

# 卒業論文

## 日程計画における作業履歴を活用した ファジィ・ランダム多目的最適化の 並列分散解法

Parallel Distributed Solution  
of Fuzzy Random Multiobjective Optimization  
Utilizing Work History in Schedule Planning

富山県立大学 電子・情報工学科

1515028 杉山 桃香

指導教員 奥原 浩之 教授

平成31年2月19日



# 目次

図一覧	iii
表一覧	iv
記号一覧	vi
第1章 序論	1
§ 1.1 本研究の背景	1
§ 1.2 本研究の目的	1
§ 1.3 本論文の概要	2
第2章 日程計画と建築業界	4
§ 2.1 日程計画における課題	4
§ 2.2 作業履歴の活用	7
§ 2.3 建築現場における課題	8
第3章 実問題への対応と並列分散解法	11
§ 3.1 ファジィ・ランダム変数	11
§ 3.2 日程計画における遺伝的アルゴリズム解法	13
§ 3.3 並列分散処理による高速化	14
第4章 提案手法	19
§ 4.1 ファジィランダム変数を導入した多目的日程計画問題	19
§ 4.2 等価確定問題への変換	21
§ 4.3 提案手法のアルゴリズム	24
第5章 結論ならび今後の課題	28
謝辞	30
参考文献	31
付録	34
A. 1 HellowWorld を並列実行するソースコード	34
A. 2 円周率計算の並列分散処理ソースコード	34

A. 3 巡回セールスマン問題を島 GA で並列分散処理するソースコード . . . . .	35
--	----

# 図一覧

2.1	アロー・ダイヤグラム	5
2.2	クリティカルパスの例	5
2.3	時間費用関数	6
2.4	現場コミュニケーションアプリ kizuku	6
2.5	データの活用例	8
3.1	ファジィ・ランダムイメージ	11
3.2	ファジィランダム変数のメンバシップ関数 $\mu_{\tilde{C}_{ij}}(\tau)$ の例	12
3.3	目的関数のメンバシップ関数 $\mu_{\tilde{C}_{ij}x_{ij}}(v)$ の例	12
3.4	GA のフローチャート	14
3.5	パレート最適解の例	14
3.6	並列分散 GA のイメージ	15
3.7	実験に用いるラズベリーパイ 8 台	15
3.8	mpich とラズベリーパイ 8 台を用いた Hellow World の並列実行結果	16
3.9	円周率計算の台数ごとの処理時間グラフ	17
3.10	MPI による並列分散 GA のフロー	18
3.11	並列分散 GA を用いた TPU 問題台数ごとの処理時間のグラフ	18
4.1	ファジィ・ランダム多目的日程計画問題	19
4.2	クリティカルパスの所要時間を最小化	20
4.3	多目的最適化問題	20
4.4	不確定な時間費用関数	21
4.5	ファジィ目標 $\tilde{G}$ のメンバシップ関数 $\mu_{\tilde{G}}$	21
4.6	満足基準値を用いた確率最大化モデル	23
4.7	データ収集から式変換までのフロー	26
4.8	提案手法の全体的なフロー	27

## 表一覽

3.1	mpich とラズベリーパイ 8 台を用いた円周率計算の処理時間 . . . . .	17
3.2	並列分散 GA を用いた TPU 問題台数ごとの処理時間 . . . . .	18



# 記号一覧

以下に本論文において用いられる用語と記号の対応表を示す.

用語	記号
先行作業	$i$
後続作業	$j$
プロジェクトの総作業数	$n$
従事者グループ	$k$
依頼候補の従事者グループ数	$w$
作業の所要時間	$t_{ij}$
作業を従事者グループに依頼したときの費用 (ファジィ・ランダム変数)	$\tilde{c}_{ik}$
クリティカルパスを選択する 0-1 変数	$x_{ij}$
依頼する従事者グループを選択する 0-1 変数	$y_{ik}$
ファジィ・ランダム変数のメンバシップ関数	$\mu_{\tilde{c}_{ik}}$
中心値 (平均値)	$\bar{d}_{ik}$
左右の広がりパラメータ	$\beta_{ik}\gamma_{ik}$
ファイジィ目標	$\tilde{G}$
ファジィ目標の最良値	$g^0$
ファジィ目標の最悪値	$g^1$
ファイジィ目標のメンバシップ関数	$\mu_{\tilde{G}}$
ファジィ目標は満たされる可能性の度合い (可能性測度)	$\Pi_{\tilde{c}_{ik}}$
満足基準値	$h$
擬逆関数	$L^*(h)\mu_{\tilde{G}}^*(h)$
確率変数の分布関数	$F$





## 序論

### § 1.1 本研究の背景

現在、少子高齢化による労働者人口の減少は1つの社会的課題となっている。国立社会保障・人口問題研究所の調査で、2030年には、人口の1/3近くが65歳以上の高齢者になると推計されている。そして、このまま対策がないと、人手不足によるGDPの減少などの問題が考えられる[1]。この問題の対策として、AIの導入や出生率の増加政策などが挙げられるが、今回はその対策の一つである「最適な人員・費用追加による生産性の向上」に着目した。

労働人口の減少で生産性の向上や作業効率の向上が注目されるなか、今特に、人手不足が深刻である建築・土木工事の現場においても、作業効率アップによる生産性の向上を目指した政策が考えられている。住宅建築は長い期間をかけて行うため、工程が進むほど後戻りが難しくなる。よって、最初にプロジェクト全体の大まかな流れを掴んでおくことが重要である。しかし、現場におけるスケジューリング管理は非常に難しく、天災や事故などが原因で工程の変更や作業の中止が起こることがある。

このように、最適な人員・費用の追加を考える日程計画では、各作業における作業員ごとの作業時間や費用などの情報が必要になる。しかし、それらの要素は前述したような不確実で不確定な要素である。つまり、実問題を考える際には、不確実性と不確定性の両方の性質を含む問題としてみる必要がある。

従来の日程計画問題の多くは、この両方の性質を同時に考慮できていない。したがって、不確実性と不確定性の両方の性質を考慮することで、より実問題に対応した日程計画問題を考えることが出来ると考える。

### § 1.2 本研究の目的

本研究では、「人手不足」による建築現場での作業効率の低下と職人さんの負担を軽減することを大きな目的とした日程計画を考える。

その目的を満たすために必要となることをまとめる。最初に、今の建築現場における日程計画について理解し、現状の課題と、その課題を解消して作業効率を上げるために必要な仕組みを考える。主な建築現場では、工程会議で事前にプロジェクトに必要な日数や費用の見積もりを行い、工程計画というものを考える。しかし、日数や費用は不確実で不確定な要素であり、何らかの事象によって変動してしまう。よって、今の建築現場の工事に対して、事前に考えられる日程計画は十分に活用できていない。更に、深刻な人手不足により職人さんを確保することができず、予定通りに作業が進まないということも考えられる。他にも、未知のシステムに対する現場の拒絶反応や、現場作業の環境の変化によるシステムの柔軟不足なども課題である。

このような課題を解消するため、本研究では次の三つのことを考慮した日程計画問題の提案を行う。一つ目は、日程計画を考えるとときに必要となる費用の不確実性と不確定性を考慮することで、より実問題に対応できるように工夫する。本研究では、作業情報の不確定性と不確実性の両方の性質を同時に表現するため、ファジィ・ランダム変数 [2, 3] の概念を導入する。

二つ目は、不確実で不確定な要素を考慮した上で、職人さんごとの各作業に対する作業効率を考え、各作業に対して最適な職人さんの選択・割り当てを補助することである。

三つ目は、今、現場作業に導入されているシステムを有効活用することである。未知のシステムに対する作業員の戸惑いや不安な感情は、かえって作業効率を悪くしてしまう可能性がある。よって、本研究では、現状導入されている現場管理システムから得られるデータを有効活用できるような日程計画問題を考え、今あるシステムを拡張することで実用できるような仕組みを提案する。つまり、提案する日程計画問題が、より実問題に対応しやすいようにするため、現場におけるデータの収集方法を考え、そのデータを活用しパラメータの設定が行えるような定式化と式変形を行う。

更に、日程計画問題はファジィ・ランダム変数を導入することにより複雑な問題になる。複雑な問題になるほど、計算時間が膨大になることが考えられるため、現場での活用を見越して処理時間の高速化も検討する必要がある。よって、解法では、遺伝的アルゴリズム (GA) を使用した並列分散処理による処理時間の高速化を考える。解法に GA を選択する理由は、GA は個体集合として多数の解候補を扱うという特徴があり、この特徴を生かし、個体ごとの処理を複数のマシンに分散させることで、容易に探索処理を並列分散することが可能であるからだ。よって、本研究では、Raspberry Pi3 を 8 台と MPI を用いて、島 GA の並列分散処理が実行できる環境の構築も行う。

## § 1.3 本論文の概要

本論文は次のように構成される。

**第 1 章** 本研究の概要と目的について説明した。

**第 2 章** 現状の建築業界における日程計画について説明する。また、現場から得られる作業情報と、活用する作業履歴についての説明、今の建築現場の課題について述べる。

**第 3 章** 一般的なファジィ・ランダム変数についての説明や、解法に用いる多目的最適化の

遺伝的アルゴリズム，高速化を目的とした並列分散環境の構築について述べる．加えて，並列分散処理による高速化を確認する動作テストについてもまとめる．

**第4章** ファジィ・ランダム変数を導入した多目的日程計画問題について説明する．更に，問題の定式化と式の等価確定変換について説明する．

**第5章** まとめと今後の課題を述べる．

# 日程計画と建築業界

## § 2.1 日程計画における課題

日程計画問題 (スケジューリング問題)

生産や建設などの1つの大きなプロジェクトの工程において、作業を効率よく進めるため、適切に仕事の順序を決定する問題を一般的にスケジューリング問題という。

かつて、大規模なプロジェクトのスケジュールや生産工程の管理には、工程・作業の進み具合を線表で表す、H.L. Gantt が考案したガント・チャートと呼ばれるツールが、主として手作業で使われていた。しかし、プロジェクトの規模の拡大や生産工程の複雑化によって、旧来の手法では処理し切れなくなりつつあり、電子計算機の登場により、これを利用した新しいプロジェクトの管理方法や多種多様な作業の日程計画法の開発が行われてきた。こうして生まれたのがプログラム・エバリュエーション・アンド・レビュー・テクニック (Program Evaluation and Review Technique : PERT) とクリティカルパスメソッド (Critical Path Method : CPM) という手法である [4]。

### PERT

PERT は、規模の大きなプロジェクトの複雑に入り組んでいる作業の流れを、図法を用いて分かりやすく表現することができる。また、各作業の所要時間のデータから、プロジェクトの完成が期日に間に合うか否か、どの作業を重点的に管理すれば、プロジェクト全体の所要時間の短縮になるかなどを判断できる日程管理を目的とした手法である [5]。

### CPM

PERT が日程だけを考えた計画法であるのに対して、CPM は PERT にコスト概念を取り入れ、日程の短縮に関するコスト・パフォーマンスを考える計画法である。

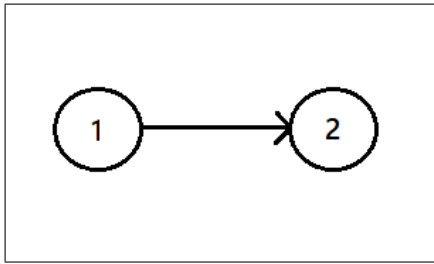


図 2.1: アロー・ダイアグラム

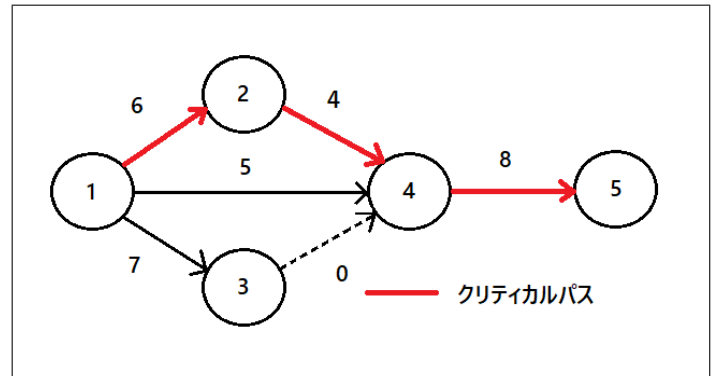


図 2.2: クリティカルパスの例

1つのプロジェクトは、ある目的を達成するための複数の作業からできている。建築現場の日程計画も、複数の作業が存在しており、それらの作業には「整地作業」をおこなってから「基盤」「骨組み」の順番で作業を行わなければならない、といったような順序関係がある。このように、ある作業に対し先行して終了していなければならない作業を先行作業、後に行う作業を後続作業と呼ぶ。このような、作業の順序関係を表現する方法に、アロー・ダイアグラムと呼ばれる図法を用いることが多い（図 2.1 参照）。

### クリティカルパス

図 2.1 のように、プロジェクトの各作業間の繋がりを可視化することで作業工程の把握・管理に役立てることができる。例えば、抽出した作業の「所要時間」「つながり」に基づき、プロジェクト全体の所要時間を算出できる。更に、この方法を用いると、全作業工程の中から「重要な作業」を特定することが可能だ。ここでの「重要な作業」とは「プロジェクト全体の所要時間の遅れ・短縮に大きく影響を与える作業」のことを指し、この作業工程をクリティカルパスと呼ぶ。

もし、プロジェクト全体の所要時間を短縮したいならば、このクリティカルパスの上にある作業の短縮に資源を費やすことが最も効果的である。

### クリティカルパスの算出方法

正式なクリティカルパスの算出には、往路時間計算（フォワードパス）と復路時間計算（バックワードパス）という方法を用いる。この場合、最早開始/終了時刻と最遅開始/終了時刻から余裕時間を求めることができ、余裕時間が0になる作業がクリティカルパスになる [6]。

しかし、本研究では所要時間が不確実で不確定な場合を想定するため、詳細なクリティカルパスの算出が困難である。したがって、上記の詳細な手法ではなく、簡易的な方法である「所要時間の積算」でクリティカルパスを算出する（図 2.2 参照）。これは、作成したプロジェクトネットワークの流れに沿って、所要時間を積算し、最も時間がかかるルートをクリティカルパスとする方法である。

詳細な方法よりは、余裕時間などの情報がない部分で劣るが、本研究で用いるクリ

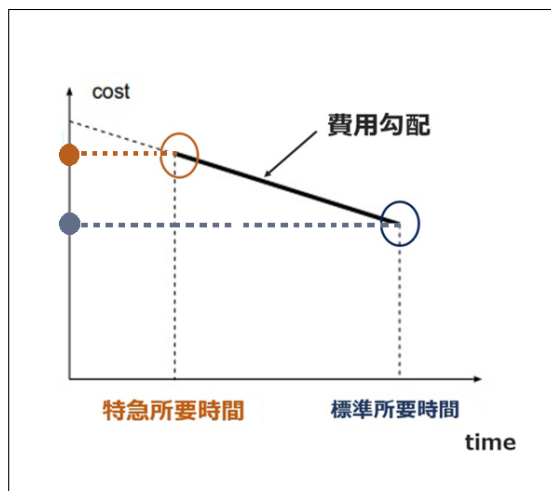


図 2.3: 時間費用関数

ティカルパスとしては、プロジェクト全体の最長所要時間が分かるだけで、十分であると考え、この方法を用いる。

本研究で考える最適な日程計画は「いかに無駄な資源をかけずに迅速にプロジェクトを遂行できるか」を目的とし、そのための最適な資源選びを補助することを目指す。よって、クリティカルパス上の作業の所要時間を最小化することを本研究の日程計画における1つ目の目的関数とする。

ここまでで、クリティカルパス上の所要時間を管理することが最効率であると述べた。しかし、CPMで所要時間の最小化を目的とする場合には作業員の追加、つまり費用の追加が伴い、これら2つの要素は本来トレードオフの関係にある。この時間と費用のトレードオフの関係を時間費用関数で表すことができる（図2.3参照）。

### 従来の日程計画問題の課題

スケジューリングは、様々なプロジェクトにおける最も重要な計画行為の1つである。そのスケジューリングの歴史は長く、数多くのスケジューリング問題に関する研究が行われてきた。そして、多くの研究成果が技術化されている。特に、計算機やソフトウェアの技術の発達により、以前よりも高度な生産システムや管理システムの実装が容易になってきている。しかし、スケジューリング問題に対する研究は数多く行われ、その技術化もされているが、そのシステムを実際に活用できていない業界は多い。なぜならば、研究においてアカデミックな視点から考えるスケジューリング問題と、現実世界の問題には多くのギャップが存在しているからである。



図 2.4: 現場コミュニケーションアプリ kizuku

#### 現実問題とのギャップの例

1. 従来の研究モデルでは、静的で確定的なものを前提としていたが、現実問題では動的で不確定なものが多く存在している。
2. 従来の研究モデルで扱う制約は、比較的単純で数も多くないが、現実問題では複雑な制約が数多く存在する。
3. 従来の研究モデルでは、コスト関数 (時間費用関数) は線形であるが、現実問題では非線形である。

このように、研究モデルと、現実問題との間には様々なギャップが存在している。研究モデルでも対応できるような単純な問題は、現実問題には殆ど存在しておらず、複雑で大規模な問題が多い [7]。したがって、これらのギャップを解消しなければ、今後スケジューリング手法が幅広い業界で活用されることは難しいと考えられる。

## § 2.2 作業履歴の活用

今、建築現場における作業を IT の技術を用いることで、少しでも作業員の負担を減らす工夫がされている。

#### 建設 BALENA<sup>1</sup>

建設業界における、お金と工事、人の流れを Web 上で管理するサービスである。工事の現場よりも、建設会社やリフォーム会社の管理部門で役立つ機能が充実している [8]。

#### 現場コミュニケーションアプリ Kizuku<sup>2</sup>

このアプリでは、建築の現場における進捗報告、確認作業、指示出しすべてを物件専用トークで管理し、自社社員や協力会社の就労者リアルタイムでやり取りができる (図 2.4)。また、このアプリでは工程スケジュール情報も管理しており、各作業工程の開始と完了の情報が分かる [9]。

今、建築の現場では現場での作業情報を管理・共有する為のアプリが導入されている。そのアプリでは、作業員の入退場や作業時間などが管理されている。したがって、このようなアプリから得られる作業履歴のビックデータから日程計画問題を考えるのに必要となるパラメータを得ることができると考える。特に、現場コミュニケーションアプリ Kizuku は現場における管理に役立つ機能が充実しているため、このアプリを活用することで、現場

<sup>1</sup><https://mag.branu.jp/archives/2442>

<sup>2</sup><https://www.ctx.co.jp/kizuku2pr/>



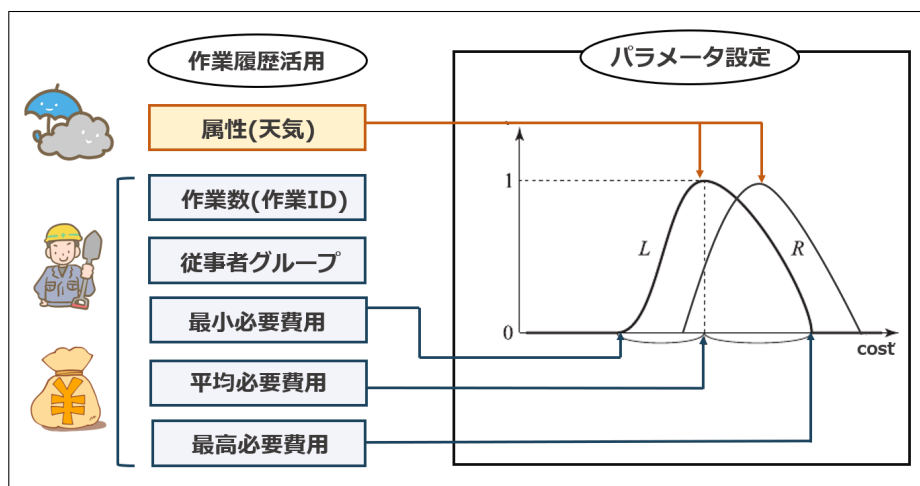


図 2.5: データの活用例

で得られたリアルタイムなデータを用いた職人さんの日程計画を管理することも可能であると考えられる。

今、このような現場アプリの登場により、以前よりも現場での作業情報の管理がしっかりとされている。よって、過去の作業履歴の情報を蓄積し、分析することが可能であると考えられる。得られる情報としては、各作業員ごとの作業ごとの所要時間と費用などがある。更に、その情報と天気の情報に合わせて考えると、属性（晴れ・雨・雪など）ごとの所要時間と費用の情報を得ることも可能である。

