するロボットであり, これらのロボットの集合を自律分散ロボット群と言う. これらのロボットが協調的に動作することで全体として一つの目標を達成する. ロボットの機能やシステム全体の仮定の違いによってさまざまなモデルが考えられるが, モデルと問題の可解性の関係を明らかにすることが研究者の興味の一つである. 近年, この自律分散ロボット群の理論的研究が盛んに行われており, 深海や宇宙など人間が直接管理することが難しいような環境において, 人間による管理の必要がない自律分散ロボット群の利用が期待されている. 自律分散ロボットは識別子を持たず, 外見上も区別できず, ロボット同士が直接通信を行うことはなく, 過去の情報を記憶することはできない. 各ロボットは他のロボットの位置を観測 (Look) し, 観測結果を入力にアルゴリズムに従って行き先を計算 (Compute) し, その行き先に移動 (Move) するというシンプルなサイクルを繰り返し問題を解く.

本研究で扱うペアロボットシステムモデルも Programmable matter のひとつである. ペアロボットシステムモデルは自律分散移動ロボットシステムのひとつとして捉えられ, 二次元格子や三角格子など離散空間上のモデルであり, システムを構成するロボットは通常の自律分散移動ロボットシステムと基本的に同様である. 具体的には識別子や通信機能を持たず, 重複検知機能を有する. また,2 台で 1 つの組を構成しており, 互いに自分のペアがどのロボットか認識できる. また, 同一点に 2 台までロボットの存在を許可している. ペア同士は同一点上または隣接点上のどちらかに存在し, 同一点上に存在するときはそれぞれのロボットは *Short* 状態, 隣接点上に存在するときは *Long* 状態をとり, 各ロボットはこの 2 状態間を交互に遷移することで全体として目的を達成する. 本研究ではこのペアロボットシステムモデルにおいて, 座標系に関して 1 軸の向きと方向を共通知識とし, 視野が距離 1 であり, 完全同期及び半同期スケジューラの下で動作するモデル上で正三角形を形成しているペアロボット群を一直線上に整列させる問題を解くアルゴリズムを提案する.

2 関連研究

自律分散ロボット群の制御に関する理論的研究においては, ロボットモデルと問題の可解性の関係を明らかにすることが研究者の興味の一つである.Prencipe[5] によってロボットが座標系に関して共通の知識を持たない連続平面において, 追加の能力を一切仮定しない場合に,

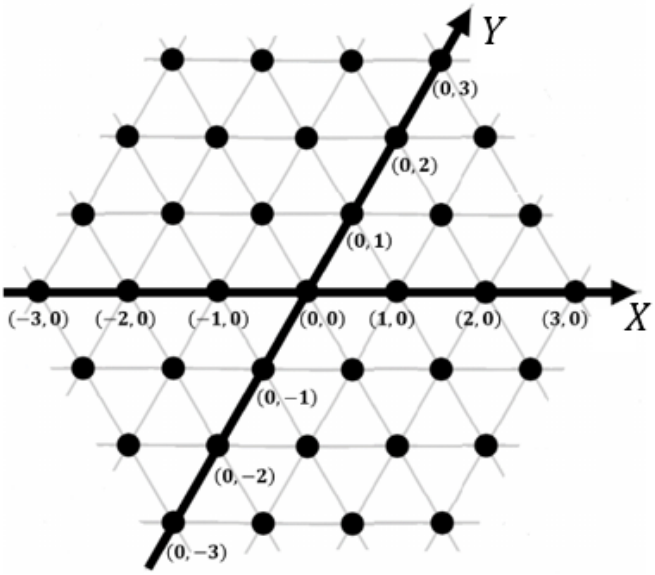


図 1　二次元三角格子座標系 $\mathbb{T}$

非同期および半同期で動作する 2 台以上のロボットの一点集合問題 (任意の位置に配置されたロボットがある一点に集合する問題) を解くアルゴリズムは存在しないことが証明されている.Flocchini ら [6] によって, ロボットが座標系について合意があると仮定し, 制限された視界 (*limitedvisibility*) でのロボットの一点集合問題が解かれている. 連続平面ではなく離散平面についてもロボットの一点集合問題は研究されている.Kiasing ら [7] や Angelo ら [8] によって, グラフ構造や格子上のロボットの一点集合問題に関して研究されている.Cieliebak ら [9] は, 格子サイズが $m \times n$ の格子上のロボットの一点集合問題について,m と n が偶数あるいは奇数のときの組み合わせそれぞれに関する可解性を示した.

山田ら [10] によるペアロボットシステムモデルは, 各ロボットはロボットの総数を知らず共通の座標系を持たないが, 単位距離と 1 軸の方向と向き, 右回りを知っているとし, 各ロボットの視野範囲は隣接頂点のみとしている. このようなモデル上のロボットで完全同期または半同期スケジューラの下で実行される, 一直線上に連続的に配置されたペアロボットの集合がその直線上を切断される事なく行進するアルゴリズムや行進中の進行方向に障害物があった場合にその障害物を被覆するアルゴリズムが提案されている.

3　モデル

3.1　座標系

すべての自律分散ロボットは二次元三角格子座標系 $\mathbb{T}$ 上に存在する.(図 1) $\mathbb{T}$ は $\mathbb{F} = \{(X + \frac{1}{2}Y, \frac{\sqrt{3}}{2}Y) | X, Y \in \mathbb{Z}\}$ で定義された点の集合であり,$\mathbb{F}$ に対応する頂点の集合 $\mathbb{G} = \{(X, Y) | X, Y \in \mathbb{Z}\}$ を持つ. $\mathbb{T}$ の原点を $\mathbb{G}$ 上の点 $(X = 0, Y = 0)$ とする. $\mathbb{G}$ における任意の Y に対して,$(0, Y)$ で表される点すべてを含む直線を Y 軸の方向とし $(0, Y)$ から $(0, Y + 1)$ への方向を Y 軸の正の向き,$(0, Y)$ から $(0, Y - 1)$ への方向を Y 軸の負の向きとする. 同様に任意の X に対しても,$(X, 0)$ から $(X + 1, 0)$ への方向を X 軸の正の向き,$(X, 0)$ から $(X - 1, 0)$ への方向を X 軸の負の向きと定義する. また,$(0, Y)$ と $(0, Y + 1)$ の距離を $\mathbb{T}$ の単位長さ 1 として定義する.

この時 $\mathbb{G}$ 上の任意の 2 点 $u = (u_X, u_Y), v = (v_X, v_Y)$ 間の距離 $dist(u, v)$ を次のように定める.

$$dist(u,v) = \begin{cases} |u_X - v_X| + |u_Y - v_Y| \\ \qquad ((u_X - v_X)(v_Y - u_Y) \geq 0) \\ |u_X - v_X| + |u_Y - v_Y| - min(|u_X - v_X|, |u_Y - v_Y|) \\ \qquad ((u_X - v_X)(v_Y - u_Y) < 0) \end{cases}$$

3.2　スケジューラ

以下, 多くの研究で一般に用いられている標準的なロボットの実行モデルについて説明する. 各ロボットの動作は次に説明する 4 つの状態,Wait,Look,Compute,Move のうち後者の 3 つの状態を 1 サイクルとして繰り返し実行する.Wait は任意の時刻に挿入可能な状態である.

1. **Wait:** ロボットは待機状態. ロボットは無制限に待機することはできない. 開始時点ではすべてのロボットが Wait 状態である.
2. **Look:** ロボットは視野範囲内の他のロボットの位置座標を得る.
3. **Compute:** ロボットはアルゴリズムに従って行き先を計算する. ロボットは無記憶であるため, アルゴリズムの入力は直前の Look で得た情報のみである.
4. **Move:** Compute 状態において計算した行き先に向かって移動する.

また, 各ロボットに Look,Compute,Move で構成される実行サイクルを, スケジューラから指示されたタイミングで実行すると仮定する. スケジューラは, その同期の程度によって次の 3 つのモデルが定義される.(図 2)

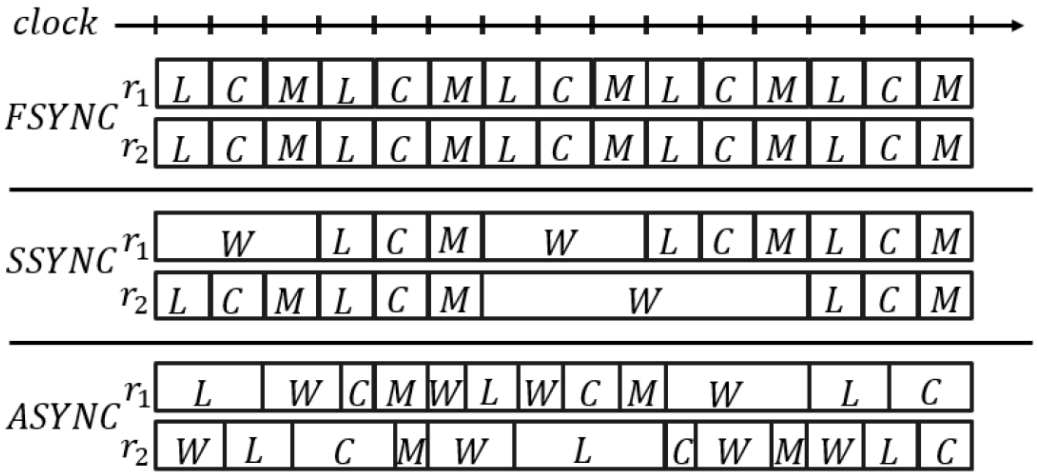


図 2　ロボットの 3 つの実行モデルの例

- **完全同期 (fully-synchronous, FSYNC):**すべてのロボットが完全に同期した時計を用いてアルゴリズムを実行できると仮定する. すなわち, すべてのロボットは同じ時刻にアルゴリズムを開始し, その後, 同じ時刻に, Look,Compute,Move をそれぞれ実行する.
- **半同期 (semi-synchronous, SSYNC):**FSYNC において, サイクルを実行しないロボットの存在を許すモデルである. ただし, すべてのロボットは無限回サイクルを実行すると仮定する.
- **非同期 (asynchronous, ASYNC):**ロボットの実行に関する仮定を一切置かないモデルである. 各ロボットのサイクルの開始時刻や Look,Compute,Move にかかる時間の上限も与えられない. そのため, ロボットが移動中に他のロボットに観測されることもある.

これらの 3 つの同期モデルに関して,SSYNC は FSYNC の実行を含み,ASYNC は SSYNC, FSYNC の実行を含む. つまり,ASYNC で, ある問題を解くアルゴリズムが存在する場合, そのアルゴリズムは SSYNC,FSYNC のロボットに対しても, その問題を正しく解くアルゴリズムである.SSYNC と FSYNC のモデル関係も同様である. 本研究では完全同期及び半同期で動作するとする.

3.3　ペアロボットモデル

n 台のロボットの集合 $\mathbb{R}$ を $\mathbb{R} = \{r_0, r_1, \cdots r_{n-1}\}$ とする. 各ロボットは独自の三角格子座標系を持ち,$\mathbb{T}$ を知らないとする. ただし,$\mathbb{T}$ の X 軸の方向と向き, 及び単位距離にのみ合意を持つとする. また, 各ロボットの座標系における原点は自身の頂点座標とする. ペアロボットシステムモデルにおけるペアロボットは, ロボット 2 台で 1 組の構成をしており,1 台のロボットが 2 組以上のペアに属することはなく, したがってシステムを構成するロボットの台数は偶数でペア数はロボットの台数の半分となる. さらに, ペアロボットシステムモデルでは同一点に 2 台までロボットが存在することが許される. ただし, 同一点上に存在する 2 台のロボットが必ずしもペアである必要はない. また, 同じペアに属するロボットは互いに自分のペアがどのロボットか認識でき, 互いに単位距離より大きく離れることはできない. つまりペアを構成する 2 台のロボットは, 同一点あるいは隣接する 2 頂点のみに存在可能であり,Move による移動先は自身の隣接点のうちの 1 点となる. したがって, 同一のペアに属するロボット $r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ と $r_j(\in \mathbb{R}, 0 \leq i \leq n-1, r_i \neq r_j)$ の位置関係によって,r_i 及び r_j の状態 $r_{i.s}, r_{j.s}$ を以下のように定める.(図 3)

$$\begin{aligned} Short : dist(r_i, r_j) &= 0 \\ Long : dist(r_i, r_j) &= 1 \end{aligned}$$

(a)*Short* 状態　　(b)*Long* 状態

図 3　ペアロボットの状態

そして, ペアが *Short* のとき, その点上における 2 台のロボットの対称性を崩すことができない. つまり,2 台のロボットの片方のみを移動させなければならないが 2 台のロボットが全く同じ情報を持つため, 片方のみを動かすということができない. そのため, 各ロボットに対し *high* または *low* の識別子 $r_{i.id}$ を定数として与え, 各ペアロボットでそれぞれ異なる識別子でペアを組むこととする. これにより *Short* 状態のときは,*high* の識別子のロボットのみを移動させることができる.

$\mathbb{G}$ 上の任意の点を $v = (v_X, v_Y)$ としたとき, ロボット $r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ の視野範囲 $\mathbb{N}(r_i)$ を以下のように定める.

$\mathbb{N}(r_i) = \{(v_X, v_Y) | dist(r_i, v) \leq 1\}$

ロボットは視野範囲内の点に関して弱重複検知機能を有する. つまり, 視野範囲内の各点に存在するロボットの台数が 0 台か 1 台かそれ以上の台数かが区別できるものとする. また, ロボット $r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ の視野範囲内の点にラベルを付ける. ロボット自身の存在する点に

は $r_{i.0}$ のラベルを付ける. それ以外の点に関しては X 軸正の向きに存在する点から順に右回りに $\{r_{i.l}|l \in [1,6]\}$ と付けるものとする.(図 4)

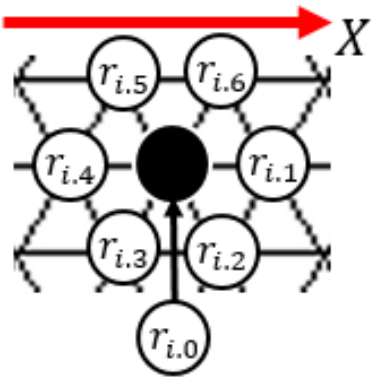


図 4　各ロボットにおける視野範囲のラベル付け

3.4　三角形直線化問題

本研究ではペアロボットシステムモデルにおいて, 正三角形を形成しているペアロボット群を一直線に整列させる問題を扱う.　ロボットの集合を $\mathbb{R} = \{r_0, r_1, \cdots, r_{n-1}\}$ とする.　ロボット $r_i(\in \mathbb{R})$ の任意の時刻 $t = \{0, 1, 2, \cdots\}$ での座標を $(r_{i.X}(t), r_{i.Y}(t))$ とし, 集合 $\mathbb{C}(t) = \{(r_{i.X}(t), r_{i.Y}(t))|0 \leq i \leq n-1\}$ を時刻 t における $\mathbb{G}$ 上の $\mathbb{R}$ の配置状況とする. また, ペアロボットの集合を $\mathbb{P} = \{p_k = (r_{2k}, r_{2k+1})|r_{2k}, r_{2k+1} \in \mathbb{R}, 0 \leq k \leq \frac{n}{2}-1\}$ とする. $\mathbb{T}$ の X 軸上に 1 辺を持つ正三角形 T の辺上及び内部の頂点の集合を $\mathbb{S} = \{s_1, s_2, \cdots, s_l\}$ とし,$s_i(\in \mathbb{S}, 1 \leq i \leq l)$ の座標を $(s_{i.X}, s_{i.Y})$ とする. ただし,l は T の X 軸上の辺に含まれる頂点数 L に対して以下の式を満たす離散値である.

$$l = \sum_{i=1}^{L} i$$

このとき, ロボットの台数 $n = 2l$ とする.

本問題の初期状況 $\mathbb{C}_{init}$ を $\mathbb{S}$ のすべての点に $Short$ 状態のペアロボットが存在する状況とする.(図 5)

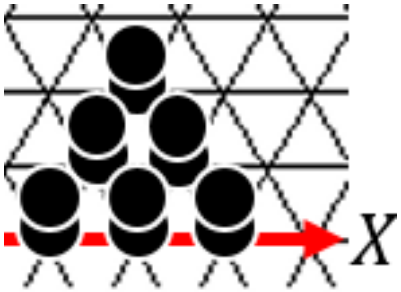


図 5　初期状況の例

そして, 目的状況 $\mathbb{C}_{goal}$ を以下のように定める.(図 6)

1. $\forall r_i, r_{i.Y}(goal) = 0$
2. 任意のロボット間のすべての頂点上にロボットが存在する

図 6　目的状況の例

このとき本問題を以下のように定める.

定義 1. 与えられた $\mathbb{C}_{init}$ を初期状況とし, 有限時間内に $\mathbb{C}_{goal}$ を満たす状況に到達させる問題を三角形直線化問題という.

4　提案アルゴリズム

本章では,3 章で紹介したモデルの基で, 三角形直線化問題を解くアルゴリズムを提案する. 本研究では, アルゴリズムの実行を以下の形式 (ガード付きアクション) で表す.

$$< label >::< guard > \rightarrow < action >$$

各アクションはラベル付けされており, 各アクションのガード $< guard >$ は Look の結果からなる論理式で表される. ロボットはガードが真の場合のみ命令文 $< action >$ を実行する.

4.1　アルゴリズムの基本戦略

最初に, 三角形直線化のための基本戦略を述べる. 提案アルゴリズムでは与えられた三角形の底辺 ($\mathbb{T}$ の X 軸上の辺) 以外のロボットを底辺を含む直線上 ($\mathbb{T}$ の X 軸上) に移動させることを考える. まず, 与えられた三角形の辺の内, 底辺ではない 1 辺 M 上に存在するロボットを底辺の延長線上に配置する. この際,M にも底辺にも属するロボットは底辺の延長線上を移動し, 底辺に属せずに M にのみ属するロボットは M 上を底辺の存在する方に向かって移動していく. これにより M 上のロボットはいづれすべて底辺の延長線上に到達する. この時,M 上に存在したロボットを除くと M より辺の長さが 1 だけ小さい正三角形が新たに出現する. その正三角形に対して先ほどと同じ操作を行う. 正三角形ができなくなるまでこれを繰り返すことで問題の解決を図った.(図 7)

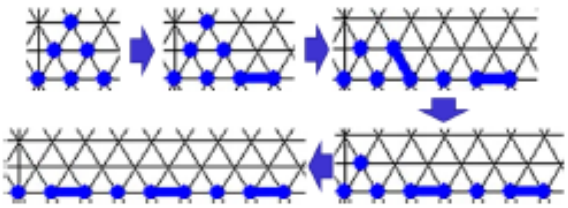


図 7　アルゴリズムの基本戦略

4.2 完全同期アルゴリズム

4.2.1 完全同期アルゴリズム

完全同期スケジューラの下で三角形直線化問題を解くアルゴリズムを Algorithm1 に示す.

Algorithm 1 ロボット $r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ 上の完全同期スケジューラの下での三角形直線化問題を解くアルゴリズム

1: **Predicates**:
2: $Occupy(r_{i.l}|l \in [0,6]) \equiv$
$\begin{cases} 0 : r_{i.l}\text{上に存在するロボットが 0 台} \\ 1 : r_{i.l}\text{上に存在するロボットが 1 台} \\ 2 : r_{i.l}\text{上に存在するロボットが 2 台以上} \end{cases}$
3: **Actions**:
4: Rule1(F) :: $r_{i.s} = Short \wedge r_{i.id} = high \wedge Occupy(r_{i.2}) = Occupy(r_{i.6}) = 0 \longrightarrow r_i$は $r_{i.1}$ に移動
5: Rule2(F)-1 :: $r_{i.s} = Short \wedge r_{i.id} = high \wedge Occupy(r_{i.1}) = 0 \wedge Occupy(r_{i.2}) = 1 \longrightarrow r_i$は $r_{i.2}$ に移動
6: Rule2(F)-2 :: $r_{i.s} = Long \wedge Occupy(r_{i.2}) = 1 \wedge Occupy(r_{i.5}) = 2 \wedge Occupy(r_{i.1}) = Occupy(r_{i.6}) = 0 \longrightarrow r_i$は $r_{i.2}$ に移動
7: Rule3(F)-1 :: $r_{i.s} = Short \wedge r_{i.id} = high \wedge Occupy(r_{i.1}) = 0 \wedge Occupy(r_{i.6}) = 1 \longrightarrow r_i$は $r_{i.6}$ に移動
8: Rule3(F)-2 :: $r_{i.s} = Long \wedge Occupy(r_{i.6}) = 1 \wedge Occupy(r_{i.3}) = 2 \wedge Occupy(r_{i.1}) = Occupy(r_{i.2}) = 0 \longrightarrow r_i$は $r_{i.6}$ に移動
9: Rule4(F) :: $r_{i.s} = Long \wedge Occupy(r_{i.5}) = 0 \wedge Occupy(r_{i.2}) = 1 \longrightarrow r_i$は $r_{i.2}$ に移動
10: Rule5(F) :: $r_{i.s} = Long \wedge Occupy(r_{i.3}) = 0 \wedge Occupy(r_{i.6}) = 1 \longrightarrow r_i$は $r_{i.6}$ に移動
11: Rule6(F)-1 :: $r_{i.s} = Short \wedge r_{i.id} = high \wedge Occupy(r_{i.1}) = 1 \wedge Occupy(r_{i.2}) = Occupy(r_{i.6}) = 0 \longrightarrow r_i$は $r_{i.1}$ に移動
12: Rule6(F)-2 :: $r_{i.s} = Long \wedge Occupy(r_{i.1}) = 1 \wedge Occupy(r_{i.4}) = 2 \wedge Occupy(r_{i.3}) = Occupy(r_{i.5}) = 0 \longrightarrow r_i$は $r_{i.1}$ に移動
13: Rule6(F)-3 :: $r_{i.s} = Long \wedge Occupy(r_{i.1}) = 1 \wedge Occupy(r_{i.3}) = 2 \wedge Occupy(r_{i.2}) = 0 \longrightarrow r_i$は $r_{i.1}$ に移動
14: Rule6(F)-4 :: $r_{i.s} = Long \wedge Occupy(r_{i.1}) = 1 \wedge Occupy(r_{i.5}) = 2 \wedge Occupy(r_{i.6}) = 0 \longrightarrow r_i$は $r_{i.1}$ に移動

Algorithm1 の各ルールについてそれぞれ図 8 に示す. 図 8 ではルールを実行するロボットを $r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ とし, 各ルールにおける実行前と実行後の r_i 及び r_i に観測されるロボットの配置状況を示している. 黒丸でルールを実行するペアロボットを示し, 灰色の丸で観測されたほかのペアロボットを示す. また, バツ印はその点にロボットが存在しないことを示している. すべてのルールにおいて図の右方向を X 軸正の向きとしている.

Rule1(F) では, ロボット r_i が $Short$ 状態で $high$ の識別子を持ち,$r_{i.1}$,$r_{i.2}$,$r_{i.6}$ にロボットが存在しない場合,$r_{i.1}$ に移動し $Long$ 状態に遷移する. Rule2(F)-1 では, ロボット r_i が $Short$ 状態で $high$ の識別子を持ち,$r_{i.1}$ にロボットが存在せず,$r_{i.2}$ に 1 台のみロボットが存在する場合,$r_{i.2}$ に移動し $Long$ 状態に遷移する. Rule2(F)-2 では, ロボット r_i が $Long$ 状態で,$r_{i.1}, r_{i.6}$ にロボットが存在せず,$r_{i.2}$ に 1 台,$r_{i.5}$ に 2 台のロボットが存在する場合,$r_{i.2}$ に移動し $Short$ 状態に遷移する. その他のルールに関しても同様に周囲の点を観測し, 決められた移動先へ移動する.

4.2.2 完全同期アルゴリズムの証明

$r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ の X 座標と Y 座標をそれぞれ $r_{i.X}, r_{i.Y}$ と表す. 初期状態において $\exists r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.id} = high, r_{i.Y} = 0), \forall r_j(\in \mathbb{R}, 0 \leq j \leq n-1, ri \neq rj), r_{i.X} \geq r_{j.X}$ を満たす r_i を r_H とする. すなわち, 初期状況において X 軸上に存在し $high$ の識別子を持ち最も X 座標の大きいロボットを r_H とする. 初期状況においてロボットの数は有限個なので r_H は唯一に決まる. 初期状況の内,$\forall r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.Y} \geq 0)$ のときを I_A,$\forall r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.Y} \geq 0)$ のときを I_Bと定義する.初期状況が I_Aのとき,$\exists r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.Y} = 1), \forall r_j(\in \mathbb{R}, 0 \leq j \leq n-1, r_{j.Y} = 1), r_{i.X} \geq r_{j.X}$を満たすロボットを r_{max} とする (2 台存在するときは $high$ の識別子を持つロボットとする).初期状況が I_B のとき ,$\exists r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.Y} = -1), \forall rj(\leq \mathbb{R}, 0 \leq j \leq n-1, r_{j.Y} = -1), r_{i.X} \geq r_{j.X}$ を満たすロボットを r_{max} とする (2 台存在するときは $high$ の識別子を持つロボットとする). すなわち, 初期状況が I_A のときは直線 $Y = 1$ 上の最も X 座標の大きいロボットを,I_B のときは直線 $Y = -1$ 上の最も X 座標の大きいロボットを r_{max} とする. ただし,2 台存在する場合は $high$ の識別子を持つロボットを r_{max} とする. 初期状況においてロボットの数は有限個なので r_{max} は唯一に決まる.

補題 1 すべてのルールにおいて移動方向にロボットの集合が切断されることはない.

証明 Rule1(F),Rule2(F)-1,Rule3(F)-1,Rule6(F)-1 は $Short$ から $Long$ へペアロボットが状態を変化させるルールである. もともと存在していた点にどちらかのペアがルール実行後も残るため, ロボットの集合が切断されるような移動は起こらない. Rule2(F)-2 では, ルール実行前に $r_{i.5}$ に存在するロボットが Rule2(F)-1 の適用条件を満たしているため,r_i が $r_{i.2}$ に移動するのと同時に他のロボットがルール実行前に r_i が存在していた点に移動する. したがってこのルールによる移動方向のロボットの切断は起こらない. Rule3(F)-2 では, ルール実行前に $r_{i.3}$ に存在するロボットが Rule3(F)-1 の適用条件を満たしているため,r_i が $r_{i.6}$ に移動するのと同時に他のロボットがルール実行前に r_i が存在していた点に移動する. したがってこのルールによる移動方向のロボットの切

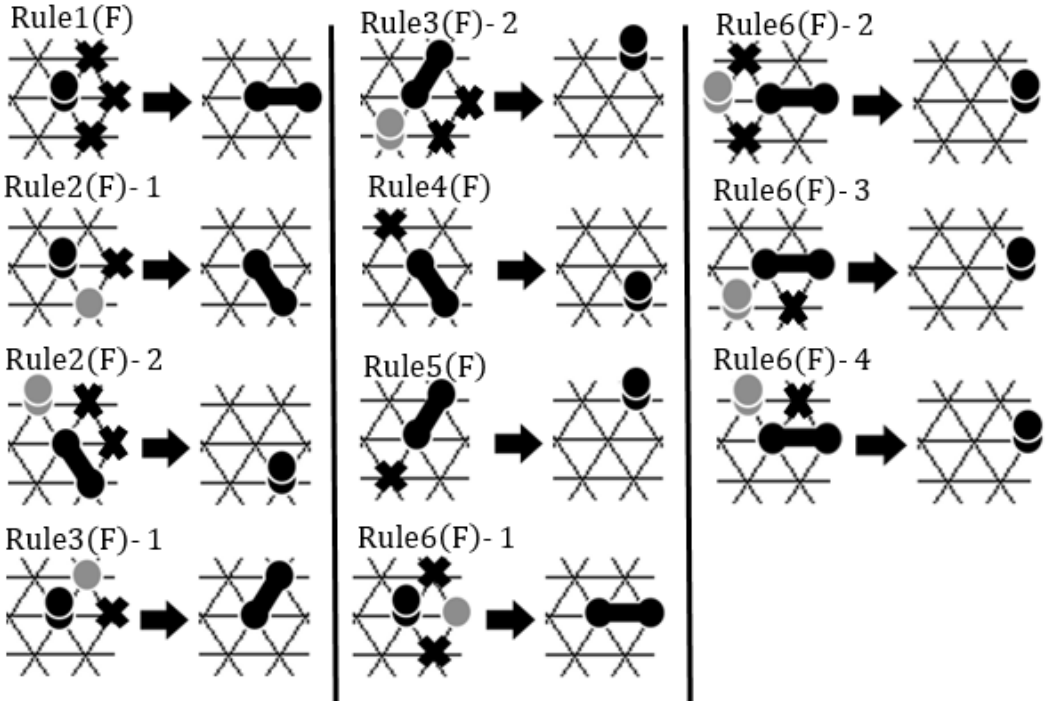


図 8　完全同期アルゴリズムのルール

断は起こらない. Rule4(F),5(F) では, ルール実行前において,r_i の隣接頂点の内. 移動方向の点に他のペアロボットが存在しないのは明らかである. したがってこれらのルールの実行によってロボットの移動方向の切断が起こることはない. Rule6(F)-2,Rule6(F)-3,Rule6(F)-4 ではそれぞれ $r_{i.4}$ に存在するロボットが Rule6(F)-1 の適用条件を,$r_{i.3}$ に存在するロボットが Rule3(F)-1 の適用条件を,$r_{i.5}$ に存在するロボットが Rule2(F)-1 の適用条件を満たしているため,,r_i が $r_{i.1}$ に移動するのと同時に他のロボットがルール実行前に r_i が存在していた点に移動する. したがってこれらのルールによる移動方向のロボットの切断は起こらない. 以上より, すべてのルールにおいて移動方向にロボットの集合が切断されることはない. □

補題 2　r_{max} は有限時間内に X 軸に移動し r_{max} でなくなる. また, 一度 X 軸上の点に移動したロボットは二度と X 軸の外には出られない.

