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8日

目 次

図一覧

表一覧

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)	$\tilde{\tilde{c}}_{ik}$
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	$\mu_{\tilde{\tilde{c}}_{ik}}$
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

第1章

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，人手不足による国内総生産（Gross Domestic Product : 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「人員・費用などの資源の最適な配分による生産性の向上」に着目した．

本研究では，労働人口の減少により，様々な業界で作業効率や生産性の向上が注目されるなか，特に人手不足が深刻である建築・土木工事の現場に焦点を当てて考える．住宅建築などの大きな工事は，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把握しておくことが重要である．しかし，作業中に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ため，現場におけるスケジュール管理は非常に困難とされている．

また，資源の最適な配分を考え生産性の向上を図る上で，各作業における職人さんごとの作業の所要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「天候」などの不確実な要素から影響を受ける不確定なものである．つまり，実問題における，資源の最適な配分を目的とした日程計画を考える際には，不確定性と不確定性の両方の性質を含む問題として考える必要がある．

一方で，従来の建築現場における日程計画問題では，この両方の性質を同時に考慮できていない．したがって，不確定性と不確実性の両方の性質を考慮し，より実問題に対応できる日程計画問題に改良する必要があると考える．

§ 1.2　本研究の目的

本研究では,「建築現場における，資源の最適な配分による生産性の向上」を目的とした日程計画を考える．この目的を達成するために，建築現場における日程計画の現状と問題点について理解し，その問題点を解消して作業効率を上げるために必要な仕組みについて考える．また，実問題への対応を考えた日程計画問題にするため，本研究では次の二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きに必要となる費用の不確定性と天候の不確実性を考慮することである．このとき，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] の概念を用いる．二つ目は，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，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問題のパラメータ設定に活用し，今あるシステムを拡張することで実用できるようなモデルを提案する．

更に，以上のことを考慮した提案モデルを，実際に遺伝的アルゴリズムなどの解法を用いて解くことができるように，モデルの定式化と式の等価確定変換を行う．また，ファジィ・ランダム変数を導入することで複雑な問題になり，処理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．したがって，遺伝的アルゴリズムを使用した並列分散処理による処理時間の高速化を目指し，Raspberry Pi 3（Model B）を 8 台と MPI を用いて，並列分散 GA が実行できる環境の構築と高速化の動作テストを行い，提案モデルの解法として問題なく用いることができるかを確認する．

§ 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 従来の日程計画と今の建築現場における課題について述べ，これからの建築現場に必要な仕組みと，既存のシステムから得られる作業情報の活用について説明する．

第 3 章 一般的なファジィ・ランダム変数や解法に用いる遺伝的アルゴリズムについて述べる．また，高速化を目的とした並列分散環境の構築ついて説明し，構築した環境での動作テストについてまとめる．

第 4 章 ファジィ・ランダム変数を導入した提案モデルについて説明する．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．

第2章

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.

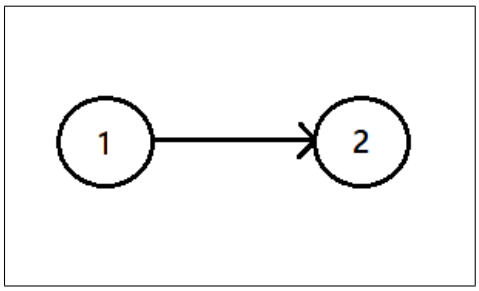

図 2.1: アロー・ダイヤグラム

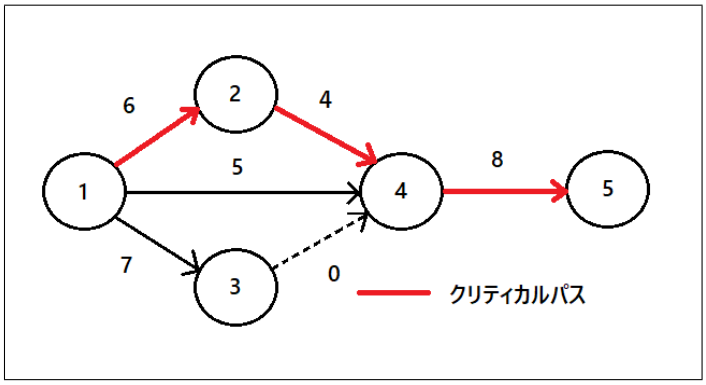

図 2.2: クリティカルパスの例

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・ダイヤグラムと呼ばれる図法を用いることが多い（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定で不確実な場合を想定するため，上記の詳細なクリティカルパスの算出ではなく，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．

従来の日程計画問題の課題

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，そのシステムを実際に活用できていない業界は多く，節 2.1 で説明した PERT や CPM などの手法も十分に活用できていない．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問題と，実問題には多くのギャップが存在しているからである．

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は，不規則かつ不確実なものを前提としていない．
3. 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．
4. 従来の研究モデルでは，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，研究モデルと実問題の間には多くのギャップが存在している．従来のモデルでも対応できるような単純な問題は，実問題には殆ど存在しておらず，実問題には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　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いては，職人さん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 [8].