本研究では、このようなデータがアプリから得られると想定し、過去の作業履歴を活用したスケジューリングが可能であるとする。

#### データの設定例

入力データには、作業工程数と従事者グループ数、属性数、作業にかかる所要時間の最大日数と最小日数、作業にかかる最大費用と最小費用を用いる。

このように、作業履歴から得られるデータを入力データとしたとき、属性における各従事者グループごとの作業所要時間の最大値と最小値、平均値などを出力することができる。同時に、その条件のときにかかる費用の最大値と最小値、平均値も出力できる。

このようにして、過去の作業履歴から収集できるデータを用いて、日程計画に必要なパラメータを設定することができる（図 2.5 参照）。

## § 2.3 建築現場における課題

節 2.1 で述べた PERT や CPM などの手法は住宅建築の日程計画においても、古くから用いられてきた手法であり、現在も CPM を活用した日程計画を作業前の工程会議にて決めことがある。しかし、工程会議で事前に立てられた日程計画は、現状ほとんど活用されてい

ない。なぜならば、現場の作業では天候や事故などによる予期せぬ作業の中止や延期が多く、最悪の場合、翌日には計画と大きくズレが生じてしまうことさえあるからだ。このように、事前に立てた日程計画からのズレが積み重なり、工期遅れに陥ることが多々ある。

この「工期遅れ」に陥る原因は、天候による工事の中止や予期せぬトラブルもあるが、単純に「着工遅れ」が原因であることが多いとされている。そして、この着工遅れの原因として考えられるのが「人手不足」である。労働人口の減少が問題になっている今、特に人手不足が深刻な建築業界では、多くの大工さんが一度に多くの現場を掛け持ちしている。そのため、いざ工事に取り掛かろうとしたときに、必要な職人さんを確保することができず着工遅れになる。

このように、今の建築現場では事前に決められた日程計画を活用できていない。しかし、このまま工期遅れが当たり前の工事を行ってはいは、従事者の方々への負担が大きくなる一方である。結果、建築現場は「過剰労働」「人手不足」といった悪循環を繰り返すことになる。

#### 現場に CPM 手法を導入するときの問題点

節 2.1 で説明したようなネットワーク理論を中心とした PERT や CPM などの手法が、これまでのバーチャート式の工程計画に代わり、一部では実用化されつつある。しかし、これらの手法、特に CPM 手法を導入した工程計画の策定を試みるが、現場への導入にはいくつかの問題点がある。問題点は大きく分けて 4 つある [10]。

##### CPM 計算自身の問題

1. 計算時間の短縮
2. アルゴリズムの単純化と簡便法の開発

##### CPM 計算データの問題

1. 作業所要時間の見積もりの精度
2. 費用の勾配算定の精度

##### 計算策定に関する問題点

1. 工程実施が、初期計画から外れた場合の対処方法
2. 気象などの自然現象が工期におよぼす影響の解析

##### 工事に科学的管理計画技術を導入するための現場の施工体制

1. 理解できないシステムに対する現場の拒絶反応
2. システムにトラブルが発生したときの現場での対応の難しさ

そこで、今の建築現場に必要とされる仕組みは何かを考える必要がある。様々な工期遅れの原因が考えられるそのなかで、予期せぬトラブルによる遅れを考慮して作業効率アップに繋げることは難しい。しかし、天候の予測や職人さん選びによる作業効率アップは可能であると考えられる。また、CPMの導入課題にもあるが「計算時間の短縮」つまり、処理時間の高速化は、よりリアルタイムな管理が求められる現場作業において欠かせない要素であると考えられる。そのため、今の建築現場の生産性を向上させるために必要とされる仕組みは次のようなものであると考えた。

今の現場に必要な仕組み

1. 職人さんの工事スケジュールの把握と最適な割り振り
2. 天候などの不確実性・不確定性を考慮した日数・費用の見積もり
3. システム処理時間の高速化

## 実問題への対応と並列分散解法

### § 3.1 ファジィ・ランダム変数

従来、曖昧な場合と確率的な場合は混同して用いられてきた。しかし、可能性と確率性というものは、本来は違うものである。そして、現実問題には「確率変動要素」と「ファジィ要素」が両方同時に存在し、または両方の要素を併せもつ要因がある。具体例を挙げると、「投票率」などがその例である。天候という確率的要素により投票所に来る人数が異なり、晴れ・曇り・雨などの属性ごとに確率変動の実現値があいまいな数（ファジィ数）となる。このような、確率変動の実現値がファジィ数である数をファジィ・ランダム変数という（図 3.1 参照）[ [11]]。

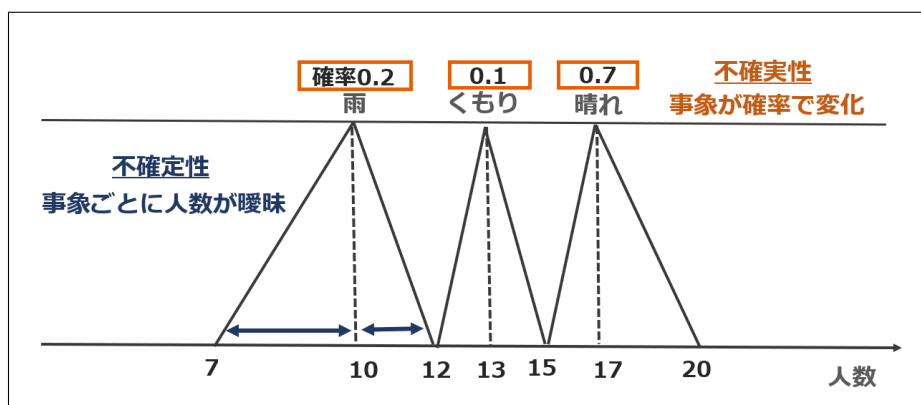


図 3.1: ファジィ・ランダムイメージ

つまり、ファジィ・ランダム変数は、確率変数やファジィ数を拡張したもので、不確定性（ファジィ性）と不確実性（ランダム性）の両方を表現することができる。既存研究の中でいくつかの提案がなされている。その中でも、ファジィランダム変数は基礎理論から応用

面まで幅広く研究がなされており，数理計画問題をベースとした意思決定に関する研究においても有益な結果が得られている [9, 10]．ファジィランダム変数は式 (3.1) のメンバシップ関数で定義される（図 3.2 参照）．

$$\mu_{\tilde{C}_{ij}}(\tau) = \begin{cases} L\left(\frac{\bar{d}_{ij} - \tau}{\bar{\beta}_{ij}}\right), & \text{if } \tau \leq \bar{d}_{ij}, \\ R\left(\frac{\tau - \bar{d}_{ij}}{\bar{\gamma}_{ij}}\right), & \text{otherwise.} \end{cases} \quad (3.1)$$

ここで， $\bar{d}_{ij}$  は中心（平均）で， $\bar{\beta}_{ij}, \bar{\gamma}_{ij}$  は左右の広がりを表すパラメータである．目的関数の係数は  $L-R$  ファジィ数において，中心，広がりのパラメータが確率変数となっているファジィ・ランダム変数であるため，各目的関数は，拡張原理に基づく  $L-R$  ファジィ数の演算により，式 (3.2) のようなメンバシップ関数で特徴づけられるファジィランダム変数となる（図 3.3 参照）．

$$\mu_{\tilde{C}_{ij}x_{ij}}(v) = \begin{cases} L\left(\frac{\bar{d}_{ij}x_{ij} - v}{\bar{\alpha}_{ij}x_{ij}}\right), & \text{if } v \leq \bar{d}_{ij}x_{ij}, \\ R\left(\frac{v - \bar{d}_{ij}x_{ij}}{\bar{\beta}_{ij}x_{ij}}\right), & \text{otherwise.} \end{cases} \quad (3.2)$$

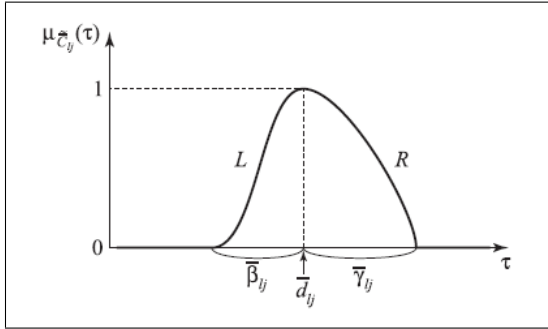


図 3.2: ファジィランダム変数のメンバシップ関数  $\mu_{\tilde{C}_{ij}}(\tau)$  の例

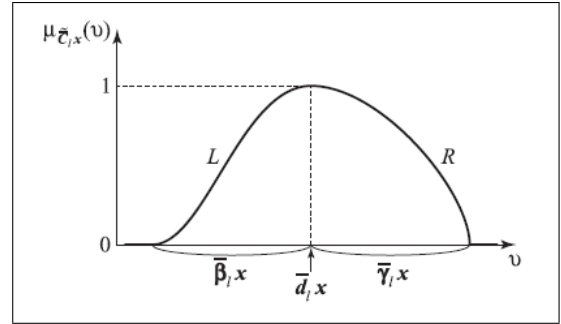


図 3.3: 目的関数のメンバシップ関数  $\mu_{\tilde{C}_{ij}x_{ij}}(v)$  の例

## § 3.2 日程計画における遺伝的アルゴリズム解法

日程計画問題の解法として、従来多くの解法が挙げられてきた。それらの解法を大きく分けると「厳密解法」と「近似解法」の二つに分けられる。

### 厳密解法

最適化問題に対して、最適性の保証された解を求める解法のこと、主に NP 困難な問題に対して用いられる。NP 困難である問題とは、問題の規模が大きくなると爆発的に計算時間が増加するような問題である。解法としては、分枝限定法などがある [ [12]]。

### 近似解法

厳密解ではなく、ある程度良いとされる解、近似解を求める解法のこと、厳密解法では解を求めることが困難である場合に用いられる。厳密解法に比べて、近似解を得るまでにかかる時間は短くなるが、厳密解ではないため、解に対する信憑性についての評価が必要になる。

前述した厳密解法を用いて、現実問題の複雑な日程計画問題を解くと、実行可能な解が求まらなかったり、膨大な計算時間がかかるため、直接用いることは困難とされている。よって、現実問題のような複雑で大規模な問題を解く場合には、近似解法である遺伝的アルゴリズムやタブ探索などのメタヒューリスティックスに基づいたシステムが用いられている [7]。更に、現実問題における日程計画問題の多くは、単目的な問題ではなく、多目的な問題がほとんどである。

#### 多目的最適化問題

最適化問題とは、目的関数の引数を調整して目的関数を最小化、もしくは最大化する問題のことを指す。その中でも、目的関数が複数のものを多目的最適化問題という。現実問題において、単目的な問題は稀であり、多くの問題が多目的最適化問題であるとされる。一般的に、多目的最適化問題においては、目的関数の間に一方が良くなれば他方が悪くなるトレードオフの関係がある。したがって、多目的最適化問題において、最適解は1つではない（図 3.4 参照）。

現実問題に近い問題を解くことを考え、本研究では、近似解法の一つである遺伝的アルゴリズムを用いたパレート最適解の導出を検討する。

### 遺伝的アルゴリズム

遺伝的アルゴリズム (Genetic Algorithm : GA) とは自然界における生物の進化モデル、すなわち世代を形成している個体の集合（個体群）の中で、環境への適応度の高い個体が次世代により多く生き残り、交叉や突然変異を起こしながら次の世代を形成していく過程を模した最適化手法である（図 3.5 参照）。これまでに、多くのスケジューリング問題に対して GA の適用が試みられてきた [ [13]]。

GA は、遺伝的操作・適応度計算に対する膨大な計算コストが要求される。よって、並列計算機によるアルゴリズムの並列化について様々な検討がなされている [ [14]]。

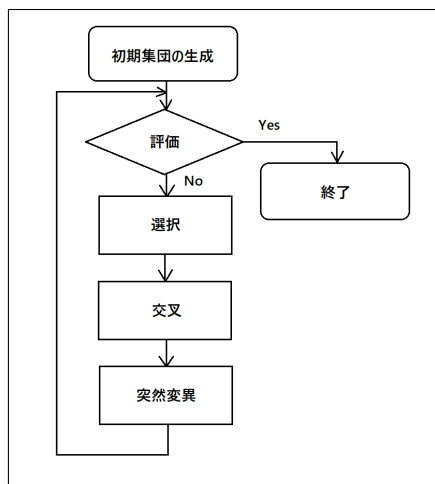


図 3.4: GA のフローチャート

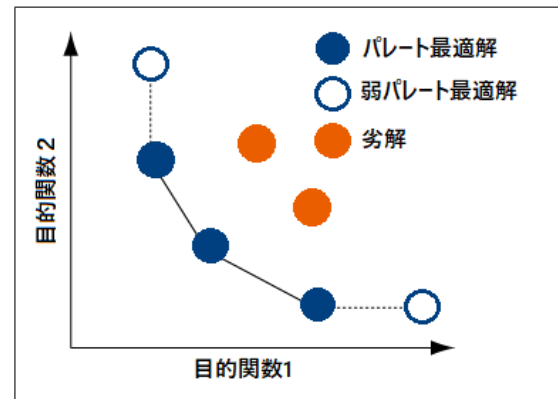


図 3.5: パレート最適解の例

### 並列分散 GA

並列分散 GA とは、全体の個体群を、いくつかの島に分散させ、その中でのみ遺伝的操作を行うモデルを指す。島ごとに異なる交叉率や突然変異率等のパラメータを設定出来るため、パラメータ設定の困難さを緩和する効果がある。基本的に島の中でのみ遺伝的操作が行われるため、各島ごとに異なる局所優良解に到達する事が期待出来る。様々な実験により、並列分散 GA では、全体を一つにして行なう単純 GA と比較して、最適解発見確率が格段に高まる事が確認されている。更に、島と島の個体同士を交換（移住）することで局所解に陥るのを防ぎ、島ごとの多様性を維持することが出来るため、解の探索能力が高まる（図 3.6 参照）。

また、並列分散処理になるので、異なるコンピュータを島ごとに割り当てる事が可能となり、計算時間の大幅な短縮も可能である。

## § 3.3 並列分散処理による高速化

本研究では、節 3.2 で述べた多目的最適化の解法に対して、Raspberry Pi8 台と MPI を用いて並列分散処理を行う（図 3.7 参照）。

Raspberry Pi<sup>3</sup>とは――

Raspberry Pi とは、内臓ハードディスクなどを搭載しない代わりに、電源や SD カードストレージを装着することによって使用できる「ワンボードマイコン」と呼ばれるハードウェアのこと。2012 年により安価な教育用のシングルボードコンピュータとして開発された [ [15]]。

<sup>3</sup><https://jp.rs-online.com/web/p/products/1239470/>

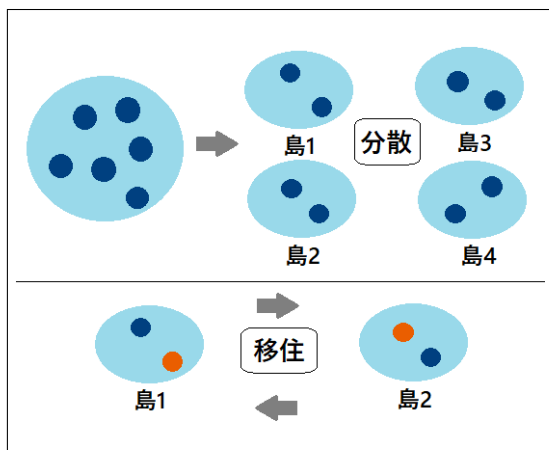


図 3.6: 並列分散 GA のイメージ



図 3.7: 実験に用いるラズベリーパイ 8 台

## MPI の概要

MPI とは Message Passing Interface の略で、分散メモリ間のメッセージ通信の実行基盤の 1 つである。ある 1 つの目的を複数のコンピュータを用いて、分散・並列処理する際に用いられる。MPI には、mpich や Open MPI などのいくつかの実装系があり、本研究では、現状フリーで広く使用されている mpich を用いる [ [16]]。

## Raspberry Pi の環境構築

本研究では、複数マシンにまたがって複数のプロセスを起動し、それぞれの間で互いに通信し、並列処理を行う。このとき、外部ネットワークを利用してプロセス間通信を行うため、セキュアシェル（SecureShell：SSH）とネットファイルシステム（Network File System：NFS）の設定、hosts ファイルを準備する必要がある。

また、複数ノード上での複数プロセスによる並列処理を行うとき、担当分の処理を行うと同時に並列処理や結果の集約を担当する「マスターノード」とマスターノードと連携して担当分の処理を行う「スレーブノード」の 2 種類に分けられる。

### 手順 1

まず、ノード間並列を行う前に、パスワードなしの ssh 接続ができる状態にしておく必要がある。そのため、各マシンの IP アドレスを固定し、全てのマシンに互いの鍵を送信する。全てのマシンにパスワードなしの ssh 接続ができることが確認できたら、全てのマシンに mpich をインストールする。

### 手順 2

次に、NFS の設定を行う。NFS とは、分散ファイルシステムの一つで、あるマシンが所有する 1 つのディレクトリを、他の複数のノードで共有して使うための仕組みである。MPI を使う際には、実行ファイルが全てのマシンで同じ位置に置かれている必要がある。よって、NFS を用いて、マスターノードの作業ディレクトリをスレーブノードに共有する。

### 手順 3

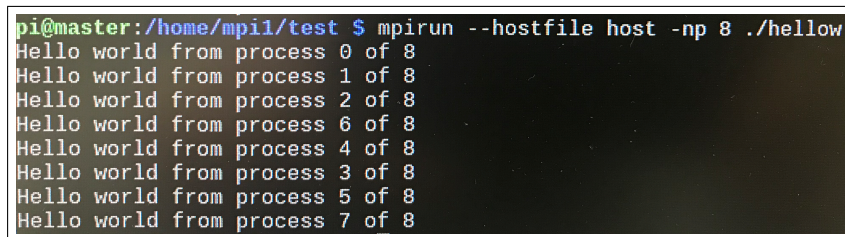


最後に、各マシンの IP アドレスまたはホスト名を書いた hosts ファイルを「マスターノード」の実行プログラムと一緒に置く．この hosts ファイルとは、ドメインネームシステム（Domain Name System : DNS）よりも先に参照される IP アドレスとドメイン名の一覧である．DNS とは、インターネットの重要な基盤技術の 1 つで、ドメイン名と IP アドレスの対応付けなどを行うシステムである．

更に、この構築した環境でプログラムが問題なく動くかを確認するために、ラズベリーパイ 8 台を使った「hello world の並列処理」「円周率計算の並列分散処理」「並列分散 GA を用いた巡回セールスマン問題の処理時間比較」を行う．

### 動作テスト 1: Hellow World の並列処理

動作確認のため、ラズベリーパイ 8 台で「Hellow World」を表示させるプログラムを動かした (図 3.8 参照)．MPI によって起動された各プロセスには、「ランク」と呼ばれる 0 から始まる識別番号が付与される．図 3.8 の左側の数字が「ランク」を表しており、右側の数字が動いているプロセスの総数である．また、このランクの値によって各プロセスの処理を制御することができる [ [16]]．



```
pi@master:/home/mpil/test $ mpirun --hostfile host -np 8 ./hellow
Hello world from process 0 of 8
Hello world from process 1 of 8
Hello world from process 2 of 8
Hello world from process 6 of 8
Hello world from process 4 of 8
Hello world from process 3 of 8
Hello world from process 5 of 8
Hello world from process 7 of 8
```

図 3.8: mpich とラズベリーパイ 8 台を用いた Hellow World の並列実行結果

図 3.8 を見ると、問題なく 8 台分の「Hellow World」が表示されており、0 から 7 のランクの値も表示されている．よって、並列処理を正常に行える環境であることが確認できた．