証明　補題 1 より I_A のときは $r_{max.2}$ に,I_B のときは $r_{max.6}$ にロボットが存在する.r_H は Rule1,6(F) のいづれかを常に実行し *Short* と *Long* を交互に遷移し, これと補題 1 より I_A のときは $r_{max.2}$,I_B のときは $r_{max.6}$ に存在するロボットは有限時間内に *Long* になる.したがって r_{max} は有限時間内に $r_{max.2}$ または $r_{max.6}$ に *Long* のロボットを観測し Rule2,3,4,5(F) のいずれかを実行し X 軸上に移動する.Rule1(F) は Y 座標の変わらない移動であり, そのほかのルールはすべてロボットを観測した点への移動となっている. したがって補題 1 より一度 X 軸上の点に移動したロボットは二度と X 軸の外に出ることはない.

□

補題 3　すべてのロボットは目的状況になっていなければ必ず r_{max} が存在する.

証明　初期状況であれば, 明らかに r_{max} が存在する. さらに,$r_{max.4}$ の方向の直線上 ($Y = 1$ または $Y = -1$) にはロボットが全て連結状態であり,r_{max} のロボットが存在する限り, 全員動作しない. ある状況において,r_{max} が存在するのであれば補題 2 より r_{max} は有限時間内に X 軸に移動し r_{max} でなくなる. この動作により新たな r_{max} が存在しなくなるのであれば,$r_{max.3}$ または $r_{max.5}$ 方向の直線上にロボットが 1 つもないときである.かつ, その動作により $r_{max.4}$に存在するロボットが r_{max}となる.$r_{max.4}$にロボットが存在しない場合は X 軸上以外にロボットが 1 つも存在しなくなり, これは目的状況である.

□

補題 1,2,3 から以下の定理が成り立つ.

定理　完全同期スケジューラの下で Algorithm1 は三角形直線化問題を解く.

4.3　半同期アルゴリズム

4.3.1　半同期アルゴリズム

半同期スケジューラの下で三角形直線化問題を解くアルゴリズムを Algorithm2 に示す.

Algorithm2 の各ルールについてそれぞれ図 9 に示す. 図 9 ではルールを実行するロボットを $r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ とし, 各ルールにおける実行前と実行後の r_i 及び r_i に観測されるロボットの配置状況を示している. 黒丸でルールを実行するペアロボットを示し, 灰色の丸で観測されたほかのペアロボットを示す. また, バツ印はその点にロボットが存在しないことを示している. すべての

Algorithm 2 ロボット $r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ 上の半同期スケジューラの下での三角形直線化問題を解くアルゴリズム

1: **Predicates**:
2: $Occupy(r_{i.l}|l \in [0,6]) \equiv$
$$\begin{cases} 0 : r_{i.l}\text{上に存在するロボットが}0\text{台} \\ 1 : r_{i.l}\text{上に存在するロボットが}1\text{台} \\ 2 : r_{i.l}\text{上に存在するロボットが}2\text{台以上} \end{cases}$$
3: **Actions**:
4: Rule1(S) :: $r_{i.s} = Short \wedge r_{i.id} = high \wedge Occupy(r_{i.1}) = Occupy(r_{i.2}) = 0 \longrightarrow r_i$は $r_{i.1}$ に移動
5: Rule2(S)-1 :: $r_{i.s} = Short \wedge r_{i.id} = high \wedge Occupy(r_{i.1}) = 0 \wedge Occupy(r_{i.2}) = 1 \longrightarrow r_i$は $r_{i.2}$ に移動
6: Rule2(S)-2 :: $r_{i.s} = Long \wedge Occupy(r_{i.0}) = 2 \wedge Occupy(r_{i.1}) = Occupy(r_{i.6}) = 0 \wedge Occupy(r_{i.2}) = 1 \longrightarrow r_i$は $r_{i.2}$ に移動
7: Rule3(S)-1 :: $r_{i.s} = Short \wedge r_{i.id} = high \wedge Occupy(r_{i.1}) = 0 \wedge Occupy(r_{i.6}) = 1 \longrightarrow r_i$は $r_{i.6}$ に移動
8: Rule3(S)-2 :: $r_{i.s} = Long \wedge Occupy(r_{i.0}) = 2 \wedge Occupy(r_{i.1}) = Occupy(r_{i.2}) = 0 \wedge Occupy(r_{i.6}) = 1 \longrightarrow r_i$は $r_{i.6}$ に移動
9: Rule4(S) :: $r_{i.s} = Short \wedge r_{i.id} = high \wedge Occupy(r_{i.1}) = 1 \wedge Occupy(r_{i.2}) = Occupy(r_{i.6}) = 0 \longrightarrow r_i$は $r_{i.1}$ に移動
10: Rule5(S) :: $r_{i.s} = Long \wedge Occupy(r_{i.0}) = 2 \wedge Occupy(r_{i.1}) = 1 \wedge Occupy(r_{i.2}) = Occupy(r_{i.6}) = 0 \longrightarrow r_i$は $r_{i.1}$ に移動
11: Rule6(S) :: $r_{i.s} = Long \wedge Occupy(r_{i.0}) = 1 \wedge Occupy(r_{i.2}) = 1 \wedge Occupy(r_{i.5}) = 0 \longrightarrow r_i$は $r_{i.2}$ に移動
12: Rule7(S) :: $r_{i.s} = Long \wedge Occupy(r_{i.0}) = 1 \wedge Occupy(r_{i.6}) = 1 \wedge Occupy(r_{i.3}) = 0 \longrightarrow r_i$は $r_{i.6}$ に移動

ルールにおいて図の右方向を X 軸正の向きとしている.

Rule1(S) では, ロボット r_i が $Short$ 状態で $high$ の識別子を持ち,$r_{i.1}$,$r_{i.2}$,$r_{i.6}$ にロボットが存在しない場合,$r_{i.1}$ に移動し $Long$ 状態に遷移する. Rule2(S)-1 では, ロボット r_i が $Short$ 状態で $high$ の識別子を持ち,$r_{i.1}$ にロボットが存在せず,$r_{i.2}$ に 1 台のみロボットが存在する場合,$r_{i.2}$ に移動し $Long$ 状態に遷移する. Rule2(S)-2 では, ロボット r_i が $Long$ 状態で,$r_{i.0}$ にロボットが 2 台存在し,$r_{i.1}$ にロボットが存在せず,$r_{i.2}$ に 1 台のみロボットが存在する場合,$r_{i.2}$ に移動し $Short$ 状態に遷移する. その他のルールに関しても同様に周囲の点を観測し, 決められた移動先へ移動する.

4.3.2 半同期アルゴリズムの証明

$r_i(\in \mathbb{R}, 0 \leq i \leq n-1)$ の X 座標と Y 座標をそれぞれ $r_{i.X}, r_{i.Y}$ と表す. 初期状態において $\exists r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.id} = high, r_{i.Y} = 0), \forall r_j(\in \mathbb{R}, 0 \leq j \leq n-1, ri \neq rj), r_{i.X} \geq r_{j.X}$ を満たす r_i を r_H とする. すなわち, 初期状況において X 軸上に存在し $high$ の識別子を持ち最も X 座標の大きいロボットを r_H とする. 初期状況においてロボットの数は有限個なので r_H は唯一に決まる. 初期状況の内,$\forall r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.Y} \geq 0)$ のときを I_A,$\forall r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.Y} \geq 0)$ のときを I_Bと定義する.初期状況が I_Aのとき,$\exists r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.Y} = 1), \forall r_j(\in \mathbb{R}, 0 \leq j \leq n-1, r_{j.Y} = 1), r_{i.X} \geq r_{j.X}$を満たすロボットを r_{max} とする (2 台存在するときは $high$ の識別子を持つロボットとする).初期状況が I_B のとき ,$\exists r_i(\in \mathbb{R}, 0 \leq i \leq n-1, r_{i.Y} = -1), \forall rj(\leq \mathbb{R}, 0 \leq j \leq n-1, r_{j.Y} = -1), r_{i.X} \geq r_{j.X}$ を満たすロボットを r_{max} とする (2 台存在するときは $high$ の識別子を持つロボットとする). すなわち, 初期状況が I_A のときは直線 $Y = 1$ 上の最も X 座標の大きいロボットを,I_B のときは直線 $Y = -1$ 上の最も X 座標の大きいロボットを r_{max} とする. ただし,2 台存在する場合は $high$ の識別子を持つロボットを r_{max} とする. 初期状況においてロボットの数は有限個なので r_{max} は唯一に決まる.

補題 4 すべてのルールにおいて移動方向にロボットの集合が切断されることはない.

証明 Rule1(S),Rule2(S)-1,Rule3(S)-1,Rule4(S) は $Short$ から $Long$ へペアロボットが状態を変化させるルールである. もともと存在していた点にどちらかのペアがルール実行後も残るため, ロボットの集合が切断されるような移動は起こらない. Rule2(S)-2,Rule3(S)-2,Rule5(S) では, ルール実行前に $r_{i.0}$ に他のペアに属する $Long$ 状態のロボット r_j が存在する. しかし,r_j のペアは $r_{i.3}, r_{i.4}, r_{i.5}$ のいづれかにペアのロボットが存在し, いづれの場合もいかなるルールの適用条件も満たさない. したがって r_j はこのルールを実行するサイクル内には移動しないので, これらのルールの実行による移動方向への切断は起きない. Rule6(S),Rule7(S) では, ルール実行前において,r_i の隣接頂点の内. 移動方向の点に他の

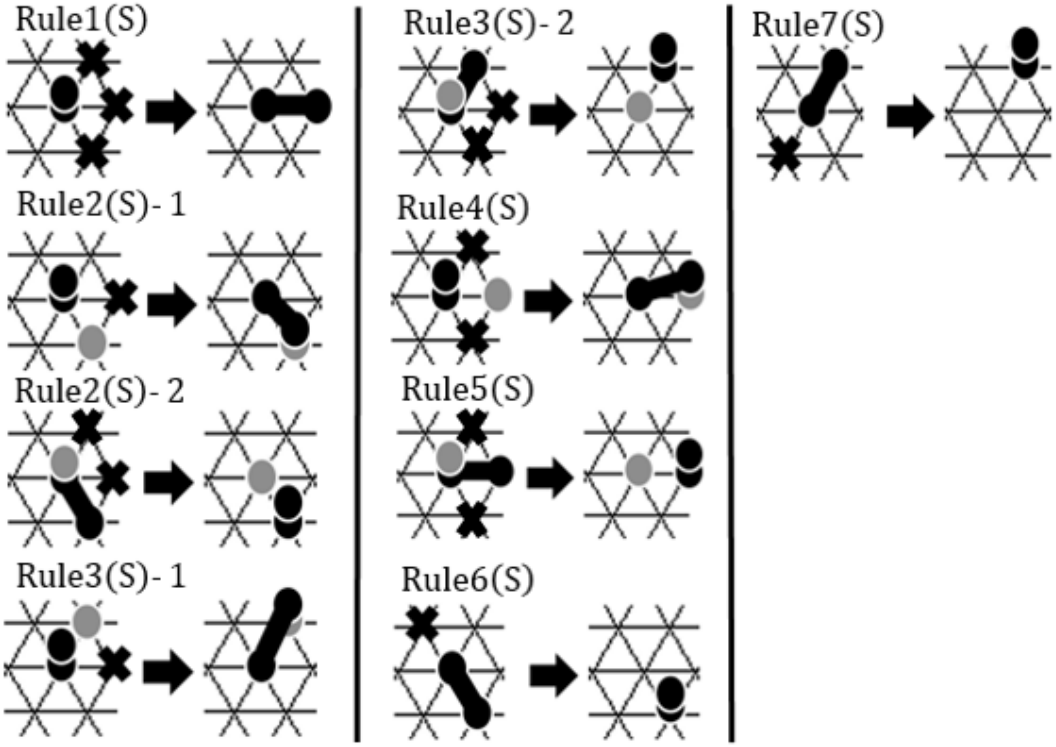


図 9　半同期アルゴリズムのルール

ペアロボットが存在しないのは明らかである. したがってこれらのルールの実行によってロボットの移動方向の切断が起こることはない. 以上より, すべてのルールにおいて移動方向にロボットの集合が切断されることはない. □

補題 5　r_{max} は有限時間内に X 軸に移動し r_{max} でなくなる. また, 一度 X 軸上の点に移動したロボットは二度と X 軸の外には出られない.

証明　補題 4 より I_A のときは $r_{max.2}$ に,I_B のときは $r_{max.6}$ にロボットが存在する.r_H は Rule1,5(S) のいづれかを常に実行し *Short* と *Long* を交互に遷移し, これと補題 4 より I_A のときは $r_{max.2}$,I_B のときは $r_{max.6}$ に存在するロボットは有限時間内に *Long* になる.したがって r_{max} は有限時間内に $r_{max.2}$ または $r_{max.6}$ に *Long* のロボットを観測し Rule2,3,6,7(S) のいずれかを実行し X 軸上に移動する.Rule1(S) は Y 座標の変わらない移動であり, そのほかのルールはすべてロボットを観測した点への移動となっている. したがって補題 4 より一度 X 軸上の点に移動したロボットは二度と X 軸の外に出ることはない. □

補題 6　すべてのロボットは目的状況になっていなければ必ず r_{max} が存在する.

証明　初期状況であれば, 明らかに r_{max} が存在する. さらに,$r_{max.4}$ の方向の直線上 ($Y = 1$ または $Y = -1$) にはロボットが全て連結状態であり,r_{max} のロボットが存在する限り, 全員動作しない. ある状況において,r_{max} が存在するのであれば補題 5 より r_{max} は有限時間内に X 軸に移動し r_{max} でなくなる. この動作により新たな r_{max} が存在しなくなるのであれば,$r_{max.3}$ または $r_{max.5}$ 方向の直線上にロボットが 1 つもないときである.かつ, その動作により $r_{max.4}$ に存在するロボットが r_{max} となる.$r_{max.4}$ にロボットが存在しない場合は X 軸上以外にロボットが 1 つも存在しなくなり, これは目的状況である. □

補題 4,5,6 から以下の定理が成り立つ.

定理　半同期スケジューラの下で Algorithm2 は三角形直線化問題を解く.

5　おわりに

本研究では, ペアロボットシステムモデルにおいて各ロボットの視野範囲を隣接頂点までとし, 共通座標系なしで各ロボットの座標系の 1 軸の向きと方向に共通知識を与えたときに, 正三角形を形成しているペアロボット群を一直線上に整列させる問題 (三角形直線化問題) を完全同期及び半同期スケジューラの下で解くアルゴリズムを提案した. しかし, 識別子 $r_{i.id}$ が対称性を崩す以上にロボットの進行方向の情報も与えてしまっているなどといった課題もあり, より良いモデルを提案していく必要がある.

参考文献

[1] K. C. Cheung, E. D. Demaine, J. R. Bachrach, andS. Grifith, "Programmable assembly with universallyfoldable strings (moteins)," *IEEE Transactions onRobotics*, 27(4):718-729, 2011.

[2] R. Nagpal, A. Kondacs, and C. Chang, "Programming methodology for biologically-inspired self-assembling systems," Technical report, AAAI Spring Symposium on Computational Synthesis, 2003.

[3] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman, "Design and self-assembly of two-dimensional dna crystals," *Nature*, 394(6693):539-544, 1998.

[4] D. Woods, H.-L. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin, "Active self-assembly of algorithmic shapes and patterns in polylogarithmic time," In *ITCS*, pages 353-354, 2013.

[5] G.Prencipe, "On the feasibility of gathering by autonomous mobile robots," in *SIROCCO* 2005,*LNCS 3499*,pp.246-.

[6] P.Flocchini, G.Prencipe, N.Santoro, and P.Widmayer, "Gathering of asynchronous robots with limited visibility," *TCS 2005*, vol.337,no.1-3,pp.147-168.

[7] R.Klasing, E.Markou, and A.Pelc, "Gathering asynchronous oblivious mobile robots in a ring," *TCS*, vol.390,no.1,pp.27-39,2008.

[8] G.D'Angelo, G.Stefano, R.Klasing, and A.Navarra, "Gathering of robots on anonymous grids without multiplicity detection," in *SIROCCO* 2012 , *LNCS* 7355, pp.327-338.

[9] M.Cieliebak, P.Flocchini, G.Prencipe, and N.Santoro, "Solving the robots gathering problem," *LNCS* 2719,pp.1181-,(2003).

[10] 山田涼斗, 金鎔煥, 片山喜章, 和田幸一, "ペアロボットモデルにおける直進行進と物体被覆アルゴリズムについて," 第 14 回情報科学ワークショップ,2018.

Adaptive Path Finding for Thousands of Interacting Agents

Keisuke Okumura[1], Yasumasa Tamura[1], Xavier Défago[1]

[1] School of Computing, Tokyo Institute of Technology
okumura.k@coord.c.titech.ac.jp, {tamura, defago}@c.titech.ac.jp

The Multi-agent Path Finding (MAPF) problem consists in all agents having to move to their own destinations while avoiding collisions [10]. MAPF is now receiving a lot of attention due to its high practicality, e.g., traffic control [1], automated warehouse [12] or airport surface operation [4], etc. Numerous optimal solvers for MAPF have been proposed so far, e.g., search-based solvers [2]. However, finding an optimal solution is known as NP-hard [13] and lacks of scalability for the number of agents. In realistic applications to the problem, MAPF must be solved iteratively, studied as *lifelong* MAPF [3] or *online* MAPF [11]. Due to the above reasons, sub-optimal solvers such as prioritized planning [9] are practical to use.

In this talk, we first define an abstract model to address the behaviors of multiple moving agents, called *iterative* MAPF, which generalizes classical MAPF and lifelong MAPF. Then, we present a novel approach for iterative MAPF, that we call Priority Inheritance with Backtracking (PIBT) [5]. PIBT relies on prioritized planning with a single time window. *Priority inheritance*, originally considered in resource scheduling problems [8], is introduced to deal effectively with priority inversion in path adjustment. Priority inheritance is accompanied by *backtracking* protocol to prevent being agent stuck. When the environment is a graph such that all pairs of adjacent nodes belong to a simple cycle of length 3 or more (e.g., biconnected), regardless of the number of agents, PIBT ensures *reachability*, i.e., all agents are guaranteed to reach their own destinations within finite time after being given. Our implementation of PIBT is designed to be decentralized without global communication. Experimental results over various scenarios confirm that PIBT is adequate both for finding paths in large environments with many agents, as well as for conveying packages in an automated warehouse.

We also introduce two variants of PIBT. First, Windowed PIBT(winPIBT) [7] is a generalized algorithm of PIBT with respect to the time window. PIBT plans the paths of all agents one step at a time, i.e., the time window size is just one, and this locality causes inefficient path planning in some cases. winPIBT expands the time window by enabling retroactive priority inheritance while ensuring the reachability. Empirical results show that winPIBT mitigates livelock situations occurring in PIBT, and plans more efficient paths depending on the window size, especially in graphs containing narrow passages. Second, PIBT with temporally inflation (PIBT+TI) [6] aims at extending target graphs, e.g., graphs containing tree structures, by introducing temporary inflation of agent priorities. Although we have not shown the reachability of PIBT+TI, empirical results demonstrate that PIBT+TI mitigates deadlock situations occurring in PIBT while keeping path efficiency.

Finally, we discuss several future directions of PIBT including robot implementations, and make effort to build up the way of adaptive path finding for thousands of interacting agents.

REFERENCES

[1] K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, 31:591–656, 2008.

[2] A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Tenth Annual Symposium on Combinatorial Search*, 2017.

[3] H. Ma, J. Li, T. Kumar, and S. Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 837–845. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[4] R. Morris, C. S. Pasareanu, K. S. Luckow, W. Malik, H. Ma, T. S. Kumar, and S. Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop: Planning for Hybrid Systems*, 2016.

[5] K. Okumura, M. Machida, X. Défago, and Y. Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 535–542, 7 2019.

[6] K. Okumura, Y. Tamura, and X. Défago. Prioritized planning with temporary inflation for iterative multi-agent path finding. In *the Eighteenth Fourum on Information Technology (FIT)*, 2019.

[7] K. Okumura, Y. Tamura, and X. Défago. winpibt: Extended prioritized algorithm for iterative multi-agent path finding. *arXiv preprint arXiv:1905.10149*, 2019.

[8] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on computers*, 39(9):1175–1185, 1990.

[9] D. Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[10] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. *arXiv preprint arXiv:1906.08291*, 2019.

[11] J. Švancara, M. Vlk, R. Stern, D. Atzmon, and R. Barták. Online multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7732–7739, 2019.

[12] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9, 2008.

[13] J. Yu and S. M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, 2013.

Two-player Competitive Diffusion Game: Graph Classes and the Existence of a Nash Equilibrium

Naoka Fukuzono[1], Tesshu Hanaka[2], Hironori Kiya[1], Hirotaka Ono[1], and Ryogo Yamaguchi[3]

[1] Nagoya University
[2] Chuo University
[3] Development Bank of Japan

Abstract. The competitive diffusion game is a game-theoretic model of information spreading on a graph proposed by Alon et al. (2010). In the model, a player chooses an initial vertex of the graph, from which information by the player spreads through the edges connected with the initial vertex. If a vertex that is not yet influenced by any information receives information by a player, it is influenced by the information and it diffuses it to adjacent vertices. A vertex that simultaneously receives two or more types of information does not diffuse any type of information from then on. The objective of a player is to maximize the number of vertices influenced by the player's information. In this paper, we investigate the existence of a pure Nash equilibrium of the two-player competitive diffusion game on chordal and its related graphs. We show that a pure Nash equilibrium always exists on block graphs, split graphs and interval graphs, all of which are well-known subclasses of chordal graphs. On the other hand, we show that there is an instance with no pure Nash equilibrium on (strongly) chordal graphs; the boundary of the existence of a pure Nash equilibrium is found.

Keywords: Nash equilibrium · Competitive diffusion game · Algorithmic game theory · Chordal graph.

1 Introduction

The competitive diffusion game is a game-theoretic model of information spreading on a graph proposed by Alon et al. [1]. It is introduced in order to study several competitive diffusion phenomena on social network services (SNS), such as Facebook and Twitter. For example, viral marketing is a typical commercial activity utilizing information diffusion phenomena on a social network. A game-theoretical setting happens when several companies want to sell interoperable products via viral marketing.

In the model, each player has its own information and wants to spread it to vertices in a graph. To this end, a player chooses an initial vertex of the graph, from which information by the player spreads through the edges connected with the initial vertex. If a vertex that is not yet influenced by any information receives information by a player, it is influenced by the information and it diffuses

the information to adjacent vertices. A vertex that simultaneously receives two or more types of information by multiple players does not diffuse any type of information from then on. The objective of a player is to maximize the number of vertices affected by the player's information. These settings are interpreted in real world situations as follows: A graph is a social network and each vertex represents a person (potential customer) and an edge represents that two persons corresponding end vertices are friends in an SNS. Players are commercial companies that want to sell interoperable products via viral marketing. Each company asks a person on the SNS to advertise its own product by paying some amount of money. The person receiving money recommends the product of the company to his/her friends. A person receives a recommendation of a product from a friend, he/she decides to buy the product and newly recommends the product of the company to his/her friends. Sometimes a person simultaneously receives two types of recommendations. Then he/she gets confused, and he/she does not buy any of the products and recommend anything. This is a simplest model and we can consider more generalized models.

In analyses of game-theoretic models, one of typical approaches is to focus on Nash equilibria. This is because finding a Nash equilibrium is related to predict behaviours of rational players. Although it is known that there exists a mixed-strategy Nash equilibrium for every finite game, a pure Nash equilibrium does not always exist. In fact, in the competitive diffusion game, there is a graph under which no pure Nash equilibrium exists even for the two-player case [1], whereas a graph in some restricted graph classes such as cycles always has a pure Nash equilibrium for any number of players [3]. If a game has a pure Nash equilibrium, it implies that it is relatively easy to analyze. From such a viewpoint, several studies try to find classes of graphs under which a pure Nash equilibrium always exists. For more details, see the following subsection.

1.1 Related work

There are many studies that focus on the existence of a pure Nash equilibrium of the two-player competitive diffusion game. For example, Alon et al. give a graph with diameter 3 has no pure Nash equilibrium [1]. Takehara et al. give a stronger example, a graph with diameter 2 has no pure Nash equilibrium [14]. On the other hand, Small and Mason show that a pure Nash equilibrium always exists on trees [12]. Roshabin shows that a pure Nash equilibrium always exists on cycles and grid graphs [11], and Sukenari et al. shows that a pure Nash equilibrium always exists on torus grid graphs [13]. These results are about the two-player competitive diffusion game. For three or more players, the situation is different. For example, in most of the cases, a path always has a pure Nash equilibrium. The exception is the case where the number of players is 3 and the number of vertices is at least 6. On the other hand, a cycle always has a pure Nash equilibrium for the case where the number of players and the number of vertices are arbitrary [3].

If the number k of players is bounded by a constant, it can be done in polynomial time to check whether a given graph has a pure Nash equilibrium or

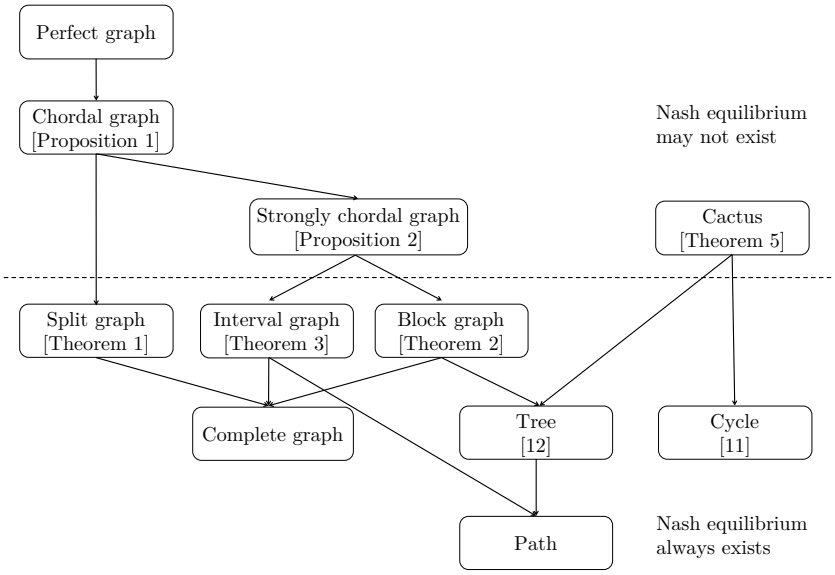


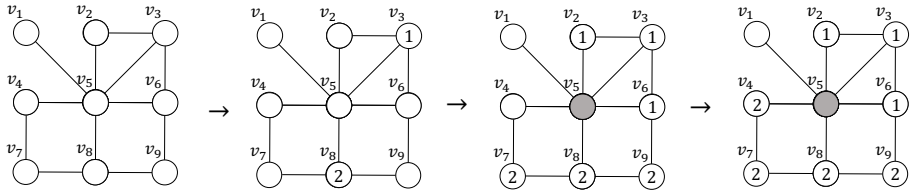
Fig. 1. Graph classes and the existence of a pure Nash equilibrium. Connections between two graph classes imply that the above one is a super class of the below one.

not, because the number of combinations of strategies is $O(n^k)$, where n is the number of vertices. On the other hand, it is not trivial to check the existence of a pure Nash equilibrium for general k. Etesami and Basar show that the decision problem of the existence a pure Nash equilibrium for general k is NP-complete [5]. Furthermore, Ito et al. show that the decision problem of the existence a pure Nash equilibrium is W[1]-hard when it is parameterized by k [8].

1.2 Our results

In this paper, we investigate the existence of a pure Nash equilibrium of the two-player competitive diffusion game on chordal and its related graphs. A graph is called *chordal* if every induced cycle in the graph should have exactly three vertices. The class of chordal graphs is a well-known graph class in many research fields, and they are also called rigid circuit graphs or triangulated graphs. Particularly in the algorithm theory, it is considered very important, because many NP-hard optimization problems in general graphs can be solved in polynomial time if the input graph is chordal. This is a motivation that we focus on chordal graphs. Furthermore, chordal graphs and its related graph classes are intensively and extensively studied, and there are many well-known graph classes. For example, trees are also chordal.