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

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

図 2.3: 現場コミュニケーションアプリ kizuku

§ 2.3　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[1]

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 [9].

現場コミュニケーションアプリ Kizuku[2]

このアプリでは，建築の現場における進歩報告，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とリアルタイムでやり取りができる（図 2.3）．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 [10].

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ため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．

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

[1]https://mag.branu.jp /archives/2442
[2]https://www.ctx.co.jp/kizuku2pr/

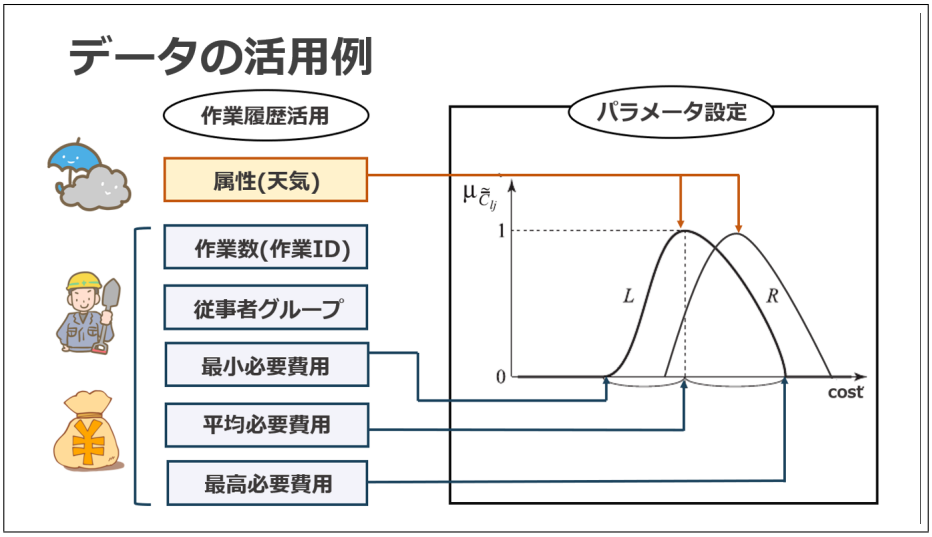

図 2.4: 作業履歴を活用したパラメータの設定

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，職人さんごとの作業にかかる所要時間と費用などが挙げられる．更に，その情報と天気の情報を合わせて考えると，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日程計画が可能であると考える．

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 (職人さんの数)，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費用の最大値と最小値，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 2.4 参照）．

第3章

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，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・雨などの属性ごとで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].