### 動作テスト 2 :円周率計算の並列分散処理

動作テスト 1 で Hellow World を 8 台分問題なく表示させることができた．動作テスト 2 では、円周率計算の並列分散処理を行い、計算の処理時間が問題なく計測できるかを確認する．更に、1 台、2 台、4 台、8 台での処理時間を比較し、計算の高速化ができていないかの確認も行う（表 3.1, 3.9 参照）．

図 3.9 のように、問題なく 8 台での並列分散処理を行ったときの処理時間を計測できた．また、図 3.10 をみると、分散する台数を増やすにつれ、計算にかかる処理時間を短縮できていることが確認できる．

表 3.1: mpich とラズベリーパイ 8 台を用いた円周率計算の処理時間

台数	処理時間
1 台	51 秒
2 台	25 秒
4 台	21 秒
8 台	17 秒

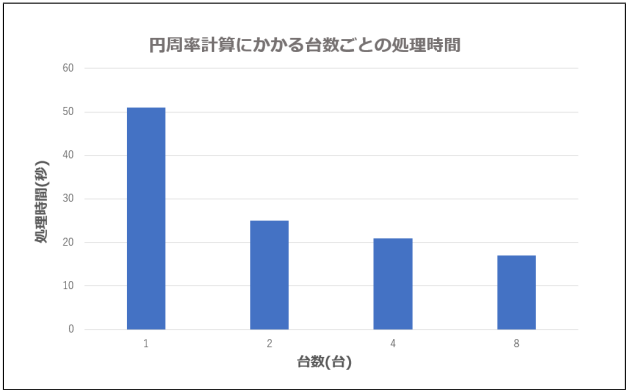


図 3.9: 円周率計算の台数ごとの処理時間グラフ

動作テスト 3: 並列分散 GA による巡回セールスマン問題

本研究では、日程計画の解法として、MPI を用いた並列分散 GA の活用を検討する (図 3.10 参照)．そのための、動作テストとして巡回セールスマン問題に対しての台数ごとの処理時間の変化を比較する [17]．

巡回セールスマン問題 (TSP 問題) —

多くの都市と各都市間の移動コストがあたえられたとき、全ての都市を一度だけ回り戻ってくるルートのうち、最小のコストになる回り方を求める問題のことである．

計算に用いたパラメータ —

マップは 30 × 30，都市数は 50 都市，遺伝子数 800 個，世代数は 50000 世代，移住数は 120 遺伝子 (15 %)，[端末数は 1 台、2 台、4 台、8 台それぞれの台数で処理時間を比較する．

動作テスト 1，動作テスト 2 と動作テスト 3 より，並列分散処理による高速化が可能であることが確認できた．最後の動作テスト 3 を行ったときのみ 4 台から 8 台へ分散台数を増やしても処理時間が短縮されなかった (表 3.2, 図 3.11 参照)．これは，実験中のマシンの様子から，サーバーやネットワーク機器などのハードウェアの処理能力不足や通信のオーバーヘッドによるものであると考えられる．

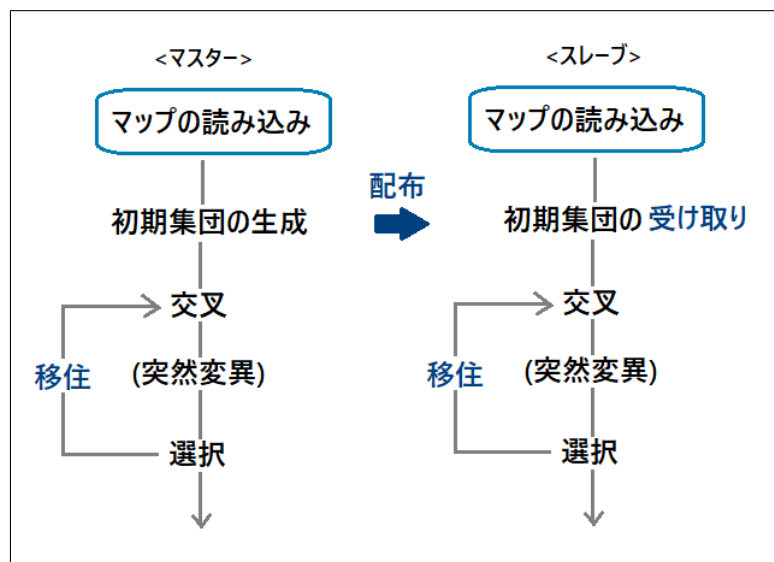


図 3.10: MPI による並列分散 GA のフロー

表 3.2: 並列分散 GA を用いた TPU 問題台数ごとの処理時間

台数	処理時間
1 台	51 秒
2 台	25 秒
4 台	21 秒
8 台	17 秒

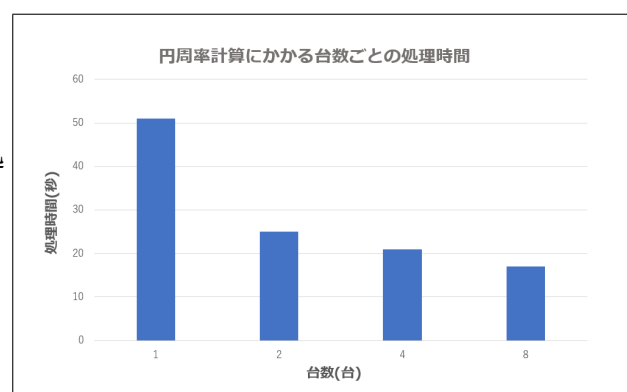


図 3.11: 並列分散 GA を用いた TPU 問題台数ごとの処理時間のグラフ

## 提案手法

### § 4.1 ファジィランダム変数を導入した多目的日程計画問題

本研究で、提案するモデルについて説明する。節 2.4 で述べた建築現場の悩みと節 3.1 で述べた従来の日程計画問題の課題の内容から、建築現場における作業効率アップを目指す日程計画を考えた。

提案モデルでは、特に建築現場の悩みを解消するために必要とされる「職人さんの最適な割り振り」と従来の日程計画問題の課題とされる「天候・日数・費用などの動的で不確定・不確実な要素の見積もり」を考慮したファジィ・ランダム多目的日程計画問題についてまとめる（図 4.1 参照）。

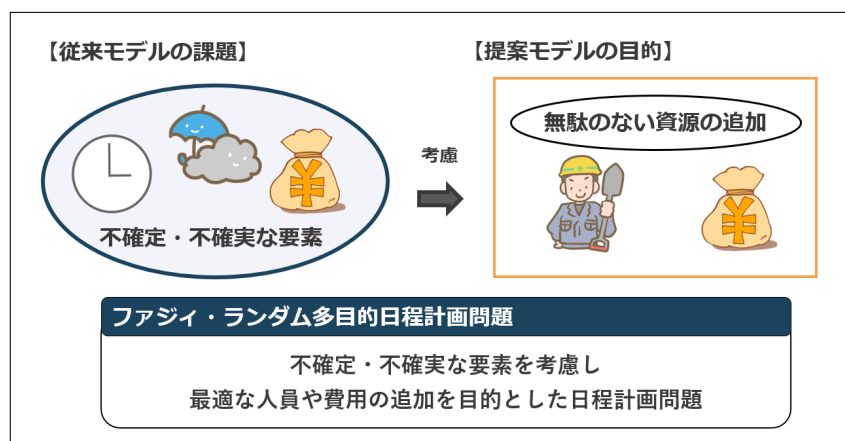


図 4.1: ファジィ・ランダム多目的日程計画問題

図 4.1 のように、本研究では、住宅建築における日程計画は、天候という「確率変動要素」と費用という「ファジィ要素」が両方同時に存在すると考え、これらの要素を考慮するファジィ・ランダム多目的日程計画を考えた。具体的な作業の順序関係を可視化した住宅建築におけるプロジェクトネットワークの例を次に示す（図 4.2 参照）。

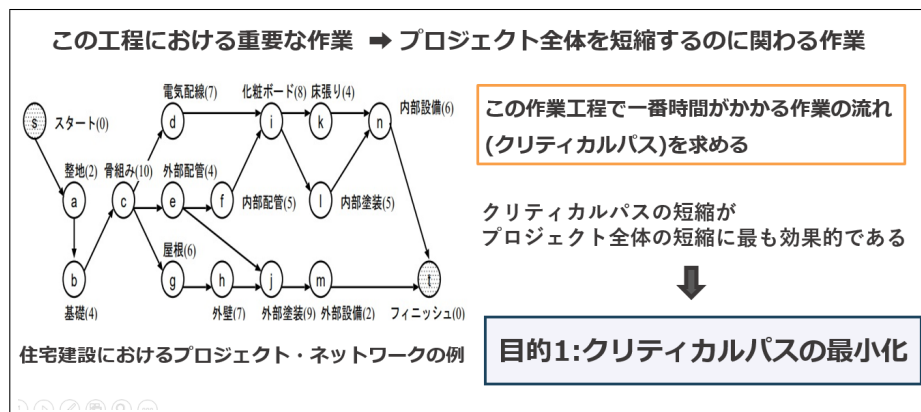


図 4.2: クリティカルパスの所要時間を最小化

このような、プロジェクトネットワークも活用しつつ、算出したクリティカルパス上にある作業の所要時間の最小化を目指す。更に、本研究では所要時間と費用の両方の最小化を目的とした多目的日程計画問題を考える（図 4.3 参照）。

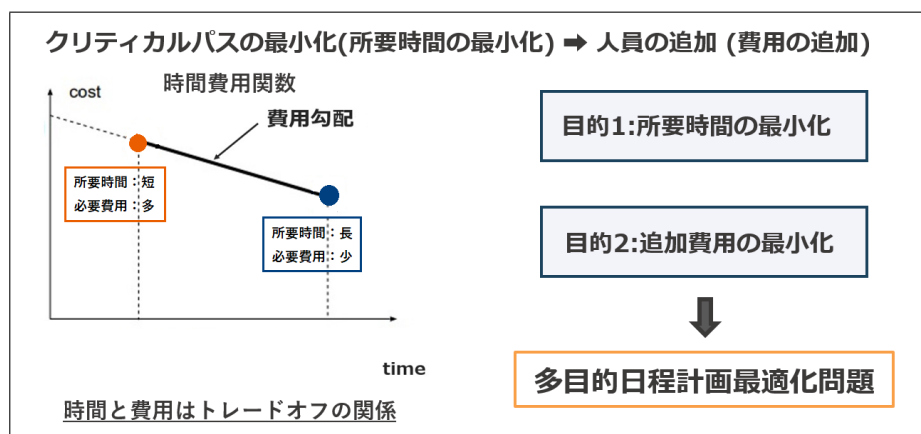


図 4.3: 多目的最適化問題

従来、所要時間と費用の関係は、節 2.3 で説明したような時間費用関数で特徴づけられると考えられる。しかし、提案モデルでは、図 4.3 のような静的で確定的な時間費用関数ではなく、動的で不確定な時間費用関数によって特徴づけられる（図 4.4 参照）。

この問題では、従事者グループごとの費用のファジィ性・ランダム性を両方表現するため、費用を最小化するという目的関数の係数にファジィ・ランダム変数を用いる。この問題を解くことにより、天候ごとの各作業における作業日数と費用が見積もることができる。したがって、各作業における最適な従事者グループの割り当てを考えることが可能になる。

## 提案手法の定式化

本研究では、考えたファジィ・ランダム多目的日程計画問題を次のように定式化した。

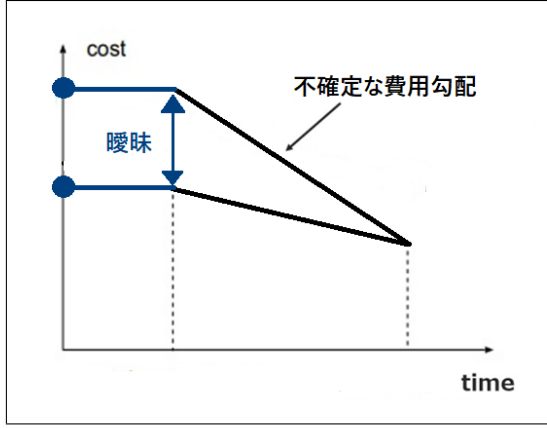


図 4.4: 不確定な時間費用関数

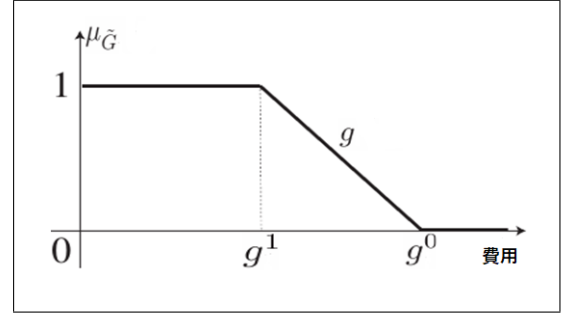


図 4.5: ファジィ目標  $\tilde{G}$  のメンバシップ関数  $\mu_{\tilde{G}}$

$$\begin{aligned}
 & \text{minimize} \quad \max \left\{ \sum_{i=1}^n \sum_{j \in N_i} \sum_{k=1}^w t_{ijk} x_{ij} y_{ik} \right\} \\
 & \text{minimize} \quad \sum_{i=1}^n \sum_{j \in N_i} \sum_{k=1}^w \tilde{c}_{ik} x_{ij} y_{ik} \\
 & \text{subject to} \quad \left. \begin{aligned}
 & \sum_{i=1}^n \sum_{j \in N_i} x_{ij} - \sum_{i=1}^n \sum_{j \in N_i} x_{ji} = \begin{cases} 1, (i = s), \\ 0, (i \neq s, t), \\ -1, (i = t). \end{cases} \\
 & x_{ij} \in \{0, 1\}, \quad i = 1, \dots, n, j = 1, \dots, n; \\
 & y_{ik} \in \{0, 1\}, \quad i = 1, \dots, n, k = 1, \dots, w;
 \end{aligned} \right\} \quad (4.1)
 \end{aligned}$$

式 (4.1) は、クリティカルパス上の作業の総所要時間と総費用の最小化するための解を求める定式化である。ここでの解とは、クリティカルパス上の作業とその従事者グループの組合せである。ここで、 $i$  は先行作業、 $j$  は後続作業、 $n$  はプロジェクトの総作業数である。また、 $k$  は作業を受け持つ従事者グループ、 $w$  は依頼候補の従事者グループ数を表している。

よって、 $t_{ijk}$  は作業  $i$  から作業  $j$  開始前までを従事者グループ  $k$  が受け持ったときの所要時間、 $\tilde{c}_{ik}$  は作業  $i$  を従事者グループ  $k$  に依頼したときの費用を表している。 $x_{ij}$  は作業  $i$  から作業  $j$  を選択する 0-1 変数、 $y_{ik}$  は作業  $i$  を従事者グループ  $k$  に依頼するかを選択する 0-1 変数である。また、 $\tilde{c}_{ik}$  の各要素は、節 3.1 で述べたメンバシップ関数により特徴づけられるファジィ・ランダム変数を要素とする係数ベクトルである。

## § 4.2 等価確定問題への変換

ここで、ファジィ・ランダム変数を含む式をそのまま取り扱うことは困難であるため、確率計画問題から多目的日程計画問題へ等価確定変換する必要がある。

そこで、本研究では、式 (4.1) のファジィ・ランダム変数を係数に含む目的関数に対して、可能性測度と確率最大化モデルに基づき式変形を行う [18] [19] [20]。

### ファジィ目標 $\tilde{G}$ の導入

ファジィランダム変数を含む2つ目の目的関数に対して、ファジィ目標を導入する。ファジィ目標  $\tilde{G}$  のメンバシップ関数  $\mu_{\tilde{G}}$  は次の式 (4.2) によって特徴づけられる。式 (4.2) によって特徴づけられるファジィ目標のメンバシップ関数の例を図示する (図 4.5 参照)。

$$\mu_{\tilde{G}} = \begin{cases} 1, & \text{if } y < g^1, \\ g(y), & \text{if } g^1 \leq y \leq g^0, \\ 0, & \text{if } y > g^0. \end{cases} \quad (4.2)$$

ここで、 $g^1$  は最良値 (最小費用)、 $g^0$  は最悪値 (最高費用) である。 $g$  は狭義単調減少連続関数である。

目的関数のメンバシップ関数  $\mu_{\tilde{c}_{ik}x_{ij}y_{ik}}$  を可能性分布とみなし、このとき、その可能性分布の下でファジィ目標  $\tilde{G}$  が満たされる可能性の度合いを  $\Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G})$  とし、可能性測度を用いて次の式 (4.3) ように与えられる。可能性の度合い  $\Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G})$  を図示した (図 4.6 参照)。

$$\Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G}) = \sup \min \{ \mu_{\tilde{c}_{ik}x_{ij}y_{ik}}, \mu_{\tilde{G}} \} \quad (4.3)$$

原問題に対して、ファジィ目標が満たされる可能性の度合い (可能性測度) を最大化する問題へ変換して考えると、式 (4.4) のようになる。

$$\left. \begin{array}{ll} \text{minimize} & \max \left\{ \sum_{i=1}^n \sum_{j \in N_i} \sum_{k=1}^w t_{ijk} x_{ij} y_{ik} \right\} \\ \text{maximize} & \Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G}) \\ \text{subject to} & \end{array} \right\} \quad (4.4)$$

$$\sum_{i=1}^n \sum_{j \in N_i} x_{ij} - \sum_{i=1}^n \sum_{j \in N_i} x_{ji} = \begin{cases} 1, & (i = s), \\ 0, & (i \neq s, t), \\ -1, & (i = t). \end{cases}$$

$$x_{ij} \in \{0, 1\} \in X, i = 1, \dots, n, j = 1, \dots, n;$$

$$y_{ik} \in \{0, 1\} \in Y, i = 1, \dots, n, k = 1, \dots, w;$$

ここで、制約条件の  $X, Y$  は問題の実行可能領域を表している。可能性測度の導入により、費用のファジィ性を確定的に取り扱うことができるため、問題は確率計画問題となる。

### 確率最大化モデルによる変換

ここでは、問題における可能性の度合い  $\Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G})$  の最大化を  $\Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G})$  がある満足

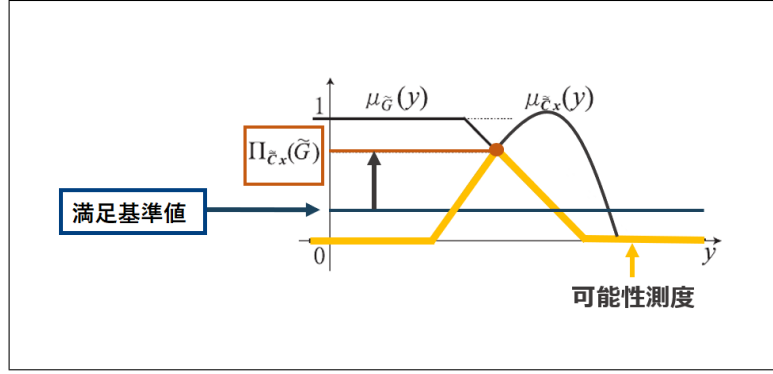


図 4.6: 満足基準値を用いた確率最大化モデル

基準値  $h$  以上となる確率を最大化するという確率最大化モデルに基づき,  $\Pr[\Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G}) \geq h]$  の最大化に置き換える (図 4.6 参照). 問題は式 (4.5) のように定式化される.

$$\begin{aligned}
 & \text{minimize } \max \left\{ \sum_{i=1}^n \sum_{j \in N_i} \sum_{k=1}^w t_{ijk} x_{ij} y_{ik} \right\} \\
 & \text{maximize } \Pr[\Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G}) \geq h] \\
 & \text{subject to} \\
 & \sum_{i=1}^n \sum_{j \in N_i} x_{ij} - \sum_{i=1}^n \sum_{j \in N_i} x_{ji} = \begin{cases} 1, (i = s), \\ 0, (i \neq s, t), \\ -1, (i = t). \end{cases} \\
 & x_{ij} \in \{0, 1\}, \in X \quad i = 1, \dots, n, j = 1, \dots, n; \\
 & y_{ik} \in \{0, 1\}, \in Y \quad i = 1, \dots, n, k = 1, \dots, w;
 \end{aligned} \quad (4.5)$$