We obtain the following results: We show that a pure Nash equilibrium always exists on block graphs, split graphs and interval graphs, all of which are well-known subclasses of chordal graphs. In particular, block graphs is a super class



(a) Initial state (b) Time 1 (b) Time 2 (b) Time 3

Fig. 2. An example how two-player competitive diffusion game goes. Vertices with 1 and 2 stand for vertices dominated by p_1 and p_2, respectively. White vertices are undominated vertices and a grey vertex is a neutral vertex. (a) A graph is an initial state. (b) At time 1, p_1 chooses v_3 and p_2 chooses v_8. (c) At time 2, v_2 and v_6 are dominated by p_1 and v_7 and v_9 are dominated by p_2. Vertex v_5 becomes a neutral vertex. (d) At time 3, v_4 is dominated by p_2. Since no player can dominate a vertex any more, the game ends. In the end of the game, v_1 is an undominated vertex. The utility of p_1 is $U_1(v_3, v_8) = 3$ and the utility of p_2 is $U_2(v_3, v_8) = 4$.

of trees. On the other hand, we show that there is a (strongly) chordal graph that has no pure Nash equilibrium; the boundary of the existence of a pure Nash equilibrium is found. The results are summarized in Figure 1.

The rest of the paper is organized as follows. In Section 2, we define several notations and terminology, and introduce graph classes. Section 3 is the main part of this paper. We show that a pure Nash equilibrium always exists on block graphs, split graphs and interval graphs, and give a (strongly) chordal graph that has no pure Nash equilibrium. Finally, we present the existence of a pure Nash equilibrium for some related graph classes and settings in Section 4.

2 Preliminaries

In this paper, we use the standard graph notations. Let $G = (V, E)$ be an undirected conneted graph where $|V| = n$ and $|E| = m$. For $V' \subseteq V$, we denote by $G[V']$ a subgraph induced by V'. We denote by $N(v)$ the set of neighbors of v. A vertex set C is called a *clique* if $G[C]$ is a complete graph. Moreover, a clique C is *maximal* if there is no clique C' such that $C \subseteq C'$.

2.1 Competitive diffusion game

Let p_1 and p_2 be players 1 and 2, respectively. Also, let $G = (V, E)$ be an undirected connected graph. Then the two-player competitive diffusion game on G proceeds as follows (see also Figure 2).

Time 1. Each player chooses one vertex $v \in V$. The vertex v is called an *initial vertex*. If v is chosen by only one player, we say v is *dominated* by p. Otherwise, that is, if two players choose v, the vertex v is a *neutral* vertex. In

the subsequent time, no player can dominate the neutral vertex. A vertex is called *undominated* if it is neither a dominated vertex nor a neutral vertex.

Time t **(**$t \geq 2$**).** A vertex $v \in V$ is dominated by a player p at time t if (i) v is neither neutral nor dominated by any player by time $(t-1)$, and (ii) v has a neighbor dominated by p, but does not have a neighbor dominated by the other player p'. If v satisfies (i) and has both neighbors of p and p', then v becomes a neutral vertex at time t. When no player can dominate a vertex any more, the game ends.

When player p chooses a vertex $s \in V$ at time 1, we call a vertex s the *strategy* of p. For two players p_1 and p_2, a *strategy profile* $\mathbf{s} = (s_1, s_2)$ is a pair of strategies of p_1 and p_2. If p_1 changes the strategy s_1 to s_1', we denote it by $(s_1, s_2) \to (s_1', s_2)$. Similarly, if p_2 changes the strategy s_2 to s_2', we denote it by $(s_1, s_2) \to (s_1, s_2')$. For a strategy profile (s_1, s_2), we say s_i *dominates* vertices dominated by p_i in the end of a game. For a strategy profile $\mathbf{s}$, the utility $U_i(\mathbf{s})$ of p_i is the sum of vertices dominated by p_i at the end of a game. In Figure 2, the utility of p_1 is $U_1(v_3, v_8) = 3$ and the utility of p_2 is $U_2(v_3, v_8) = 4$.

Then we define a pure Nash equilibrium in the two-player competitive diffusion game.

Definition 1. *A strategy profile* $\mathbf{s} = (s_1, s_2)$ *is called a* pure Nash equilibrium *if there is no vertex* $v \in V$ *such that* $U_1(v, s_2) > U_1(s_1, s_2)$ *or* $U_2(s_1, v) > U_1(s_1, s_2)$*, that is, if no player can increase the utility by changing the strategy.*

We call the two-player competitive diffusion game *2-CDG* for short. Also, we simply use term "Nash equilibrium" instead of pure Nash equilibrium" from here on. If both p_1 and p_2 choose $v \in V$ as initial vertices, the utilities of p_1 and p_2 are 0 because v is a neutral vertex and other vertices are undominated in the end of a game. Then, a player has an incentive to change the strategy from v to another vertex because if two players choose different vertices, the utilities of them are at least 1. This implies that the strategy profile (v, v) for any $v \in V$ cannot be a Nash equilibrium. Thus, we suppose that two players choose different vertices as initial vertices.

2.2 Graph classes

In this subsection, we define several graph classes. A graph $G = (V, E)$ is a *chordal graph* if every cycle of length at least 4 has a chord, or equivalently every induced cycle has exactly 3 vertices [4]. A graph $G = (V, E)$ is a *strongly chordal graph* if it is chordal graph and includes no n-sun (for $n \geq 3$) as an induced subgraph [6]. A cycle is a graph with closed circuits. A graph $G = (V, E)$ is a *block graph* if all the two connected components are cliques [7]. By the definition of a block graph, a tree is a block graph. A graph $G = (V, E)$ is a *split graph* if V can be partitioned in an independent set I and a clique C [10]. A graph $G = (V, E)$ is a *interval graph* if it has an intersection model consisting of intervals on a real line corresponding to a vertex such that there is an edge in G if and only if two lines are intersect [9]. For more information about graph classes, see [2].

3 The existence of a Nash equilibrium

In this section, we investigate the existence of Nash equilibrium in 2-CDG.

3.1 Split graph

In this section, we show a Nash equilibrium always exist in two-player competitive diffusion game on a split graph.

Theorem 1. *In 2-CDG on any split graph, a Nash equilibrium always exists. Furthermore, an equilibrium can be found in linear time.*

To show Theorem 1, we prove the following three lemmas.

Lemma 1. *Let $G = (C \cup I, E)$ be a split graph where C forms a clique and I is an independent set. If the strategy profile of p_1 and p_2 is (u, v) where $u, v \in C$, the utilities of p_1 and p_2 are $U_1(u, v) = |N(u)| - |N(u) \cap N(v)| + 1$ and $U_2(u, v) = |N(v)| - |N(v) \cap N(u)| + 1$, respectively.*

Proof. First, we can observe that every $w \in C \backslash \{u, v\}$ becomes a neutral vertex because C forms a clique. For a vertex w in I, we consider the following cases: (a) w is adjacent to both u and v, (b) w is adjacent to u but not adjacent to v, (c) w is not adjacent to u but adjacent to v, and (d) w is adjacent to neither vertex u nor v.

In case (a), w becomes a neutral vertex because it is adjacent to both vertex u and v. In case (b), w is dominated by p_1. In case (c), w is dominated by p_2. In case (d), w is dominated by neither p_1 nor p_2 because every vertex in C adjacent to w becomes a neutral vertex. Therefore, the utility of p_1 is $U_1(u, v) = |N(u)| - |N(u) \cap N(v)| + 1$. This represents the number of vertices satisfying case (b) plus u. Similarly, we have $U_2(u, v) = |N(v)| - |N(v) \cap N(u)| + 1$. □

Lemma 2. *On any split graph $G = (C \cup I, E)$, if both p_1 and p_2 choose vertices in C, there is no strategy to choose a vertex in I that increases the utilities of p_1 and p_2.*

Proof. If p_1 changes the strategy from u to $x \in I$, the utility of p_1 becomes $U_1(x, v) = 1 \leq U_1(u, v)$ by Lemma 1. Also, if p_2 changes the strategy from v to $y \in I$, the utility of p_2 becomes $U_2(u, y) = 1 \leq U_1(u, v)$. Therefore, each player does not change the strategy to choose a vertex in I. □

From Lemmas 1 and 2, we obtain Lemma 3 that concludes Theorem 1.

Lemma 3. *There is a Nash equilibrium (u, v) for $u, v \in C$.*

Proof. For the sake of contradiction, we suppose that there is no Nash equilibrium (u, v) for any $u, v \in C$. By assumption, if p_1 and p_2 choose vertices $u, v \in C$, respectively, at least one player changes the strategy. Moreover, the strategy must take a vertex in C by Lemma 2. Thus, each player repeatedly

continues to take a different vertex in C from the current vertex for the other player's strategy.

Without loss of generality, suppose that p_1 takes u^1 and p_2 takes v^1. From Lemma 1, we have $U_1(u^1, v^1) = |N(u^1)| - |N(u^1) \cap N(v^1)| + 1$ and $U_2(u^1, v^1) = |N(v^1)| - |N(v^1) \cap N(u^1)| + 1$. By assumption, there is a vertex such that at least one player can increase own utility by changing own strategy. From Lemma 2, such a vertex is in C. Without loss of generality, we assume that there is a vertex $u^1 \in C$ satisfying $U_1(u^1, v^1) < U_1(u^2, v^1)$. The utility of p_1 is $U_1(u^2, v^1) = |N(u^2)| - |N(u^2) \cap N(v^1)| + 1$. If p_1 continues to change the strategy, we can use the same augmentation. Thus, we can suppose that p_1 chooses u^2 just before p_2 changes the strategy.

Since $U_1(u^1, v^1) < U_1(u^2, v^1)$, we have:

$$|N(u^1)| - |N(u^1) \cap N(v^1)| < |N(u^2)| - |N(u^2) \cap N(v^1)|. \tag{1}$$

Next, since there is no Nash equilibrium on clique C, p_2 also changes the strategy from v^1 to $v^2 \in C$ in order to increase the utility of p_2. Since $U_2(u^2, v^1) < U_2(u^2, v^2)$, it holds that:

$$|N(v^1)| - |N(u^2) \cap N(v^1)| < |N(v^2)| - |N(u^2) \cap N(v^2)|. \tag{2}$$

By the same argument, p_1 changes the strategy from u^2 to u^3. Since $U_1(u^2, v^2) < U_1(u^3, v^2)$,

$$|N(u^2)| - |N(u^2) \cap N(v^2)| < |N(u^3)| - |N(u^3) \cap N(v^2)|. \tag{3}$$

Since there are at most n^2 strategy profiles, there is a sequence of strategy profiles $(u^1, v^1), (u^2, v^1), (u^2, v^2), \ldots, (u^k, v^{k-1}), (u^k, v^k), (u^1, v^k), (u^1, v^1)$ for some $k \leq n^2$. When the strategy profile (u^k, v^{k-1}) changes to (u^k, v^k), that is, $(u^k, v^{k-1}) \to (u^k, v^k)$, $U_2(u^k, v^{k-1}) < U_2(u^k, v^k)$ holds. Thus, it holds that:

$$|N(v^{k-1})| - |N(u^k) \cap N(v^{k-1})| < |N(v^k)| - |N(u^k) \cap N(v^k)|. \tag{4}$$

Similarly, when $(u^k, v^k) \to (u^1, v^k)$, we have:

$$|N(u^k)| - |N(u^k) \cap N(v^k)| < |N(u^1)| - |N(u^1) \cap N(v^k)|. \tag{5}$$

Finally, when $(u^1, v^k) \to (u^1, v^1)$,

$$|N(v^k)| - |N(u^1) \cap N(v^k)| < |N(v^1)| - |N(u^1) \cap N(v^1)|. \tag{6}$$

Summing all inequalities, we have $0 < 0$. This is a contradiction. □

3.2 Block graph

Theorem 2. *In 2-CDG on any block graph, a Nash equilibrium always exists.*

If a graph G is a complete graph, a Nash equilibrium clearly exists. Thus, we suppose that G is not a complete graph.

A vertex v is called a *cut vertex* if $G[V \setminus \{v\}]$ has at least two component. On a block graph, a non cut vertex is contained in exactly one maximal clique C and all the neighbors is in C. For a maximal clique C, we suppose that p_1 and p_2 select $x \in C$ and $y \in C$, respectively. If x is not a cut vertex, the utility of p_1 is $|\{x\}| = 1$ since x is adjacent to only vertices in C on block graph G and every vertex in $C \setminus \{x, y\}$ becomes a neutral vertex.

Suppose that x is a cut vertex in G. Then, let $D_x(C)$ be the set of vertices in connected components not containing C in $G[V \setminus \{x\}]$. Since $y \in C$, the set of vertices dominated by x is $D_x(C) \cup \{x\}$. Note that every vertex in $C \setminus \{x, y\}$ is neutral. Moreover, every vertex in C is a neutral vertex and every vertex in $V \setminus (D_x(C) \cup D_y(C) \cup C)$ is an undominated vertex. Thus, the utility of p_1 is $|D_x(C)| + 1$ if p_1 chooses $x \in C$ and p_2 chooses a vertex in $C \setminus \{x\}$ on block graph G.

For a maximal clique C on block graph G, we denote by $w(C, u)$ the number of vertices dominated by u when either p_1 or p_2 chooses $u \in C$ and the other chooses a vertex in $C \setminus \{u\}$. Note that if u is a cut vertex, then $w(C, u) = |D_u(C)| + 1$, and otherwise $w(C, u) = |\{u\}| = 1$. Also, we suppose that a maximal clique $C = \{u_1^C, \ldots, u_k^C\}$ satisfies that $w(C, u_1^C) \geq \cdots \geq w(C, u_k^C)$.

Lemma 4. *Let $\mathcal{C}$ be the set of maximal cliques in G. Also, let C^* be a maximal clique such that $w(C^*, u_2^{C^*}) = \max_{C \in \mathcal{C}} w(C, u_2^C)$. Then the strategy profile $(u_1^{C^*}, u_2^{C^*})$ is a Nash equilibrium.*

Proof. Suppose that strategy profile $(u_1^{C^*}, u_2^{C^*})$ is not a Nash equilibrium. First, we show that p_1 does not change the strategy. Since $C^* = \{u_1^{C^*}, \ldots, u_k^{C^*}\}$ satisfies that $w(C^*, u_1^{C^*}) \geq \cdots \geq w(C^*, u_k^{C^*})$, p_1 does not change the strategy from $u_1^{(C^*)}$ to $u \in C^* \setminus \{u_1^{C^*}, u_2^{C^*}\}$. If $u_1^{C^*}$ is not a cut vertex, the utility of p_1 is 1. Then we have $w(C, u_1^{C^*}) = \cdots = w(C, u_k^{C^*}) = 1$. This implies G is a complete graph, and hence $(u_1^{C^*}, u_2^{C^*})$ is a Nash equilibrium. This is a contradiction. Suppose that $u_1^{C^*}$ is a cut vertex. Then if p_1 selects $u_1^{C^*}$ and p_2 selects the vertex $u_2^{C^*}$, the utility of p_1 is $w(C, u_1^{C^*}) = |D_{u_1^{C^*}}(C^*)| + 1$. Even if p_1 changes the strategy from $u_1^{C^*}$ to $w \in D_{u_1^{C^*}}(C^*)$, the utility of p_1 does not increase because w can only dominate vertices in $D_{u_1^{C^*}}(C^*) \setminus \{u_1^{C^*}\}$. Note that $u_1^{C^*}$ becomes either a neutral vertex or a vertex dominated by $C^* \setminus \{w\}$ when p_1 selects w. Moreover, even if p_1 changes the strategy from $u_1^{C^*}$ to $w \in V \setminus (D_{u_1^{C^*}}(C^*) \cup C^*)$, the utility of p_1 also does not increase because w can only dominate at most $|D_{u_3^{C^*}}(C^*)| + 1$ vertices. Therefore, p_1 does not change the strategy.

Suppose that p_2 changes the strategy from $u_2^{C^*}$ to $w \in V \setminus \{u_1^{C^*}, u_2^{C^*}\}$. As with the case of p_1, p_2 does not change the strategy from $u_2^{C^*}$ to $w \in V \setminus D_{u_1^{C^*}}(C^*)$ because the number of vertices dominated by w is no more than that of vertices dominated by $u_2^{C^*}$. The remaining case is when p_2 changes the strategy from $u_2^{C^*}$ to $w \in D_{u_1^{C^*}}(C^*)$. Let C' be another maximal clique that contains $u_1^{C^*}$ on block graph G. Note that C' belongs to $D_{u_1^{C^*}}(C^*)$. Suppose that p_2 changes

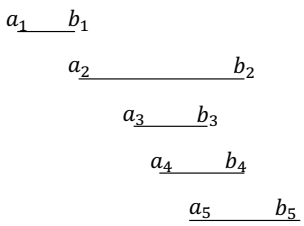


Fig. 3. The interval representation sorted in nonincreasing order of the initial endpoints a_i's.

the strategy to $w \in C'$. However, since $w(C^*, u_2^{C^*}) = \max_{C \in \mathcal{C}} w(C, u_2^C)$ by assumption, we have $w(C^*, u_2^{C^*}) \geq w(C, u_2^{C'})$. Thus, p_2 does not change the strategy from $u_2^{C^*}$ to $w \in C'$. Moreover, p_2 does not change the strategy to $w' \in D_{u_1^{C^*}}(C^*) \setminus C'$ because the number of vertices dominated by w' is not greater than that of vertices dominated by w. Therefore, p_2 does not change the strategy to $w \in D_{u_1^{C^*}}(C^*)$. This contradicts $(u_1^{C^*}, u_2^{C^*})$ is not a Nash equilibrium. □

Since the strategy profile of Lemma 4 always exists, a Nash equilibrium always exists. This concludes the proof of Theorem 2.

3.3 Interval graph

Theorem 3. *In 2-CDG on any interval graph, a Nash equilibrium always exists.*

We assume that an interval graph $G = (V, E)$ is given by corresponding intervals $\mathcal{I} = \{I_1, \cdots I_n\}$, where each interval $I_i = [a_i, b_i]$ $(i = 1, \cdots n)$ of two integer $a_i \leq b_i$ corresponds to vertex i. The endpoint a_i of I_i is called the *initial endpoint* and the other endpoint b_i is called *the terminal endpoint*. We assume that $\{I_1, \cdots I_n\}$ are sorted in nonincreasing order of the initial endpoints a_i's (see Figure 3). On an interval graph, there is an edge if and only if two intervals are intersect. That is, if interval I_i intersects interval I_j, two vertices i and j are adjacent. If interval $I_i \in \mathcal{I}$ properly contains interval $I_j \in \mathcal{I}$, $N(j) \subseteq N(i)$ holds in G. We remark that a player p chooses a vertex v means p chooses the corresponding interval I_i and vice versa in 2-CDG on any interval graph.

An interval graph is called a *proper interval graph* if no interval properly contains any other interval. An interval graph is called a *unit interval graph* if each interval has unit length. The classes of proper interval graphs and unit interval graphs are known to be equivalent [2]; every proper interval graph has an unit interval representation, and vice versa. In the following, we assume that a proper interval graph is given by a unit interval representation.

We first show that a Nash equilibrium always exists in 2-CDG on any proper interval graph. Without loss of generality, we suppose that $U_1(x, y) \geq U_2(x, y)$ for any strategy profile (x, y).

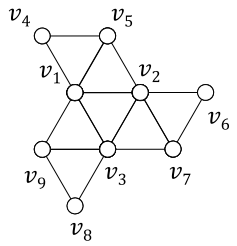


Fig. 4. A chordal graph with no Nash equilibrium.

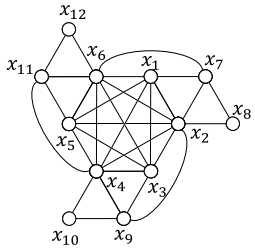


Fig. 5. A strongly chordal graph with no Nash equilibrium.

Lemma 5. *Suppose that $G = (V, E)$ is a proper interval graph. Let $\{x^*, y^*\}$ be an edge satisfying $U_2(x^*, y^*) = \max_{\{x,y\}\in E} U_2(x, y)$. Then (x^*, y^*) is a Nash equilibrium in 2-CDG on G.*

Proof. We show this lemma by contradiction. Suppose that strategy profile (x^*, y^*) is not a Nash equilibrium; a player has an incentive to change the strategy. Clearly p_2 does not change the strategy since $U_2(x^*, y^*) = \max_{\{x,y\}\in E} U_2(x, y)$ holds. Thus p_1 is the player that changes the strategy x^*. Without loss of generality, $a_{x^*} \leq a_{y^*}$, that is, the initial point of I_x is on the left side of a_y by the unit interval representation. Let D_{x^*} be the set of vertices such that every vertex v satisfies $a_v \leq a_{x^*}$. Moreover, let D_{y^*}be the set of vertices such that every vertex v satisfies $a_{y^*} \leq a_v$. Then we observe that p_1 dominates vertices in $D_{x^*} \setminus (N(x^*) \cap N(y^*))$. Also, p_2 dominates vertices in $D_{y^*} \setminus (N(x^*) \cap N(y^*))$. Since $U_1(x^*, y^*) \geq U_2(x^*, y^*)$, p_1 does not change the strategy to any vertex v satisfying $a_{y^*} \leq a_v$. Moreover, any vertex v satisfying $a_v \leq a_{x^*}$ can dominate only $D_{x^*} \setminus ((N(x^*) \cap N(y^*)) \cup \{x^*\})$ and the utility of p_1 does not increase when p_1 changes the strategy to v. Thus, p_1 does not change the strategy from x^* to any vertex in V. This implies (x^*, y^*) is a Nash equilibrium. □

Lemma 5 implies that a Nash equilibrium always exists in 2-CDG on any proper interval graph. Actually, a similar lemma holds for interval graphs, though we omit the proof due to the space limitation. The following Lemma 6 implies Theorem 3.

Lemma 6. *Suppose that $G = (V, E)$ is an interval graph. Let $\{x^*, y^*\}$ be an edge satisfying $U_2(x^*, y^*) = \max_{\{x,y\}\in E} U_2(x, y)$. Then (x^*, y^*) is a Nash equilibrium in 2-CDG on G.*

3.4 Chordal graph and Strongly chordal graph

In this section, we show that a Nash equilibrium does not always exist in the two-player competitive diffusion game if the graph is chordal or strongly chordal.

Proposition 1. *There is a chordal graph with* 9 *vertices and diameter* 3 *that has no Nash equilibrium in 2-CDG.*

Proposition 2. *There is a strongly chordal graph with* 12 *vertices and diameter* 3 *that has no Nash equilibrium in 2-CDG.*

The concrete instances of these propositions are shown in Figures 4 and 5, and Table 1 shows the pay-off matrix for the instance in Figure 4, though we omit the one for Figure 5. In the table, each element (v_i, v_j) in the pay-off matrix represents $(U_1(v_i, v_j), U_2(v_i, v_j))$, and we leave the elements of lower triangle of the matrix empty for legibility. By using Table 1, we can verify that one player has an incentive of changing the strategy for every strategy profile. For example, we start at (v_1, v_4), whose element is $(7, 1)$. Here, p_2 has an incentive to move to v_3, whose utilities are $(3, 4)$. Then, p_1 has an incentive to move to v_2, whose utilities are $(4, 3)$. This procedure continues as (v_2, v_1) and (v_3, v_1), which is essentially equivalent to (v_1, v_3); it is an endless loop. We can find similar endless loops in Figure 5.

Table 1. The pay-off matrix in the chordal graph of Fig. 4

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
v_1	$(0,0)$	$(4,3)$	$(3,4)$	$(7,1)$	$(6,1)$	$(6,2)$	$(5,2)$	$(6,1)$	$(6,2)$
v_2		$(0,0)$	$(4,3)$	$(6,1)$	$(6,2)$	$(7,1)$	$(6,1)$	$(6,2)$	$(5,2)$
v_3			$(0,0)$	$(6,2)$	$(5,2)$	$(6,1)$	$(6,2)$	$(7,1)$	$(6,1)$
v_4				$(0,0)$	$(1,7)$	$(5,3)$	$(3,5)$	$(3,5)$	$(2,5)$
v_5					$(0,0)$	$(5,2)$	$(3,4)$	$(5,3)$	$(4,3)$
v_6						$(0,0)$	$(1,7)$	$(5,3)$	$(3,5)$
v_7							$(0,0)$	$(5,2)$	$(5,2)$
v_8								$(0,0)$	$(1,7)$
v_9									$(0,0)$

4 Some other results

In this section, we briefly present two related results, though we omit all of the proofs and detailed explanation due to the space limitation. One is about vertex-weighted cycles (Theorem 4), and the other is about cacti (Theorem 5). In the vertex-weighted model, the utility is defined by not the number of influenced vertices but the total weights of influenced vertices. A cactus is a connected graph in which any two simple cycles have at most one vertex in common. Intuitively, given a block graph, we can obtain a cactus by removing the internal edges in all the cliques with size at least 4.

Theorem 4 and the results of unweighted case [3] contrast, and Theorems 5 and 2 contrast.

Theorem 4. *In 2-CDG, any vertex weighted cycle of length at most* 5 *always has a Nash equilibrium, whereas for any length at least* 6 *there are a cycle that has no Nash equilibrium.*

Theorem 5. *In 2-CDG, any vertex-weighted cactus whose induced cycles consist of at most* 5 *always has a Nash equilibrium. On the other hand, there is an unweighted cactus containing a cycle with length at least 6 that has no Nash equilibrium.*

References

1. Alon, N., Feldman, M., Procaccia, A.D., Tennenholtz, M.: A note on competitive diffusion through social networks. Information Processing Letters **110**(6), 221–225 (2010)
2. Brandstadt, A., Spinrad, J.P., et al.: Graph classes: a survey, vol. 3. SIAM (1999)
3. Bulteau, L., Froese, V., Talmon, N.: Multi-player diffusion games on graph classes. Internet Mathematics **12**(6), 363–380 (2016)
4. Dirac, G.A.: On rigid circuit graphs. In: Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg. vol. 25, pp. 71–76. Springer (1961)
5. Etesami, S.R., Basar, T.: Complexity of equilibrium in competitive diffusion games on social networks. Automatica **68**, 100–110 (2016)
6. Farber, M.: Characterizations of strongly chordal graphs. Discrete Mathematics **43**(2-3), 173–189 (1983)
7. Harary, F.: A characterization of block-graphs. Canadian Mathematical Bulletin **6**(1), 1–6 (1963)
8. Ito, T., Otachi, Y., Saitoh, T., Satoh, H., Suzuki, A., Uchizawa, K., Uehara, R., Yamanaka, K., Zhou, X.: Competitive diffusion on weighted graphs. In: Workshop on Algorithms and Data Structures. pp. 422–433. Springer (2015)
9. Lekkeikerker, C., Boland, J.: Representation of a finite graph by a set of intervals on the real line. Fundamenta Mathematicae **51**(1), 45–64 (1962)
10. Roberts, F.S., Spencer, J.H.: A characterization of clique graphs. Journal of Combinatorial Theory, Series B **10**(2), 102–108 (1971)
11. Roshanbin, E.: The competitive diffusion game in classes of graphs. In: International Conference on Algorithmic Applications in Management. pp. 275–287. Springer (2014)
12. Small, L., Mason, O.: Nash equilibria for competitive information diffusion on trees. Information Processing Letters **113**(7), 217–219 (2013)
13. Sukenari, Y., Hoki, K., Takahashi, S., Muramatsu, M.: Pure nash equilibria of competitive diffusion process on toroidal grid graphs. Discrete Applied Mathematics **215**, 31–40 (2016)
14. Takehara, R., Hachimori, M., Shigeno, M.: A comment on pure-strategy nash equilibria in competitive diffusion games. Information Processing Letters **112**(3), 59–60 (2012)

ビザンチンエージェントの封じ込めと合意形成

半澤　陽 *　　山内　由紀子 †

概要

ビザンチン故障プロセス集合が刻々と変化する合意形成問題を移動ビザンチン合意問題と呼ぶ．Inoue ら (2018) は，ビザンチン故障を引き起こすビザンチンエージェントの移動を分散システムの一部に封じ込めるために通信リンクのブロックを提案した．本研究では，完全グラフ上でビザンチンエージェントの移動経路に基づく通信リンクのブロックによる封じ込めの不可能性，通信リンクの永続的切断による封じ込めの可能性を示す．さらに，通信リンクの一時的切断の場合，閉路をもつ分散システムでの封じ込めの不可能性を示す．さらに，各操作を行いつつ移動ビザンチン合意問題を解く分散アルゴリズムを示す．

1　はじめに

分散システムにおける最も基本的な課題のひとつに，プロセス間で合意を形成する**合意問題**がある．故障したプロセスが任意に振る舞う故障を**ビザンチン故障**と呼び，この故障の下での合意形成問題を**ビザンチン合意問題**と呼ぶ．n をプロセス数，t を故障プロセス数とした時，Lamport らは $n > 3t$ の場合，完全グラフ上で合意を形成する分散アルゴリズムを提案し，$n \leq 3$ の場合，合意を形成する分散アルゴリズムが存在しないことを示した [5]．Garay は故障プロセス集合が継続的に変化する**移動ビザンチン故障**の下での合意形成問題を提案した [3]．この論文ではプロセス自身が故障していた事を認識できる**故障感知性**をもち，常に正常なプロセスが少なくとも 1 つ存在するという仮定の下，$n > 6t$ の場合に**移動ビザンチン合意問題**を解く分散アルゴリズムが示された [3]．Buhrman らは故障の原因となるビザンチンエージェントが送信メッセージと共にプロセス間を移動することで故障プロセス集合が変化するとし，各プロセスが故障感知性をもち，常に正常なプロセスが少なくとも 1 つ存在するという仮定の下で，$n > 3t$ の場合に完全グラフにおいて移動ビザンチン合意問題を解く分散アルゴリズムを提案した [1]．Sasaki らは，故障感知性がない場合に常に正常なプロセスが少なくとも 1 つ存在する場合に，$n > 6t$ の場合に完全グラフにおいて移動ビザンチン合意問題を解く分散アルゴリズムを提案した [6]．