ここで, 任意の根元事象に対して  $\Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G}) \geq h$  は

$$\begin{aligned}
 & \Pi_{\tilde{c}_{ij}x_{ij}}(\tilde{G}) \geq h \\
 \Leftrightarrow & \sup \min \{ \mu_{\tilde{c}_{ik}x_{ij}y_{ik}}, \mu_{\tilde{G}} \} \geq h \\
 \Leftrightarrow & \exists y : \mu_{\tilde{c}_{ik}x_{ij}y_{ik}} \geq h, \quad \mu_{\tilde{G}} \geq h \\
 \Leftrightarrow & \exists y : L \left( \frac{\bar{d}x_{ij}y_{ik} - y}{\beta x_{ij}y_{ik}} \right) \geq h, \quad R \left( \frac{y - \bar{d}x_{ij}y_{ik}}{\bar{\gamma}x_{ij}y_{ik}} \right) \geq h, \quad \mu_{\tilde{G}}(y) \geq h \\
 \Leftrightarrow & \exists y : \{ \bar{d} - L^*(h)\bar{\beta} \} x_{ij}y_{ik} \leq y \leq \{ \bar{d} + R^*(h)\bar{\gamma} \} x_{ij}y_{ik}, \quad y \leq \mu_{\tilde{G}}^*(h) \\
 \Leftrightarrow & \{ \bar{d} - L^*(h)\bar{\beta} \} x_{ij}y_{ik} \leq \mu_{\tilde{G}}^*(h).
 \end{aligned}$$

のように変形することができる. ここで,  $L^*(h)$  及び  $\mu_{\tilde{G}}^*(h)$  は擬逆関数であり,

$$\begin{aligned}
 L^*(h) &= \sup \{ r | L(r) \geq h, r \geq 0 \} \quad (0 < h \leq 1) \\
 \mu_{\tilde{G}}^*(h) &= \sup \{ r | \mu_{\tilde{G}}(r) \geq h \} \quad (0 < h \leq 1)
 \end{aligned}$$



というように表わされる．さらに，確率変数  $\bar{t}$  の分布関数を  $F(\cdot)$  とすると，問題は

$$\begin{aligned}
\Pr[\Pi_{\tilde{c}_{ij}x_{ij}}(\tilde{G}) \geq h] &= \Pr[\{\bar{d} - L^*(h)\bar{\beta}\}x_{ij}y_{ik} \leq \mu_{\tilde{G}}^*(h)] \\
&= \Pr[\{(d^1 + \bar{t}d^2) - L^*(h)(\beta^1 + \bar{t}\beta^2)\}x_{ij}y_{ik} \leq \mu_{\tilde{G}}^*(h)] \\
&= \Pr\left[\bar{t} \leq \frac{\{L^*(h)\beta^1 - d^1\}x_{ij}y_{ik} + \mu_{\tilde{G}}^*(h)}{\{d^2 - L^*(h)\beta^2\}x_{ij}y_{ik}}\right] \\
&= F\left(\frac{\{L^*(h)\beta^1 - d^1\}x_{ij}y_{ik} + \mu_{\tilde{G}}^*(h)}{\{d^2 - L^*(h)\beta^2\}x_{ij}y_{ik}}\right)
\end{aligned}$$

となる．最終的に，定式化した式 (4.1) のファジィ・ランダム多目的日程計画問題は式 (4.6) のような等価な多目的計画問題へ置き換えることができる．

$$\left. \begin{array}{ll} \text{minimize} & \max \left\{ \sum_{i=1}^n \sum_{j \in N_i} t_{ijk} x_{ij} y_{ik} \right\} \\ \text{maximize} & F\left(\frac{-\{d^1 - L^*(h)\beta^1\}x_{ij}y_{ik} + \mu_{\tilde{G}}^*(h)}{\{d^2 - L^*(h)\beta^2\}x_{ij}y_{ik}}\right) \\ \text{subject to} & \end{array} \right\} \quad (4.6)$$

$$\begin{aligned}
\sum_{i=1}^n \sum_{j \in N_i} x_{ij} - \sum_{i=1}^n \sum_{j \in N_i} x_{ji} &= \begin{cases} 1, (i = s), \\ 0, (i \neq s, t), \\ -1, (i = t). \end{cases} \\
x_{ij} \in \{0, 1\}, & \in X \quad i = 1, \dots, n, j = 1, \dots, n; \\
y_{ik} \in \{0, 1\}, & \in Y \quad i = 1, \dots, n, k = 1, \dots, w;
\end{aligned}$$

### § 4.3 提案手法のアルゴリズム

まず，ファジィ・ランダム多目的日程計画問題を解くために必要となるデータの収集を行い，そのデータを用いたパラメータの設定方法を以下に示す．

**Step0:** 建築現場における作業情報を現場コミュニケーションアプリなどを活用して蓄積する．収集するデータは，「属性数(天候)」「作業数(作業ID)」「従事者数」「従事者グループごとの各作業にかかった日数」「従事者グループごとの各作業にかかった費用」である．

次に，対象とするプロジェクトにおける重要な作業の導出とを行う．ここで，重要な作業とは，プロジェクトを遂行する上で最も時間がかかる作業の流れであるクリティカルパスを示す．クリティカルパスの算出方法は，節 2.1 で説明した方法を用いる．

**Step1:** 蓄積された過去の作業履歴のデータから費用のファジィ・ランダム変数を特徴づけるためのメンバシップ関数に必要な左右の広がりパラメータ  $\beta_{ik}, \gamma_{ik}$  と中心値  $\bar{d}_{ik}$  を設定する．左の広がりパラメータ  $\beta_{ik}$  は作業履歴の中の最小費用を，右の広がりパラメータ  $\gamma_{ik}$  には最大費用を，そして，中心値  $\bar{d}_{ik}$  には費用の平均値を設定する．

**Step2:** 作業履歴から収集したデータにより，意思決定者の判断でファジィ目標の最良値  $g_0$  と最悪値  $g_1$  を設定し，ファジィ目標  $G$  のメンバシップ関数  $\mu_{\bar{G}}$  を設定する．更に，設定したファジィ目標を満たす可能性が満足基準値  $h$  より大きくなる確率を最大化する目的関数へ変換する．

**Step3:** Step0 で収集したデータを参考に，プロジェクトにおけるクリティカルパスの導出を行う．本研究では，この導出したクリティカルパス上の所要時間と費用最小化するような職人さんの選択を行い，クリティカルパス上の作業効率の向上を図る（図 4.7 参照）．

ここまでで，必要なデータの収集とクリティカルパスの導出を行った．更に，クリティカルパス上の作業に対してパラメータの設定を行い，これらの設定したパラメータから可能性測度を用いた確率最大化モデルを用いた式変形を行った．

次に，MPI と RaspberryPi を用いた並列分散処理の流れを説明する．解法は節 3.3 と節 3.4 で説明したものを用いる（図 4.8 参照）．

**Step4:** 等価変換した目的関数を並列分散 GA を用いて解く．まず，RaspberryPi3 のマスター (1 台) で初期集団を生成する．

**Step5:** マスターで生成した初期集団を各スレーブ (7 台) へ MPISend() を使い配布する．配布した初期集団を MPIRecv() を用いて各スレーブにて受け取る．

**Step6:** 各スレーブで遺伝子操作（交叉・突然変異・選択・移住）を行う．移住は一定の世代間隔でランダムに選んだ遺伝子を他のスレーブへ送信する．

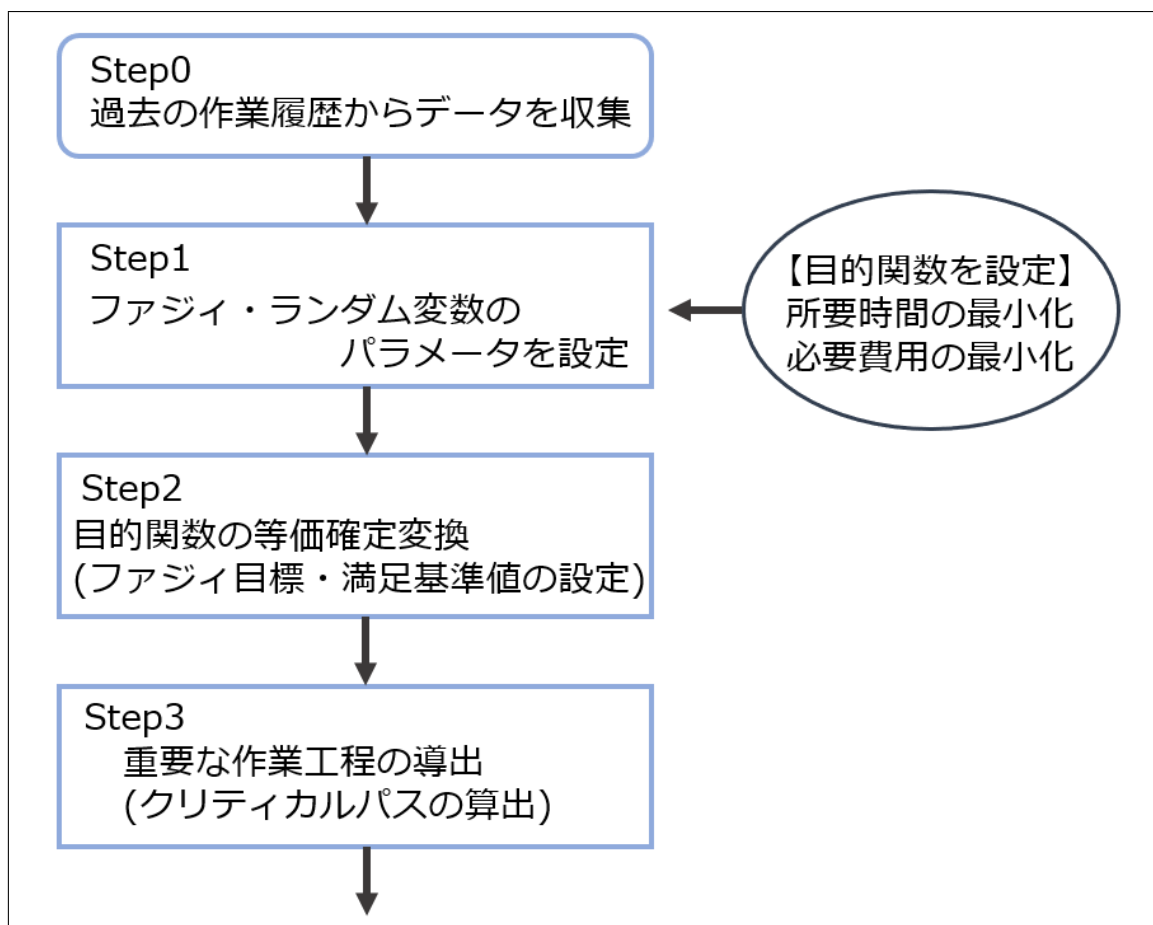


図 4.7: データ収集から式変換までのフロー

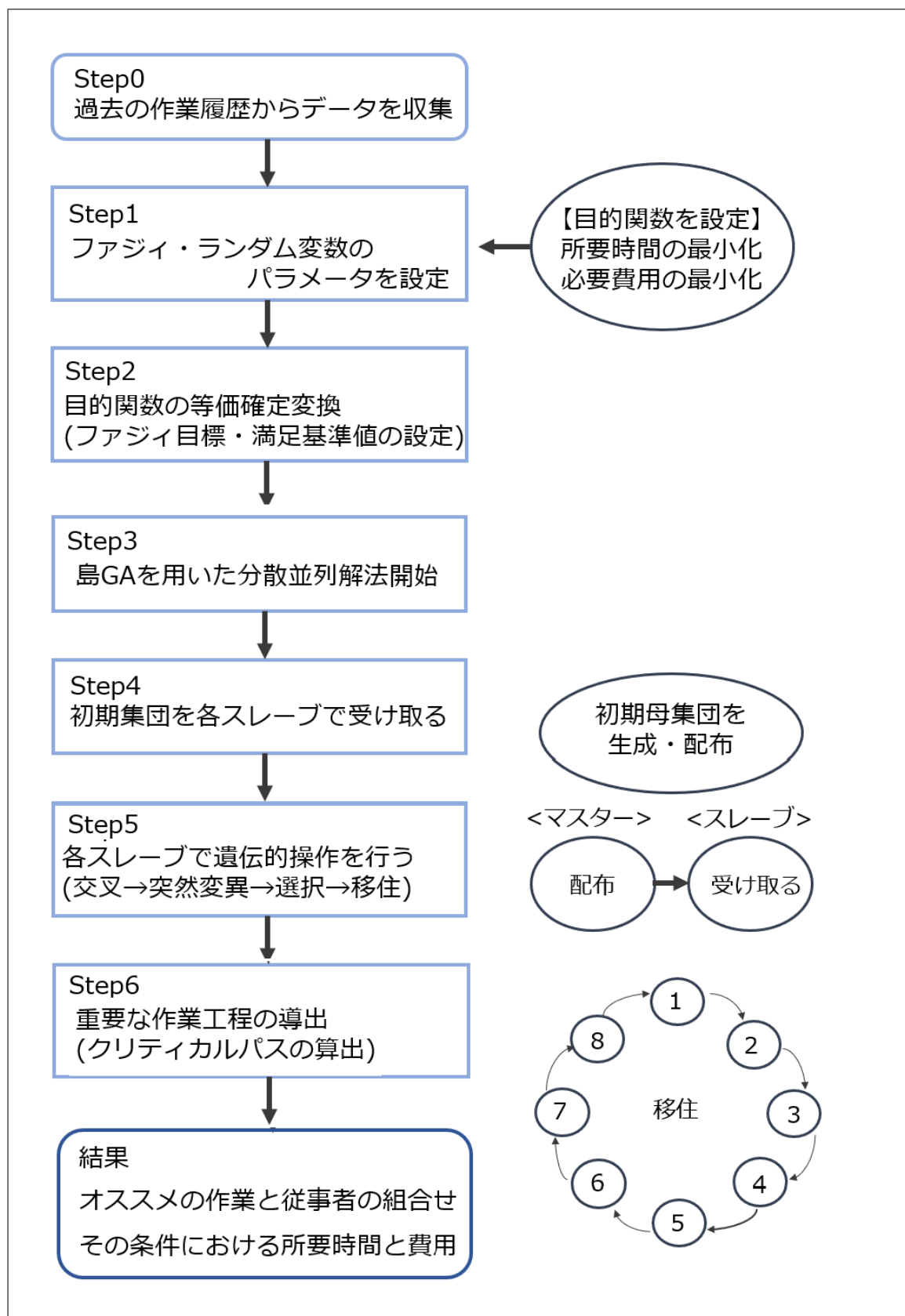


図 4.8: 提案手法の全体的なフロー

### 結論ならび今後の課題

本研究の目的は、建築業界における日程計画を考える際に、不確実性・不確定性を含む作業情報に着目し、過去の作業履歴を実際の現場で活用されている現場コミュニケーションアプリなどを利用し蓄積・分析できるとし、取得したデータからパラメータ設定ができるような日程計画問題の提案を行うことである。提案する問題に対しての定式化、データの設定方法から解法までのアルゴリズムを示し、高速化の環境構築までを行った。本研究で

提案したファジィ・ランダム日程計画問題の五つの特徴をまとめる。一つ目の特徴は、既存研究の多くは静的で確定的なものを前提としており、現実問題への活用が困難であったが、現場の作業情報の不確実性と不確定性の両方を表現するために、問題の目的関数の係数にファジィ・ランダム変数を導入した点。二つ目は、建築現場の悩みから、今の現場に求められる仕組みとされる、職人さんの最適な割り振りを補助できる点。三つ目は、時間費用関数によって表現される所要時間と費用のトレードオフの関係を、所要時間と費用の最小化の多目的最適化問題にすることで、両方のバランスを考えた日程計画を考えることができる点。四つ目は、プロジェクトにおけるクリティカルパス上の費用と所要時間の最小化を目的とすることで、全工程のうちどの作業と職人の組合せを優先して管理すべきかが分かる点。五つ目は、処理時間の高速化を目的とした並列分散環境の構築を行った点。結論として、建築現場に必要とされる、「気象などの不確実・不確定な要素を考慮した所要時間と費用の見積もり」と「職人さんの最適な選択・割り振りの補助」を行いつつ、今の現場環境からでも得られるデータの範囲内で、取得したデータからパラメータを設定が行えるファジィ・ランダム多目的日程計画を提案することができた。本研究の研究成果は、過去の作業

履歴から取得できるデータを、不確実で不確定な要素、本研究では作業にかかる費用を見積もるために活用するモデルの提案と定式化を行ったことである。提案したモデルは、建築現場に限らず、様々な業界における不確実で不確定な要素の見積もりや、最適な人員選択の補助が可能であると考えられる。

今後の課題として、まずは提案したファジィ・ランダム多目的日程計画を解き、従来のファジィ・ランダム変数を用いていないモデルとの比較・検証を行い提案モデルの有効性を示す必要がある。更に、本研究で提案したファジィ・ランダム多目的日程計画問題では、最

初にクリティカルパスを求め、そのクリティカルパスにかかる所要時間と費用が最小になるような職人さんの選択を行うことを考えた。しかし、クリティカルパス上の工程を管理することが、最も全体の作業効率の向上に効果的であるとされているが、現実問題で管理する工程がクリティカルパス上の作業だけでは実用するには不十分である。そこで、ファジィ・ランダム変数を導入したクリティカルパス上への作業へ優先的に最適な人員の割り当てを行いつつ、他の作業の人員も最低限確保できるような問題への改良も検討すべきである。更に、職人さんの最適な割り当てという部分に注視して考えると、今後の展望として、

本研究で提案したような一つの大きなプロジェクトの中の作業のみではなく、いくつかの大きなプロジェクトの作業に対する職人さんのファジィ・ランダム日程計画問題も考えられる。なぜならば、建築現場における職人さん達の多くは、深刻な人手不足により、いくつも工事現場を掛け持ちしているためである。本研究では、より現実問題に対応できるモデルにするため「作業費用の不確実性・不確定性の考慮」を考えたが、現実問題に対する日程計画問題には、前述したようなプロジェクトの掛け持ちのような複雑な制約を持つものも多く、まだ様々な方面からアプローチし、改善する余地があると考えられる。

# 謝辞

本研究を遂行するにあたり，多大なご指導と終始懇切丁寧なご鞭撻を賜った富山県立大学電子・情報工学科の奥原浩之教授に深甚な謝意を表します．最後になりましたが，多大な協力をして頂いた研究室の同輩諸氏に感謝致します．

2019 年 2 月

杉山 桃香

## 参考文献

- [1] “国内人口推移が2030年の「働く」にどのような影響を及ぼすか”, <https://www.recruit-ms.co.jp/research/2030/report/trend1.html>
- [2] H. Kwakernaak, “Fuzzy random variable-I,” *Information Sciences*, vol. 15, pp. 1-29, 1978.
- [3] M.L. Puri, and D.A. Ralescu, “Fuzzy random variables,” *Journal of Mathematical Analysis and Applications*, vol. 114, pp. 409-422, 1986.
- [4] 飯田耕司, “不確実性への挑戦 意思決定分析の理論”, 三恵社, 2006.
- [5] “PERT と CPM” ,<http://d-engineer.com/industrialeng/pert.html>
- [6] “クリティカルパスでプロジェクト遅延要因を以前に把握せよ” ,<https://pmkuma.com/critical-path-method/>
- [7] 木瀬 洋, “スケジューリング研究の過去・現在・未来”
- [8] “建築現場をIT化する「ダンドリ」ツール6選”, <https://mag.branu.jp/archives/2442>
- [9] “現場コミュニケーションアプリ Kizuku”, <https://www.ctx.co.jp/kizuku2pr/>
- [10] 吉川和広, 春名 攻, “ネットワーク手法による施工計画のシステムアプローチに関する研究-CPM 計算の簡便化-”
- [11] David L. Applegate, Robert E. Bixby, Vasek Chvatal, William J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton Univ Pr, 2007
- [12] M.Gen, R. Cheng, *Genetic Algorithms, Engineering Design*, John Wiley, Sons, 1997.
- [13] 玉置 久, 森 正勝, 荒木 光彦, “遺伝アルゴリズムを用いたパレート最適解集合の生成法”
- [14] “Raspberry Pi とは”, <https://furien.jp/columns/58/>
- [15] “MPIの環境構築や基本コマンドのまとめ” ,<https://qiita.com/kkk627/items/49c9c35301465f6780f>
- [16] “MPI を用いた並列プログラミングの概要” ,<http://www.cenav.org/raspi4b/>
- [17] “MPI 並列プログラミング” ,<http://www.matlab.nitech.ac.jp/kazu-k/mpi.html>
- [18] Hideki Katagiri , “Interactive multiobjective fuzzy random linear programming: Maximization of possibility and probability”
- [19] M. Sakawa, I. Nishizaki, H. Katagiri, *Fuzzy Stochastic Multiobjective Programming*, Springer, 2011
- [20] 椎名孝之, “確率計画法”, 朝倉書店, 2015.