一時故障に対する故障耐性として，自己安定性が提案されている [2]．**自己安定プロトコル**とは，任意のシステム状況から開始しても目的のシステム状況へ収束することを保証する分散アルゴリズムである．Inoue らは，故障感知性に加え故障直後のプロセスがビザンチンエージェントの移動先のプロセスを判別できる探知可能モデルを導入し，プロセスが隣接プロセスからのメッセージを受信しない通信リンクの**ブロック**を複数行うことで単一のビザンチンエージェントを分散システムの一部に封じ込めながら，分散システム上に全域木を構成する自己安定プロトコルを提案した [4]．本稿では，ビザンチンエージェントの移動を分散システムの一部に制限することをビザンチンエージェントの**封じ込め**と呼ぶ．

本研究では，通信リンクのブロック，切断によるビザンチンエージェントの封じ込めと合意形成を考える．ここで，**切断**とは通信リンクを双方向にブロックすることである．

* 九州大学大学院システム情報科学府，hanzawa@tcs.inf.kyushu-u.ac.jp
† 九州大学大学院システム情報科学研究院，yamauchi@inf.kyushu-u.ac.jp

2 準備

プロセス集合を $\Pi = \{p_1, p_2, ..., p_n\}$，通信リンクの集合を E とし，無向グラフ $G = (\Pi, E)$ と表す．グラフ G は完全グラフを仮定する．p_i と p_j について $\{p_i, p_j\} \in E$ である時，p_i と p_j は**隣接**していると言い，隣接プロセスは相互にメッセージの送受信による通信を行うことができる．p_i の固有の識別子を i とし，プロセスはメッセージの送信時に自身の識別子をメッセージに添付することで，受信したメッセージを判別することができる．各プロセスは**ラウンド**毎に動作し，各ラウンドでメッセージの**送信**，**受信**，**内部計算**を行う．このモデルを**同期モデル**と呼ぶ．各プロセスは受信したメッセージに基づいて内部計算を行い，自身の値を更新し，次のラウンドで送信するメッセージを決定する．送信されたメッセージは同じラウンド内で必ず受信される．各プロセスはいくつかの内部変数を管理しており，これらの変数の値がプロセスの状態を表す．

プロセスが任意の振る舞いを行う故障を**ビザンチン故障**と呼ぶ．**故障プロセス**は隣接プロセスそれぞれに異なるメッセージを送信することができる二地点間通信を行うものとするが識別子を変えることはできないとする．故障プロセスの集合を F と表す．各プロセスは故障プロセスを知ることはできない．$\Pi \backslash F$ に含まれるプロセスを**正常プロセス**と呼ぶ．故障プロセス集合 F がラウンド毎に変化する故障を**移動ビザンチン故障**と呼ぶ．本研究では，正常プロセスに**ビザンチンエージェント**が訪れることで，故障プロセス集合が変化すると考える．ビザンチンエージェントはメッセージの送信時に故障プロセスから移動し，受信時に隣接プロセスに移動する．分散システム中に高々 t 個のビザンチンエージェントが存在するとする．直前のラウンドでビザンチンエージェントがいたプロセスを *cured* **プロセス**と呼ぶ．本研究では *cured* プロセスはそのラウンドでは送信を行わず，受信したメッセージをもとに正しい動作に復帰するとする．そして，自身が故障プロセスであった事を認識できる**故障感知性**をもつ．

移動ビザンチン合意問題は，移動ビザンチン故障が存在する分散システム上で各プロセスに初期値 $d_i \in \{0, 1\}$ を与え，正常プロセスが以下の条件を満たすように合意値を決定する問題である [4]．

（**決定性**）：全ての正常プロセスは合意値 v_i をいずれ決定する．
（**合意性**）：全ての正常プロセスは同じ値で合意しなければならない．
（**妥当性**）：全ての正常プロセスの初期値が同じ値であれば，その値で合意しなければならない．
（**合意維持性**）：一度正常プロセス間で合意に達すると，その後もその値で正常プロセス間で合意を維持しなければならない．

本研究では故障を起こすことのない常に正常なプロセスが少なくとも 1 台存在すると仮定する[*1]．

Inoue らは，*cured* プロセスがビザンチンエージェントの移動先のプロセスを判別できる**探知可能モデル**と判別できない**探知不可能モデル**を提案した [4]．p_i が故障プロセスの場合，探知可能モデルでは p_i は変数 $dest_{p_i}$ を持ち $dest_{p_i}$ は $\{\perp, 1, ..., n\}$ の要素を値とする．ビザンチンエージェントが p_i から p_j へ移動した次のラウンドに $dest_{p_i} = j$ となる．探知不可能モデルは，常に $dest_{p_i} = \perp$ の状態である．Inoue らは，探知可能モデルで隣接プロセスからのメッセージを受信しない通信リンクの**ブロック**を提案した．p_i が p_j のメッセージを受信しない時，p_i は p_j をブロックしていると言う．

本研究では，通信リンクを双方向にブロックする通信リンクの**切断**を導入する．双方向にブロックした通信リンク上でその後二度と通信を行えないとき，その切断を**永続的切断**と呼ぶ．

3 ビザンチンエージェントの封じ込め

$t = 1$ の場合を想定する．探知可能モデルでは，通信リンクのブロックや切断を行うことでビザンチンエージェントの移動を制限できる可能性がある．

[*1] 故障しないプロセスが存在しなければ，正常プロセス間で合意することができない [3]．

ビザンチンエージェントが同じプロセスに常に留まっている状況では，通信リンクのブロックや切断を行うことはできずビザンチンエージェントの封じ込めは行えない．本章では，ビザンチンエージェントが移動し続ける状況を想定しビザンチンエージェントの封じ込めを以下のように定義する．

定義 3.1. (ビザンチンエージェントの封じ込め) $X \subsetneq \Pi$ の全プロセスからの通信リンクを $\Pi \backslash X$ の全プロセスがブロックしており，かつ，ビザンチンエージェントが X に含まれるプロセスにいる時，ビザンチンエージェントを X に封じ込めたと言う．

ビザンチンエージェントの移動先のプロセスからの通信リンクのみをブロックする手順を**ビザンチンエージェントの移動経路に基づく**通信リンクのブロックと呼ぶ．p_i が p_j からの通信リンクをブロックしている時，ビザンチンエージェントは p_j から p_i へ移動することができない．p_j は p_i からの通信リンクをブロックすることはできないので，複数本通信リンクをブロックしてもビザンチンエージェントを封じ込められないことがわかる．一方，通信リンクの永続的切断を行うことで，ビザンチンエージェントを封じ込めることができる．

定理 3.1. $t = 1$ で各プロセスがビザンチンエージェントの移動経路に基づく通信リンクのブロックを行う場合，ビザンチンエージェントを封じ込めることはできない．

定理 3.2. $t = 1$ で各プロセスがビザンチンエージェントの移動経路に基づく通信リンクの永続的切断を行う場合，単一のプロセスにビザンチンエージェントを封じ込めることができる．

これまでは切断した通信リンク上でその後二度と通信が行えない永続的切断による封じ込めについて考えた．次に，切断した通信リンクを持つプロセスを訪れたビザンチンエージェントによって切断している通信リンクの記録が改ざんされる場合について考える．記録の改ざんによって切断した通信リンク上で再び通信が行われる可能性がある切断を**一時的切断**と呼ぶ．

定理 3.3. $t = 1$ で各プロセスがビザンチンエージェントの移動経路に基づく通信リンクの一時的切断を行う場合について，単一のプロセスでビザンチンエージェントを封じ込めできるのは，グラフ G が木構造である場合，またその場合に限る．

定理 3.4. $t = 1$ で各プロセスがビザンチンエージェントの移動経路に基づく通信リンクの一時的切断を行う場合について，ビザンチンエージェントを封じ込めできない場合は，グラフ G が閉路を持つ場合，またその場合に限る．

4 ビザンチン合意アルゴリズム

本節では，完全グラフ上でビザンチンエージェントの移動経路に基づく通信リンクのブロック，切断それぞれの手法で移動ビザンチン合意問題を解くアルゴリズムを与える．各プロセスはブール値を保持できる配列 $block$ を管理し，p_i の $block_{p_i}[j] = true$ である時，p_i は p_j からのメッセージを受信しない．各プロセスの初期値を $d_i \in \{0, 1\}$ とする．3 つのラウンドを 1 フェーズとし，各フェーズで**コーディネータ**となるプロセスを変更しながら値の更新を行う．正常プロセスがコーディネータとなる時，そのプロセスが $(n-4)$ 個以上のプロセスからブロック，切断されていない時，次のフェーズの第 1 ラウンドで全ての正常プロセス間で合意することができる．その他の場合は常に正常なプロセスがコーディネータとなれば正常プロセス間で合意することができる．通信リンクのブロックを用いたアルゴリズムを bMopt を Algorithm 1 に示す．第 3 ラウンドで $cured$ 状態のプロセスは Algorithm 2 に示した RECONSTRUCT を実行することで自身の値を決定する．通信リンクのブロックは Algorithm 3 に示した BLOCK を実行する．プロトコル bMopt について，以下の定理が成り立つ．

定理 4.1. $n > 3$，$t = 1$ の時，プロトコル $bMopt$ は完全グラフ上で移動ビザンチン合意問題を解く．

Algorithm 1 Protocol bMopt at process p_i

v_i; プロセス p_i の値
c_i; 各フェーズでのコーディネーターとなるプロセスの識別子
MV ; プロセスが受信した値を格納する一次元配列
C ; 受信した 0, 1 に対するカウンター
D ; 受信した ⊥ ,0,1 に対するカウンター

ECHO ; プロセスが受信した値を格納する二次元配列

1: $v_i \leftarrow d_i$
2: **for** phase $s = 1$ to ∞ **do**
3: *Round* 1
4: send v_i to all processes
5: receive(MV[·])
6: **if** cured **then**
7: $BLOCK$
8: **for** $j = 0$ to 1 **do**
9: $C[j]$ $= \#\ of\ j's$ in MV
10: $v_i = \begin{cases} 0 & if\ C[0] \geq n' - t \\ 1 & if\ C[1] \geq n' - t \\ \perp & otherwise \end{cases}$
11: *Round* 2
12: send v_i to all processes
13: receive(MV[·])
14: **if** cured **then**
15: $BLOCK$
16: **for** $j = \perp$ *to* 1 **do**
17: $D[j]$ $= \#\ of\ j's\ in$ MV
18: $v_i = \begin{cases} 0 & if\ D[0] > t \\ 1 & if\ D[1] > t \\ \perp & otherwise \end{cases}$
19: *Round* 3
20: send MV[·] to all processes
21: **for** *each* i **do**
22: ECHO[i, ·]=MV[·] received from i
23: **if** cured **then**
24: $BLOCK$
25: **if** cured in Round 2 **then**
26: RECONSTRUCT
27: $c_i \leftarrow s\ mod\ n$
28: **if** $v_i = \perp$ *or* $D[v_i] < n' - t$ **then**
29: $v_i = max(0, v_{c_i})$

Algorithm 2 RECONSTRUCT

1: **for** *all* $j \in \Pi$ **do**
2: **if** $w \in \{0, 1\}$ occurs at least $n' - t$ times in column j of ECHO[i, ·] **then**
3: $SV_i[j] \leftarrow w$
4: **else**
5: $SV_i[j] \leftarrow \perp$
6: **for** $j = \perp$ to 1 **do**
7: $D[j]$ $= \#$ of $j's$ in SV
8: $v_i = \begin{cases} 0 & (D[0] > t) \\ 1 & (D[1] > t) \\ \perp & (otherwise) \end{cases}$

Algorithm 3 BLOCK

1: **if** $dest_{p_i} > 0$ **then**
2: $block_{p_i}[dest_{p_i}] \leftarrow true$

通信リンクを永続的，一時的切断する場合の移動ビザンチン合意問題を解くアルゴリズムを dMopt に示す．bMopt と dMopt の相違点は以下の 2 点である．1 点目は，BLOCK の操作を行う時に，通信リンクを双方向ブロックする手続きを行う．2 点目は，第 1 ラウンドで *cured* 状態のプロセスは，第 2 ラウンドでは t 個以上の値を受信した時その値を採用し，それ以外の時は値を ⊥ とする．プロトコル dMopt について，以下の定理が成り立つ．

定理 4.2. $n > 3$，$t = 1$ の時，プロトコル *dMopt* は完全グラフ上で移動ビザンチン合意問題を解き，ビザンチンエージェントが移動し続ける場合，ビザンチンエージェントを封じ込める．

定理 4.3. プロトコル *dMopt* は高々 $3n$ ラウンドで，移動ビザンチン合意問題を解く．

5 まとめ

本研究では，1 台のビザンチンエージェントを封じ込める手法とトポロジーによる封じこめ，$n > 3$ で移動ビザンチン合意問題を解くアルゴリズムを提案した．今後の課題は，より一般的なネットワークの形として動的グラフでのビザンチンエージェントの封じ込めと移動ビザンチン合意問題を解くアルゴリズムの考案である．

参考文献

[1] H. Burhman, J. Garay, and J. Hoepman. Optimal Resiliency against Mobile Faults. Proc. of FTCS 1995, pp. 83–88, 1995.

[2] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. Communications of the ACM, Vol. 17, pp. 643–644, 1974.

[3] J. Garay. Reaching (and Maintaining) Agreement in the Presence of Mobile Faults. Proceedings. of WDAG 1994, pp. 253–264, 1994.

[4] K. Inoue, H. Kakugawa, and T. Masuzawa.

Strongly stabilizing protocol for spanning tree construction with mobile Byzantine. 第14回情報科学ワークショップ予稿集, pp. 258–264, 2018.

[5] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, Vol. 4, pp. 382–402, 1982.

[6] T. Sasaki, Y. Yamauchi, S. Kijima, and M. Yamashita. Mobile Byzantine Agreement on Arbitrary Network. Proc. of OPODIS 2013, pp. 236–250, 2013.

視界に制限のある自律移動ロボット群によるグリッド探索

長濵 将太　　大下 福仁　　井上 美智子
奈良先端科学技術大学院大学
{nagahama.shota.nl1,f-oosita,kounoe}@is.naist.jp

概要

本稿で扱う研究は，視界に制限のある自律移動ロボット群による $m \times n$ グリッドの停止探索アルゴリズムの検討である．各ロボットは定数距離 ϕ 以内のノードを観測でき，他のロボットからも観測可能な定数 l 個の色を持つライトを利用できる．また，左右の概念である chirality を共有しているときと共有していないときの両方の場合を検討する．本稿では，我々が設計したアルゴリズムによる探索手法を概説する．その後，具体例として，完全同期モデル，$\phi = 1$，$l = 3$，chirality の共有ありでの探索アルゴリズムを紹介する．

1　はじめに

1.1　背景と動機

自律移動ロボットの協調に関する研究が分散コンピューティングの分野で注目を集めている．これらの研究は与えられたタスクを達成できる必要最小限のロボットの能力を明らかにすることに焦点を当てている．ロボットの動作のモデルとして，LCM（Look-Compute-Move）モデル [1] がよく使用されている．LCM モデルでは，各ロボットは「観測（Look）」「計算（Compute）」「移動（Move）」の 3 つのフェーズから成るサイクルを繰り返し実行する．観測フェーズでは，ロボットは他のロボットの位置を観測する．計算フェーズでは，観測結果を入力として自身のアルゴリズムを実行し，次の移動先を計算する．移動フェーズでは，計算フェーズで計算した移動先に移動する．

ロボットの能力については，様々なモデルが考えられている．必要最小限のロボットの能力を明らかにするために，多くの研究は弱い能力のロボットを仮定している．具体的には，ロボットは全て区別不可能で同じアルゴリズムを実行し，メモリを持たず，他のロボットとの直接的な通信手段を持たない（観測フェーズで他のロボットの位置を観測することにより間接的に通信している）．加えて，ロボットが左右の概念（chirality と呼ばれる）を共有していない場合 [4] や，視界に制限のある場合，つまり，観測フェーズで定数距離 ϕ 以内のロボットしか観測できない場合 [12] も考えられている．一方，弱い能力では困難なタスクを達成するために，各ロボットにライト [14] を持たせることで能力を強めたモデルも考えられている．このモデルでのロボットは，1 回の LCM サイクルごとに定数個の色を持つ集合から 1 色選んでライトを照らすことができる．このライトは不揮発性であり，定数サイズのメモリとして使用できる．加えて，ロボットが観測フェーズで他のロボットの位置を観測するとき，同時にそのロボットのライトの色も観測できる．ロボットの動作環境のモデルとしては，ユークリッド平面などの連続的な環境 [1–3] や，グラフネットワークなどの離散的な環境 [4–6] がこれまでに考えられている．

グラフネットワークにおける基本的なタスクの 1 つに探索（exploration）があり，永続探索（perpetual exploration）と停止探索（terminating exploration）の 2 種類がよく研究されている．永続探索の達成条件は，全てのロボットが全てのノードを無限回訪問することである．停止探索の達成条件は，各ノードを少なくとも 1 体のロボットが訪問したうえで，全てのロボットが移動を終えることである．視界に制限のないロボットの場合，永続探索はリング [4] やグリッド [7] で研究され，停止探索はリング [5, 6]，木 [8]，グリッド [9]，トーラス [10]，任意のネットワーク [11] で研究されている．これらの研究では，探索に必要な最少のロボット数を明らかにすることを目的としている．

本稿では，視界に制限のあるロボットによる探索に焦点を当てる．視界に制限のあるロボットによる探索はリング [12, 13, 15]，無限グリッド [16] で研究されている．しかしながら，同一のモデルによる他のネットワークの探索は我々の知る限り研究されていない．

1.2 成果

我々は，視界に制限のあるロボットによる $m \times n$ グリッドの停止探索アルゴリズムを設計した．提案アルゴリズムにおけるロボットのモデルとロボット数との対応を表 1 に示す．ロボットのモデル（同期モデル，観測可能な距離 ϕ，ライトの色数 l，chirality の共有の有無）によって，アルゴリズムが必要とするロボット数 k は異なっている．なお，ライトの色数が 1 のロボットは，ライト無しのロボットと等価である．これらの結果は，各モデルにおいて探索に必要なロボット数の上界を示しているといえる．

本稿では，我々が設計したアルゴリズムによる探索手法を概説する．その後，具体例として，完全同期モデル，$\phi = 1$，$l = 3$，chirality の共有ありでの探索アルゴリズムを紹介する．

2 定義

■システムモデル システムは $m \times n$ グリッド ($m \geq 3, n \geq 3$) と k 体のロボットから成る．本稿では，以下を満たすグラフ $G = (V_{tx}, E_{dge})$ を $m \times n$ グリッドと定義する（V_{tx} はノードの集合，E_{dge} はエッジの集合）．

1. $V_{tx} = \{(i,j) \,|\, i \in \{0,1,\ldots,m\} \wedge j \in \{0,1,\ldots,n\}\}$
2. $E_{dge} = \{((i_1,j_1),(i_2,j_2)) | (i_1,j_1) \in V_{tx} \wedge (i_2,j_2) \in V_{tx} \wedge |i_1 - j_1| + |i_2 - j_2| = 1\}$

グリッドの大きさを示す m,n やノードのラベル $(0,0),(0,1),\ldots,(m-1,n-1)$ をロボットは知ることができない．グリッド上の 2 つのノード間の距離は，ノード間の最短経路上のエッジの個数とする．2 体のロボット a,b 間の距離は，a と b がそれぞれ存在するノード間の距離とする．

ロボットは全員が同一（identical）である．つまり，同じアルゴリズムを実行し，ユニークな識別子を持たない．各ロボットは自身や他のロボットから見えるライトを 1 つ持つ．各ロボットは自身のライトの色を集合 Col から 1 つ選び，メモリとして保持することができる．Col の要素数が有限であるとき，利用できる色の個数を l で表す（つまり，$l = |Col|$）．本稿では，$l = 2$ の場合は $Col = \{W, G\}$，$l = 3$ の場合は $Col = \{W, G, B\}$ とする（$l = 1$ の場合のアルゴリズムは紹介しない）．ライトの他にロボットはメモリを持たない．各ロボットは他のロボットの位置や色の観測という形で情報を集め，自身の色の変更や移動という形で情報を伝えることで，他のロボットと通信することができる．各ロボットは定数距離 ϕ ($\phi > 0$) 以内のロボットの位置と色を観測することができる．ロボットは全員が同一であるから，ϕ の値は全ロボットで等しいものとする．

各ロボットは「観測」「計算」「移動」の 3 つのフェーズから成る LCM サイクルを繰り返すことでアルゴリズムを実行する．観測フェーズでは，ロボットは距離 ϕ 以内のロボットの位置と色をスナップショットとして観測する．計算フェーズではロボットは観測フェーズでの観測結果にしたがって，次の色と移動を決定する．色を変える場合は，計算フェーズの終了時に変える．移動すると決めた場合は，移動フェーズで隣のノードに移動する．実行の非同期性をモデル化するために，スケジューラという，各ロボットがいつフェーズを実行するかを決める概念を導入する．スケジューラがロボット r にフェーズを実行させるとき，スケジューラが r を作動させる，という．完全同期，半同期，非同期の 3 種類の同期モデルを考える．全ての同期モデルで，時間は無限の時刻の列 $0,1,2,\ldots$ で表される．ロボットはこの時刻を知ることができない．完全同期モデルと半同期モデルでは，ある時刻 t で作動させられるロボット全てが，1 つのサイクルを時刻 t と $t+1$ の間に同時に実行する．完全同期モデルでは，スケジューラは各時刻に全てのロボットを作動させる．半同期モデルでは，スケジューラは各時刻に 1 体以上のロボットを選び，選んだロボットを作動させる．非同期モデルでは，スケージュラはロボットのサイクルを非同期に作動させる．言い換えると，各フェーズ間の時間は有限だが，予測不可能である．また，スケジューラは公平であると仮定する．つまり，各ロボットは無限回サイクルを実行する．

■状況 ノード (i,j) にいるロボットの色の多重集合を $M_{i,j}$ と表記する．ノード (i,j) にロボットがいなければ，$M_{i,j} = \emptyset$ である．ロボットがいるノードの個数を q，ロボットがいるノードの集合を $P = \{(i_1,j_1),(i_2,j_2),\ldots,(i_q,j_q)\}$ とする．このとき，システムの状況を集合 $C = \{((i_1,j_1),M_{i_1,j_1}),((i_2,j_2),M_{i_2,j_2}),\ldots,((i_q,j_q),M_{i_q,j_q})\}$ と定義する．

■ビュー ロボットが観測フェーズを実行したとき，ロボットは距離 ϕ 以内の部分的な状況をビューとして得る．ノード (i,j) 上のロボット r が得るビューを考える．c_r を r の色とする．$\phi = 1$ のとき，ビュー $V_1 = (c_r, M_{i-1,j}, M_{i,j-1}, M_{i,j}, M_{i,j+1}, M_{i+1,j})$ が得られる．$\phi = 2$ のとき，ビュー $V_2 = (c_r, M_{i-2,j}, M_{i-1,j-1}, M_{i-1,j}, M_{i-1,j+1}, M_{i,j-2}, M_{i,j-1},$

表 1　$m \times n$ グリッドの停止探索における提案アルゴリズム

ロボットのモデル				ロボット数 k
同期モデル	観測可能な距離 ϕ	ライトの色数 l	chirality の共有	
半同期/非同期	1	3	なし	6
			あり	3
	2	2	なし	4
			あり	3
		3	なし	3
			あり	2
完全同期	1	2	なし	5
			あり	3
		3	なし	4
			あり	2
	2	1	なし	4
			あり	3
		2	なし	3
			あり	2

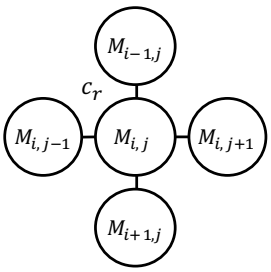
$M_{i,j}, M_{i,j+1}, M_{i,j+2}, M_{i+1,j-1}, M_{i+1,j}, M_{i+1,j+1}, M_{i+2,j}$) が得られる．ノード (i', j') ($|i - i'| + |j - j'| \leq \phi$) が存在しない場合は，$M_{i',j'} = \perp$ とする．

ロボットは東西南北のような大域的な方位を認識する能力を持っていないため，上述のビューを $\pi/2, \pi, 3\pi/2$ それぞれの角度で回転させた結果を，元のビューと区別して認識することはできない．例えば，V_1 を時計回りに $\pi/2$ 回転させた結果は $V_{1,\pi/2} = (c_r, M_{i,j-1}, M_{i+1,j}, M_{i,j}, M_{i-1,j}, M_{i,j+1})$ であり，ロボット r は V_1 と $V_{1,\pi/2}$ を区別することができない．また，ロボットが共通の chirality を有していないとき，ロボットはこれらの（回転させた結果を含む）ビューの鏡像も元のビューと区別して認識することができない．例えば，V_1 の鏡像は $V_{1,mirror} = (c_r, M_{i-1,j}, M_{i,j+1}, M_{i,j}, M_{i,j-1}, M_{i+1,j})$ であり，ロボット r は V_1 と $V_{1,mirror}$ を区別することができない．このようにして，各観測フェーズでロボット r が得たビューについて，そのビューを含む区別できないビューの集合を IV_r とおく．

r のビューが何かしらの対称性を持つとき，r から見て複数の方向が同一に見える場合がある．r がいる (i, j) を中心とする点対称なビューでは，r は $(i-1, j)$ のある方向と $(i+1, j)$ のある方向を区別できず，$(i, j-1)$ のある方向と $(i, j+1)$ のある方向を区別することもできない．加えて，ロボットが共通の chirality を有していない場合，(i, j) を通る直線を対称軸とする線対称なビューでは，対称軸に垂直な直線が示す 2 方向（例えば，$(i, j), (i+1, j)$ を通る直線を対称軸とするビューでは，$(i-1, j)$ のある方向と $(i+1, j)$ のある方向）を区別することができない．これらの同一に見える方向のいずれかに向かって r が移動しようとする場合，実際に移動する方向は，同一に見える複数の方向の中からスケジューラによって選ばれる．ただし，本稿で示すアルゴリズムはスケジューラによる移動方向の決定が起こらないよう作られている．

■アルゴリズム　本テーマではアルゴリズムをルールの集合として記述する．各ルールはラベルとガードとアクションの組合せで記述される．ラベルは各ルールをラベル付けしている．ガードはルールが実行可能となる条件であり，ロボットのビューで記述される．すなわち，ロボット r が直近の観測フェーズで得たビューと区別できないビューの集合 IV_r の要素のいずれかが，アルゴリズム中のガードのいずれかに一致していれば，r はそのガードに対応するラベルのルールを実行可能である．ルールが実行可能であれば，ロボットは対応するアクションにしたがって計算フェーズと移動フェーズで色の変更や移動を行う．

区別できないビューの集合の要素が複数のガードと一致した場合，対応するのルールのいずれかがスケジューラによって選ばれ，実行可能となる．ただし，本稿で提



$Rule: c_{new}, Movement$

図 1　アルゴリズムの各ルールの記述形式

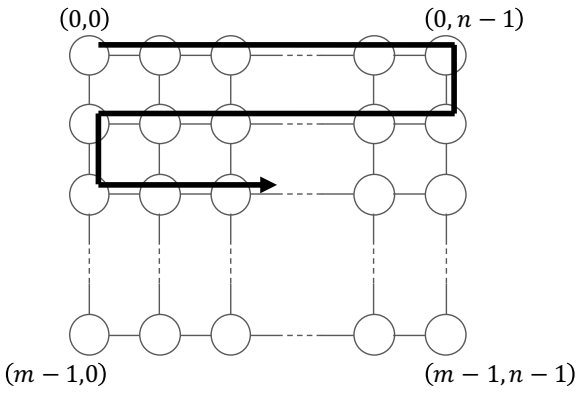


図 2　提案アルゴリズムによる $m \times n$ グリッドの探索順路

案するアルゴリズムでは，そのような状況は発生しない．

■実行　初期状況 C_0 からの実行は，任意の $j > 0$ に対して以下を満たすような状況のシーケンス $E = C_0, C_1, \ldots$ で定義される．

1. $C_{j-1} \neq C_j$
2. C_j は C_{j-1} からいくつかのロボットが移動または色の変更をすることで得られる．
3. C_{j-1} から C_j までの間に移動または色の変更をする各ロボット r について，その移動や色の変更を自身のアルゴリズムと $C_{j'}$ でのビューに従って決定するような，状況のインデックス j' $(0 \leq j' \leq j)$ が存在する．

■探索問題　問題 $\mathcal{P}$ は実行の集合として定義される．ある実行 E と問題 $\mathcal{P}$ について $E \in \mathcal{P}$ が成り立つならば，E は $\mathcal{P}$ を解く．初期状況 C_0 からのアルゴリズム $\mathcal{A}$ の任意の実行が $\mathcal{P}$ を解くならば，$\mathcal{A}$ は C_0 から $\mathcal{P}$ を解く．$\mathcal{A}$ が C_0 から $\mathcal{P}$ を解くような初期状況 C_0 が存在することを，$\mathcal{A}$ は $\mathcal{P}$ を解く，と表記する．状況 C と問題 $\mathcal{P}$ について，初期状況 C から $\mathcal{P}$ を解くアルゴリズムが存在するならば，C は $\mathcal{P}$ に対して可解である．本稿では，以下に定義する停止探索問題を扱う．

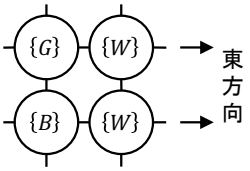
定義 1　停止探索は，各ノードを少なくとも 1 体のロボットが訪問したうえで，ルールを実行可能なロボットがいなくなるような実行の集合である．

■ルールの記述　本稿では，アルゴリズムの各ルールを図 1 の形式で記述する．本稿で詳細なアルゴリズムを紹介するのは $\phi = 1$ の場合のみであるため，$\phi = 1$ でのルールの記述形式のみを述べる．$Rule$ はルールのラベルである．図 1 中のグラフはガードであり，ビュー $V_1 = (c_r, M_{i-1,j}, M_{i,j-1}, M_{i,j}, M_{i,j+1}, M_{i+1,j})$ を表す．$M_{i',j'} = \emptyset$ $(|i - i'| + |j - j'| \leq \phi)$ の場合は，対応するノードの中に $\emptyset$ と書く代わりに，何も書かない．$M_{i',j'} = \perp$ の場合は，対応するノードの中に $\perp$ と書く代わりに，そのノードを黒色に塗りつぶす．$M_{i',j'}$ が $\emptyset$ と $\perp$ のどちらかであればよい場合は，対応するノードを灰色に塗りつぶす．ロボット r の区別できないビューの集合 IV_r の要素のいずれかが V_1 に一致すれば，r はそのルールを実行可能である．このとき，r は $c_{new}, Movement$ で表されるアクションを実行することができる．c_{new} はロボットの新しい色を表し，$Movement$ は移動を表す．$Movement$ は $Idle, \leftarrow, \rightarrow, \uparrow, \downarrow$ のいずれかで表される．$Idle$ は，ロボットが移動しないことを表す．$\leftarrow$ は，ガードの $M_{i,j-1}$ に対応するノードへ移動することを表す．$\rightarrow$ は，ガードの $M_{i,j+1}$ に対応するノードへ移動することを表す．$\uparrow$ は，ガードの $M_{i-1,j}$ に対応するノードへ移動することを表す．$\downarrow$ は，ガードの $M_{i+1,j}$ に対応するノードへ移動することを表す．ただし，本稿で紹介するアルゴリズムのルールには，$Idle$ と $\uparrow$ は使われていない．

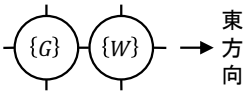
3　探索手法の概略

本章では，我々が設計したアルゴリズムによる探索手法を概説する．説明の簡単化のため，グリッド上の大域的な方角を定義する．ノード (i, j) から見て，$(i, j+1)$ が見える方向を東，$(i, j-1)$ が見える方向を西，$(i+1, j)$ が見える方向を南，$(i-1, j)$ が見える方向を北とする．我々が設計したアルゴリズムは全て，図 2 の矢印に従う順序でノードを探索していく．つまり，グリッドの北西の隅から探索を開始して，以下の一連の動作を南端の隅（m が奇数の場合は南東の隅，m が偶数の場合は南西の隅）に到達するまで繰り返す．

1.（東への直進）グリッドの東端まで一直線に進む．



(A) 共通のchiralityを有していない場合



(B) 共通のchiralityを有している場合

図 3　完全同期かつ $\phi = 1$，$l = 3$ の場合の東へ直進するときの隊列

2.（西への折り返し準備）南に距離 1 だけ下がりつつ，西に折り返せるように整列する．
3.（西への直進）グリッドの西端まで一直線に進む．
4.（東への折り返し準備）南に距離 1 だけ下がりつつ，東に折り返せるように整列する．

東または西への直進を行うとき，ロボットが共通の chirality を有しているのであれば，ロボットは 1 列に並んでいる状況を維持しながら直進する．ロボットが共通の chirality を有していないのであれば，ロボットは 2 列に並んでいる状況を維持しながら直進する．2 列に並ぶ理由は，グリッドの端に到達してから南に下がるために，ロボットが左右（北と南）を区別する必要があることである．2 列に並んだロボットの各列にとって，もう一方の列は北と南を区別する目印となる．このため，表 1 について，ロボットが共通の chirality を有している場合は，そうでない場合より少ないロボット数でアルゴリズムを設計できている．

各ロボットは自身の LCM サイクルごとに，隊列が組まれていることを自身のビューによって確認し，隊列に応じた進行方向に移動する．完全同期の場合，隊列内の全ロボットが同時に動作するため，全く同じ隊列を維持したまま移動することができる．例えば，完全同期かつ $\phi = 1$，$l = 3$ の場合，ロボットは図 3 に示す隊列を維持しながら東へ直進する．また，完全同期かつ $\phi = 2$，$l = 2$ で共通の chirality を有していない場合，ロボットは図 4 に示す隊列を維持しながら東へ直進する．

こうしてグリッドの東端に到達した後，ロボットは南へ下がりつつ西へ折り返す準備をする．共通の chirality を有していない場合は，東へ直進するときの隊列と鏡像の関係にある隊列を形作る．これにより，東へ直進するときと同じルールによって西へ直進することができる．共通の chirality を有している場合は，東へ直進するときの隊列と鏡像の関係ではない隊列を構成する必要がある．なぜなら，東端に到達したときに chirality を利用して南へ下がっている場合，鏡像の関係にある隊列でグリッドの西端に到達したときは北へ上がってしまうからである．例えば，図 3 の例と同じモデルである完全同期かつ $\phi = 1$，$l = 3$ の場合，ロボットは西へ直進するときに図 5 に示す隊列を維持している．

この後に西へ直進してグリッドの西端に到達したら，ロボットは南へ下がりつつ東へ折り返す準備をする．例えば，図 3，図 5 と同じモデルである完全同期かつ $\phi = 1$，$l = 3$ の場合，ロボットは図 3 に示した隊列を再び形作る．このような動作の繰り返しにより，ロボットは全ノードを訪問して南端の隅に到達することで，グリッドの停止探索を達成する．

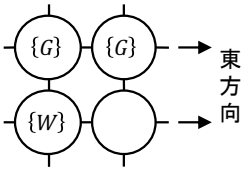
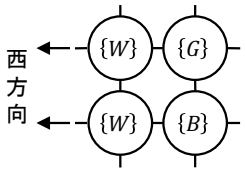
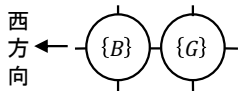


図 4　完全同期かつ $\phi = 2$，$l = 2$ で共通の chirality を有している場合の東へ直進するときの隊列



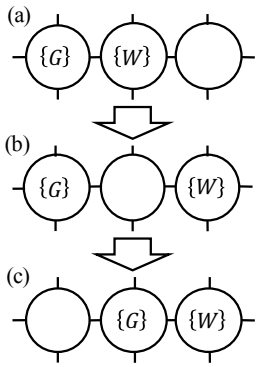
(A) 共通のchiralityを有していない場合



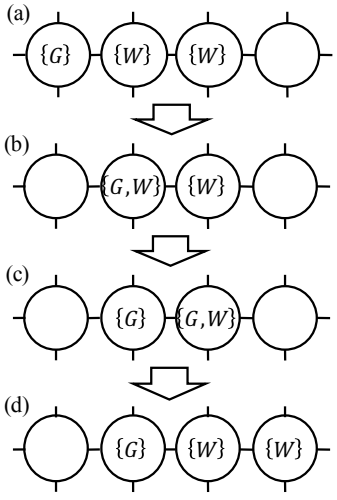
(B) 共通のchiralityを有している場合

図 5　完全同期かつ $\phi = 1$，$l = 3$ の場合の西へ直進するときの隊列

■半同期/非同期での直進方法　図 3，図 4，図 5 に示した例は完全同期の場合で，スケジューラが全ロボットを同時に作動させることを前提としていた．半同期/非同期の場合は，ほとんどの状況でロボットが 1 体ずつ動作するようにアルゴリズムを設計している．もし，ある状況で複数のロボットが動作できる場合，スケジューラがどのロボットを作動させるかによって実行が分岐する．



(A) $\phi = 2$の場合



(B) $\phi = 1$の場合

図 6　半同期/非同期かつ $l = 3$ で共通の chirality を有している場合の東へ直進するときの隊列

分岐が多ければ多いほど，多数の実行が考えられる．これらの実行のいずれにおいても探索を達成できるようにアルゴリズムは設計されなければならないため，アルゴリズムを設計するときはこのような分岐を減らすことが望ましい．そのため，可能な限り各状況でロボットが 1 体ずつ動作するようにアルゴリズムを設計している．

例えば，半同期/非同期かつ $l = 3$ で共通の chirality を有している場合，ロボットは図 6 に示すように隊列を変えていくことで東に移動する（図 6 の隊列では 2 色しか使われていないが，残りの 1 色は西へ折り返すときに使われる）．図 6 (A) の $\phi = 2$ の場合は，(a) の状況から，色 W のロボットが色 G のロボットから離れるように移動し，(b) の状況となる．この状況から，色 G のロボットが色 W のロボットへ近づくように移動し，(c) の状況となる．この状況は (a) の状況から各ロボットが距離 1 ずつ東へ移動した結果であり，以上の 2 種類の動作を繰り返すことで東へ移動し続けることができる．図 6 (B) の $\phi = 1$ の場合は，(a) の状況から，色 G のロボットが隣の色 W のロボットへ向かって移動し，(b) の状況となる．この状況から，色 G のロボットと同じノードにいる色 W のロボットが，自身の色を G にして隣の色 W のロボットへ向かって移動し，(c) の状況となる．この状況から，色 W のロボットと同じノードにいる色 G のロボットが，自身の色を W にして隣の色 G のロボットから離れるように移動し，(d) の状況となる．この状況は (a) の状況から各ロボットが距離 1 ずつ東へ移動した結果であり，以上の 3 種類の動作を繰り返すことで東へ移動し続けることができる．

4　探索アルゴリズムの一例

我々が設計したアルゴリズムの一つである，完全同期かつ $\phi = 1$，$l = 3$ で共通の chirality を有しているロボット 2 体による，$m \times n$ グリッドの停止探索アルゴリズムを図 7 に示す．このアルゴリズムの実行を初期状況 $C_0 = \{((0,0),\{G\}),((0,1),\{W\})\}$ から開始することで停止探索を達成できる．

■東への直進　色 G のロボットと色 W のロボットが横 1 列に並んでいる状況で，色 W のロボットがルール $R1$，色 G のロボットがルール $R2$ を同時に実行することで，これら 2 体のロボットは東へ直進することができる．

■西への折り返し準備　アルゴリズムの実行における西への折り返し準備の過程を図 8 に示す．東への直進を続けた 2 体のロボットは，やがてグリッドの東端に到達する（図 8（a））．この状況から，ルール $R3$ によって，色 W のロボットは自身の色を G に変え，グリッドの外側と向かい合って右側に移動する（つまり，南に下がる）．これと同時に，ルール $R2$ によって，色 G のロボットが移動する．結果，色 G のロボット 2 体がグリッドの東端で縦 1 列に並んでいる状況となる（図 8（b））．この状況から，ルール $R4$ によって，より南側のロボットが自身の色を B に変え，グリッドの内側に移動する．同時に，ルール $R5$ によって，もう一方のロボットが南側のロボットに付いて行く形で移動する．結果，色 B のロボットと色 G のロボットが横 1 列に並んでいる状況となる（図 8（c））．

■西への直進　色 B のロボットと色 G のロボットが横 1 列に並んでいる状況で，色 B のロボットがルール $R6$，色 G のロボットがルール $R7$ を同時に実行することで，これら 2 体のロボットは西へ直進することができる．

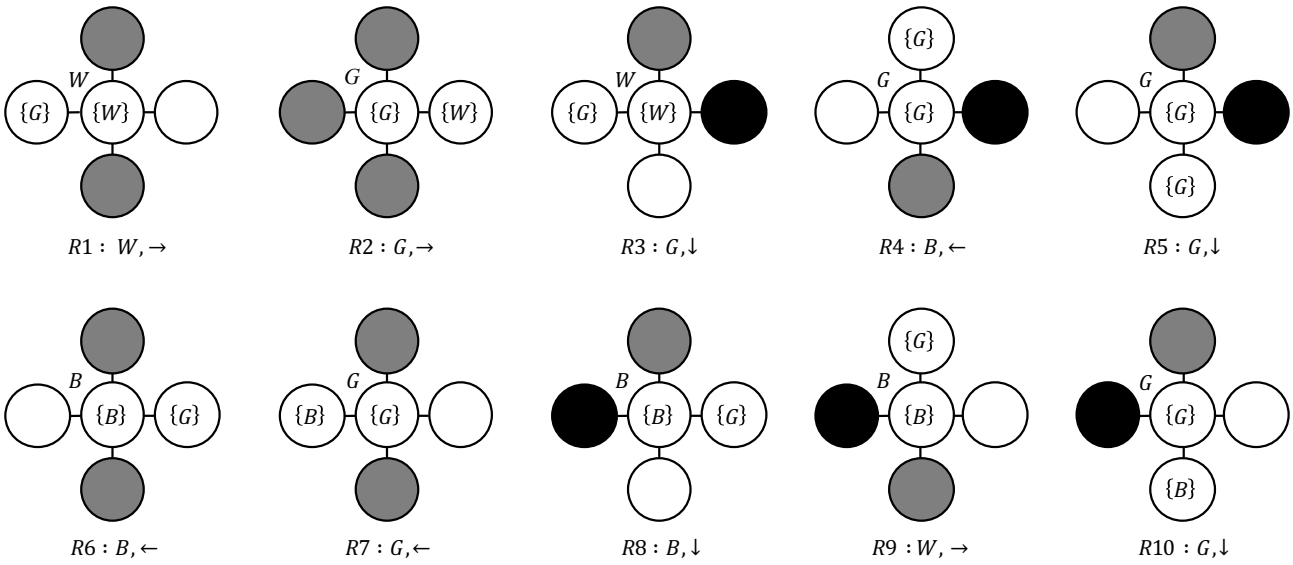


図 7　完全同期かつ $\phi = 1$，$l = 3$ で共通の chirality を有しているロボット 2 体による $m \times n$ グリッドの停止探索アルゴリズム

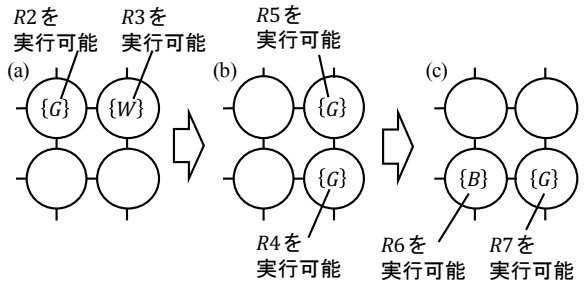


図 8　完全同期かつ $\phi = 1$，$l = 3$ で共通の chirality を有しているロボット 2 体による西への折り返し準備の過程

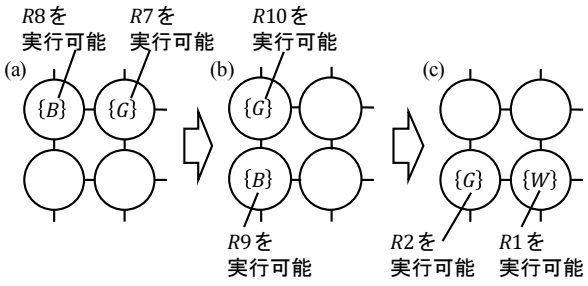


図 9　完全同期かつ $\phi = 1$，$l = 3$ で共通の chirality を有しているロボット 2 体による東への折り返し準備の過程

■東への折り返し準備　アルゴリズムの実行における東への折り返し準備の過程を図 9 に示す．西への直進を続けた 2 体のロボットは，やがてグリッドの西端に到達する（図 9（a））．この状況から，ルール $R8$ によって，色 B のロボットはグリッドの外側と向かい合って左側に移動する（つまり，南に下がる）．これと同時に，ルール $R7$ によって，色 G のロボットが移動する．結果，色 G のロボットと色 B のロボットがグリッドの西端で縦 1 列に並んでいる状況となる（図 9（b））．この状況から，ルール $R9$ によって，色 B のロボットが自身の色を W に変えてグリッドの内側に移動する．同時に，ルール $R10$ によって，色 G のロボットがもう一方のロボットに付いて行く形で移動する．結果，色 G のロボットと色 W のロボットが横 1 列に並んでいる状況となり，再び東へ直進することが可能となる（図 9（c））．

■探索の終了　ロボットが全ノードを訪問して隅に到達した後，ルールを実行可能なロボットがいなくなり，探索が終了する．m が奇数の場合は，色 G のロボットと色 W のロボットが東に直進することで南端のノードを順に訪問し，南東の隅に到達する．色 W のロボットが南東の隅のノード $(m-1, n-1)$ を訪問したとき，状況は $\{((m-1, n-2), \{G\}), ((m-1, n-1), \{W\})\}$ である．この状況では，色 W のロボットが実行可能なルールは存在しない．一方，色 G のロボットはルール $R2$ によって移動する．結果，状況は $\{((m-1, n-1), \{W, G\})\}$ となる．この状況では，いずれかのロボットが実行可能なルールは存在せず，以降はどのロボットもルールを実行しない．m が偶数の場合は，色 B のロボットと色 G のロボットが西に直進することで南端のノードを順に訪問し，南西の隅に到達する．色 B のロボットが南西の隅のノード $(m-1, 0)$ を訪問したとき，状況は

$\{((m-1,0),\{B\}),((m-1,1),\{G\})\}$ である．この状況では，色 B のロボットが実行可能なルールは存在しない．一方，色 G のロボットはルール $R7$ によって移動する．結果，状況は $\{((m-1,0),\{B,G\})\}$ となる．この状況では，いずれかのロボットが実行可能なルールは存在せず，以降はどのロボットもルールを実行しない．このようにして，ロボットは全ノードを訪問した後にルールを実行しなくなり，停止探索を達成できる．

5 まとめ

本稿では，視界に制限のある自律移動ロボット群による $m \times n$ グリッドの停止探索アルゴリズムを検討した．本稿で扱ったモデルは，ロボットは定数距離 ϕ 以内のノードを観測でき，他のロボットからも観測可能な定数 l 個の色を持つライトを利用できるというものであった．また，左右の概念である chirality を共有しているときと共有していないときの両方の場合を扱った．

今後の課題を 3 点挙げる．第 1 に，検討したモデルにおいて探索に必要なロボット数の下界を示すことである．第 2 に，検討したモデルより制約の強い場合（例えば，非同期かつライトの色数が 1 の場合）に探索を達成するアルゴリズムが存在するか否かを示すことである．第 3 に，同様のモデルによる他のネットワークの探索問題を検討することである．

参考文献

[1] I. Suzuki and M. Yamashita, "Distributed anonymous mobile robots: Formation of geometric patterns," SIAM Journal on Computing, vol.28(4), pp.1347-1363, March 1999.

[2] P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, "Gathering of asynchronous robots with limited visibility," Theoretical Computer Science, vol.337(1-3), pp.147-168, June 2005.

[3] N. Fujinaga, Y. Yamaguchi, H. Ono, S. Kijima, M. Yamachita, "Pattern formation by oblivious asynchronous mobile robots," SIAM Journal on Computing, vol.44(3), pp.740-785, June 2015.

[4] L. Blin, A. Milani, M. Potop-Butucaru, S. Tixeuil, "Exclusive perpetual ring exploration without chirality," Proc. 24th International Symposium on Distributed Computing, pp.312-327, Cambridge, MA, USA, September 2010.

[5] S. Devismes, A. Lamani, F. Petit, S. Tixeuil, "Optimal probabilistic ring exploration by semi-synchronous oblivious robots," Theoretical Computer Science, vol.498, pp.10-27, August 2013.

[6] P. Flocchini, D. Ilcinkas, A. Pelc, N. Santoro, "Computing without communicating: Ring exploration by asynchronous oblivious robots," Algorithmica, vol.65(3), pp.562-583, January 2013.

[7] F. Bonnet, A. Milani, M. Potop-Butucaru, S. Tixeuil, "Asynchronous exclusive perpetual grid exploration without sense of direction," Proc. 15th International Conference on Principles of Distributed Systems, pp.251-265, Toulouse, France, December 2011.

[8] P. Flocchini, D. Ilcinkas, A. Pelc, N. Santoro, "Remembering without memory: Tree exploration by asynchronous oblivious robots," Theoretical Computer Science, vol.411(14-15), pp.1583-1598, March 2010.

[9] S. Devismes, A. Lamani, F. Petit, P. Raymond, S. Tixeuil, "Optimal grid exploration by asynchronous oblivious robots," Proc. 14th International Symposium on Stabilization, Safety, and Security of Distributed Systems, pp.64-76, Toronto, Canada, October 2012.

[10] S. Devismes, A. Lamani, F. Petit, S. Tixeuil, "Optimal torus exploration by oblivious robots," Proc. Third International Conference on Networked Systems, pp.183-199, Agadir, Morocco, May 2015.

[11] J. Chalopin, P. Flocchini, B. Mans, N. Santoro, "Network exploration by silent and oblivious robots," Proc. 36th International Workshop on Graph Theoretic Concepts in Computer Science, pp.208-219, Zarós, Crete, Greece, June 2010.

[12] A.K. Datta, A. Lamani, L.L. Larmore, F. Petit, "Ring exploration by oblivious agents with local vision," Proc. IEEE 33rd International Conference on Distributed Computing Systems, pp.347-356, Philadelphia, USA, July 2013.

[13] A.K. Datta, A. Lamani, L.L. Larmore, F. Petit, "Ring exploration by oblivious robots with vision limited to 2 or 3," Proc. 15th International Symposium on Stabilization, Safety, and Security of

Distributed Systems, pp.363-366, Osaka, Japan, November 2013.

[14] S. Das, P. Flocchini, G. Prencipe, N. Santoro, "Autonomous mobile robots with lights," Theoretical Computer Science 609, pp.171-184, January 2016.

[15] F. Ooshita, S. Tixeuil, "Ring exploration with myopic luminous robots," Proc. 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems, pp.301-316, Tokyo, Japan, November 2018.

[16] Q. Bramas, S. Devismes, and P. Lafourcade, "Brief announcement: Infinite grid exploration by disoriented robots," in *SIROCCO 2019*. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02145822

Mining Game in Concurrent Chains

Tomoki Umino, Taisuke Izumi

September 7, 2019

Abstract

In the Bitcoin system, participants obtain the permission of appending a block to the tail of the chair by solving mathematical puzzles. Since they earn rewards by writing block, this process is often called *mining*. In this paper, we consider the situation where the system has a multiple number of chains, each of which offers a different amount of rewards per one block writing. Each rational player wants to write a block offering higher rewards, but too competitive writing increases the risk of dropping out the written block from the canonical history, which results in the decrease of social welfare (i.e., throughput of block writing). Here a game-theoretic problem arises: Which chain each (rational) player should choose for maximizing its local benefit? Then how bad does the social welfare become?

In this study, w e introduce a new game called *Mining Game* for the (possibly multiple) cryptocurrency system which supports simultaneous writing of multiple blocks containing no transaction conflict, and provides the analysis of Nash equilibriums there. We present the existence of a Nash equilibrium, as well as the upper bound for its *price of anarchy (PoA)*, that is, the ratio between the social welfare optimized under the central control and that in the worst-case Nash equilibrium. We obtain the PoA is bounded by $\frac{r_0}{r_m} \cdot \exp(\alpha + 1)$, where r_0 and r_m are the highest and lowest rewards respectively and α is the difficulty of puzzles relative to the total hash power of players in the system. We also show that the Mining Game is equivalent to the *Load Balancing Game* with respect to the convergence time of reaching a Nash equilibrium, presenting its explicit upper and lower bounds.

1 Introduction

The Bitcoin is one of the biggest cryptocurrency markets introduced by Satoshi Nakamoto [1], which consists of the distributed ledger technology based on the consensus algorithm. In distributed ledger, each block, a collection of transactions, is linked by a hash pointer. That is, the distributed ledger is a linked list of blocks (i.e., *blockchain*). Every node in the Bitcoin network reaches an agreement on the ledger. To avoid inconsistent forks of the chain, the system employs the *Proof of Work(PoW)* as the consensus algorithm. The PoW is a

kind of the leader election algorithms. The node paticipating the election (called *miner*) must mathematical puzzles to obtain the permission of appending the next block. Miners earns a reward if it succeeds to append a block, whose ingredient is typically block rewards and the fee for processing the transactions written in the block. For winning the PoW leader election, each miner has to solve a mathematical puzzle. The miner solving that first becomes the leader for the next block.

If two or more miners succeeded in solving the puzzle at the same time, it potentially causes an inconsistent fork of the ledger. In that case one of the forks eventually survives.

1.1 Our result

In this paper, we consider the situation where the system has a multiple number of chains, each of which offers a different amount of rewards per one block writing. Each rational player wants to write a block offering higher rewards, but too competitive writing increases the risk of dropping out the written block from the canonical history, which results in the decrease of social welfare (i.e., throughput of block writing). Here a game-theoretic problem arises: Which chain each (rational) player should choose for maximizing its local benefit? Then how bad does the social welfare become? To this goal, we newly introduce an n-player noncooperative game called the *mining game*. It consist of a set of n players V and a set of m blocks B, where $V = \{v_0, v_1, \cdots, v_{n-1}\}$ and $B = \{b_0, b_1, \cdots, b_{m-1}\}$. A block $b_i \in B$ has a corresponding reward r_i and a player v_i has the hash-power p_i, where $0 < p_i < 1$. The task of solving puzzles is modeled as a stochastic process. A player v_i succeeds in mining with probability p_i. Each player selects a block which he mine among the B. A player v_i which selects b_i as his own action can get a corresponding reward r_i if other players which select b_i fail in mining and he succeed in mining. Therefore, the utility of a player which selects b_i as his own action is the expected reward. The utility of the system is the expected number of written blocks. In this study, we prove that there is a Nash equilibrium in the Mining Game and calculate the quality of the Nash equilibrium. Furthermore, we show that the Mining Game is equivalent to the Load Balancing Game from the perspective of the convergence time to reach a Nash equilibrium. Our contributions are as follows.

1. We formalize strategic mining in the cryptocurrency which allows multiple block writing without conflict (Section 2).

2. We prove that there exists a Nash equilibrium in such a game, and present an upper bound on the quality of Nash equilibria, which is measured by the ratio between the utility of the system which is controlled in the centralized manner and the utility of the system by rational players is $\frac{r_0}{r_m} \cdot \exp(\alpha + 1)$ if the difficulty of the puzzle is controlled by the system, where r_0 and r_m are a highest reward and a lowest reward respectively and α is a variable which is controlled by the system(Section 3).

3. We prove that the mining game is equivalent to the *load balancing game* with respect to the convergence time to reach a Nash equilibrium and present an upper bound and a lower bound on the convergence time. Under several assumptions, there exists a case which requires at least $(\frac{n}{m-1})^{m-1}/(2(m-1)!)$ steps to reach a Nash equilibrium for the Mining Game, and any instance of the Mining Game reaches to a Nash equilibrium at most n steps under the best assumption (Section 4).

2 Preliminary

2.1 Blockchain System

In this paper, we consider the blockchain system which supports m parallel writing of blocks. The system has a set $B = \{b_0, b_1, \ldots, b_{m-1}\}$ of m blocks to be appended to chains, and a set $V = \{v_0, v_1, \ldots, v_{n-1}\}$ of n players choose one of the blocks in B and tries to append it by solving the corresponding puzzle. Despite the nature of full asynchrony and long-liveness in block-chain systems, we regard the task of writing blocks as a one-shot synchronous task for simplicity. Specifically, the run of the system proceeds with the consecutive two phases defined as follows:

1. **Block selection phase**: Each player decides a block which it will write to the corresponding chain. It is assumed that players can see other players' choice and can change their own decision.

2. **Mining phase**: Each player tries to obtain the permission of writing the chosen block by solving the corresponding mining puzzle. A player v_i obtains the permission for block b_i if and only if it succeeds in solving the mining puzzle of b_i and no other player succeeds, Then v_i earns the corresponding reward r_i. [1]

Each player v_i has a parameter p_i called *hash power* which indicates its own computational capacity per unit time. Let h_i be the difficulty of a puzzle corresponding to block b_i. We model the task of solving a puzzle as a stochastic process. The player v_i solves the mining puzzle of b_i within the mining phase with probability p_i/h_i. To simplify the model, we assume $h_i = \beta$ for all i, Then, the summation of p_i/β over all i obviously represents the expected number of players that can solve a puzzle. We also denote $\alpha = \sum_i (p_i/\beta)/m$. Intuitively , the value of α means the throughput ratio(i.e., the number of blocks simultaneously written). In well-managed systems, we can expect that β is appropriately controlled so that α does not become so high. By normalizing the values of hash power, we can assumes $\beta = 1$ without loss of generality.

[1] While we assume a deadline of the mining phase in this model, actual blockchain systems are asynchronous and thus instead of deadlines,the player solving the puzzle first gets the permission. However, this difference is not essential for the analyses shown in this paper.