# 付録

## A. 1 HellowWorld を並列実行するソースコード

RaspberryPi8 台で HellowWorld を並列実行するソースコード A.1 をしめす.

ソースコード A. 1: hellow.c

```
1 #include <stdio.h>
2 #include "mpi.h"
3 int main( int argc, char *argv[] )
4 {
5     int rank;
6     int size;
7     double t0,t1,t2,t_w;
8     MPI_Status stat;
9     MPI_Init(&argc,&argv );
10    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
11    MPI_Comm_size(MPI_COMM_WORLD, &size);
12    MPI_Barrier(MPI_COMM_WORLD);
13
14    t1=MPI_Wtime();
15    printf( "Hello world from process %d of %d\n", rank, size );
16    MPI_Barrier(MPI_COMM_WORLD);
17    t2=MPI_Wtime();
18    t0=t2-t1;
19    MPI_Reduce(&t0, &t_w, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
20    MPI_Finalize();
21    printf("%f\n",t_w);
22    return 0;
23 }
```

## A. 2 円周率計算の並列分散処理ソースコード

RaspberryPi8 台で円周率計算を並列分散処理するソースコード A.2 をしめす.

ソースコード A. 2: pi.c

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <mpi.h>
4 void main(int argc, char* argv[]){
5     double PI25DT = 3.141592653589793238462643;
6     double mypi, pi, h, sum, x, f, a;
7     double t1, t2, t0, t_w;
8     int n, myid, numprocs, i, rc;
9     int ierr;
10
11    MPI_Status stat;
```

```

12 MPI_Init(&argc, &argv);
13 MPI_Comm_rank(MPI_COMM_WORLD, &myid);
14 MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
15 if(myid==0){
16     printf("Enter the number of intervals \n");
17     scanf("%d", &n);
18     printf("n=%d \n", n);
19 }
20 MPI_Barrier(MPI_COMM_WORLD);
21 t1=MPI_Wtime();
22
23 MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
24 h = 1.0 / n;
25 sum = 0.0;
26 for(i=myid+1; i<=n; i+=numprocs){
27     x = h * (i - 0.5);
28     sum = sum + 4.0 / (1.0 + x*x);
29 }
30 mypi = h * sum;
31 MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
32 if(myid == 0){
33     printf("pi is approximately: %18.16lf Error is: %18.16lf \n", pi,fabs(pi-
        PI25DT));
34 }
35 MPI_Barrier(MPI_COMM_WORLD);
36 t2 = MPI_Wtime();
37 t0= t2-t1;
38 MPI_Reduce(&t0, &t_w, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
39
40 if(myid==0){
41     printf("execution time = : %8.4lf [sec.] \n", t_w);
42 }
43 rc= MPI_Finalize();
44
45 }

```

---

## A. 3 巡回セールスマン問題を島GAで並列分散処理するソースコード

RaspberryPi8 台で巡回セールスマン問題を島GAで並列分散処理するソースコード A.3 をしめす。

ソースコード A. 3: tspga.c

```

1 // #define DEFAULT_SOURCE
2 // #define BSD_SOURCE
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <stdlib.h> /* rand() */
6 #include <time.h> /* srand() */
7 #include "mpi.h"
8
9 #define NODE 50 /* 都市の数 */
10 #define MAX_DIST 30 /* 都市間の距離の最大値 */
11 #define GENOM 100 /* 遺伝子数 */
12 #define LIMIT 50000 /* 計算回数 */
13 #define EVOLUTION 1 /* 突然変異発生確率[%] */
14
15 #define MIG_INTERVAL 100 /* 移住間隔/世代 */
16 #define MIG_RATE 15 /* 移住率[%] */
17
18 #define DEBUG1 0 /* 関数確認用 */
19 #define DEBUG2 0 /* 経路が正しいかどうか確認 */
20 #define WRITEFILE 1 /* ファイルへの書き込み用の出力 */
21
22 typedef struct {
23     double dist[NODE]; /* town[now].dist[go] */
24     int flag; /* 訪問確認用旗 */
25 } TOWN;
26
27 TOWN town[NODE];
28
29 typedef struct {
30     double distance; /* 距離 */
31     int route[NODE+1]; /* 経路 */
32     int flag; /* 旗 */
33 } RESULT;
34
35 RESULT result[GENOM];
36
37 int champ = 0;
38 double fast = NODE * MAX_DIST;
39
40 void func(void); /* 母集団生成 */
41 void makelist(void); /* 都市間の距離リスト作成 */
42 void tournament(int rank); /* 淘汰 */
43 void calc_dist(RESULT *res); /* 距離計算 */
44 void cross(void); /* 交叉 */
45 void variation(int np); /* 突然変異 */
46 void migrate(int rank, int np, MPI_Status *status); /* 移住 */
47
48 /* main

```

```

*****
*/
49
50 int main(int argc, char **argv) {
51     int i, j, k, min, sec;
52     int start, end;
53     //double t1, t2, t0, t_w;
54     time_t t1, t2;
55     int rank, np; /* ランク、端末数 */
56
57     MPI_Status status;
58     MPI_Init(&argc, &argv);
59     MPI_Comm_rank(
60         MPI_COMM_WORLD, &rank);
61     MPI_Comm_size(
62         MPI_COMM_WORLD, &np);
63
64     double fast_rank[np], fast_all;
65     makelist();
66     int distribution[GENOM][NODE+1];
67
68     /* --- 親 (rank=0) --- */
69     if (rank == 0)
70     {
71         start = time(&t1); /* 開始時間取得 */
72         srand((unsigned)time(NULL));
73         func(); /* 母集団初期化 */
74
75         /* 母集団の配分 */
76         for (i = 1; i < np; i++) { //i: 端末(rank)
77             for (j = 0; j < (GENOM); j++) { //j: 遺伝子数
78                 for (k = 0; k < NODE+1; k++) { //k: 経路
79                     distribution[j][k] = result[j].route[k];
80                 }
81             }
82             MPI_Send(distribution, (GENOM)*(NODE+1),
83                     MPI_INT, i, 99, MPI_COMM_WORLD);
84         }
85
86         /* --- 子 (rank!=0) --- */
87         else
88         {
89             /* 母集団の受け取り */
90             MPI_Recv(distribution, (GENOM)*(NODE+1), MPI_INT, 0,
91                     99, MPI_COMM_WORLD, &status);
92             for (i = 0; i < (GENOM); i++)
93             { //i: 遺伝子数
94                 for (j = 0; j < NODE+1; j++)
95                 { //j: 経路
96                     result[i].route[j] =
97                         distribution[i][j];
98                 }
99             }
100         }
101     }

```