2.2 Mining Game

Mining Game(MG) is defined by 4-tuple (n, m, P, R), where n is the number of players, m is the number of blocks, $P = (p_0, p_1, \ldots, p_{n-1})$ is the vector representing the hash-power of each player , and $R = (r_0, r_1, \ldots, r_{m-1})$ is the vector representing the reward of each block. A strategy space S_i for any player v_i ($i \in [0, n-1]$) is the block set B. Hence the global strategy space $\mathcal{S}$ is B^n. We denote by s_i ($i \in [0, n-1]$) the action of a player v_i. Given a global strategy $s = (s_0, s_2, \ldots, s_{n-1}) \in \mathcal{S}$, we define a set of miners who are mining a block b_j as $V_j(s) = \{v_i \mid s_i = b_j\}$. Given a global strategy s, we define the expected utility $u(s, v_i)$ for player v_i as follows.

$$u(s, v_i) = r_i p_i \prod_{v_j \in V_{s_i}(s) \setminus \{v_i\}} (1 - p_j)$$

The expected utility $U(s)$ for the blockchain system is defined by expected number of written blocks.

$$\begin{aligned} U(s) &= \sum_{v_i \in [0,n-1]} \frac{u(s, v_i)}{r_i} \\ &= \sum_{v_i \in [0,n-1]} p_i \prod_{v_j \in V_{s_i}(s) \setminus \{v_i\}} (1 - p_j). \end{aligned}$$

Given a global strategy s, we denote a set of actions without v_i as s_{-i}. We often denote s as (s_i, s_{-i}). Especially, we denote a global strategy which a player v_i changes his own action from s_i to s'_i as (s', s_{-i}).

2.3 Nash Equilibrium and Price of Anarchy

The Nash equilibrium is a solution concept for the n players non-cooperative games. It is defined as follows.

Definition 1 *Let V be a set of players, S be a global strategy space, $u : S \times V \to \mathbb{R}$ be a utility function, and G be a game defined by V, S, and u. A global strategy s^* is a* Nash equilibrium *if the following condition holds for any player $v_i \in V$.*

$$u(s^*, v_i) \geq u((s_i, s^*_{-i}), v_i)(s_i \in S_i).$$

The Nash equilibrium implies that any player cannot increase his utility by changing only his own action. That is, no player has an incentive to change its action.

Given a global strategy s, if a player v_i can increase his utility by changing his own action from s_i to s', then we call this action a *better response step*. We denote by $s^0 \to s^1$ the transition of global strategies by the better response step of a v_i. By the definition of the better response step, if there is no player which can do better response step in a global strategy s^*, then s^* is a Nash equilibrium.

Let $U(s) : S \to \mathbb{R}$ be the function which takes a global strategy s and returns the utility of the system. While every player does not have any incentive to change his own action in Nash equilibria, it does not implies that the utility of the system is maximized. The concept of *price of anarchy(PoA)* is quantifies the degradation of the system utility caused by the selfish behavior of each agent, which is defined as follows:

Definition 2 *Let $U(s) : S \to \mathbb{R}$ be the utility function of a system and E be a set of Nash equilibriums. PoA is defined as follows.*

$$PoA = \frac{\max_{s \in S} U(s)}{\min_{e \in E} U(s)}$$

3 Existence of Pure Nash Equilibrium and Bound for Price of Anarchy

First, we show that there exists a Nash equilibrium for any instance of MG. Letting s be any strategy, and s' be that induced from s by any better response step, a function $f : S \to \mathbb{R}$ is called a *ordinal potential function* if $f(s) > f(s')$ holds for any such pair s and s'. It is easy to see that any finite game with an ordinary potential function has a pure nash equilibrium [3]. We have the following lemma.

Lemma 1 *For any instance I of MG, there exists the Nash equilibrium s^*.*

Proof 1 *We show that MG has an ordinal potential function. Given a global strategy s, we define a density $D_i(s)$ for a block b_i as follows.*

$$D_i(s) = r_i \cdot P_i(s)$$
$$P_i(s) = \begin{cases} \prod_{v_l \in V_i(s)} (1 - p_l) & V_i(s) \neq \emptyset \\ 1 & V_i(s) = \emptyset \end{cases}$$

Suppose that v_x in strategy s changes its action from b_i to b_j, and the global strategy results in s'. Then, we have the following equality.

$$\begin{aligned} & D_i(s) + D_j(s) - (D_i(s') + D_j(s')) \\ &= r_i \cdot P_i(s) + r_j \cdot P_j(s) - r_i \cdot \frac{1}{1 - p_x} \cdot P_i(s) - r_j \cdot (1 - p_x) \cdot P_j(s) \\ &= r_i \cdot P_i(s)(1 - \frac{1}{1 - p_x}) + r_j \cdot P_j(s)(1 - 1 + p_x) \\ &= r_j \cdot P_j(s) \cdot p_x - r_i \cdot P_i(s) \cdot \frac{p_x}{1 - p_x} \\ &= u(s', v_x) - u(s, v_x) \end{aligned}$$

The inequality $u(s', v_x) - u(s, v_x) > 0$ if and only if the change of v_x's action from b_i to b_j is a better response step. That is, $D_i(s) + D_j(s)$ for any s decreases if we

update s by any better response step. In addition, any better response step from b_i to b_j affects only the densities of b_i and b_j, and thus $D(s)$ $\sum_{b_i \in B \setminus b_i, b_j} D_i(s) = \sum_{b_i \in B \setminus b_i, b_j} D_i(s')$ holds. It implies that $D(s) = \sum_{b_i \in B} D_i(s)$ is an ordinal potential function. The lemma is proved..

Next we present an upper bound of the PoA of the mining game. First, we show a property of Nash equilibrium of MG.

Lemma 2 *Let s be a global strategy, and $v_x \in V_i(s)$ be the player whose hash power is the minimum in $V_i(s)$. If the change of v_x's action from b_i to b_j is not a better response step in a global strategy s, no player in $V_i(s)$ can increase its utility by changing its action to b_j.*

Proof 2 *Let v_y be any player in $V_i(s)$. Since v_x's action from b_i to b_j is not a better response step, we have $u(s, v_x) > u((j, s_{-x}), v_x)$ holds. We obtain the following inequality.*

$$
\begin{aligned}
& u(s, v_x) > u((j, s_{-x}), v_x) \\
& \Leftrightarrow r_i \cdot p_x \cdot \prod_{v_l \in V_i(s) \setminus \{v_x\}} (1 - p_l) > r_j \cdot p_x \cdot \prod_{v_l \in V_j(s)} (1 - p_l) \\
& \Leftrightarrow \frac{r_i}{r_j} > \frac{\prod_{v_l \in V_j(s)} (1 - p_l)}{\prod_{v_l \in V_i(s) \setminus \{v_x\}} (1 - p_l)}
\end{aligned}
$$

Since the right side of this inequality is monotonically decreasing in terms of p_y. Thus for any $v_y \in V_i$, we have

$$
\begin{aligned}
& \frac{r_i}{r_j} > \frac{\prod_{v_l \in V_j(s)} (1 - p_l)}{\prod_{v_l \in V_i(s) \setminus \{v_x\}} (1 - p_l)} \geq \frac{\prod_{v_l \in V_j(s)} (1 - p_l)}{\prod_{v_l \in V_i(s) \setminus \{v_y\}} (1 - p_l)} \\
& \Leftrightarrow u(s, v_y) > u((j, s_{-y}), v_y).
\end{aligned}
$$

That is, v_y's action from b_i to b_j is not a better response step.

We show the main theorem.

Theorem 1 *The PoA of the mining game is bounded as follows:*

$$PoA \leq \frac{r_0}{r_m} \cdot \exp(\alpha + 1)$$

Proof 3 *Given a Nash equilibrium s^*, Let $\hat{u}_i$ be the logarithmic utility of $u(s^*, v_i)$ for player v_i defined as follows:*

$$
\begin{aligned}
\hat{u}_i &= \log u(s^*, v_i) \\
&= \log r_{s_i^*} + \log p_i + \sum_{v_l \in V_{s_i^*}(s) \setminus \{v_i\}} \log(1 - p_l)
\end{aligned}
$$

Using $\hat{u_i}$, we obtain the following description of the system utility $U(s)$.

$$\begin{aligned} U(s) &= \sum_{b_i \in B} \sum_{v_j \in V_i(s)} p_j \times \prod_{v_l \in V_i(s) \setminus \{v_i\}} (1 - p_l) \\ &= \sum_{b_i \in B} \sum_{v_j \in V_i(s)} \exp(\hat{u_j} - \log r_i) \\ &= \sum_{b_i \in B} \exp(-\log r_i) \sum_{v_j \in V_i(s)} \exp(\hat{u_j}) \end{aligned}$$

Let $w_i = \sum_{v_j \in V_i(s)} p_j$ and $\hat{w} = \frac{\sum_{b_i \in B} w_i}{m}$. Using the Taylor expansion of $\log(1+x)(-1 < x \leq x)$, $x - \frac{x^2}{2} < \log(1+x) < x$. Now, we obtain the lower bound on $U(s)$.

$$\begin{aligned} U(s) &> \sum_{b_i \in B} \exp(-\log r_i) \sum_{v_j \in V_i(s)} \exp(\log r_j + \log p_j - \sum_{v_l \in V_i(s) \setminus \{v_j\}} p_l) \\ &= \sum_{b_i \in B} \sum_{v_j \in V_i(s)} p_j \cdot \exp(-w_i + p_j) \\ &= \sum_{b_i \in B} \exp(-w_i) \sum_{v_j \in V_i(s)} p_j \cdot \exp(p_j) \\ &> \sum_{b_i \in B} \exp(-w_i) \sum_{v_j \in V_i(s)} p_j \\ &= \sum_{b_i \in B} \exp(-w_i) \cdot w_i \end{aligned}$$

For any block $b_i \in B$, Let q_i be the player with the lowest-power hash power in V_i. By the definition on Nash Equilibria, the following inequality holds for any block b_i by Lemma 2.

$$r_i \cdot p_{q_i} \cdot \prod_{v_l \in v_i \setminus \{q_i\}} (1 - p_l) > r_j \cdot p_{q_i} \cdot \prod_{v_l \in V_j(s)} (1 - p_l)$$

$$\log r_i + \log p_{q_i} + \sum_{v_l \in V_i(s) \setminus \{q_i\}} \log(1 - p_l) > \log r_j + \log p_{q_i} + \sum_{v_l \in V_j(s)} \log(1 - p_l)$$

$$\log r_i + \sum_{v_l \in V_i(s) \setminus \{q_i\}} \log(1 - p_l) > \log r_j + \sum_{v_l \in V_j(s)} \log(1 - p_l)$$

$$\log r_i - w_i + q_i > \log r_j - w_j$$

Let $w_i' = w_i - \log r_i$. Then it follows

$$w_i' - q_i < w_j'$$

Since the same discussion also holds for another block b_j. Thus we have $w'_j - q_j < w'_i$. It follows that $w'_i - q_i < w'_j < w'_i + q_j$ holds for any 2 blocks $b_i, b_j \in B$. Let $w'_{min} = \min_{b_i \in B} w'_i$. For any block b_i, $w'_i < w'_{min} + q_i$ holds. And we define $\hat{w'} = \frac{\sum_{b_i \in B} w'_i}{m}$. $w'_{\min} = \beta \hat{w'}$, where β is an appropriate value less than one. Therefore, $w'_i < \beta \hat{w'} + q_i$ holds. We obtain the following bound.

$$\begin{aligned}
U(s) &> \sum_{b_i \in B} \exp(-w'_i - \log r_i) \cdot w_i \\
&= \sum_{b_i \in B} \frac{1}{r_i} \cdot \exp(-w'_i) \cdot w_i \\
&> \sum_{b_i \in B} \frac{1}{r_i} \cdot \exp(-\beta \hat{w'} - q_i) \cdot w_i \\
&= \exp(-\beta \hat{w'}) \sum_{b_i \in B} \frac{1}{r_i} \cdot \exp(-q_i) \cdot w_i \\
&> \exp(-1) \cdot \exp(-\beta \hat{w'}) \sum_{b_i \in B} \frac{1}{r_i} \cdot w_i \\
&> \exp(-1) \cdot \exp(-\beta \hat{w'}) \cdot \frac{1}{r_0} \sum_{b_i \in B} w_i \\
&= \exp(-1) \cdot \exp(-\beta \hat{w'}) \cdot \frac{1}{r_0} \cdot \alpha m
\end{aligned}$$

The following inequality holds for $\hat{w'}$.

$$\begin{aligned}
\hat{w'} &= \frac{\alpha m - \sum_{b_i \in B} \log r_i}{m} \\
&< \frac{\alpha m - m \log r_m}{m} \\
&= \alpha - \log r_m
\end{aligned}$$

Since $U(s)$ is expected number of written blocks in the system. $U(s) < \sum_{i \in [0, n-1]} p_i = \alpha m$ obviously holds. Finally, the following inequality holds.

$$\begin{aligned}
PoA &< \frac{\alpha m}{\exp(-1) \cdot \exp(-\beta \hat{w'}) \cdot \frac{1}{r_0} \cdot \alpha m} \\
&= \frac{1}{\exp(-1) \cdot \exp(-\beta \hat{w'}) \cdot \frac{1}{r_0}} \\
&= r_0 \cdot \exp(\beta \hat{w'} + 1).
\end{aligned}$$

By the fact of $\beta < 1$, we have

$$\begin{aligned}
&< r_0 \cdot \exp(\hat{w'} + 1) \\
&< r_0 \cdot \exp(\alpha - \log r_m + 1) \\
&= \frac{r_0}{r_m} \cdot \exp(\alpha + 1).
\end{aligned}$$

4 Bounds for Convergence Time

In this section, we discuss about the convergence time of the mining game, i.e., the number of better response steps necessary to reach a Nash equilibrium. For analyses, we introduce yet another game called *load balancing game* (LBG), and shows that a special case of MG is reducible from LBG. It implies that the convergence-time lower bound for LBG also applies to MG. We also present an upper bound for (general) MG, which is not based on the reduction but the proof is very similar with that for LBG.

4.1 Load Balancing Game

We consider a system consisting of a set $J = \{j_0, j_1, \cdots, j_{n-1}\}$ of n jobs and a set $M = \{m_0, m_1, \cdots, m_{m-1}\}$ of m machines. Each job has a weight and each machine has a capacity. If many jobs concentrate in a single machine, it takes a longer time to process them because the resource of the machine is shared among assigned jobs. Thus, each rational player tries to find the machine where its own job can be processed fast. A load balancing game is define by a 4-tuple (n, m, W, c), where $W = (w_0, w_1, \cdots, w_{n-1})$ is the job-Weight vector and and $C = (c_0, c_1, \cdots, c_{m-1})$ is the machine-capacity vector. The weight of a job j_i is w_i, and the capacity of a machine m_i is c_i. Each job selects a machine to process itself. Thus a strategy space S_i for a job j_i is M and a global strategy space S is M^n. Given a global strategy s where $s = (s_0, s_1, \cdots, s_{n-1})$, we denote by $M_i(s)$ the set of jobs selecting m_i. as their own actions, i.e., $M_i(s) = \{j_i \in J \mid s_i = m_i\}$. Given a global strategy s, we define the cost $l(s, j_i)$ of a job j_i as follows.

$$l(s, j_i) = \frac{\sum_{j_k \in M_{s_i}(s)} w_k}{c_{s_i}}$$

Each job aims to minimize its cost. Throughout this paper, we consider a simplified version of LBG, assuming $c_i = 1$ for any $i \in [0, n-1]$. Eyal Even-Dar, et al. proved that there exists a Nash equilibrium in the Load Balancing Game, and present upper and lower bounds for the convergence time under several assumptions [4]. More precisely, they introduce the best response policy as the assumption on the behavior of each player, where any job j_i having two or more better response steps always selects the action which minimize its cost. They also introduce two policies of the scheduler (i.e., the choice of jobs), called *max-Weight job strategy* and *min-Weight job strategy*. Let $A(s)$ be the set of jobs having at least one better response step in s. Under the max-weight job strategy, a job with the maximum weight in $A(s)$ is chosen by the scheduler, In contrast, the min-Weight job strategy, a job with the minimum weight in $A(s)$ is chosen. The following bounds are proved in [4].

Lemma 3 *For the Max Weight Job strategy with best response policy, the Load Balancing Game reaches to a Nash equilibrium at most n steps.*

Lemma 4 *For the Min Weight Job strategy with best response policy, there are a global strategy s that requires at least $(\frac{n}{m-1})^{m-1}/(2(m-1)!)$ steps to reach a Nash equilibrium.*

4.2 Reduction from The Load Balancing Game

In the mining game, a player migrates to a block with higher density as a better response step. As mentioned above, the density of a block b_i is defined by the product of a block reward r_i and the fail probability of all the players in V_i. While this function is highly nonlinear, taking logarithm provides an additive (i.e., linear) form. In other words, the mining game is considered as a load balancing game on the exponent of utility functions, which is the main trick of our reduction. The actual reduction is very simple, which is stated as follows:

1. Associate each block $b_i \in B$ with each machine in $m_i \in M$.
2. Associate each player $v_i \in V$ with each jon $j_i \in J$.
3. Set reward $b_i = c_i$. Since we assume $c_i = 1$ for any i, it implies $r_i = 1$ for any i.
4. Set hash power $p_i = 1 - \exp(-w_i)$.

In the following argument, let I_l be any instance of the load balancing game, and I_m be the reduced instance of the mining game. Let $g : B^n \to M^n$ be the function naturally defined for mapping a strategy of I_m to that of I_l. It is obvious that g is obviously a bijection. Let g^{-1} be the inverse function of g. First, we show that Nash equilibria in I_l and I_m has one-to-one correspondence.

Lemma 5 *A strategy s^* of I_m is a Nash equilibrium if and only if $g(s^*)$ is a Nash equilibrium of I_l.*

Proof 4 *We only show that $g(s)$ is a Nash equilibrium if s^* is a Nash equilibrium. The opposite direction is easily proved by tracing the following argument inversely. For any player $v_i \in V$ and any block $b_j \in B \setminus \{s_i^*\}$, the following inequality holds.*

$$\frac{p_i}{1-p_i} \cdot \prod_{v_l \in V_{s_i^*}(s^*)} (1-p_l) > p_i \cdot \prod_{v_l \in V_j(s^*)} (1-p_l)$$

$$\frac{1}{1-p_i} \cdot \prod_{v_l \in V_{s_i^*}(*)} (1-p_l) > \prod_{v_l \in V_j(s^*)} (1-p_l)$$

Taking the logarithm of both sides

$$-\log(1-p_i)+\sum_{v_l\in V_{s_i^*}(s^*)}\log(1-p_l)>\sum_{v_l\in V_j(s^*)}\log(1-p_l)$$

$$w_i-\sum_{v_l\in V_{s_i^*}(s^*)}w_l>-\sum_{v_l\in V_j(s^*)}w_l$$

$$\sum_{v_l\in V_{s_i^*}(s^*)}w_l<\sum_{v_l\in V_j(s^*)}w_l+w_i$$

The summation $\sum_{v_l\in V_i(s^*)}w_l$ for block b_i is the total weight of machine m_i in $g(s^*)$. The above inequality implies that in $g(s^*)$ there are no job which can decrease its cost by changing its own action. That is, $g(s^*)$ is a Nash equilibrium in the I_l.

Next, we show that any better response step in I_m corresponds to a better response step of I_l.

Lemma 6 *Any strategy change in I_m from s to s' is a better response step only if the corresponding change in I_l from $g(s)$ to $g(s')$ is a better response step.*

Proof 5 *Suppose that s changes to s' by a better response step of player v_i in the I_m. Then, the following inequality holds.*

$$\prod_{v_l\in V_{s_i}(s)\setminus\{v_i\}}(1-p_l)<\prod_{v_l\in V_{s_i'}(s')}(1-p_l)$$

Taking the logarithm of both sides, we can get the following formula.

$$\sum_{v_l\in V_{s_i}(s)}w_l>\sum_{v_l\in V_{s_i'}(s')}w_l+w_i$$

It implies that any better response step on the I_m is also a better response step in I_l.

Theorem 2 *For the min-hash-power strategy with the best response policy, there exists a global strategy which requires at least $\frac{n}{m-1}^{m-1}/2((m-1)!)$ better response steps to reach a Nash equilibrium.*

Proof 6 *For the min-weight job strategy with the best response policy, there exists a global strategy which requires at least $\frac{(\frac{n}{m-1})^{m-1}}{2(m-1)!}$ steps to reach a Nash equilibrium for the Load Balancing Game [4]. Since the min-weight job strategy obviously corresponds to the min-hash-power strategy by Lemmas 5 and 6, the theorem is proved.*

Next, we show that for the Max Hash-power strategy with best response policy, any instance of the Mining Game reach to a Nash equilibrium at most n better response steps. First, we derive some general properties. The next lemma states the highest density cannot increase.

Lemma 7 *The highest density among the blocks either remains the same or decreases while the system reaches to a Nash equilibrium.*

Proof 7 *If a player v_x migrates from a block b_i to a block b_j in a global strategy s as a better response step, then the following formula holds.*

$$\frac{p_x}{1-p_x} \cdot D_i(s) < p_x \cdot D_j(s)$$
$$D_i(s) < (1-p_x) \cdot D_j(s)$$

Thus, $D_i(s) < (1-p_x) \cdot D_j(s) < D_j(s)$ and $D_i(s) < \frac{1}{1-p_x} \cdot D_i(s) < D_j(s)$. Furthermore, players who select the block with highest density cannot increase their utilities by migrating to other blocks. Therefore, the highest density among the blocks either remains the same or decreases while the system reaches to a Nash equilibrium.

Theorem 3 *For the Max Hash-power strategy with best response policy, any instance of the Mining Game reaches to a Nash equilibrium at most n better response steps.*

Proof 8 *Suppose that a player v_i has migrated to block b_{max} with highest density on a global strategy s and the s transits to a new global strategy s'. By the lemma 7, the only reason that v_i wishes to switch block is that another player migrated to the b_{max}. Now we consider the necessary condition of another player who makes v_i migrate to another block. Let v_j be a player who has migrated to the b_{max} on the s'. Since v_i migrated to the b_{max}, the block $b_{s'_j}$ is the only block where v_i can increases his utility by migrating. If v_i can increase his utility by migrating to the $b_{s'_j}$, then the following formula holds.*

$$\frac{p_i}{1-p_j} \cdot D_{s'_j}(s') > p_i \cdot \frac{1-p_j}{1-p_i} \cdot D_{max}(s')$$
$$\frac{1}{1-p_j} \cdot D_{s'_j}(s') > \frac{1-p_j}{1-p_i} \cdot D_{max}(s')$$

Since the v_j migrated to the b_{max} and increased his utility, $D_{max}(s') > \frac{1}{1-p_j} D_{s'_j}(s')$ Thus, $\frac{1-p_j}{1-p_i}$ must be less than 1. Therefore, the v_i can increase his utility by migrating to the $b_{s'_j}$ if $p_i < p_j$. Under the Max Hash-power strategy only players with lower hash-power can arrive in the subsequent, so each player stabilizes after the first migration, and the theorem follows.

References

[1] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[2] Cryptocurrency market state visualization. https://coin360.com/, Accessed 2019-08-26

[3] D. Monderer and L.S. Shapley. Potential games. Games and Economic Behavior, 14:124–143, 1996

[4] Even-Dar E., Kesselman A., Mansour Y. (2003) Convergence Time to Nash Equilibria. In: Baeten J.C.M., Lenstra J.K., Parrow J., Woeginger G.J. (eds) Automata, Languages and Programming. ICALP 2003. Lecture Notes in Computer Science, vol 2719. Springer, Berlin, Heidelberg

地図を持つエージェントの高速なランデブーについて

柿澤一輝　泉泰介　北村直暉

n 個の頂点から構成される連結無向グラフを考える．ランデブー問題とは，異なる頂点に配置された 2 体以上のエージェントを，単一の頂点上に集合させるという問題である．本研究では同期システムにおける，地図を持つ 2 体のエージェントによるランデブー問題を考える．このシステムにおいて，各エージェントは事前に地図と呼ばれるグラフ全体のトポロジ情報を把握しており，その中における自身の初期位置についても知識を持つものとするが，他方のエージェントの初期位置は知らない．各ラウンドでエージェントは、現在滞在している頂点に留まるか，接続辺を辿って隣接頂点へ移動するという動作を行う．ランデブーアルゴリズムを評価するための指標は，各エージェントが動作を開始してからランデブーを達成するまでの同期ラウンド数とする．グラフにおける 2 体のエージェントの初期位置間の距離を d とすると，$\Omega(d)$ ラウンドがランデブーアルゴリズムの自明な下界となる．本研究が対象とするモデルでの既存研究として，Collins らによる結果 [1] が知られており，$O(d\log^2 n)$ ラウンドでランデブーを達成するアルゴリズムを提案している．また，最悪時の下界として $\Omega(d\log n/\log\log n)$ ラウンドを必要とするようなインスタンスの存在も示している．本研究では，2 エージェントの初期位置が距離 d 以下のすべての頂点対から一様ランダムに選ばれるとしたとき，平均 $O(d\log^{3/2} n)$ ラウンドでランデブーを達成するアルゴリズムを提案する．

参考文献

[1] Andrew Collins, Jurek Czyzowicz, Leszek Gąsieniec, Adrian Kosowski, and Russell Martin. Synchronous rendezvous for location-aware agents. In *International Symposium on Distributed Computing*, pp. 447–459. Springer, 2011.

Evaluation and Ranking of Replica Deployments in Geographic State Machine Replication

Shota Numakura
Dept. of Computer Science and Eng.
Toyohashi University of Technology
Toyohashi, Japan
numakura.shota@usl.cs.tut.ac.jp

Junya Nakamura
Information and Media Center
Toyohashi University of Technology
Toyohashi, Japan
junya@imc.tut.ac.jp

Ren Ohmura
Dept. of Computer Science and Eng.
Toyohashi University of Technology
Toyohashi, Japan
ren@tut.jp

***Abstract*—Geographic state machine replication (SMR) is a replication method in which replicas of a service are located on multiple continents to improve the fault tolerance of a general service. Nowadays, geographic SMR is easily realized using public cloud services; SMR provides extraordinary resilience against catastrophic disasters. Previous studies have revealed that the geographic distribution of the replicas has a significant influence on the performance of the geographic SMR; however, the optimal way for a system integrator to deploy replicas remains unknown. In this paper, we propose a method to evaluate and rank replica deployments to assist a system integrator in deciding a final replica deployment. In the method, we also propose a novel evaluation function that estimates a latency of SMR protocols with round-trip time (RTT). To demonstrate the effectiveness of the proposed method, we build thousands of geographic SMRs on Amazon Web Services and present experimental results. The results show that the proposed method that estimates a latency based on RTTs can generate consistent rankings with reasonable calculation time.**

***Index Terms*—fault tolerance, state machine replication, geo-replication, replica deployment, public cloud**

I. Introduction

Services running on the client-server model may crash or behave unintentionally from time to time due to software bugs or attacks by malicious users. To prevent such problems and continuously provide services to clients, fault tolerance is an important consideration. *State machine replication* (SMR) [1] is commonly used to improve fault tolerance by replicating a service over multiple replicas. In SMR, the replicated service is called replicas, and the state of all replicas are kept consistent by executing a replication protocol. Hence, using this method, an active operation can be continued as a whole even if a failure occurs in a part of the replicas. Several SMR protocols have been proposed in previous studies [2]–[8].

An SMR that deploys replicas on a continental scale is called *geographic SMR* [9]–[14]. Replicas in geographic SMR are separated by a large distance to withstand a catastrophic disaster, such as an earthquake. If some of the replicas fail, the service can be continued by the replicas in other *sites* (regions). With the development of public cloud services after the 2000s, geographic SMR can be easily realized.

Although geographic SMR could have been easily implemented, ways of obtaining the best performance using the optimal replica deployment remain unclear. Performance of a replica deployment depends on several factors, including the location of the leader replica, distances between replicas, and distances between clients and replicas. For example, if replicas are deployed in nearby regions, the time taken for request processing can be shortened, but the fault tolerance will be reduced. In contrast, if the replicas are distributed farther apart from one another, the fault tolerance will increase, but the processing time for a normal request will be slower.

In this paper, we propose a performance-estimation method to determine the optimal replica deployment for building a service using geographic SMR. First, we define the task to find the optimal replica deployment among all possible candidates as *replica deployment decision problem*, which requires to output a ranking of all possible replica deployments sorted by their latencies. The proposed method solves this problem by using an evaluation function that estimates a latency of each replica deployment based on the *round-trip time* (RTT), which is generally regarded as an important parameter in geographic SMR. Although it is unrealistic to actually build all possible replica deployments and measure their latencies, RTTs can be measured relatively easily. Therefore, this evaluation function is practical and can be used to select the optimal deployment for actual service construction.

Finally, we conduct an experimental evaluation using Amazon Web Services with 15 regions to demonstrate the effectiveness and practicality of the proposed method. In the experiment, we actually build thousands of geographic replications and measure their latencies; then we create the measured latency ranking and compare it against the rankings generated by the proposed method. The results exhibit that the proposed method with the RTT-based evaluation function can generate a consistent ranking with reasonable calculation time.

In particular, this paper makes the following contributions:

1) It presents a new method that generate a ranking to assist deciding a replica deployment for geographic SMR.
2) It also presents a evaluation function that consistently calculates latency of a replica deployment by using round-trip time between sites, which can be easily measured compared with the actual latency of the deployment.
3) It conducts exhaustive experiments with thousands of replications built on Amazon Web Services, and evalu-

ates the proposed method and the evaluation function.