```

92     }
93     calc_dist(&result[i]); //距離計算
94 }
95 srand((unsigned)time(NULL));
96 }
97
98 /* ループ */
99 i = 0;
100 while (i < LIMIT) {
101     cross(); /* 交叉 */
102     tournament(rank);
103     if ((rand() % 100) < EVOLUTION) {
104         variation(np); /* 突然変異 */
105         tournament(rank); /* 淘汰 */
106     }
107     i++;
108
109     if (i % MIG_INTERVAL == 0) {
110         /* 結果表示 */
111         if (rank == 0) {
112             fast_all = fast;
113             for (j = 1; j < np; j++) {
114                 //printf("%d\t%6.2f\n", j,
115                     fast_all);
116                 MPI_Recv(&fast_rank[j],
117                     1, MPI_DOUBLE, j,
118                     50,
119                     MPI_COMM_WORLD,
120                     &status);
121                 if (fast_all > fast_rank[j])
122                     fast_all = fast_rank[j];
123
124                 //printf("%d\t%6.2f\n", j,
125                     fast_rank[j]);
126             }
127
128             #if WRITEFILE
129             end = time(&t2); /* 終了時間取得 */
130             min = end - start;
131             sec = min / 60;
132             min -= sec * 60;
133
134             printf("%d\t%6.2f\n", i,
135                 fast_all);
136             printf("exec time;;; %d:%02d\n",
137                 sec, min);
138             //printf("%6.2f\n", fast_all);
139         }
140
141         #endif
142     }
143     else {
144         MPI_Send(&fast, 1,
145             MPI_DOUBLE, 0, 50,
146             MPI_COMM_WORLD);
147     }
148     printf("%6.2f\n", fast);
149 }
150
151 void func() {
152     #if DEBUG1
153     puts("--func");
154
155     if (np-1) /* 0より大きいとき移住 */
156     {
157         MPI_Barrier(
158             MPI_COMM_WORLD);
159         migrate(rank, np, &status); /*
160             移住 */
161     }
162 } //loop end
163
164 get_char();get_char();
165 #if WRITEFILE
166 //nanosleep(1000*rank*100);
167 printf("%2d> %6.2f: ", rank, result[
168     champ].distance);
169 for (i = 0; i < NODE+1; i++) {
170     printf("%2d ", result[champ].route[
171         i]);
172 }
173 puts("");
174
175 MPI_Barrier(MPI_COMM_WORLD);
176 ;
177 #endif
178
179 if (rank == 0)
180 {
181     fast_all = fast;
182     for (i = 1; i < np; i++) {
183         MPI_Recv(&fast_rank[i], 1,
184             MPI_INT, i, 50,
185             MPI_COMM_WORLD, &
186             status);
187         if (fast_all > fast_rank[i])
188             fast_all = fast_rank[i];
189     }
190
191     printf("the shortest
192         distance: %.2f\n",
193         fast_all);
194 }
195
196 #if WRITEFILE
197 //nanosleep(1000*np*100);
198 // printf("exec time;;; %d:%02d
199     \n", sec, min);
200 #endif
201 }
202
203 else
204 {
205     MPI_Send(&fast, 1, MPI_INT, 0,
206         50, MPI_COMM_WORLD);
207 }
208
209 MPI_Finalize(); /* MPI終了 */
210 }
211
212 /* func
213     ****
214 */
215 void func() {
216     #if DEBUG1
217     puts("--func");
218
219     if (np-1) /* 0より大きいとき移住 */
220     {
221         MPI_Barrier(
222             MPI_COMM_WORLD);
223         migrate(rank, np, &status); /*
224             移住 */
225     }
226 } //loop end
227
228 get_char();get_char();
229 #if WRITEFILE
230 //nanosleep(1000*rank*100);
231 printf("%2d> %6.2f: ", rank, result[
232     champ].distance);
233 for (i = 0; i < NODE+1; i++) {
234     printf("%2d ", result[champ].route[
235         i]);
236 }
237 puts("");
238
239 MPI_Barrier(MPI_COMM_WORLD);
240 ;
241 #endif
242
243 if (rank == 0)
244 {
245     fast_all = fast;
246     for (i = 1; i < np; i++) {
247         MPI_Recv(&fast_rank[i], 1,
248             MPI_INT, i, 50,
249             MPI_COMM_WORLD, &
250             status);
251         if (fast_all > fast_rank[i])
252             fast_all = fast_rank[i];
253     }
254
255     printf("the shortest
256         distance: %.2f\n",
257         fast_all);
258 }
259
260 #if WRITEFILE
261 //nanosleep(1000*np*100);
262 // printf("exec time;;; %d:%02d
263     \n", sec, min);
264 #endif
265 }
266
267 else
268 {
269     MPI_Send(&fast, 1, MPI_INT, 0,
270         50, MPI_COMM_WORLD);
271 }
272
273 MPI_Finalize(); /* MPI終了 */
274 }
275
276 /* func
277     ****
278 */
279 void func() {
280     #if DEBUG1
281     puts("--func");
282
283     if (np-1) /* 0より大きいとき移住 */
284     {
285         MPI_Barrier(
286             MPI_COMM_WORLD);
287         migrate(rank, np, &status); /*
288             移住 */
289     }
290 } //loop end
291
292 get_char();get_char();
293 #if WRITEFILE
294 //nanosleep(1000*rank*100);
295 printf("%2d> %6.2f: ", rank, result[
296     champ].distance);
297 for (i = 0; i < NODE+1; i++) {
298     printf("%2d ", result[champ].route[
299         i]);
300 }
301 puts("");
302
303 MPI_Barrier(MPI_COMM_WORLD);
304 ;
305 #endif
306
307 if (rank == 0)
308 {
309     fast_all = fast;
310     for (i = 1; i < np; i++) {
311         MPI_Recv(&fast_rank[i], 1,
312             MPI_INT, i, 50,
313             MPI_COMM_WORLD, &
314             status);
315         if (fast_all > fast_rank[i])
316             fast_all = fast_rank[i];
317     }
318
319     printf("the shortest
320         distance: %.2f\n",
321         fast_all);
322 }
323
324 #if WRITEFILE
325 //nanosleep(1000*np*100);
326 // printf("exec time;;; %d:%02d
327     \n", sec, min);
328 #endif
329 }
330
331 else
332 {
333     MPI_Send(&fast, 1, MPI_INT, 0,
334         50, MPI_COMM_WORLD);
335 }
336
337 MPI_Finalize(); /* MPI終了 */
338 }
339
340 /* func
341     ****
342 */
343 void func() {
344     #if DEBUG1
345     puts("--func");
346
347     if (np-1) /* 0より大きいとき移住 */
348     {
349         MPI_Barrier(
350             MPI_COMM_WORLD);
351         migrate(rank, np, &status); /*
352             移住 */
353     }
354 } //loop end
355
356 get_char();get_char();
357 #if WRITEFILE
358 //nanosleep(1000*rank*100);
359 printf("%2d> %6.2f: ", rank, result[
360     champ].distance);
361 for (i = 0; i < NODE+1; i++) {
362     printf("%2d ", result[champ].route[
363         i]);
364 }
365 puts("");
366
367 MPI_Barrier(MPI_COMM_WORLD);
368 ;
369 #endif
370
371 if (rank == 0)
372 {
373     fast_all = fast;
374     for (i = 1; i < np; i++) {
375         MPI_Recv(&fast_rank[i], 1,
376             MPI_INT, i, 50,
377             MPI_COMM_WORLD, &
378             status);
379         if (fast_all > fast_rank[i])
380             fast_all = fast_rank[i];
381     }
382
383     printf("the shortest
384         distance: %.2f\n",
385         fast_all);
386 }
387
388 #if WRITEFILE
389 //nanosleep(1000*np*100);
390 // printf("exec time;;; %d:%02d
391     \n", sec, min);
392 #endif
393 }
394
395 else
396 {
397     MPI_Send(&fast, 1, MPI_INT, 0,
398         50, MPI_COMM_WORLD);
399 }
400
401 MPI_Finalize(); /* MPI終了 */
402 }
403
404 /* func
405     ****
406 */
407 void func() {
408     #if DEBUG1
409     puts("--func");
410
411     if (np-1) /* 0より大きいとき移住 */
412     {
413         MPI_Barrier(
414             MPI_COMM_WORLD);
415         migrate(rank, np, &status); /*
416             移住 */
417     }
418 } //loop end
419
420 get_char();get_char();
421 #if WRITEFILE
422 //nanosleep(1000*rank*100);
423 printf("%2d> %6.2f: ", rank, result[
424     champ].distance);
425 for (i = 0; i < NODE+1; i++) {
426     printf("%2d ", result[champ].route[
427         i]);
428 }
429 puts("");
430
431 MPI_Barrier(MPI_COMM_WORLD);
432 ;
433 #endif
434
435 if (rank == 0)
436 {
437     fast_all = fast;
438     for (i = 1; i < np; i++) {
439         MPI_Recv(&fast_rank[i], 1,
440             MPI_INT, i, 50,
441             MPI_COMM_WORLD, &
442             status);
443         if (fast_all > fast_rank[i])
444             fast_all = fast_rank[i];
445     }
446
447     printf("the shortest
448         distance: %.2f\n",
449         fast_all);
450 }
451
452 #if WRITEFILE
453 //nanosleep(1000*np*100);
454 // printf("exec time;;; %d:%02d
455     \n", sec, min);
456 #endif
457 }
458
459 else
460 {
461     MPI_Send(&fast, 1, MPI_INT, 0,
462         50, MPI_COMM_WORLD);
463 }
464
465 MPI_Finalize(); /* MPI終了 */
466 }
467
468 /* func
469     ****
470 */
471 void func() {
472     #if DEBUG1
473     puts("--func");
474
475     if (np-1) /* 0より大きいとき移住 */
476     {
477         MPI_Barrier(
478             MPI_COMM_WORLD);
479         migrate(rank, np, &status); /*
480             移住 */
481     }
482 } //loop end
483
484 get_char();get_char();
485 #if WRITEFILE
486 //nanosleep(1000*rank*100);
487 printf("%2d> %6.2f: ", rank, result[
488     champ].distance);
489 for (i = 0; i < NODE+1; i++) {
490     printf("%2d ", result[champ].route[
491         i]);
492 }
493 puts("");
494
495 MPI_Barrier(MPI_COMM_WORLD);
496 ;
497 #endif
498
499 if (rank == 0)
500 {
501     fast_all = fast;
502     for (i = 1; i < np; i++) {
503         MPI_Recv(&fast_rank[i], 1,
504             MPI_INT, i, 50,
505             MPI_COMM_WORLD, &
506             status);
507         if (fast_all > fast_rank[i])
508             fast_all = fast_rank[i];
509     }
510
511     printf("the shortest
512         distance: %.2f\n",
513         fast_all);
514 }
515
516 #if WRITEFILE
517 //nanosleep(1000*np*100);
518 // printf("exec time;;; %d:%02d
519     \n", sec, min);
520 #endif
521 }
522
523 else
524 {
525     MPI_Send(&fast, 1, MPI_INT, 0,
526         50, MPI_COMM_WORLD);
527 }
528
529 MPI_Finalize(); /* MPI終了 */
530 }
531
532 /* func
533     ****
534 */
535 void func() {
536     #if DEBUG1
537     puts("--func");
538
539     if (np-1) /* 0より大きいとき移住 */
540     {
541         MPI_Barrier(
542             MPI_COMM_WORLD);
543         migrate(rank, np, &status); /*
544             移住 */
545     }
546 } //loop end
547
548 get_char();get_char();
549 #if WRITEFILE
550 //nanosleep(1000*rank*100);
551 printf("%2d> %6.2f: ", rank, result[
552     champ].distance);
553 for (i = 0; i < NODE+1; i++) {
554     printf("%2d ", result[champ].route[
555         i]);
556 }
557 puts("");
558
559 MPI_Barrier(MPI_COMM_WORLD);
560 ;
561 #endif
562
563 if (rank == 0)
564 {
565     fast_all = fast;
566     for (i = 1; i < np; i++) {
567         MPI_Recv(&fast_rank[i], 1,
568             MPI_INT, i, 50,
569             MPI_COMM_WORLD, &
570             status);
571         if (fast_all > fast_rank[i])
572             fast_all = fast_rank[i];
573     }
574
575     printf("the shortest
576         distance: %.2f\n",
577         fast_all);
578 }
579
580 #if WRITEFILE
581 //nanosleep(1000*np*100);
582 // printf("exec time;;; %d:%02d
583     \n", sec, min);
584 #endif
585 }
586
587 else
588 {
589     MPI_Send(&fast, 1, MPI_INT, 0,
590         50, MPI_COMM_WORLD);
591 }
592
593 MPI_Finalize(); /* MPI終了 */
594 }
595
596 /* func
597     ****
598 */
599 void func() {
600     #if DEBUG1
601     puts("--func");
602
603     if (np-1) /* 0より大きいとき移住 */
604     {
605         MPI_Barrier(
606             MPI_COMM_WORLD);
607         migrate(rank, np, &status); /*
608             移住 */
609     }
610 } //loop end
611
612 get_char();get_char();
613 #if WRITEFILE
614 //nanosleep(1000*rank*100);
615 printf("%2d> %6.2f: ", rank, result[
616     champ].distance);
617 for (i = 0; i < NODE+1; i++) {
618     printf("%2d ", result[champ].route[
619         i]);
620 }
621 puts("");
622
623 MPI_Barrier(MPI_COMM_WORLD);
624 ;
625 #endif
626
627 if (rank == 0)
628 {
629     fast_all = fast;
630     for (i = 1; i < np; i++) {
631         MPI_Recv(&fast_rank[i], 1,
632             MPI_INT, i, 50,
633             MPI_COMM_WORLD, &
634             status);
635         if (fast_all > fast_rank[i])
636             fast_all = fast_rank[i];
637     }
638
639     printf("the shortest
640         distance: %.2f\n",
641         fast_all);
642 }
643
644 #if WRITEFILE
645 //nanosleep(1000*np*100);
646 // printf("exec time;;; %d:%02d
647     \n", sec, min);
648 #endif
649 }
650
651 else
652 {
653     MPI_Send(&fast, 1, MPI_INT, 0,
654         50, MPI_COMM_WORLD);
655 }
656
657 MPI_Finalize(); /* MPI終了 */
658 }
659
660 /* func
661     ****
662 */
663 void func() {
664     #if DEBUG1
665     puts("--func");
666
667     if (np-1) /* 0より大きいとき移住 */
668     {
669         MPI_Barrier(
670             MPI_COMM_WORLD);
671         migrate(rank, np, &status); /*
672             移住 */
673     }
674 } //loop end
675
676 get_char();get_char();
677 #if WRITEFILE
678 //nanosleep(1000*rank*100);
679 printf("%2d> %6.2f: ", rank, result[
680     champ].distance);
681 for (i = 0; i < NODE+1; i++) {
682     printf("%2d ", result[champ].route[
683         i]);
684 }
685 puts("");
686
687 MPI_Barrier(MPI_COMM_WORLD);
688 ;
689 #endif
690
691 if (rank == 0)
692 {
693     fast_all = fast;
694     for (i = 1; i < np; i++) {
695         MPI_Recv(&fast_rank[i], 1,
696             MPI_INT, i, 50,
697             MPI_COMM_WORLD, &
698             status);
699         if (fast_all > fast_rank[i])
700             fast_all = fast_rank[i];
701     }
702
703     printf("the shortest
704         distance: %.2f\n",
705         fast_all);
706 }
707
708 #if WRITEFILE
709 //nanosleep(1000*np*100);
710 // printf("exec time;;; %d:%02d
711     \n", sec, min);
712 #endif
713 }
714
715 else
716 {
717     MPI_Send(&fast, 1, MPI_INT, 0,
718         50, MPI_COMM_WORLD);
719 }
720
721 MPI_Finalize(); /* MPI終了 */
722 }
723
724 /* func
725     ****
726 */
727 void func() {
728     #if DEBUG1
729     puts("--func");
730
731     if (np-1) /* 0より大きいとき移住 */
732     {
733         MPI_Barrier(
734             MPI_COMM_WORLD);
735         migrate(rank, np, &status); /*
736             移住 */
737     }
738 } //loop end
739
740 get_char();get_char();
741 #if WRITEFILE
742 //nanosleep(1000*rank*100);
743 printf("%2d> %6.2f: ", rank, result[
744     champ].distance);
745 for (i = 0; i < NODE+1; i++) {
746     printf("%2d ", result[champ].route[
747         i]);
748 }
749 puts("");
750
751 MPI_Barrier(MPI_COMM_WORLD);
752 ;
753 #endif
754
755 if (rank == 0)
756 {
757     fast_all = fast;
758     for (i = 1; i < np; i++) {
759         MPI_Recv(&fast_rank[i], 1,
760             MPI_INT, i, 50,
761             MPI_COMM_WORLD, &
762             status);
763         if (fast_all > fast_rank[i])
764             fast_all = fast_rank[i];
765     }
766
767     printf("the shortest
768         distance: %.2f\n",
769         fast_all);
770 }
771
772 #if WRITEFILE
773 //nanosleep(1000*np*100);
774 // printf("exec time;;; %d:%02d
775     \n", sec, min);
776 #endif
777 }
778
779 else
780 {
781     MPI_Send(&fast, 1, MPI_INT, 0,
782         50, MPI_COMM_WORLD);
783 }
784
785 MPI_Finalize(); /* MPI終了 */
786 }
787
788 /* func
789     ****
790 */
791 void func() {
792     #if DEBUG1
793     puts("--func");
794
795     if (np-1) /* 0より大きいとき移住 */
796     {
797         MPI_Barrier(
798             MPI_COMM_WORLD);
799         migrate(rank, np, &status); /*
800             移住 */
801     }
802 } //loop end
803
804 get_char();get_char();
805 #if WRITEFILE
806 //nanosleep(1000*rank*100);
807 printf("%2d> %6.2f: ", rank, result[
808     champ].distance);
809 for (i = 0; i < NODE+1; i++) {
810     printf("%2d ", result[champ].route[
811         i]);
812 }
813 puts("");
814
815 MPI_Barrier(MPI_COMM_WORLD);
816 ;
817 #endif
818
819 if (rank == 0)
820 {
821     fast_all = fast;
822     for (i = 1; i < np; i++) {
823         MPI_Recv(&fast_rank[i], 1,
824             MPI_INT, i, 50,
825             MPI_COMM_WORLD, &
826             status);
827         if (fast_all > fast_rank[i])
828             fast_all = fast_rank[i];
829     }
830
831     printf("the shortest
832         distance: %.2f\n",
833         fast_all);
834 }
835
836 #if WRITEFILE
837 //nanosleep(1000*np*100);
838 // printf("exec time;;; %d:%02d
839     \n", sec, min);
840 #endif
841 }
842
843 else
844 {
845     MPI_Send(&fast, 1, MPI_INT, 0,
846         50, MPI_COMM_WORLD);
847 }
848
849 MPI_Finalize(); /* MPI終了 */
850 }
851
852 /* func
853     ****
854 */
855 void func() {
856     #if DEBUG1
857     puts("--func");
858
859     if (np-1) /* 0より大きいとき移住 */
860     {
861         MPI_Barrier(
862             MPI_COMM_WORLD);
863         migrate(rank, np, &status); /*
864             移住 */
865     }
866 } //loop end
867
868 get_char();get_char();
869 #if WRITEFILE
870 //nanosleep(1000*rank*100);
871 printf("%2d> %6.2f: ", rank, result[
872     champ].distance);
873 for (i = 0; i < NODE+1; i++) {
874     printf("%2d ", result[champ].route[
875         i]);
876 }
877 puts("");
878
879 MPI_Barrier(MPI_COMM_WORLD);
880 ;
881 #endif
882
883 if (rank == 0)
884 {
885     fast_all = fast;
886     for (i = 1; i < np; i++) {
887         MPI_Recv(&fast_rank[i], 1,
888             MPI_INT, i, 50,
889             MPI_COMM_WORLD, &
890             status);
891         if (fast_all > fast_rank[i])
892             fast_all = fast_rank[i];
893     }
894
895     printf("the shortest
896         distance: %.2f\n",
897         fast_all);
898 }
899
900 #if WRITEFILE
901 //nanosleep(1000*np*100);
902 // printf("exec time;;; %d:%02d
903     \n", sec, min);
904 #endif
905 }
906
907 else
908 {
909     MPI_Send(&fast, 1, MPI_INT, 0,
910         50, MPI_COMM_WORLD);
911 }
912
913 MPI_Finalize(); /* MPI終了 */
914 }
915
916 /* func
917     ****
918 */
919 void func() {
920     #if DEBUG1
921     puts("--func");
922
923     if (np-1) /* 0より大きいとき移住 */
924     {
925         MPI_Barrier(
926             MPI_COMM_WORLD);
927         migrate(rank, np, &status); /*
928             移住 */
929     }
930 } //loop end
931
932 get_char();get_char();
933 #if WRITEFILE
934 //nanosleep(1000*rank*100);
935 printf("%2d> %6.2f: ", rank, result[
936     champ].distance);
937 for (i = 0; i < NODE+1; i++) {
938     printf("%2d ", result[champ].route[
939         i]);
940 }
941 puts("");
942
943 MPI_Barrier(MPI_COMM_WORLD);
944 ;
945 #endif
946
947 if (rank == 0)
948 {
949     fast_all = fast;
950     for (i = 1; i < np; i++) {
951         MPI_Recv(&fast_rank[i], 1,
952             MPI_INT, i, 50,
953             MPI_COMM_WORLD, &
954             status);
955         if (fast_all > fast_rank[i])
956             fast_all = fast_rank[i];
957     }
958
959     printf("the shortest
960         distance: %.2f\n",
961         fast_all);
962 }
963
964 #if WRITEFILE
965 //nanosleep(1000*np*100);
966 // printf("exec time;;; %d:%02d
967     \n", sec, min);
968 #endif
969 }
970
971 else
972 {
973     MPI_Send(&fast, 1, MPI_INT, 0,
974         50, MPI_COMM_WORLD);
975 }
976
977 MPI_Finalize(); /* MPI終了 */
978 }
979
980 /* func
981     ****
982 */
983 void func() {
984     #if DEBUG1
985     puts("--func");
986
987     if (np-1) /* 0より大きいとき移住 */
988     {
989         MPI_Barrier(
990             MPI_COMM_WORLD);
991         migrate(rank, np, &status); /*
992             移住 */
993     }
994 } //loop end
995
996 get_char();get_char();
997 #if WRITEFILE
998 //nanosleep(1000*rank*100);
999 printf("%2d> %6.2f: ", rank, result[
1000     champ].distance);
1001 for (i = 0; i < NODE+1; i++) {
1002     printf("%2d ", result[champ].route[
1003         i]);
1004 }
1005 puts("");
1006
1007 MPI_Barrier(MPI_COMM_WORLD);
1008 ;
1009 #endif
1010
1011 if (rank == 0)
1012 {
1013     fast_all = fast;
1014     for (i = 1; i < np; i++) {
1015         MPI_Recv(&fast_rank[i], 1,
1016             MPI_INT, i, 50,
1017             MPI_COMM_WORLD, &
1018             status);
1019         if (fast_all > fast_rank[i])
1020             fast_all = fast_rank[i];
1021     }
1022
1023     printf("the shortest
1024         distance: %.2f\n",
1025         fast_all);
1026 }
1027
1028 #if WRITEFILE
1029 //nanosleep(1000*np*100);
1030 // printf("exec time;;; %d:%02d
1031     \n", sec, min);
1032 #endif
1033 }
1034
1035 else
1036 {
1037     MPI_Send(&fast, 1, MPI_INT, 0,
1038         50, MPI_COMM_WORLD);
1039 }
1040
1041 MPI_Finalize(); /* MPI終了 */
1042 }
1043
1044 /* func
1045     ****
1046 */
1047 void func() {
1048     #if DEBUG1
1049     puts("--func");
1050
1051     if (np-1) /* 0より大きいとき移住 */
1052     {
1053         MPI_Barrier(
1054             MPI_COMM_WORLD);
1055         migrate(rank, np, &status); /*
1056             移住 */
1057     }
1058 } //loop end
1059
1060 get_char();get_char();
1061 #if WRITEFILE
1062 //nanosleep(1000*rank*100);
1063 printf("%2d> %6.2f: ", rank, result[
1064     champ].distance);
1065 for (i = 0; i < NODE+1; i++) {
1066     printf("%2d ", result[champ].route[
1067         i]);
1068 }
1069 puts("");
1070
1071 MPI_Barrier(MPI_COMM_WORLD);
1072 ;
1073 #endif
1074
1075 if (rank == 0)
1076 {
1077     fast_all = fast;
1078     for (i = 1; i < np; i++) {
1079         MPI_Recv(&fast_rank[i], 1,
1080             MPI_INT, i, 50,
1081             MPI_COMM_WORLD, &
1082             status);
1083         if (fast_all > fast_rank[i])
1084             fast_all = fast_rank[i];
1085     }
1086
1087     printf("the shortest
1088         distance: %.2f\n",
1089         fast_all);
1090 }
1091
1092 #if WRITEFILE
1093 //nanosleep(1000*np*100);
1094 // printf("exec time;;; %d:%02d
1095     \n", sec, min);
1096 #endif
1097 }
1098
1099 else
1100 {
1101     MPI_Send(&fast, 1, MPI_INT, 0,
1102         50, MPI_COMM_WORLD);
1103 }
1104
1105 MPI_Finalize(); /* MPI終了 */
1106 }
1107
1108 /* func
1109     ****
1110 */
1111 void func() {
1112     #if DEBUG1
1113     puts("--func");
1114
1115     if (np-1) /* 0より大きいとき移住 */
1116     {
1117         MPI_Barrier(
1118             MPI_COMM_WORLD);
1119         migrate(rank, np, &status); /*
1120             移住 */
1121     }
1122 } //loop end
1123
1124 get_char();get_char();
1125 #if WRITEFILE
1126 //nanosleep(1000*rank*100);
1127 printf("%2d> %6.2f: ", rank, result[
1128     champ].distance);
1129 for (i = 0; i < NODE+1; i++) {
1130     printf("%2d ", result[champ].route[
1131         i]);
1132 }
1133 puts("");
1134
1135 MPI_Barrier(MPI_COMM_WORLD);
1136 ;
1137 #endif
1138
1139 if (rank == 0)
1140 {
1141     fast_all = fast;
1142     for (i = 1; i < np; i++) {
1143         MPI_Recv(&fast_rank[i], 1,
1144             MPI_INT, i, 50,
1145             MPI_COMM_WORLD, &
1146             status);
1147         if (fast_all > fast_rank[i])
1148             fast_all = fast_rank[i];
1149     }
1150
1151     printf("the shortest
1152         distance: %.2f\n",
1153         fast_all);
1154 }
1155
1156 #if WRITEFILE
1157 //nanosleep(1000*np*100);
1158 // printf("exec time;;; %d:%02d
1159     \n", sec, min);
1160 #endif
1161 }
1162
1163 else
1164 {
1165     MPI_Send(&fast, 1, MPI_INT, 0,
1166         50, MPI_COMM_WORLD);
1167 }
1168
1169 MPI_Finalize(); /* MPI終了 */
1170 }
1171
1172 /* func
1173     ****
1174 */
1175 void func() {
1176     #if DEBUG1
1177     puts("--func");
1178
1179     if (np-1) /* 0より大きいとき移住 */
1180     {
1181         MPI_Barrier(
1182             MPI_COMM_WORLD);
1183         migrate(rank, np, &status); /*
1184             移住 */
1185     }
1186 } //loop end
1187
1188 get_char();get_char();
1189 #if WRITEFILE
1190 //nanosleep(1000*rank*100);
1191 printf("%2d> %6.2f: ", rank, result[
1192     champ].distance);
1193 for (i = 0; i < NODE+1; i++) {
1194     printf("%2d ", result[champ].route[
1195         i]);
1196 }
1197 puts("");
1198
1199 MPI_Barrier(MPI_COMM_WORLD);
1200 ;
1201 #endif
1202
1203 if (rank == 0)
1204 {
1205     fast_all = fast;
1206     for (i = 1; i < np; i++) {
1207         MPI_Recv(&fast_rank[i], 1,
1208             MPI_INT, i, 50,
1209             MPI_COMM_WORLD, &
1210             status);
1211         if (fast_all > fast_rank[i])
1212             fast_all = fast_rank[i];
1213     }
1214
1215     printf("the shortest
1216         distance: %.2f\n",
1217         fast_all);
1218 }
1219
1220 #if WRITEFILE
1221 //nanosleep(1000*np*100);
1222 // printf("exec time;;; %d:%02d
1223     \n", sec, min);
1224 #endif
1225 }
1226
1227 else
1228 {
1229     MPI_Send(&fast, 1, MPI_INT, 0,
1230         50, MPI_COMM_WORLD);
1231 }
1232
1233 MPI_Finalize(); /* MPI終了 */
1234 }
1235
1236 /* func
1237     ****
1238 */
1239 void func() {
1240     #if DEBUG1
1241     puts("--func");
1242
1243     if (np-1) /* 0より大きいとき移住 */
1244     {
1245         MPI_Barrier(
1246             MPI_COMM_WORLD);
1247         migrate(rank, np, &status); /*
1248             移住 */
1249     }
1250 } //loop end
1251
1252 get_char();get_char();
1253 #if WRITEFILE
1254 //nanosleep(1000*rank*100);
1255 printf("%2d> %6.2f: ", rank, result[
1256     champ].distance);
1257 for (i = 0; i < NODE+1; i++) {
1258     printf("%2d ", result[champ].route[
1259         i]);
1260 }
1261 puts("");
1262
1263 MPI_Barrier(MPI_COMM_WORLD);
1264 ;
1265 #endif
1266
1267 if (rank == 0)
1268 {
1269     fast_all = fast;
1270     for (i = 1; i < np; i++) {
1271         MPI_Recv(&fast_rank[i], 1,
1272             MPI_INT, i, 50,
1273             MPI_COMM_WORLD, &
1274             status);
1275         if (fast_all > fast_rank[i])
1276             fast_all = fast_rank[i];
1277     }
1278
1279     printf("the shortest
1280         distance: %.2f\n",
1281         fast_all);
1282 }
1283
1284 #if WRITEFILE
1285 //nanosleep(1000*np*100);
1286 // printf("exec time;;; %d:%02d
1287     \n", sec, min);
1288 #endif
1289 }
1290
1291 else
1292 {
1293     MPI_Send(&fast, 1, MPI_INT, 0,
1294         50, MPI_COMM_WORLD);
1295 }
1296
1297 MPI_Finalize(); /* MPI終了 */
1298 }
1299
1300 /* func
1301     ****
1302 */
1303 void func() {
1304     #if DEBUG1
1305     puts("--func");
1306
1307     if (np-1) /* 0より大きいとき移住 */
1308     {
1309         MPI_Barrier(
1310             MPI_COMM_WORLD);
1311         migrate(rank, np, &status); /*
1312             移住 */
1313     }
1314 } //loop end
1315
1316 get_char();get_char();
1317 #if WRITEFILE
1318 //nanosleep(1000*rank*100);
1319 printf("%2d> %6.2f: ", rank, result[
1320     champ].distance);
1321 for (i = 0; i < NODE+1; i++) {
1322     printf("%2d ", result[champ].route[
1323         i]);
1324 }
1325 puts("");
1326
1327 MPI_Barrier(MPI_COMM_WORLD);
1328 ;
1329 #endif
1330
1331 if (rank == 0)
1332 {
1333     fast_all = fast;
1334     for (i = 1; i < np; i++) {
1335         MPI_Recv(&fast_rank[i], 1,
1336             MPI_INT, i, 50,
1337             MPI_COMM_WORLD, &
1338             status);
1339         if (fast_all > fast_rank[i])
1340             fast_all = fast_rank[i];
1341     }
1342
1343     printf("the shortest
1344         distance: %.2f\n",
1345         fast_all);
1346 }
1347
1348 #if WRITEFILE
1349 //nanosleep(1000*np*100);
1350 // printf("exec time;;;
```

```

186 #endif
187 int i, j, go;
188
189 for (i = 0; i < GENOM; i++)
190 {
191     result[i].distance = 0;
192     result[i].route[0] = 0;
193     result[i].route[NODE] = 0;
194
195     for (j = 1; j < NODE; j++) {
196         town[j].flag = 0;
197     }
198     town[0].flag = 1;
199
200     for (j = 1; j < NODE; j++) {
201         do {
202             go = rand() % (NODE-1) +
203                 1;
204             while (town[go].flag); //do-
205                 while end
206             result[i].route[j] = go;
207             town[go].flag++;
208         } //for(j) end
209         calc_dist(&result[i]); /* 距離計算
210             */
211     } //for(i) end
212
213 #if DEBUG1
214     puts("---func end");
215 #endif
216 }
217
218 /* calc_dist
219     ****
220     */
221 void calc_dist(RESULT *res) {
222     int i, now, go;
223
224     /* 経路が正しいかどうか確認 */
225 #if DEBUG2
226     double judge = 0, dist_judge = 0;
227     for (i = 1; i < NODE; i++) {
228         judge += res->route[i];
229         dist_judge += i;
230     }
231     if (judge != dist_judge) {
232         puts("route error!");
233         exit(1);
234     }
235 #endif
236
237     res->distance = 0;
238     now = 0;
239     for (i = 1; i < NODE+1; i++) {
240         go = res->route[i];
241         res->distance += town[now].dist[
242             go];
243         now = go;
244     }
245 }

```