II. Background

A. State Machine Replication

State machine replication (SMR) [1] is a replication method for the client-server model. In SMR, the server is modeled by a state machine; thus, on receipt of a message, the server changes its state and sends messages to other processes if necessary. The server's role is replicated over n replicas that independently operate the functions on distinct hosts and interact with clients via request and response messages.

Client requests to be executed are submitted to all replicas, and the order in which different replicas receive these requests may differ due to variations in the communication delays. Therefore, the replicas execute a replication protocol to guarantee that they process requests in the same order to maintain consistency. After a replica processes a request, it replies to the client with the execution result.

There are two variations of SMR; SMR that can withstand crash failures (resp. Byzantine failures) is called CFT SMR (resp. BFT SMR). The number of faulty replicas that a replication can tolerate f is related to n as follows [15]: $n \geq 2f + 1$ for CFT SMR and $n \geq 3f + 1$ for BFT SMR. Hereafter, we assume BFT SMR and $n = 4$ (i.e., $f = 1$); however, the proposed method is applicable for any n and f of BFT SMR and CFT SMR.

B. Related Work

The problem of determining the optimal replica deployment has been extensively studied in the field of data replication. Cook et al. formulated the time required to read and write data as a cost in a simple read-write policy (when reading a data object, refer to one replica. When writing data, a client transfer the data to all servers that have its replica) and proved that this problem is NP-complete [16]. They also proposed an approximation algorithm for the problem. Although the target replication problem is different, their formulation is very similar to the evaluation function proposed in this paper. The survey by Sen et al. [17] provides a comprehensive overview of the previous studies on the data location optimization problem using mathematical models.

In the field of geographic SMR, there are a few methods that optimize a replica deployment [10], [11]. In [10], Liu and Vukolić proposed two methods for geographic SMR: Droppy that dynamically relocates a set of replication leaders according to given replication settings and workload situations, and Dripple that divides the replicated system state into multiple partitions so that Droppy can efficiently relocate the leaders. Eischer and Distler proposed Archer [11] that relocates leaders based on their response times as measured by clients. A Hash-chain-based technique was employed in the protocol to allow clients to detect illegal phases caused by Byzantine replicas to prevent such replicas from being wrongly assigned as leaders.

In this paper, we propose a method that can help identify the best replica deployment when building an geographic SMR. The proposed method differs from these prior studies in several ways. First, the proposed method can be used with any replication protocol by defining an evaluation function to calculate the estimated latency of different replica deployments. In contrast, although Droppy and Archer can dynamically relocates the leader replica locations, they only support leader-based replication protocols. Second, the proposed method can also identify the best replica deployment from all possible replica deployments; this complements these existing methods, which are limited to determining an assignment of replication roles to the replicas in a replication.

III. Replica Deployment Decision Problem

We formally define the problem addressed herein as a *replica deployment decision problem*. In the definition, we call a location wherein a replica (or a client) can be deployed to as a *site*[1]. In the problem, the following inputs are provided by a user.

- n: the number of replicas that the user wants to deploy
- SC: a set of candidate sites wherein replicas can be deployed
- C: a set of client locations

The goal of this problem is to output a ranking[2] of replica deployments sorted by latency (of course, a replica deployment with smaller latency is ranked higher). The user will then choose the final replica deployment for the SMR from this ranking. Here, latency is defined as the time taken by a client from sending a request to the replicas until receiving its response.

IV. Proposed Method

In this section, we propose a method to solve the replica deployment decision problem and to determine the optimal replica deployment from all the possible deployments for geographic SMR. Using the proposed method, any replication configuration can be evaluated without actually building it.

A. Overview

Figure 1 illustrates the overview of the proposed method and the method consists of the following steps:

1) First, a set, DC, of all possible replica deployments is created based on SC and n. Each replica deployment is expressed as a pair of locations for the leader and the other replicas[3].
2) Next, for each replica deployment $x \in DC$, its latency is estimated using the evaluation function $f(x, C)$ based on the measured RTTs. This function is further described in Section IV-B.

[1]For example, if the SMR is built on a public cloud service, each region is a site; if it is built in facilities on premises, each data center is a site.

[2]The proposed method outputs not only the best replica deployment, but also the whole ranking of all possible deployments, because the best deployment may not be acceptable for some reason other than latency.

[3]Here, we assume rotating coordinator-based SMR protocols similar to those [4], [6], [7], [18]. If the proposed method is applied to leader-less SMR protocols similar to those [2], [3], [5], then each replica deployment is simply expressed as a set of replica locations of size n.

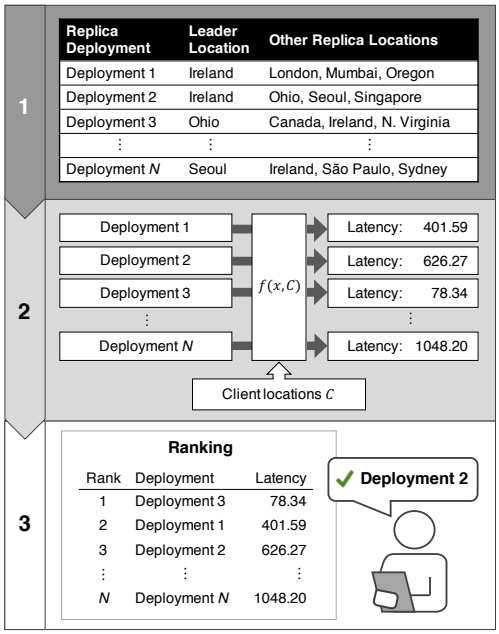


Fig. 1. Overview of the proposed method

3) The elements in DC are sorted based on their calculated latency; the sorted result is outputted as the ranking for the inputs.

Thus, the replica deployment with the shortest latency is ranked as the best replica deployment.

B. Evaluation Function $f(x, C)$

The evaluation function $f(x, C)$ outputs an estimated latency based on replica deployment, x, and the client locations, C by tracing message transmissions specific to a replication protocol being used. The function plays an important role in the proposed method.

1) Approach: If site candidates SC is large, it is impractical to actually build SMRs with all possible replica deployments to evaluate their latencies. Therefore, the evaluation function estimates them based on round-trip time (RTT) between sites, which can be measured more easily, and outputs as an latency for that deployment. In other words, before using the proposed method, a user must measure RTTs between candidate sites in advance. Here, the time required for message processing in a replica is disregarded because the communication delay between replicas is relatively large compared with the processing time in a geographic SMR.

Assuming that a latency can only be estimated from the communication time, two factors must be considered: the types of message communications (i.e., *message transmission patterns*) that constitute latency and the communication time between sites. The message transmission pattern can be found by referring to an SMR protocol used in a replication. Then, for a given set C of clients and replica locations x, the function simulate the transmission and receipt of messages based on the message transmission pattern of the replication protocol and the measured RTTs.

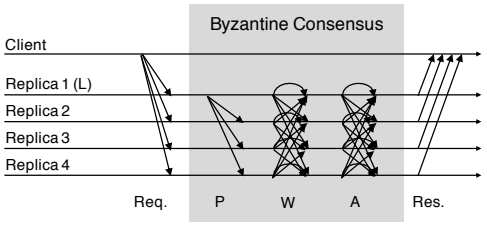


Fig. 2. The message transmission pattern for Mod-SMaRt protocol [7] in BFT-SMaRt [8]. Replica 1 is the leader replica and Req., P, W, A, and Res., indicate Request, Propose, Write, Accept, and Response messages, respectively.

Here, we model the message transmission pattern of Mod-SMaRt [7] of BFT-SMaRt [8] as an example; however, we believe the same approach can be applied to other SMR protocols. In Mod-SMaRt, a special replica (called a *leader* replica) determines the order in which requests are executed and communicates this order to the other replicas. The message transmission pattern involves five types of messages that are exchanged among the client and replicas to process the request, as shown in Fig. 2. First, the client sends a request to each replica (Request). When the leader replica receives the request, it sends Propose messages to each replica to propose a candidate value for agreement (Propose). Then, Write and Accept messages are exchanged between all replicas to confirm the validity of the candidate values to determine the final agreed value (Write and Accept). Finally, the replicas execute the ordered request and return the result to the client (Response). Hereafter, an RTT and a message transmission delay between sites a and b is denoted as $\mathrm{RTT}(a, b)$ and $\mathrm{delay}(a, b) = \mathrm{RTT}(a, b)/2$, respectively.

2) Latency Formulation: The evaluation function f estimates a latency of each client location $c \in C$, and outputs the average of these latencies as follows:

$$f(x, C) = \sum_{c \in C} f_c(x, c)/|C|, \tag{1}$$

where f_c is a evaluation function for a single client. Hereafter, we explain how $f_c(x, c)$ calculates a latency on a replica deployment. The message pattern of Mod-SMaRt comprises five parts as depicted in Fig. 2, and we denote the timings of these parts by S_{req}, S_{pro}, S_{wrt}, S_{acc}, and S_{res}, respectively. If necessary, we denote the timing for a specific replica r_i by adding a superscript such as S^i_{pro}.

First, we calculate the timing S_{req} at which the leader receives a request. In the replication protocol, a request message is sent from a client to each replica although only the leader replica processes the request in the fault-free case; thus, S_{req} can be expressed as the average of the RTTs from each client c to the leader replica l:

$$S_{req} = \sum_{c \in C} \mathrm{delay}(c, l)/|C|. \tag{2}$$

Then, the leader sends the request to each replica as Propose messages; the timing S^i_{pro} at which the replica r_i receives the

Propose message is expressed as follows:

$$S^i_{pro} = S_{req} + \mathrm{delay}(l, r_i). \quad (3)$$

When a replica receives the Propose message, it broadcasts a Write message to all replicas. Each replica accepts the Write message when it receives the same Write messages from a majority $\lceil (n+1)/2 \rceil$ of the replicas. The timing S^i_{wrt} at which replica r_i accepts the Write messages can be calculated based on the timing at which the replica r_i receives the Write message sent from replica r_j:

$$S^i_{wrt} = \mathrm{find}(T^i_{wrt}, \lceil (n+1)/2 \rceil), \quad (4)$$

where $t_{wrt}(r_i, r_j) = S^j_{pro} + \mathrm{delay}(r_j, r_i)$, $T^i_{wrt} = \{t \mid t_{wrt}(r_i, r_j), 0 \leq j < n\}$, and $\mathrm{find}(S, k)$ is a function that returns the k-th smallest element of set S.

An Accept message is sent in the same way as Write messages. Therefore, if we define $t_{acc}(r_i, r_j) = S^i_{wrt} + \mathrm{delay}(r_j, r_i)$, S^i_{acc} is

$$S^i_{acc} = \mathrm{find}(T^i_{acc}, \lceil (n+1)/2 \rceil), \quad (5)$$

where $T^i_{acc} = \{t \mid t_{acc}(r_i, r_j), 0 \leq j < n\}$.

Finally, when a replica receives a majority of Accept messages, it executes the request and sends the execution result to the client as a Response message. When a client receives the same response message from $f+1$ distinct replicas, it accepts the result. Therefore,

$$f_c(x, c) = S_{res} = \mathrm{find}(T_{res}, f+1), \quad (6)$$

where $T_{res} = \{t \mid S^i_{acc} + \mathrm{delay}(r_i, c), 0 \leq i < n\}$.

V. Evaluation

In this section, we examine the effectiveness of the proposed method described in Section IV. First, the evaluation of replica deployments in terms of the RTT is verified in Section V-A. Next, the latencies of thousands of replica deployments on a public cloud service are measured to evaluate the accuracy of the ranking generated by the proposed method in Section V-B. Finally, Section V-C characterizes the time that it takes to generate a ranking.

All experiments are conducted using Amazon Web Services EC2, a representative public cloud service. We use 15 regions[4] of Amazon EC2 as site candidates SC for replica deployments (i.e., $|SC| = 15$). Replica and client programs are executed on Ubuntu Server 16.04 (64 bit). For replicas and clients, we use t2.micro instances that have one vCPU, 1 GiB memory, EBS storage, and a network interface of "Low to Moderate" performance.

A. Validation of the Use of RTTs

The proposed method calculates lantecies based on the RTTs between sites. Here, we evaluate whether it is appropriate to use the RTTs for estimating lantecy and how long the generated ranking is valid.

[4]N. Virginia, Ohio, N. California, Oregon, Mumbai, Seoul, Singapore, Sydney, Tokyo, Canada Central, Frankfurt, Ireland, London, Paris, São Paulo

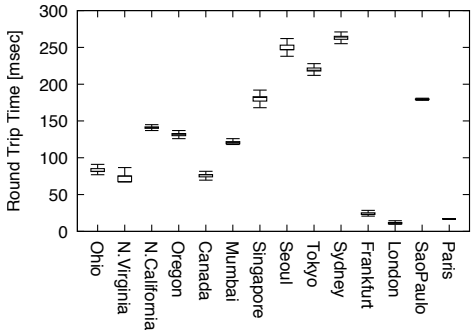


Fig. 3. Distribution of RTT from Ireland to each region during term A.

1) Method: An instance is deployed in each of the regions, and the `ping` command is executed against the instances in the other 14 regions every two seconds. RTTs were measured during the following periods (all times are displayed in UTC in 24-h notation):

- Term A: March 7, 19:27 – 22:13, 2018
- Term B: January 11, 11:14 – January 28, 3:41, 2019
- Term C: April 15, 15:48 – April 23, 11:15, 2019

2) Results and Discussion: RTTs measured during Term C are shown as a boxplot in Fig. 3 (only the results for the Ireland instance are shown due to space limitations). Although RTT varied from region to region, these variations were small. The largest variation was observed between the Ireland and Singapore regions, and its mean and standard deviation were 180.3 and 24.1 ms, respectively.

Next, we compare the RTTs from Ireland to Singapore (where the largest variations were observed) during terms A, B, and C. The average RTTs were 175.3 ms in term A, 179.8 ms in term B, and 180.3 ms in term C. Over the 13 months between term A and term C, RTT increased by 5 ms. Although this may seem like a small difference, if similar changes occurred between all regions, it is likely that the ranking generated by the proposed method would change considerably.

To investigate how these difference affects a replica deployment ranking, we generated two rankings from the RTTs measured during Terms A and C with the client location Multiple (see Section V-B1 for its definition). Figure 4 shows the correlation between these rankings. We can observe that the RTT changes affected the ranking certainly, especially for the 2000–5000 ranks. The largest difference happened on the replica deployment of Tokyo (leader), Canada, Oregon, and Singapore. The deployment was in 3523rd place in the term A ranking, while it was in 2688th place in the term C ranking.

The results indicate that the RTT variations in the public cloud are sufficiently small in the short term; thus, estimating a replica deployment in terms of based on the RTTs between sites is valid. In contrast, RTTs between regions changed over long periods (on the order of one year). Therefore, a replica deployment that is found to be optimal may no longer be optimal after a long time has passed, suggesting that replicas should be relocated periodically to maintain optimal performance.

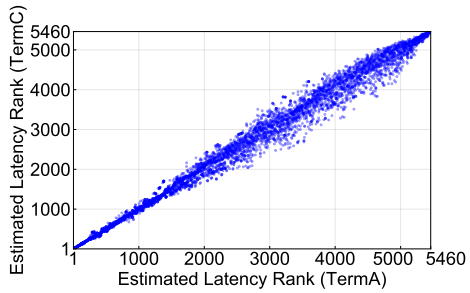


Fig. 4. Difference between the rankings of Terms A and C.

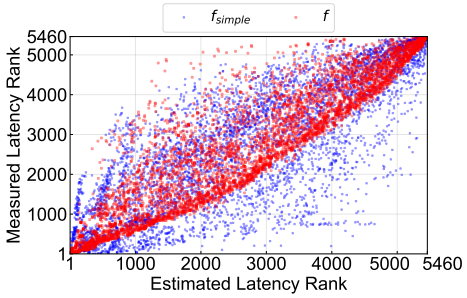


Fig. 5. Scatter plots of measured latency rank and estimated latency rank (C = Sydney, $|DC| = 5460$). Each plotted point represents the latency of a replica deployment (red for f and blue for f_{simple}). The horizontal axis represents the ranking derived from the latencies output by the proposed method with f or f_{simple}, and the vertical axis represents the ranking derived from the measured latencies.

B. Ranking Accuracy

Here, we discuss the accuracy of a ranking generated by the proposed method by comparing the rankings with those derived from the experimentally measured latencies of all possible replica deployments.

1) Method: We introduce a baseline evaluation function $f_{simple}(x, C)$ to compare the accuracy of the evaluation function of the proposed method. This function roughly estimates a latency based on a simplified message pattern for Mod-SMaRt. First, it divides the pattern into three parts: Request, Byzantine Consensus, and Response as Fig. 2 and calculates their timings S_{req}, S_{con}, and S_{res} as follows: S_{req} is the average of the half RTTs from each client to the leader replica. S_{con} is the sum of the half RTTs between all pairs of replicas. S_{res} is the average of the half RTTs from each replica to each client. Finally, this function outputs the sum of these timings as a latency.

In this experiment, all possible replica deployments are built on AWS and the latency of each one is measured. We do not assume that multiple replicas are deployed in the same region. Since $|SC| = 15$ and $n = 4$, the total number of possible replica deployments $|DC| = |SC| \times {}_{|SC|-1}C_{n-1} = 5,460$. If replicas are deployed to the same combination of regions, the location of the leader replica may differ; hence, such deployments are considered independently.

As with replicas, it is assumed that the clients are also located in the AWS regions. To evaluate the effects of the number and locations of clients, clients are placed in geographically distant regions, namely Ireland, Sydney, and N. Virginia. The case wherein multiple clients are placed in multiple regions (we call this deployment as "Multiple") is also evaluated: 10 clients are placed in Ireland, 3 clients are placed in Sydney, and 5 clients are placed in N. Virginia.

SMR is built using the open-source SMR library BFT-SMaRt [8][5]. A replication is build to withstand Byzantine failures; the tolerable number of failures is $f = 1$ and the number of replicas is $n = 4$. The defaults are used for all other BFT-SMaRt settings.

All latencies are measured using the sample programs LatencyClient and LatencyServer bundled in BFT-SMaRt. LatencyClient periodically sends requests to the service and measures the latency. LatencyServer is a dummy service that provides no functionality; it simply returns a response immediately after receiving a request from a client. The payload sizes of the requests and responses are 1,024 bytes. LatencyClient sends requests 50 times every 2 sec. The top 10% (i.e., the highest five values) and bottom 10% (i.e., the lowest five values) of the measured values are considered as outliers and disregarded; the average of the other values (40 values in total) is considered as the latency of the replica deployment. The latency is estimated with the average RTTs measured during Term C in Section V-B1.

[5]https://github.com/bft-smart/library/releases/tag/v1.1-beta

TABLE I
RMSE AND CORRELATION COEFFICIENT (CC)

Client location	RMSE		CC	
	f_{simple}	f	f_{simple}	f
Ireland	759.411	620.686	0.884	0.922
N. Virginia	722.516	548.982	0.895	0.939
Sydney	985.598	638.275	0.804	0.918
Multiple	697.473	629.228	0.902	0.920

2) Results and Discussion: Figure 5 shows the correlations between the rankings generated via the proposed method using the evaluation functions. Due to space limitations, only the results for the multiple are shown. Table I also shows the root mean square error (RMSE) calculated based on the ideal ranking (i.e., $y = x$), which perfectly matches the ranking based on the measured latencies, and the correlation coefficient (CC) for each client location. The results indicate that the RMSE was lower and the CC was higher (exceeding 0.91 in all cases) for f than for f_{simple} for all client locations. This implies that f yielded more accurate rankings by tracing the communications between the replicas in detail.

These experiments confirmed that the proposed method can generate consistent rankings in various client locations. Further, it was revealed that the rankings generated by f are more accurate than those generated by f_{simple} (particularly for the higher-ranked deployments). Hence, a higher reproducibility of the replication protocol can reproduce more accurate replica deployment ranking.

TABLE II
CALCULATION TIME BY SIZE OF SITE CANDIDATES SC ($n = 4$)

$\|SC\|$	**Time t [sec]**	$\|DC\|$	$t/\|DC\|$ **[msec]**
15	12.9	5,460	2.36
20	39.5	19,380	2.04
25	110.5	50,600	2.18
30	227.3	109,620	2.07

TABLE III
CALCULATION TIME BY THE NUMBER OF REPLICAS n ($|SC| = 15$)

n	**Time t [sec]**	$\|DC\|$	$t/\|DC\|$ **[msec]**
4	12.9	5,460	2.36
7	429.1	45,045	9.53
10	792.1	30,030	26.38
13	77.7	1,365	56.90

C. Calculation Time to Generate a Ranking

Finally, we evaluate the calculation time required to generate a ranking with the proposed method[6]. The ranking calculation times of f_{simple} and f were 1.88s and 10.88s, respectively for $n = 4$, that is, f_{simple} is about five times faster than f. This finding indicates that more time is required to calculate the estimated latency using the evaluation function and to improve the reproducibility of the communication.

Next, we investigate the influence of the size of SC on the calculation time with f. Table II shows the resulting calculation times for different $|SC|$ values and the corresponding calculation times per replica deployment $t/|DC|$. The result shows that as the size of SC increased, the calculation time required to generate the rankings considerably increased because the total number of replica deployments $|DC|$ increases exponentially with $|SC|$.

Furthermore, the influence of the number of replicas n on the calculation time with f. Table III shows the calculation time t for different $|DC|$ values and the corresponding calculation times per replica deployment, $t/|DC|$, as n is varied. The result shows that t and $|DC|$ were maximized at different values of n (10 and 7, respectively) because the calculation time of a replica deployment $t/|DC|$ increases as n increases.

These measurement results reveal that the rankings for replica deployments can be calculated in several hundred seconds when the replica number and site number are relatively small. This is considered a reasonable calculation time since a deployed SMR is typically operated for more than one year. In contrast, if large numbers of replicas and SC are used, the calculation time becomes very high. In such a case, some changes need to be made so that the solution is still practical, e.g., calculations latencies in parallel, discarding replica deployments that seems to be slow, and so on.

VI. CONCLUSION

In this paper, we addressed on the difficulty of determining the optimal replica deployment for geographic state machine replication by proposing a novel method to generate a ranking of all possible replica deployments. We introduced an evaluation function that estimates a latency of each replica deployment based on the RTTs between sites, which are easy to measure without actually building the deployments. Hence, all possible replica deployments can be evaluated and ranked accordingly to determine the optimal replica deployment for geographic SMR. We confirmed the validity of evaluating replica deployments in terms of their RTTs. After that, we measured the latencies of thousands of replica deployments built on Amazon Web Services, and ranked the deployments accordingly. Then, we compared this experimentally derived ranking with those rankings generated using the proposed method. The results exhibited that the proposed method can create a ranking with sufficient accuracy in a reasonable time.

[6]All the rankings were calculated by the program implemented with Python 3.6 on the following PC: Intel Core i5 7400, Windows 10 Home 64-bit.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Numbers JP16K16035 and JP18K18029.

REFERENCES

[1] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, 1990.

[2] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo, "RITAS: Services for Randomized Intrusion Tolerance," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 1, pp. 122–136, 2011.

[3] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *CRYPTO*, 2001.

[4] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, 2002.

[5] J. Nakamura, T. Araragi, S. Masuyama, and T. Masuzawa, "Efficient Randomized Byzantine Fault-Tolerant Replication based on Special Valued Coin Tossing," *IEICE Transactions on Information and Systems*, vol. E97-D, no. 2, pp. 231–244, 2014.

[6] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," in *SOSP*, 2007.

[7] J. Sousa and A. Bessani, "From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation," in *EDCC*, 2012.

[8] A. Bessani, J. a. Sousa, and E. E. P. Alchieri, "State Machine Replication for the Masses with BFT-SMaRt," in *DSN*, 2014.

[9] J. Sousa and A. Bessani, "Separating the WHEAT from the Chaff: An Empirical Design for Geo-Replicated State Machines," in *SRDS*, 2015.

[10] S. Liu and M. Vukolic, "Leader set selection for low-latency geo-replicated state machine," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1933–1946, 2017.

[11] M. Eischer and T. Distler, "Latency-Aware Leader Selection for Geo-Replicated Byzantine Fault-Tolerant Systems," in *DSN-W*, 2018.

[12] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building efficient replicated state machines for wans," in *OSDI*, 2008.

[13] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "EBAWA: Efficient byzantine agreement for wide-area networks," in *HASE*, 2010.

[14] P. Coelho and F. Pedone, "Geographic state machine replication," in *SRDS*, 2018.

[15] L. Lamport, "Lower bounds for asynchronous consensus," https://lamport.azurewebsites.net/pubs/bertinoro.pdf, 2002, [Online; accessed July 31, 2019].

[16] S. A. Cook, J. Pachl, and I. S. Pressman, "The optimal location of replicas in a network using a READ-ONE-WRITE-ALL policy," *Distributed Computing*, vol. 15, no. 1, pp. 57–66, 2002.

[17] G. Sen, M. Krishnamoorthy, N. Rangaraj, and V. Narayanan, "Facility location models to locate data in information networks: a literature review," *Annals of Operations Research*, vol. 246, no. 1-2, pp. 313–348, 2015.

[18] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.

個体群プロトコルモデルにおけるリーダ選挙

首藤裕一

大阪大学 大学院情報科学研究科

概 要

本稿では，個体群プロトコルモデルにおけるリーダ選挙について，最新の研究動向を紹介する．

1 個体群プロトコルモデル

個体群プロトコルモデル（英：Population protocols）とは，計算資源が極めて制約されたセンサノードによって構成されるモバイルセンサネットワークを表現する計算モデルとして Angluin ら [AAD$^+$06] によって 2004 年に提案された．本モデルではネットワークを構成するセンサノードを**個体**と呼び，ネットワーク全体を**個体群**と呼ぶ．各個体は有限状態機械であり，常にひとつの状態を持つ．2 つの個体間で互いに自身の状態を伝え合う交流と呼ばれるイベントがときおり発生し，その際，両個体の現在の状態と，プロトコルが定める状態遷移規則によって，両個体は新たな状態に遷移する．プロトコルの実行はこの交流によって進行する．後述するように，個体群中の任意の個体ペアは幾度となく交流を繰り返すことを仮定する．また，個体群を構成する個体の数（個体数）を n で表す．ここで，個体群プロトコルモデルを直感的に理解するために，簡単な応用例として，千羽の鳥の群れの一羽一羽に小型のセンサノードすなわち個体を付けた個体群（個体数 $n = 1000$）を考える．各鳥が好き勝手に移動することで，頻繁に，2 羽の鳥が極めて近い距離に接近する事象が発生する．この事象が交流に相当する．交流の発生時，無線通信によって個体は互いの状態を教え合い，自身の状態を更新する．このような例において，任意の個体ペアが幾度となく交流を繰り返すという前述の仮定は自然である．この個体群で簡単な計算を行うことを考える．具体的には，1000 羽の鳥のうち，発熱している（体温が閾値を超えている）鳥が 10 羽以上いるかどうかを全個体に把握させるという問題を解くことを考える．実行開始時，観測対象の鳥が発熱している個体は初期状態を 1 に, そうでない個体は初期状態を 0 に設定する．交流発生時の状態遷移規則は以下のように定める．

$$(x, y) \to (x + y, 0) \text{ if } x + y < 10,$$
$$(x, y) \to (10, 10) \text{ otherwise.}$$

上記の規則は，状態 x の個体と状態 y の個体が交流したときに，前者と後者の次状態を指定する規則である．第一の規則は，x と y の和が 10 に満たないときに，2 個体の状態を加算して一方に記憶させ，他方の状態を 0 にする．第二の規則は，2 個体の状態の和が

10 以上であるときに，両個体の状態をともに 10 にする．いま，このプロトコルの実行開始時において発熱している個体の総数を f とする．第二の規則が適用されるまでは，全個体の状態の総和と f は常に一致する．したがって，$f < 10$ のとき第二規則は決して適用されることがなく，すべての個体の状態は 10 未満でありつづける．一方で，任意の個体ペア間で交流が繰り返し発生するため，発熱している個体の総数が第一規則によって少数の個体に集約されていく．したがって，$f \geq 10$ のときは，必ず第二規則がいつか適用され状態 10 の個体が少なくともひとつ発生する．状態 10 の個体がひとつでも生じると，第二規則により状態 10 の個体は増殖し，やがて全個体が状態 10 となることが保証できる．ゆえに,「1000 羽の鳥のうち，発熱している鳥が 10 羽以上いるか」という問いに対して，状態 10 の個体は ”Yes ”と出力し，それ以外の状態の個体は ”No ”と出力すれば，全個体がやがて正しい答えを出力することを保証できる．このようなプロトコルが個体群プロトコルである．

上述のプロトコルの第一規則は 2 個体の対称性を破壊していることに注意されたい．すなわち，同一の状態を持つ 2 個体が交流した場合においても 2 個体が異なる状態に遷移する．これは，交流に相当する無線通信において，交流を呼びかけた個体とその呼びかけに応答した個体が存在するはずだという仮定に基づくものであり，両個体の状態が同一であっても，**呼びかけ側**の個体（initiator）と**応答側**の個体（responder）の次状態は個別に決定して良いという意味でプロトコルの設計者に強い自由度を与える．したがって，他の一部の（分散システムの）計算モデルと異なり，システム全体の対称性は個体群プロトコルモデルでは問題にならないことが多い．交流の発生パターンについてはさまざまなモデルがあるが，個体群プロトコルを扱う多くの研究では，一様ランダムな乱択スケジューラを仮定する．これは，各時刻（=ステップ）において，個体群中から一様ランダムに選択された 2 つの個体間で交流が発生するという仮定である．すなわち，各時刻において，任意の個体ペアは確率 $1/{}_nC_2$ で選択され交流を行う．ここでいう時刻とは抽象的なものであり，実際には，交流は個体群中で同時多発的に発生するため，後に説明する期待収束時間などのプロトコルの時間計算量は，計算の完了に要したステップ数を個体数 n で割った並列時間で評価することが主流である．本稿においても，この乱択スケジューラを仮定し，各種時間計算量を並列時間で評価する．

2 リーダ選挙

入力とは独立に定義されるタスク（問題）としてもっとも重要なもののひとつにリーダ選挙問題がある．これは，個体群中のただひとつの個体をリーダとして指名する問題である．リーダ選挙問題は分散システム一般にとって重要かつ基本的な問題であるが，個体群プロトコルモデルでは特に重要な意義を持つ．というのは，個体群中にただひとつのリーダが存在することを仮定できれば，Presburger 算術で定義可能な任意の述語が極めて高速に，具体的には個体数 n に関する対数多項式並列時間で解けることが 2006 年に証明されているからである [AAE08].