```

241 /* tournament
242     ****
243     */
244 void tournament(int rank) {
245     #if DEBUG1
246         printf("--tournament %d\n", rank);
247     #endif
248     int i, n, x;
249
250     for (i = 0; i < GENOM/2; i++)
251     {
252         n = i;
253         x = GENOM-1-i;
254         if (result[n].distance > result[x].
255             distance) {
256             n = x;
257             result[i] = result[n]; /* 代入 */
258         }
259         if (result[n].distance < result[
260             champ].distance)
261             champ = n;
262     }
263
264     if (fast > result[champ].distance){
265         fast = result[champ].distance;
266     }
267 #if WRITEFILE==0
268     /* 記録表示 */
269     printf("%2d> %6.2f: ", rank, result
270         [champ].distance);
271     for (i = 0; i < NODE+1; i++) {
272         printf("%2d ", result[champ].
273             route[i]);
274     }
275     puts("\n");
276 #endif
277 }
278
279 #if DEBUG1
280     printf("--tournament %d end\n",
281         rank);
282 #endif
283 }
284
285 /* cross
286     ****
287     */
288 /* 部分交叉 */
289 void cross() {
290     #if DEBUG1
291         puts("--cross");
292     #endif
293     int i, j, k, l, a, b, x;
294     int tmp1, tmp2;
295     int flag1[NODE], flag2[NODE]; /*
296         parent1, parent2用旗 */
297     RESULT parent1, parent2;
298
299     for (i = 0; i < GENOM/2; i+=2)
300     {
301         /* 交叉するペアを選択 */
302         a = rand() % (GENOM/2);

```

```

292 do {
293     b = rand() % (GENOM/2);
294 } while (a == b);
295 parent1 = result[a];
296 parent2 = result[b];
297
298 /* 旗初期化 */
299 for (j = 0; j < NODE; j+=2) {
300     flag1[j] = 1; flag1[j+1] = 1;
301     flag2[j] = 1; flag2[j+1] = 1;
302 }
303
304 /* 距離の短いルートは交叉しない
305     ようにする */
306 for (j = 0; j < NODE; j++) {
307     if (town[parent1.route[j]].dist[
308         parent1.route[j+1]] < 2) {
309         flag1[j] = 0; flag1[j+1] = 0;
310     }
311     if (town[parent2.route[j]].dist[
312         parent2.route[j+1]] < 2) {
313         flag2[j] = 0; flag2[j+1] = 0;
314     }
315 }
316
317 /* 交叉 */
318 x = rand() % (NODE-1) + 1; /*
319     切れ目 */
320 for (j = x; j < NODE; j++) {
321     tmp1 = parent1.route[j];
322     tmp2 = parent2.route[j];
323     if (flag1[j] && flag2[j]) {
324         for (k = 1; k < NODE; k
325             ++){
326             if ((parent1.route[k] ==
327                 tmp2) && k!=j &&
328                 flag1[k]) {
329                 parent1.route[j] = tmp2;
330                 flag1[j] = 0;
331                 parent1.route[k] = tmp1;
332                 flag1[k] = 0;
333             }
334             if ((parent2.route[k] ==
335                 tmp1) && k!=j &&
336                 flag2[k]) {
337                 parent2.route[j] = tmp1;
338                 flag2[j] = 0;
339                 parent2.route[k] = tmp2;
340                 flag2[k] = 0;
341             }
342         } //for(k) end
343     } //if(j) end
344 } //for(j) end
345
346 /* 距離計算 */
347 calc_dist(&parent1);
348 calc_dist(&parent2);
349 l = GENOM/2+i;
350 result[l] = parent1;
351 result[l+1] = parent2;
352 } //for(i) end
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```



```

394     result[GENOM-i] = parent;
395 } //for(i) end
396
397 #if DEBUG1
398 puts("--variaion end");
399 #endif
400 }
401
402 /* migrate
    *****[i][j]*****
    */
403 /* 移住 */
404 void migrate(int rank, int np,
    MPI_Status *status) {
405 #if DEBUG1
406 printf("%d migrate\n", rank);
407 #endif
408 int i, j, b, x;
409 int flag[GENOM] = {0};
410 int a; /* 遺伝子数 x MIG_RATE[%]
    */
411 int immigrant[a][NODE+1];
412 a = GENOM*MIG_RATE/100;
413
414 /* 移住者を決める */
415 for (i = 0; i < a; i++) {
416     do {
417         x = rand() % GENOM;
418     } while (flag[x]);
419     flag[x] = 1;
420     for (j = 0; j < NODE+1; j++) {
421         immigrant[i][j] = result[x].route[j];
422     }
423 }
424 b = a * (NODE+1); /* 送信するデー
    タの数 */
425 /* 偶数rank 送信->受信 */
426 if (rank%2 == 0)
427 {
428     /* 送信 */
429     MPI_Send(immigrant, b,
    MPI_INT, rank+1,
430             10,
    MPI_COMM_WORLD
    );
431     /* 受信 */
432     if (rank == 0) {
433         MPI_Recv(immigrant, b,
    MPI_INT, np-1,
434                 20,
    MPI_COMM_WORLD
    , status);
435     }
436     else {
437         MPI_Recv(immigrant, b,
    MPI_INT, rank-1,
438                 20,
    MPI_COMM_WORLD
    , status);
439     }
440     /* 受信データ格納 */
441     for (i = 0; i < a; i++) {
442         do {
443             x = rand()%(GENOM/2)+(
    GENOM/2);
444         } while (flag[x]);
445         flag[x] = 1;
446         for (j = 0; j < NODE+1; j++)
447             result[x].route[j] = immigrant
    [i][j];
448     }
449     calc_dist(&result[x]);
450 }
451 }
452 /* 奇数rank 受信->送信 */
453 else
454 {
455     /* 受信 */
456     MPI_Recv(immigrant, b,
    MPI_INT, rank-1,
457             10,
    MPI_COMM_WORLD
    , status);
458     /* 受信データ格納 */
459     for (i = 0; i < a; i++) {
460         do {
461             x = rand()%(GENOM/2)
    +(GENOM/2);
462         } while (flag[x]);
463         flag[x] = 1;
464         for (j = 0; j < NODE+1; j
    ++){
465             result[x].route[j] =
    immigrant[i][j];
466         }
467         calc_dist(&result[x]);
468     }
469     /* 送信 */
470     if (rank == np-1) {
471         MPI_Send(immigrant, b,
    MPI_INT, 0,
472                 20,
    MPI_COMM_WORLD
    );
473     }
474     else {
475         MPI_Send(immigrant, b,
    MPI_INT, rank+1,
476                 20,
    MPI_COMM_WORLD
    );
477     }
478 }
479 /* 移住者を決める */
480 for (i = 0; i < a; i++) {
481     do {
482         x = rand() % GENOM;
483     } while (flag[x]);
484     flag[x] = 1;
485     for (j = 0; j < NODE+1; j++) {
486         immigrant[i][j] = result[x].route[j];

```

```

487     }
488 }
489 /* 偶数rank 受信->送信 */
490 if (rank%2 == 0)
491 {
492     /* 受信 */
493     MPI_Recv(immigrant, b,
494             MPI_INT, rank+1,
495             30,
496             MPI_COMM_WORLD,
497             status);
498     /* 受信データ格納 */
499     for (i = 0; i < a; i++) {
500         do {
501             x = rand()%(GENOM/2)+(
502                 GENOM/2);
503             } while (flag[x]);
504             flag[x] = 1;
505             for (j = 0; j < NODE+1; j++)
506             {
507                 result[x].route[j] = immigrant
508                     [i][j];
509             }
510             calc_dist(&result[x]);
511         }
512         /* 送信 */
513         if (rank == 0) {
514             MPI_Send(immigrant, b,
515                     MPI_INT, np-1,
516                     40,
517                     MPI_COMM_WORLD);
518         }
519         else {
520             MPI_Send(immigrant, b,
521                     MPI_INT, rank-1,
522                     40,
523                     MPI_COMM_WORLD);
524         }
525     }
526     /* 送信 */
527     MPI_Send(immigrant, b,
528             MPI_INT, rank-1,
529             30,
530             MPI_COMM_WORLD);
531     /* 受信 */
532     if (rank == np-1) {
533         MPI_Recv(immigrant, b,
534                 MPI_INT, 0,
535                 40,
536                 MPI_COMM_WORLD,
537                 status);
538     }
539     else {
540         MPI_Recv(immigrant, b,
541                 MPI_INT, rank+1,
542                 30,
543                 MPI_COMM_WORLD,
544                 status);
545     }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }

```

```

529     40,
530     MPI_COMM_WORLD,
531     status);
532 }
533 /* 受信データ格納 */
534 for (i = 0; i < a; i++) {
535     do {
536         x = rand()%(GENOM/2)+(
537             GENOM/2);
538     } while (flag[x]);
539     flag[x] = 1;
540     for (j = 0; j < NODE+1; j++)
541     {
542         result[x].route[j] = immigrant
543             [i][j];
544     }
545     calc_dist(&result[x]);
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }

```

1.41, 8.54, 23.32, 5.10, 25.00,  
 21.84, 16.76, 12.81, 20.22, 8.06,  
 21.02, 10.63, 8.94, 7.07,  
 559 21.47, 21.93, 14.76, 0.00, 18.03, 19.10,  
 22.47, 8.25, 12.65, 8.06, 10.44,  
 28.02, 12.17, 17.00, 9.85, 10.00,  
 27.59, 7.21, 15.26, 20.22, 16.76,  
 21.02, 19.70, 25.06, 17.09, 4.24,  
 11.05, 9.43, 6.08, 1.41, 22.20,  
 14.42, 24.21, 17.03, 11.40, 17.03,  
 16.12, 23.26, 20.25, 18.97, 23.09,  
 8.06, 24.35, 21.21, 8.06, 9.00,  
 10.00, 25.24, 12.08, 14.42,  
 560 10.30, 5.10, 8.54, 18.03, 0.00, 19.24,  
 9.49, 13.15, 6.71, 16.12, 8.60,  
 10.00, 17.12, 2.00, 8.49, 11.18,  
 10.77, 19.42, 2.83, 15.62, 20.88,  
 10.82, 18.25, 7.28, 4.12, 13.89,  
 14.04, 8.60, 14.21, 19.42, 9.90,  
 7.28, 9.00, 9.22, 6.71, 15.65, 8.06,  
 12.08, 17.46, 12.04, 18.38, 22.80,  
 8.25, 21.10, 20.25, 15.03, 20.12,  
 11.05, 6.40, 3.61,  
 561 12.00, 18.44, 10.82, 19.10, 19.24, 0.00,  
 14.00, 11.70, 20.52, 24.70, 20.00,  
 25.18, 7.00, 20.25, 18.60, 22.47,  
 21.02, 13.15, 18.60, 5.10, 4.24,  
 29.21, 33.60, 25.00, 22.20, 16.76,  
 8.54, 17.20, 21.54, 20.02, 28.84,  
 12.21, 17.00, 26.17, 16.28, 3.61,  
 11.18, 12.17, 33.42, 8.06, 35.38,  
 27.17, 27.31, 2.24, 26.68, 10.20,  
 28.16, 16.12, 19.10, 17.80,  
 562 2.00, 5.66, 7.81, 22.47, 9.49, 14.00,  
 0.00, 14.87, 15.00, 23.54, 16.12,  
 11.40, 15.65, 11.40, 15.30, 19.10,  
 7.07, 20.62, 11.05, 9.06, 17.26,  
 20.12, 27.66, 12.21, 13.60, 18.36,  
 13.60, 14.56, 20.88, 23.85, 18.87,  
 8.06, 3.00, 18.68, 12.37, 11.18,  
 6.40, 2.83, 26.93, 6.08, 27.86,  
 29.15, 15.56, 15.13, 27.20, 15.62,  
 27.66, 2.83, 14.04, 11.00,  
 563 13.60, 15.65, 7.07, 8.25, 13.15, 11.70,  
 14.87, 0.00, 10.77, 13.00, 9.43,  
 22.47, 5.66, 13.00, 8.06, 11.31,  
 20.81, 6.32, 11.00, 12.04, 10.63,  
 20.25, 22.63, 20.40, 14.14, 5.10,  
 3.16, 6.71, 9.85, 9.49, 20.62, 7.21,  
 17.03, 16.55, 7.07, 9.06, 8.49,  
 15.26, 22.67, 10.77, 25.00, 16.03,  
 21.00, 13.93, 15.13, 2.24, 16.49,  
 17.69, 9.49, 10.00,  
 564 15.13, 11.70, 10.30, 12.65, 6.71, 20.52,  
 15.00, 10.77, 0.00, 9.43, 2.24,  
 16.28, 16.12, 5.00, 3.00, 4.47,  
 17.46, 16.12, 4.12, 18.36, 20.81,  
 9.49, 13.42, 12.81, 4.47, 9.06,  
 13.04, 4.12, 7.81, 13.93, 10.05,  
 8.94, 15.30, 5.83, 4.24, 17.03,  
 10.77, 17.12, 13.04, 15.23, 14.87,  
 16.28, 11.70, 22.67, 13.60, 13.00,  
 13.42, 17.12, 1.41, 4.00,  
 565 23.19, 21.02, 17.00, 8.06, 16.12, 24.70,  
 23.54, 13.00, 9.43, 0.00, 7.62,  
 25.61, 18.36, 14.42, 8.25, 5.00,  
 26.83, 14.87, 13.42, 24.33, 23.32,  
 14.87, 11.70, 21.93, 13.45, 8.06,  
 16.16, 9.06, 3.16, 8.54, 16.55,  
 16.03, 24.35, 11.18, 11.18, 21.84,  
 18.03, 25.18, 12.37, 22.02, 15.30,  
 7.21, 20.00, 26.93, 4.24, 14.76,  
 4.12, 25.96, 9.85, 13.00,  
 566 16.00, 13.42, 10.44, 10.44, 8.60, 20.00,  
 16.12, 9.43, 2.24, 7.62, 0.00, 18.38,  
 15.00, 7.07, 1.41, 3.00, 19.24,  
 14.32, 5.83, 18.38, 19.85, 11.18,  
 13.60, 15.00, 6.71, 7.00, 12.04,  
 2.83, 5.66, 11.70, 12.00, 9.22,  
 16.76, 7.28, 4.12, 16.64, 11.18,  
 18.00, 13.45, 15.52, 15.62, 14.21,  
 13.93, 22.20, 11.66, 11.66, 11.70,  
 18.44, 2.24, 5.39,  
 567 13.34, 7.07, 16.28, 28.02, 10.00, 25.18,  
 11.40, 22.47, 16.28, 25.61, 18.38,  
 0.00, 25.32, 11.31, 18.44, 20.62,  
 5.66, 28.79, 12.81, 20.40, 28.00,  
 15.13, 24.52, 4.12, 12.21, 23.85,  
 22.56, 18.60, 24.04, 29.41, 13.04,  
 15.52, 8.54, 16.16, 16.64, 22.02,  
 15.13, 13.93, 23.35, 17.12, 23.02,  
 32.56, 8.00, 26.48, 29.83, 24.04,  
 29.41, 10.30, 16.28, 13.60,  
 568 13.89, 18.25, 9.06, 12.17, 17.12, 7.00,  
 15.65, 5.66, 16.12, 18.36, 15.00,  
 25.32, 0.00, 17.46, 13.60, 16.97,  
 22.47, 6.32, 15.52, 9.43, 5.00,  
 25.50, 28.28, 24.00, 18.97, 10.30,  
 3.16, 12.21, 15.26, 13.04, 25.63,  
 10.00, 18.38, 21.95, 12.08, 5.83,  
 10.20, 15.00, 28.32, 10.00, 30.61,  
 20.22, 25.32, 9.06, 19.92, 3.61,  
 21.54, 18.36, 14.76, 14.56,  
 569 12.08, 7.07, 9.43, 17.00, 2.00, 20.25,  
 11.40, 13.00, 5.00, 14.42, 7.07,  
 11.31, 17.46, 0.00, 7.21, 9.43,  
 12.65, 19.10, 2.00, 16.97, 21.54,  
 9.22, 16.28, 8.06, 2.24, 13.00,  
 14.32, 7.62, 12.73, 18.36, 8.60,  
 8.06, 11.00, 7.28, 6.08, 16.64, 9.22,  
 13.93, 15.52, 13.45, 16.55, 21.26,  
 8.00, 22.20, 18.60, 15.03, 18.36,  
 13.04, 5.00, 3.00,  
 570 15.03, 13.04, 9.22, 9.85, 8.49, 18.60,  
 15.30, 8.06, 3.00, 8.25, 1.41, 18.44,  
 13.60, 7.21, 0.00, 4.12, 18.87,  
 13.15, 5.66, 17.09, 18.44, 12.37,  
 15.00, 15.26, 7.28, 6.08, 10.63,  
 1.41, 5.83, 11.18, 13.04, 8.06,  
 16.16, 8.54, 3.00, 15.26, 10.05,  
 17.03, 14.87, 14.32, 17.03, 14.42,  
 14.56, 20.81, 12.08, 10.30, 12.37,

17.72, 2.24, 5.00,  
 571 19.00, 16.16, 13.34, 10.00, 11.18, 22.47,  
 19.10, 11.31, 4.47, 5.00, 3.00,  
 20.62, 16.97, 9.43, 4.12, 0.00,  
 21.93, 15.23, 8.54, 21.19, 21.93,  
 11.05, 11.31, 16.97, 8.49, 7.62,  
 14.21, 5.39, 4.12, 11.05, 12.37,  
 12.17, 19.65, 7.07, 7.07, 19.24,  
 14.14, 21.00, 11.40, 18.44, 13.89,  
 12.04, 15.26, 24.70, 9.22, 13.45,  
 8.94, 21.38, 5.10, 8.25,  
 572 9.06, 5.83, 13.89, 27.59, 10.77, 21.02,  
 7.07, 20.81, 17.46, 26.83, 19.24,  
 5.66, 22.47, 12.65, 18.87, 21.93,  
 0.00, 26.93, 13.42, 16.00, 24.33,  
 19.10, 28.02, 8.54, 14.32, 23.35,  
 20.12, 18.60, 24.70, 29.00, 17.26,  
 13.60, 4.12, 19.10, 16.40, 18.25,  
 12.53, 9.06, 27.00, 13.15, 27.17,  
 33.29, 12.65, 22.02, 30.89, 21.95,  
 30.87, 5.10, 17.00, 13.89,  
 573 19.10, 21.93, 13.04, 7.21, 19.42, 13.15,  
 20.62, 6.32, 16.12, 14.87, 14.32,  
 28.79, 6.32, 19.10, 13.15, 15.23,  
 26.93, 0.00, 17.12, 15.65, 10.05,  
 25.50, 26.08, 26.68, 20.00, 7.62,  
 7.07, 12.04, 12.21, 7.62, 26.17,  
 13.42, 23.02, 21.59, 13.04, 12.08,  
 14.42, 20.52, 26.42, 15.62, 29.07,  
 14.87, 27.07, 15.03, 15.26, 5.00,  
 17.20, 23.43, 15.03, 16.12,  
 574 11.40, 7.62, 7.81, 15.26, 2.83, 18.60,  
 11.05, 11.00, 4.12, 13.42, 5.83,  
 12.81, 15.52, 2.00, 5.66, 8.54,  
 13.42, 17.12, 0.00, 15.62, 19.70,  
 10.63, 16.76, 9.85, 3.61, 11.18,  
 12.37, 5.83, 11.40, 16.64, 10.30,  
 6.40, 11.18, 8.06, 4.12, 15.00, 7.81,  
 13.34, 16.16, 12.21, 17.49, 20.00,  
 10.00, 20.62, 17.49, 13.04, 17.46,  
 13.04, 3.61, 1.00,  
 575 7.07, 13.93, 8.06, 20.22, 15.62, 5.10,  
 9.06, 12.04, 18.36, 24.33, 18.38,  
 20.40, 9.43, 16.97, 17.09, 21.19,  
 16.00, 15.65, 15.62, 0.00, 8.94,  
 26.17, 31.76, 20.62, 19.10, 17.00,  
 9.22, 15.81, 21.21, 21.38, 25.50,  
 9.43, 12.04, 23.60, 14.32, 3.61,  
 7.81, 7.07, 31.38, 3.61, 33.02,  
 28.07, 23.32, 6.08, 27.02, 11.40,  
 28.16, 11.05, 17.00, 15.00,  
 576 15.30, 21.02, 12.37, 16.76, 20.88, 4.24,  
 17.26, 10.63, 20.81, 23.32, 19.85,  
 28.00, 5.00, 21.54, 18.44, 21.93,  
 24.33, 10.05, 19.70, 8.94, 0.00,  
 30.02, 33.24, 27.29, 23.26, 15.26,  
 7.81, 17.03, 20.25, 17.46, 29.97,  
 13.60, 20.22, 26.63, 16.64, 6.08,  
 13.15, 15.81, 33.24, 11.18, 35.47,  
 24.74, 29.12, 5.39, 24.70, 8.60,  
 26.40, 19.65, 19.42, 18.79,  
 577 21.10, 14.87, 18.44, 21.02, 10.82, 29.21,  
 20.12, 20.25, 9.49, 14.87, 11.18,  
 15.13, 25.50, 9.22, 12.37, 11.05,  
 19.10, 25.50, 10.63, 26.17, 30.02,  
 0.00, 9.49, 11.05, 7.07, 18.11,  
 22.36, 13.60, 15.13, 22.09, 2.24,  
 17.03, 18.97, 4.00, 13.42, 25.61,  
 18.38, 22.83, 8.25, 22.67, 8.06,  
 21.93, 7.28, 31.24, 18.79, 22.47,  
 17.49, 21.19, 10.77, 11.40,  
 578 28.16, 23.09, 23.71, 19.70, 18.25, 33.60,  
 27.66, 22.63, 13.42, 11.70, 13.60,  
 24.52, 28.28, 16.28, 15.00, 11.31,  
 28.02, 26.08, 16.76, 31.76, 33.24,  
 9.49, 0.00, 20.40, 14.14, 18.60,  
 25.50, 16.40, 13.89, 20.25, 11.70,  
 22.36, 27.17, 9.06, 17.49, 30.23,  
 24.17, 30.08, 1.41, 28.64, 4.12,  
 17.00, 16.76, 35.81, 14.04, 24.76,  
 12.00, 29.27, 14.76, 17.09,  
 579 13.89, 6.71, 15.03, 25.06, 7.28, 25.00,  
 12.21, 20.40, 12.81, 21.93, 15.00,  
 4.12, 24.00, 8.06, 15.26, 16.97,  
 8.54, 26.68, 9.85, 20.62, 27.29,  
 11.05, 20.40, 0.00, 8.49, 21.02,  
 21.02, 15.65, 20.62, 26.42, 9.00,  
 14.00, 9.90, 12.08, 13.93, 21.59,  
 14.14, 15.00, 19.24, 17.09, 19.00,  
 29.00, 4.12, 26.57, 26.17, 22.20,  
 25.61, 12.04, 13.04, 10.77,  
 580 14.32, 9.00, 11.40, 17.09, 4.12, 22.20,  
 13.60, 14.14, 4.47, 13.45, 6.71,  
 12.21, 18.97, 2.24, 7.28, 8.49,  
 14.32, 20.00, 3.61, 19.10, 23.26,  
 7.07, 14.14, 8.49, 0.00, 13.34,  
 15.81, 8.06, 12.21, 18.38, 6.71,  
 10.00, 13.04, 5.10, 7.07, 18.60,  
 11.31, 16.16, 13.34, 15.62, 14.32,  
 20.52, 7.28, 24.21, 17.69, 16.28,  
 17.20, 15.13, 5.10, 4.47,  
 581 17.46, 17.69, 10.77, 4.24, 13.89, 16.76,  
 18.36, 5.10, 9.06, 8.06, 7.00, 23.85,  
 10.30, 13.00, 6.08, 7.62, 23.35,  
 7.62, 11.18, 17.00, 15.26, 18.11,  
 18.60, 21.02, 13.34, 0.00, 8.25,  
 5.39, 5.00, 5.66, 19.00, 10.30,  
 20.00, 14.14, 7.21, 14.14, 12.08,  
 19.31, 18.87, 15.30, 21.47, 11.18,  
 20.62, 18.97, 10.05, 6.71, 11.40,  
 21.10, 8.25, 10.30,  
 582 12.04, 15.52, 6.32, 11.05, 14.04, 8.54,  
 13.60, 3.16, 13.04, 16.16, 12.04,  
 22.56, 3.16, 14.32, 10.63, 14.21,  
 20.12, 7.07, 12.37, 9.22, 7.81,  
 22.36, 25.50, 21.02, 15.81, 8.25,  
 0.00, 9.22, 13.00, 12.17, 22.47,  
 7.07, 16.12, 18.87, 8.94, 6.00, 7.62,  
 13.45, 25.46, 8.60, 27.66, 19.00,  
 22.20, 10.77, 18.25, 2.24, 19.65,  
 16.40, 11.66, 11.40,  
 583 14.14, 12.81, 8.06, 9.43, 8.60, 17.20,

14.56, 6.71, 4.12, 9.06, 2.83, 18.60,  
 12.21, 7.62, 1.41, 5.39, 18.60,  
 12.04, 5.83, 15.81, 17.03, 13.60,  
 16.40, 15.65, 8.06, 5.39, 9.22, 0.00,  
 6.32, 10.82, 14.14, 7.00, 15.65,  
 9.85, 2.24, 13.89, 9.00, 16.12,  
 16.28, 13.15, 18.44, 14.76, 15.30,  
 19.42, 12.65, 8.94, 13.15, 17.09,  
 3.00, 5.00,  
 584 20.40, 18.87, 14.04, 6.08, 14.21, 21.54,  
 20.88, 9.85, 7.81, 3.16, 5.66, 24.04,  
 15.26, 12.73, 5.83, 4.12, 24.70,  
 12.21, 11.40, 21.21, 20.25, 15.13,  
 13.89, 20.62, 12.21, 5.00, 13.00,  
 6.32, 0.00, 7.00, 16.49, 13.15,  
 21.93, 11.18, 8.54, 18.68, 15.13,  
 22.36, 14.32, 19.00, 17.09, 8.60,  
 19.24, 23.77, 6.32, 11.66, 7.00,  
 23.41, 7.81, 10.82,  
 585 22.83, 23.35, 16.12, 1.41, 19.42, 20.02,  
 23.85, 9.49, 13.93, 8.54, 11.70,  
 29.41, 13.04, 18.36, 11.18, 11.05,  
 29.00, 7.62, 16.64, 21.38, 17.46,  
 22.09, 20.25, 26.42, 18.38, 5.66,  
 12.17, 10.82, 7.00, 0.00, 23.35,  
 15.81, 25.61, 18.11, 12.81, 18.11,  
 17.49, 24.60, 20.88, 20.25, 23.77,  
 7.28, 25.63, 22.09, 7.81, 10.05,  
 9.90, 26.63, 13.42, 15.81,  
 586 20.00, 13.42, 18.03, 22.20, 9.90, 28.84,  
 18.87, 20.62, 10.05, 16.55, 12.00,  
 13.04, 25.63, 8.60, 13.04, 12.37,  
 17.26, 26.17, 10.30, 25.50, 29.97,  
 2.24, 11.70, 9.00, 6.71, 19.00,  
 22.47, 14.14, 16.49, 23.35, 0.00,  
 16.64, 17.46, 5.39, 13.60, 25.24,  
 17.80, 21.63, 10.44, 21.93, 10.00,  
 23.71, 5.10, 30.81, 20.59, 22.80,  
 19.42, 19.70, 11.18, 11.18,  
 587 7.28, 8.54, 1.41, 14.42, 7.28, 12.21,  
 8.06, 7.21, 8.94, 16.03, 9.22,  
 15.52, 10.00, 8.06, 8.06, 12.17,  
 13.60, 13.42, 6.40, 9.43, 13.60,  
 17.03, 22.36, 14.00, 10.00, 10.30,  
 7.07, 7.00, 13.15, 15.81, 16.64,  
 0.00, 9.90, 14.21, 5.10, 8.60, 2.00,  
 9.22, 21.95, 6.32, 23.60, 21.19,  
 15.52, 14.21, 19.42, 8.54, 20.10,  
 10.82, 7.62, 5.66,  
 588 5.00, 4.12, 10.00, 24.21, 9.00, 17.00,  
 3.00, 17.03, 15.30, 24.35, 16.76,  
 8.54, 18.38, 11.00, 16.16, 19.65,  
 4.12, 23.02, 11.18, 12.04, 20.22,  
 18.97, 27.17, 9.90, 13.04, 20.00,  
 16.12, 15.65, 21.93, 25.61, 17.46,  
 9.90, 0.00, 18.11, 13.42, 14.14,  
 8.60, 5.39, 26.31, 9.06, 26.93,  
 30.41, 13.60, 18.11, 28.23, 18.03,  
 28.46, 2.24, 14.56, 11.40,  
 589 19.31, 14.04, 15.62, 17.03, 9.22, 26.17,  
 18.68, 16.55, 5.83, 11.18, 7.28,  
 16.16, 21.95, 7.28, 8.54, 7.07,  
 19.10, 21.59, 8.06, 23.60, 26.63,  
 4.00, 9.06, 12.08, 5.10, 14.14,  
 18.87, 9.85, 11.18, 18.11, 5.39,  
 14.21, 18.11, 0.00, 10.00, 22.63,  
 15.81, 21.19, 8.25, 20.25, 9.43,  
 18.36, 9.22, 28.28, 15.26, 18.79,  
 14.21, 20.22, 7.21, 8.60,  
 590 12.04, 10.63, 6.32, 11.40, 6.71, 16.28,  
 12.37, 7.07, 4.24, 11.18, 4.12,  
 16.64, 12.08, 6.08, 3.00, 7.07,  
 16.40, 13.04, 4.12, 14.32, 16.64,  
 13.42, 17.49, 13.93, 7.07, 7.21,  
 8.94, 2.24, 8.54, 12.81, 13.60, 5.10,  
 13.42, 10.00, 0.00, 12.81, 7.07,  
 14.04, 17.20, 11.40, 19.10, 17.00,  
 14.04, 18.44, 14.87, 9.22, 15.30,  
 14.87, 2.83, 3.16,  
 591 9.22, 15.13, 7.21, 17.03, 15.65, 3.61,  
 11.18, 9.06, 17.03, 21.84, 16.64,  
 22.02, 5.83, 16.64, 15.26, 19.24,  
 18.25, 12.08, 15.00, 3.61, 6.08,  
 25.61, 30.23, 21.59, 18.60, 14.14,  
 6.00, 13.89, 18.68, 18.11, 25.24,  
 8.60, 14.14, 22.63, 12.81, 0.00,  
 7.62, 9.85, 30.00, 5.10, 31.89,  
 25.00, 23.77, 5.66, 24.19, 8.06,  
 25.50, 13.60, 15.62, 14.21,  
 592 5.39, 8.06, 1.41, 16.12, 8.06, 11.18,  
 6.40, 8.49, 10.77, 18.03, 11.18,  
 15.13, 10.20, 9.22, 10.05, 14.14,  
 12.53, 14.42, 7.81, 7.81, 13.15,  
 18.38, 24.17, 14.14, 11.31, 12.08,  
 7.62, 9.00, 15.13, 17.49, 17.80,  
 2.00, 8.60, 15.81, 7.07, 7.62, 0.00,  
 7.28, 23.71, 4.47, 25.24, 23.09,  
 16.16, 13.04, 21.38, 9.43, 22.09,  
 9.22, 9.49, 7.21,  
 593 2.00, 8.49, 8.54, 23.26, 12.08, 12.17,  
 2.83, 15.26, 17.12, 25.18, 18.00,  
 13.93, 15.00, 13.93, 17.03, 21.00,  
 9.06, 20.52, 13.34, 7.07, 15.81,  
 22.83, 30.08, 15.00, 16.16, 19.31,  
 13.45, 16.12, 22.36, 24.60, 21.63,  
 9.22, 5.39, 21.19, 14.04, 9.85,  
 7.28, 0.00, 29.41, 5.00, 30.46,  
 30.36, 18.38, 13.00, 28.64, 15.62,  
 29.27, 4.00, 16.03, 13.15,  
 594 27.51, 22.20, 23.32, 20.25, 17.46, 33.42,  
 26.93, 22.67, 13.04, 12.37, 13.45,  
 23.35, 28.32, 15.52, 14.87, 11.40,  
 27.00, 26.42, 16.16, 31.38, 33.24,  
 8.25, 1.41, 19.24, 13.34, 18.87,  
 25.46, 16.28, 14.32, 20.88, 10.44,  
 21.95, 26.31, 8.25, 17.20, 30.00,  
 23.71, 29.41, 0.00, 28.18, 3.00,  
 18.03, 15.52, 35.61, 15.00, 24.84,  
 13.04, 28.44, 14.42, 16.55,  
 595 4.12, 10.44, 5.10, 18.97, 12.04, 8.06,  
 6.08, 10.77, 15.23, 22.02, 15.52,  
 17.12, 10.00, 13.45, 14.32, 18.44,

```

13.15, 15.62, 12.21, 3.61, 11.18,
22.67, 28.64, 17.09, 15.62, 15.30,
8.60, 13.15, 19.00, 20.25, 21.93,
6.32, 9.06, 20.25, 11.40, 5.10,
4.47, 5.00, 28.18, 0.00, 29.68,
26.48, 19.72, 9.49, 25.08, 10.82,
26.00, 8.54, 13.93, 11.66,
596 28.64, 22.80, 25.00, 23.09, 18.38, 35.38,
27.86, 25.00, 14.87, 15.30, 15.62,
23.02, 30.61, 16.55, 17.03, 13.89,
27.17, 29.07, 17.49, 33.02, 35.47,
8.06, 4.12, 19.00, 14.32, 21.47,
27.66, 18.44, 17.09, 23.77, 10.00,
23.60, 26.93, 9.43, 19.10, 31.89,
25.24, 30.46, 3.00, 29.68, 0.00,
21.02, 15.03, 37.54, 18.00, 27.20,
16.03, 29.12, 16.28, 18.03,
597 28.46, 27.46, 21.84, 8.06, 22.80, 27.17,
29.15, 16.03, 16.28, 7.21, 14.21,
32.56, 20.22, 21.26, 14.42, 12.04,
33.29, 14.87, 20.00, 28.07, 24.74,
21.93, 17.00, 29.00, 20.52, 11.18,
19.00, 14.76, 8.60, 7.28, 23.71,
21.19, 30.41, 18.36, 17.00, 25.00,
23.09, 30.36, 18.03, 26.48, 21.02,
0.00, 27.20, 29.27, 3.16, 17.03,
5.00, 31.78, 16.40, 19.42,
598 17.03, 9.90, 16.76, 24.35, 8.25, 27.31,
15.56, 21.00, 11.70, 20.00, 13.93,
8.00, 25.32, 8.00, 14.56, 15.26,
12.65, 27.07, 10.00, 23.32, 29.12,
7.28, 16.76, 4.12, 7.28, 20.62,
22.20, 15.30, 19.24, 25.63, 5.10,
15.52, 13.60, 9.22, 14.04, 23.77,
16.16, 18.38, 15.52, 19.72, 15.03,
27.20, 0.00, 29.07, 24.21, 23.02,
23.35, 15.81, 12.37, 11.00,
599 13.15, 19.92, 12.81, 21.21, 21.10, 2.24,
15.13, 13.93, 22.67, 26.93, 22.20,
26.48, 9.06, 22.20, 20.81, 24.70,
22.02, 15.03, 20.62, 6.08, 5.39,
31.24, 35.81, 26.57, 24.21, 18.97,
10.77, 19.42, 23.77, 22.09, 30.81,
14.21, 18.11, 28.28, 18.44, 5.66,
13.04, 13.00, 35.61, 9.49, 37.54,
29.27, 29.07, 0.00, 28.86, 12.37,
30.36, 17.00, 21.26, 19.85,
600 26.68, 25.06, 20.22, 8.06, 20.25, 26.68,
27.20, 15.13, 13.60, 4.24, 11.66,
29.83, 19.92, 18.60, 12.08, 9.22,
30.89, 15.26, 17.49, 27.02, 24.70,
18.79, 14.04, 26.17, 17.69, 10.05,
18.25, 12.65, 6.32, 7.81, 20.59,
19.42, 28.23, 15.26, 14.87, 24.19,
21.38, 28.64, 15.00, 25.08, 18.00,
3.16, 24.21, 28.86, 0.00, 16.49,
2.24, 29.73, 13.89, 17.00,
601 14.14, 17.09, 8.06, 9.00, 15.03, 10.20,
15.62, 2.24, 13.00, 14.76, 11.66,
24.04, 3.61, 15.03, 10.30, 13.45,
21.95, 5.00, 13.04, 11.40, 8.60,
22.47, 24.76, 22.20, 16.28, 6.71,
2.24, 8.94, 11.66, 10.05, 22.80,
8.54, 18.03, 18.79, 9.22, 8.06, 9.43,
15.62, 24.84, 10.82, 27.20, 17.03,
23.02, 12.37, 16.49, 0.00, 18.03,
18.44, 11.70, 12.04,
602 27.29, 25.08, 21.02, 10.00, 20.12, 28.16,
27.66, 16.49, 13.42, 4.12, 11.70,
29.41, 21.54, 18.36, 12.37, 8.94,
30.87, 17.20, 17.46, 28.16, 26.40,
17.49, 12.00, 25.61, 17.20, 11.40,
19.65, 13.15, 7.00, 9.90, 19.42,
20.10, 28.46, 14.21, 15.30, 25.50,
22.09, 29.27, 13.04, 26.00, 16.03,
5.00, 23.35, 30.36, 2.24, 18.03,
0.00, 30.08, 13.93, 17.09,
603 4.47, 6.32, 10.63, 25.24, 11.05, 16.12,
2.83, 17.69, 17.12, 25.96, 18.44,
10.30, 18.36, 13.04, 17.72, 21.38,
5.10, 23.43, 13.04, 11.05, 19.65,
21.19, 29.27, 12.04, 15.13, 21.10,
16.40, 17.09, 23.41, 26.63, 19.70,
10.82, 2.24, 20.22, 14.87, 13.60,
9.22, 4.00, 28.44, 8.54, 29.12,
31.78, 15.81, 17.00, 29.73, 18.44,
30.08, 0.00, 16.28, 13.15,
604 14.04, 11.18, 8.94, 12.08, 6.40, 19.10,
14.04, 9.49, 1.41, 9.85, 2.24, 16.28,
14.76, 5.00, 2.24, 5.10, 17.00,
15.03, 3.61, 17.00, 19.42, 10.77,
14.76, 13.04, 5.10, 8.25, 11.66,
3.00, 7.81, 13.42, 11.18, 7.62,
14.56, 7.21, 2.83, 15.62, 9.49,
16.03, 14.42, 13.93, 16.28, 16.40,
12.37, 21.26, 13.89, 11.70, 13.93,
16.28, 0.00, 3.16,
605 11.18, 8.06, 7.07, 14.42, 3.61, 17.80,
11.00, 10.00, 4.00, 13.00, 5.39,
13.60, 14.56, 3.00, 5.00, 8.25,
13.89, 16.12, 1.00, 15.00, 18.79,
11.40, 17.09, 10.77, 4.47, 10.30,
11.40, 5.00, 10.82, 15.81, 11.18,
5.66, 11.40, 8.60, 3.16, 14.21, 7.21,
13.15, 16.55, 11.66, 18.03, 19.42,
11.00, 19.85, 17.00, 12.04, 17.09,
13.15, 3.16, 0.00
606 };
607
608
609
610
611 for (i = 0; i < NODE; i++) {
612     for (j = 0; j < NODE; j++) {
613         town[i].dist[j] = list[i][j];
614     }
615 }
616 }

```

---