リーダ選挙問題は，厳密には，一様ランダムなスケジューラによって交流が次々と発生していく結果として，やがてただひとつの個体が記号 L を出力し，それ以外のすべての個体が記号 F を出力するような状況に個体群を収束させることを目的とする問題である．リーダ選挙問題は次に示す極めて単純なプロトコル [AAD$^+$06] によって解くことができる．

表 1: 個体群モデルにおける主要なリーダ選挙プロトコル.（安定時間は期待値）

	状態数	安定時間
[AAD+06]	$O(1)$	$O(n)$
[AG15]	$O(\log^3 n)$	$O(\log^3 n)$
[AAE+17]	$O(\log^2 n)$	$O(\log^{5.3} n \cdot \log\log n)$
[AAG18]	$O(\log n)$	$O(\log^2 n)$
[GS18]	$O(\log\log n)$	$O(\log^2 n)$
[GSU18]	$O(\log\log n)$	$O(\log n \cdot \log\log n)$
[MST18]	$O(n)$	$O(\log n)$
[SOI+19]	$O(\log n)$	$O(\log n)$

- 状態空間：$\{l, f\}$
- 初期状態：l
- 出力：状態が l のとき L，状態が f のとき F
- 状態遷移規則：$(l, l) \to (l, f)$

このプロトコルにおいて各個体は l,f のいずれかの状態をとる．実行開始時の初期状態は入力にかかわらず l である．各個体は，自身の状態が l のとき，かつ，そのときのみ記号 L を出力し，そうでないとき記号 F を出力する．すなわち，状態 l の個体がリーダであり，状態 f の個体が非リーダである．交流による状態遷移は両個体がリーダのときのみ発生する．リーダどうしが交流したとき，一方（呼びかけ側）がリーダであり続け，他方（応答側）は非リーダになる．本プロトコルの実行において，実行開始時に n 個のリーダが存在し，かつ，リーダの数は 2 個以上のリーダが交流して一方が非リーダに変化することによってのみ減少するので，リーダの数が 0 になることは決してない．一方で，2 個以上のリーダが存在する場合，やがてそれらのリーダ間で交流が発生してリーダの数が 1 減る．したがって，個体群中のリーダの数はやがて 1 になる．以後，ただひとつのリーダはリーダであり続け，他の個体は常に非リーダであり続ける．よって，本プロトコルは個体群中からただひとつのリーダを選出する，すなわち，正しくリーダ選挙問題を解く．

一般に，一度その状況に到達すると，以後，問題の定める所望の性質（たとえばリーダ選挙問題の場合，ただひとつのリーダが変わらず存在し続けること）を満たし続けるような状況を安定状況と呼ぶ．そして，プロトコルが安定状況の到達するまでの時間を安定時間と呼ぶ．簡単な解析により，上記の単純なリーダ選挙プロトコルの安定時間の期待値は並列時間で $O(n)$ であることが分かる．一方で，このプロトコルは，個体の状態数を定数に制限する場合には最適であることが示されている．具体的には，Doty と Soloveichik[DS18] によって任意の定数状態プロトコルはリーダ選挙を解くのに線形時間を要することが証明されている．

この線形時間の壁は 2015 年に Alistarh と Gelashvili [AG15] によって崩された．彼らは，$O(\log^3 n)$ 状態を用いることで $O(\log^3 n)$ 時間で安定するリーダ選挙プロトコルを与えた．この後，数多くの研究が複数の研究グループによって行われ，この計算量が次々と改善されていく．詳細は表 1（主要なアルゴリズム）と 2（安定時間の下界）を参照されたい．現状，

表 2: 安定時間（期待値）の下界

	状態数	安定時間
[DS18]	$O(1)$	$\Omega(n)$
[AAE$^+$17]	$< 1/2 \log\log n$	$\Omega(n/(\mathrm{polylog}\, n))$
[SM19]	any large	$\Omega(\log n)$

計算量の最も優れたプロトコルは 2 つあり，Gąsieniec ら [GSU18] のプロトコルと著者ら [SOI$^+$19] のプロトコルである．Gąsieniec ら [GSU18] のプロトコルはわずか $O(\log\log n)$ 状態で $O(\log n \cdot \log\log n)$ 期待時間でリーダ選挙を解く．Alistarh ら [AAE$^+$17] の下界によると，対数多項式の期待時間でリーダ選挙を解くには $\Omega(\log\log n)$ 状態が必要なので，そのようなリーダ選挙プロトコルのなかでこのプロトコルの状態数は最適である．一方で，[SOI$^+$19] のプロトコルは $O(\log n)$ 状態を用いて $O(\log n)$ 期待時間でリーダ選挙を解く．Gąsieniec らのプロトコルに比べ状態数が大幅に増加しているが，このプロトコルは時間最適である．具体的には，状態数に依らずリーダ選挙を $o(\log n)$ 期待時間では解けないことが証明されている [SM19]．ワークショップ当日はこのプロトコル [SOI$^+$19] を主に紹介する予定である．

参考文献

[AAD$^+$06] D. Angluin, J Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[AAE08] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, 2008.

[AAE$^+$17] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in population protocols. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2560–2579. SIAM, 2017.

[AAG18] Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2221–2239. SIAM, 2018.

[AG15] Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*, pages 479–491. Springer, 2015.

[DS18] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018.

[GS18] Leszek Gąsieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2653–2667. SIAM, 2018.

[GSU18] Leszek Gąsieniec, Grzegorz Stachowiak, and Przemysław Uznański. Almost logarithmic-time space optimal leader election in population protocols. *arXiv preprint arXiv: 1802.06867*, 2018.

[MST18] Othon Michail, Paul G Spirakis, and Michail Theofilatos. Simple and fast approximate counting and leader election in populations. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 154–169. Springer, 2018.

[SM19] Yuichi Sudo and Toshimitsu Masuzawa. Leader election requires logarithmic time in population protocols, 2019.

[SOI+19] Yuichi Sudo, Fukuhito Ooshita, Taisuke Izumi, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Brief anouncement: Logarithmic expected-time leader election in population protocol model. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing*, page (to appear), 2019.

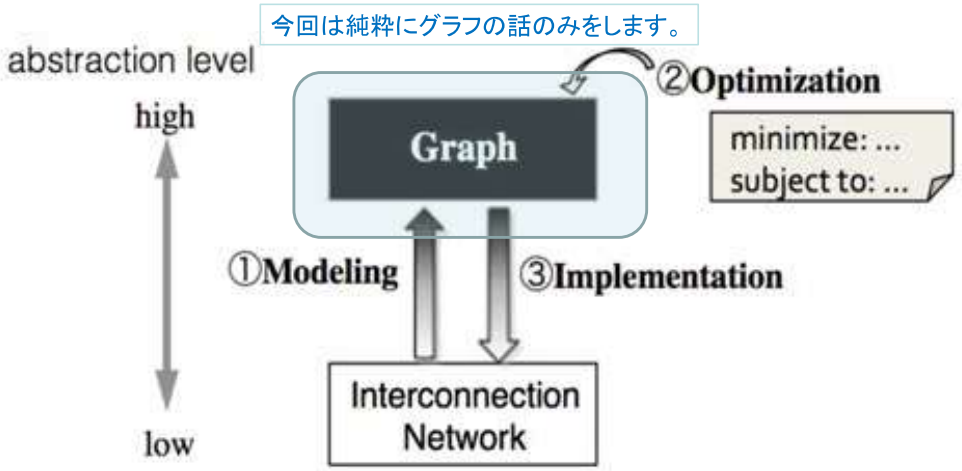
Degree/Diameter Problem for Host-Switch Graphs

広島大学　安戸僚汰

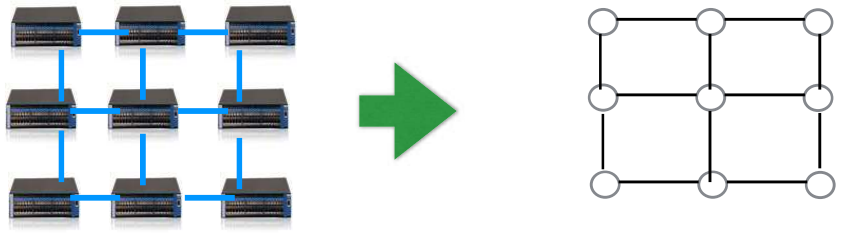
Objective

- Establish design methodology for high-performance interconnection networks of supercomputers (~100,000 nodes)
  - with the focus on modern applications with little locality

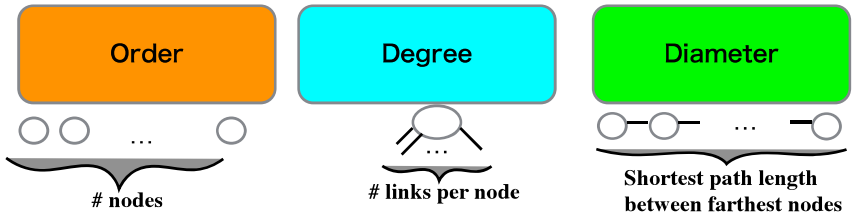
Graph-theoretical approach



The simplest graph-theoretic perspective: an interconnection network is an undirected graph



Important topological properties



Classical problem: The Degree/Diameter Problem

HIROSHIMA UNIVERSITY

Optimize (maximize):

Order — # nodes

Subject to:

Degree — # links per node?

Diameter — Shortest path length between farthest nodes

COMBINATORICS WIKI

Summarised in Combinatorics Wiki

http://combinatoricswiki.org/wiki/The_Degree/Diameter_Problem

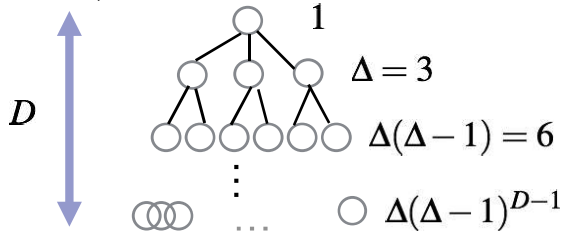
Moore graphs (optimal graphs)

HIROSHIMA UNIVERSITY

Edward F. Moore (1925-2003)

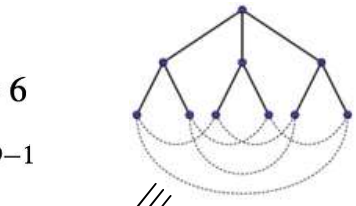
Δ : Degree, D : Diameter

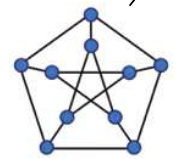
ex) $\Delta = 3$



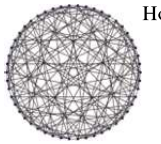
Upper bound on the order (*Moore bound*):

$$1 + \Delta \sum_{i=0}^{D-1} (\Delta - 1)^i$$





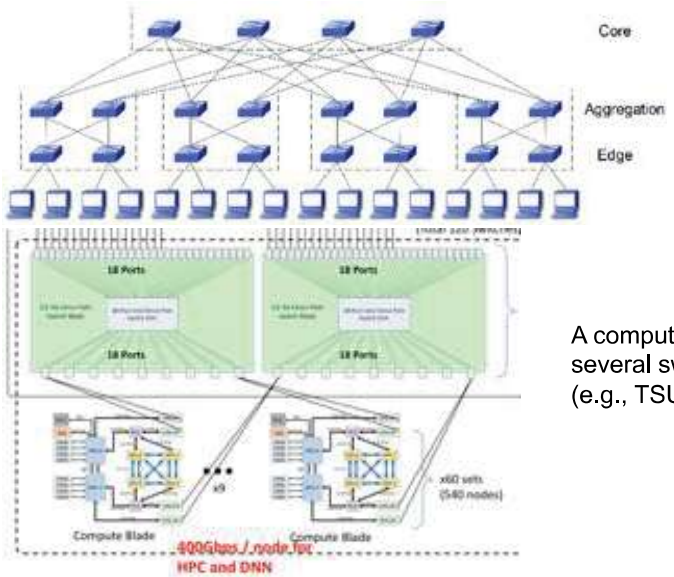
Petersen graph
$\Delta = 3, D = 2$
10 nodes



Hoffman-Singleton graph
$\Delta = 7, D = 2$
50 nodes

Some systems cannot be represented by an undirected graph…

HIROSHIMA UNIVERSITY



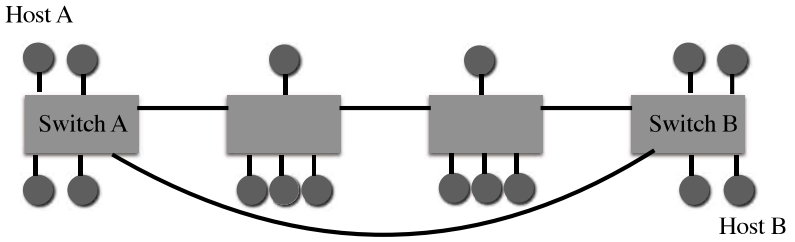
Fat-tree topology
(e.g., Tianhe-2)

A compute node is connected to several switches
(e.g., TSUBAME3.0)

http://on-demand.gputechconf.com/gtc/2017/presentation/S7813-Matsuoka-scalable.pdf.pdf

A *host-switch graph*

HIROSHIMA UNIVERSITY

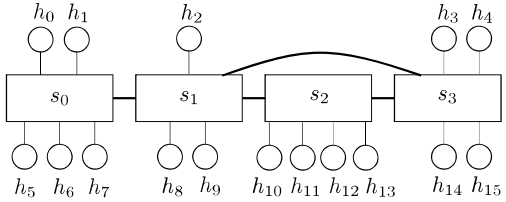


[R. Yasudo, M. Koibuchi, K. Nakano, H. Matsutani, H. Amano,
"**Order/Radix Problem: Towards Low End-to-End Latency Interconnection Networks**",
Proc. of International Conference on Parallel Processing (ICPP), Aug. 2017.]

定義. A host-switch graph

HIROSHIMA UNIVERSITY

- A host-switch graph is a 3-tuple $G = (H, S, E)$ where
  - $H = \{h_0, h_1, \ldots, h_{n-1}\}$ *is a set of n elements called* hosts nodes *(or simply hosts),*
  - $S = \{s_0, s_1, \ldots, s_{m-1}\}$ *is a set of m elements called* switches nodes *(or simply switches), and*
  - $E \subseteq \{\{s_i, s_j\} \mid s_i, s_j \in S\} \cup \{\{h_i, s_j\} \mid (h_i \in H) \wedge (s_j \in S)\}$ *is a set of unordered pairs of connected nodes called* edges.



Order n: ホスト数
Degree Δ: スイッチの最大次数
Diameter D: ホスト間最短距離の最大値

左の例では，
$n = 16, \Delta = 6, D = 4$

The Degree/Diameter Problem for host-switch graphs

HIROSHIMA UNIVERSITY

Optimize (maximize):

Order

How many **hosts**?

Subject to:

Degree

How many links **per switch**?

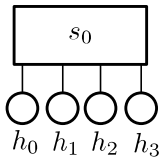
Diameter

Shortest path length between farthest hosts

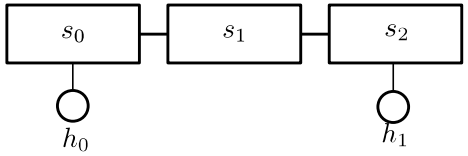
Order nの上界（自明な上界）

HIROSHIMA UNIVERSITY

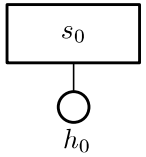
$D = 2, \Delta = 4$



$D = 4, \Delta = 2$



$\Delta = 1$



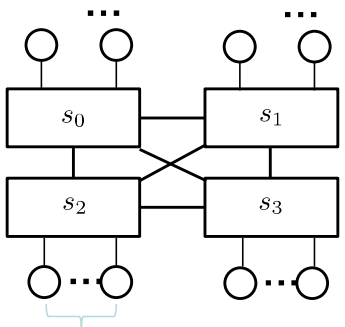
$D = 2$ または $\Delta \leq 2$ のとき，上界は Δ

Order nの上界（D = 3）

HIROSHIMA UNIVERSITY

スイッチはm-クリークを構成する。各スイッチは$m-1$個のスイッチに接合されるので，上界は

$$m(\Delta - m + 1)$$



$$\frac{\partial m(\Delta - m + 1)}{\partial m} = -2m + \Delta + 1$$

より，上界をΔで表すと

$$\frac{(\Delta+1)^2}{4} \text{ if } \Delta \text{ is odd and } \frac{\Delta(\Delta+2)}{4} \text{ if } \Delta \text{ is even.}$$

$m = 4$

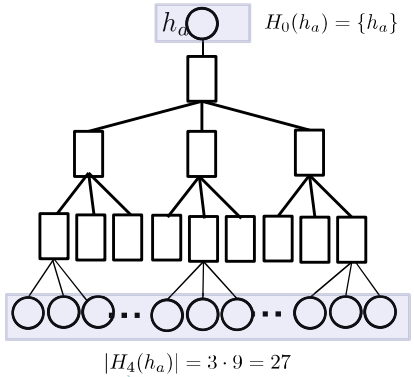
$D < 4$のときの最適解

$\Delta \backslash D$	2	3
2	host-switch graph with $n = \Delta$*	
3		2-switch clique* ($n = 4$)
4		2-switch / 3-switch clique* ($n = 6$)
5		3-switch clique* ($n = 9$)
6		3-switch / 4-switch clique* ($n = 12$)
7		4-switch clique* ($n = 16$)
8		4-switch / 5-switch clique* ($n = 20$)
9		5-switch clique* ($n = 25$)
10		5-switch / 6-switch clique* ($n = 30$)
11		6-switch clique* ($n = 36$)
12		6-switch / 7-switch clique* ($n = 42$)
13		7-switch clique*

*optimal

次に，D = 4の解を考えていく

Order nの上界（$D > 3$）



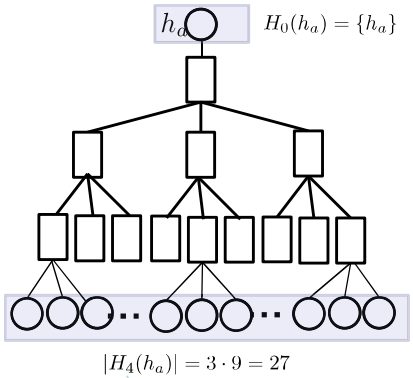
あるホストh_aについて考える。

距離1のノードは最大1，
距離2のノードは最大Δ-1，
距離3のノードは最大$(\Delta\text{-}1)^2$，
…
距離Dのノードは最大$(\Delta\text{-}1)^{D-1}$

距離D以下にホストがくると，全ホスト数は減る

$$n \leq 1 + (\Delta - 1)^{D-1}$$

Order nの上界（$D > 3$）



あるホストh_aについて考える。

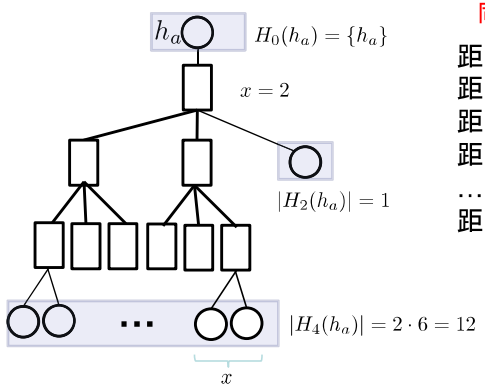
距離1のノードは最大1，
距離2のノードは最大Δ-1，
距離3のノードは最大$(\Delta\text{-}1)^2$，
…
距離Dのノードは最大$(\Delta\text{-}1)^{D-1}$

距離D以下にホストがくると，全ホスト数は減る

$$n \leq 1 + (\Delta - 1)^{D-1}$$

別のホストについて考えると，距離2に必ずホストがあり，
上界に達しない->さらに上界を改善可能

Order nの上界（$D > 3$）



ルートに近いスイッチとリーフに近いスイッチは
同数（x)のホストと接合する

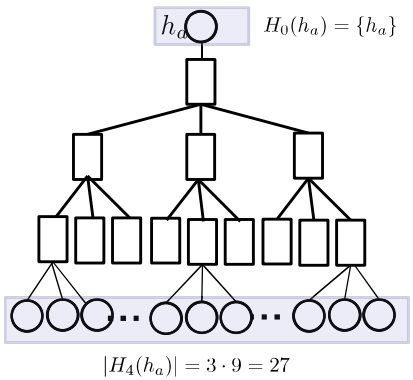
距離1のスイッチは最大1，
距離2のスイッチは最大Δ-x，
距離3のノードは最大(Δ-x)(Δ-1)，
距離4のノードは最大(Δ-x)$(\Delta\text{-}1)^2$，
…
距離Lのノードは最大(Δ-x)$(\Delta\text{-}1)^{L-2}$

最適な状態で
自分自身と距離2のホストがx個，
距離Dのホストがx (Δ-x)$(\Delta\text{-}1)^{D-3}$

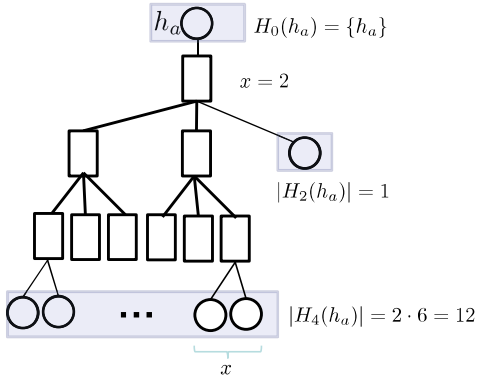
$$n \leq x + x(\Delta - x)(\Delta - 1)^{D-3}$$

最適なxを求めると…　$\left\lceil \frac{\Delta}{2} \right\rceil + \left\lceil \frac{\Delta}{2} \right\rceil \cdot \left\lfloor \frac{\Delta}{2} \right\rfloor \cdot (\Delta - 1)^{D-3}$

Order nの上界（$D > 3$）

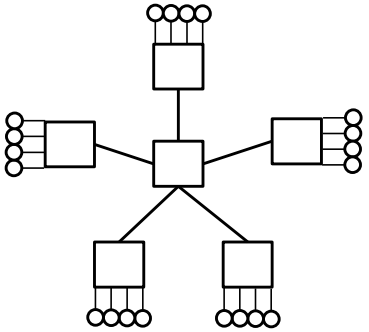


28



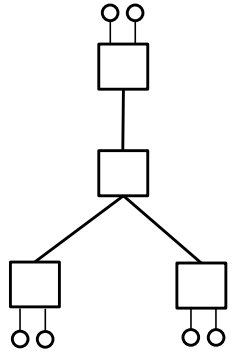
14

Star host-switch graph



$$D = 4$$

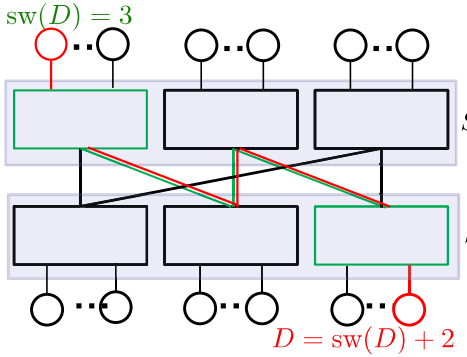
$$n = \Delta(\Delta - 1)$$

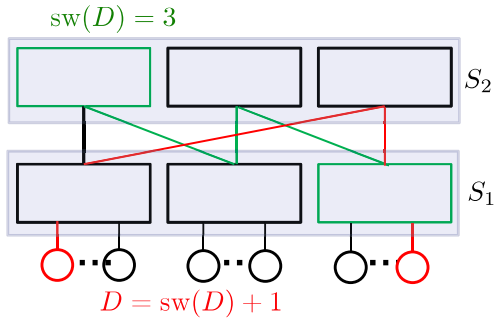


Δ=3のとき，上界に一致（最適）

Bipartite host-switch graph

スイッチが二部グラフ（bipartite graph）を構成する。二つの構成が考えられる。





D = 4のとき，完全二部グラフでないといけない
->Star host-switch graphが最適

[R. Yasudo *et al.*, IEEE Transactions on Parallel and Distributed Systems 2019]

d	Graph	N
3	Bipartite Moore graph (a projective plane)	14
4	Bipartite Moore graph (a projective plane)	26
5	Bipartite Moore graph (a projective plane)	42
6	Bipartite Moore graph (a projective plane)	62

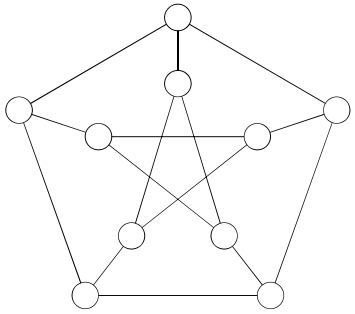
既存の解がそのまま使える

$$n = \frac{N}{2}(\Delta - d)$$

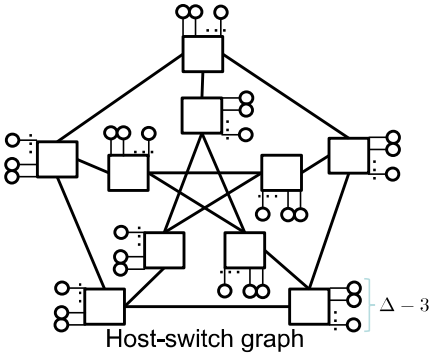
Regular host-switch graph

スイッチが無向グラフのDDP解を構成。

d	Graph	N
2	Cycle	5
3	Petersen graph [1]	10
6	Graph found by Molotsov [10]	32
7	Hoffman-Singleton graph [11]	50
17	Brown's construction [12]	274



Undirected graph (Petersen graph)



Host-switch graph

$$n = N(\Delta - d)$$

Solutions

$\Delta\backslash D$	2	3	4
2	host-switch graph with $n = \Delta$*		
3		2-switch clique* ($n = 4$)	Star* ($n = 6$)
4		2-switch / 3-switch clique* ($n = 6$)	Star ($n = 12$)
5		3-switch clique* ($n = 9$)	Star / $R_{3,2}$ ($n = 20$)
6		3-switch / 4-switch clique* ($n = 12$)	Star / $R_{3,2}$ / $R_{4,2}$ ($n = 30$)
7		4-switch clique* ($n = 16$)	$R_{5,2}$ ($n = 48$)
8		4-switch / 5-switch clique* ($n = 20$)	$R_{5,2}$ ($n = 56$)
9		5-switch clique* ($n = 25$)	$R_{7,2}$ ($n = 100$)
10		5-switch / 6-switch clique* ($n = 30$)	$R_{7,2}$ ($n = 150$)
11		6-switch clique* ($n = 36$)	$R_{7,2}$ ($n = 200$)
12		6-switch / 7-switch clique* ($n = 42$)	$R_{7,2}$ ($n = 250$)
13		7-switch clique* ($n = 49$)	$R_{7,2}$ ($n = 300$)
14		7-switch / 8-switch clique* ($n = 56$)	$R_{9,2}$ ($n = 370$)
15		8-switch clique* ($n = 64$)	$R_{10,2}$ / $B_{10,3}$ ($n = 455$)
16		8-switch / 9-switch clique* ($n = 72$)	$R_{10,2}$ / $B_{10,3}$ ($n = 546$)
17		9-switch clique* ($n = 81$)	$R_{12,2}$ / $B_{12,3}$ ($n = 665$)
18		9-switch / 10-switch clique* ($n = 90$)	$R_{13,2}$ ($n = 810$)
19		10-switch clique* ($n = 100$)	$R_{13,2}$ ($n = 972$)
20		10-switch / 11-switch clique* ($n = 110$)	$R_{13,2}$ ($n = 1134$)

Conclusions

- ホストスイッチグラフのdegree/diameter problemを紹介
- Order（ホスト数）の上界を示した
- 直径D = 4のときについて，star host-switch graph, bipartite host-switch graph, regular host-switch graphを考えた
- 大いに理論研究の余地があると考えられる