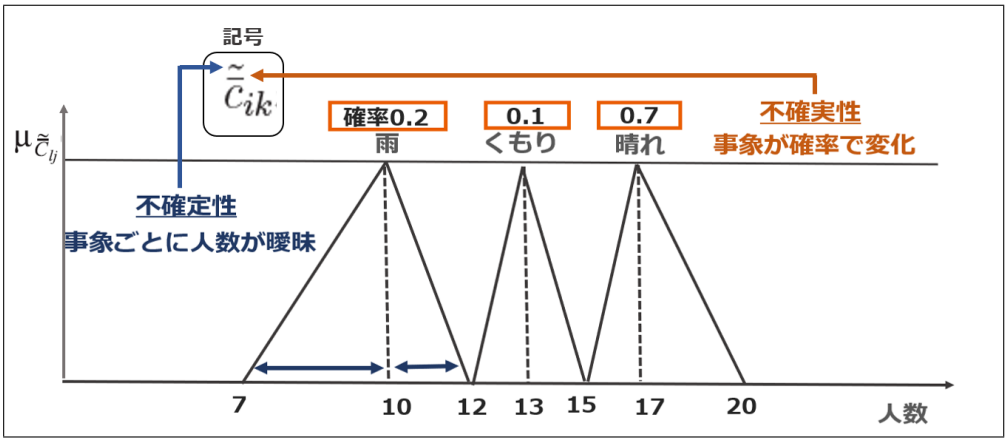

図 3.1: ファジィ・ランダムのイメージ

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

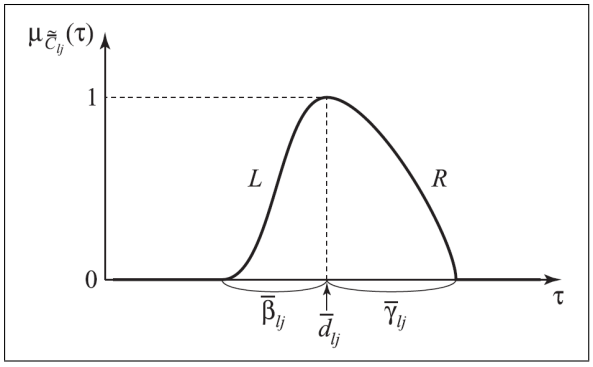

図 3.2: ファジィランダム変数のメンバシップ関数 $\mu_{\tilde{\bar{C}}_{ij}}(\tau)$ の例

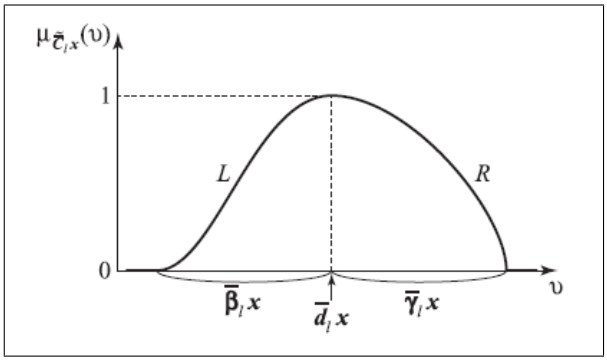

図 3.3: 目的関数のメンバシップ関数 $\mu_{\tilde{\bar{C}}_{ij}x_{ij}}(v)$ の例

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 [12]．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）.

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

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）.

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

§ 3.2　遺伝的アルゴリズムによる解法

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で，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 [13].

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で，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，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，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．更に，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．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．つまり，多目的最適化問題とは，このいくつかある最適解の集合 (パレート最適フロント) から少なくとも 1 つの解を探索する問題である（図 3.4 参照）.

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）．これまでにも，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 [14]．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 [15].

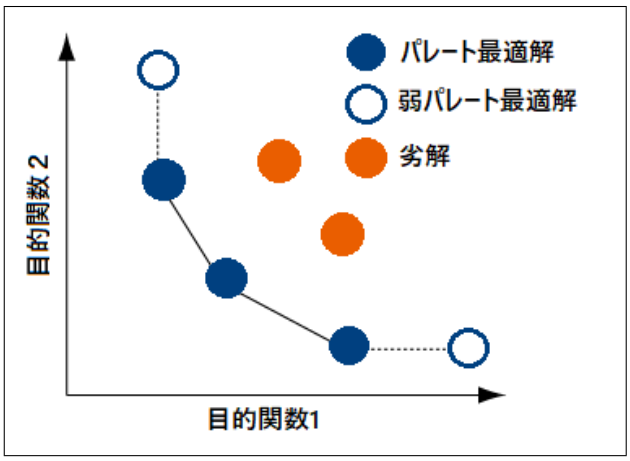

図 3.4: パレート最適解の例

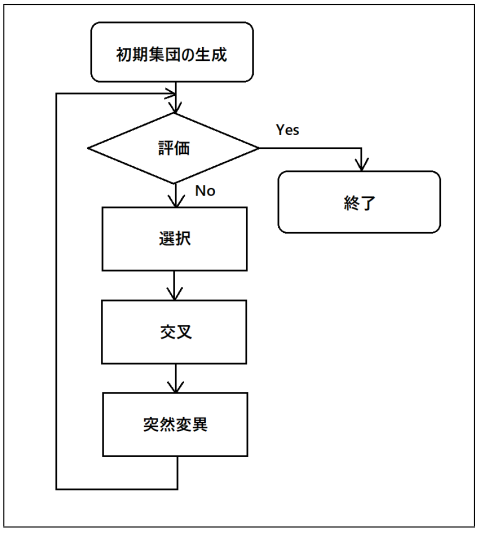

図 3.5: GA のフローチャート

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できるため，解の探索能力が高まる（図 3.6 参照）．また，並列分散処理では，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 Pi 3（Model B）[3]8 台と MPI を用いて並列分散処理を行う（図 3.7 参照）．

[3]https://jp.rs-online.com/web/p/products/1239470/

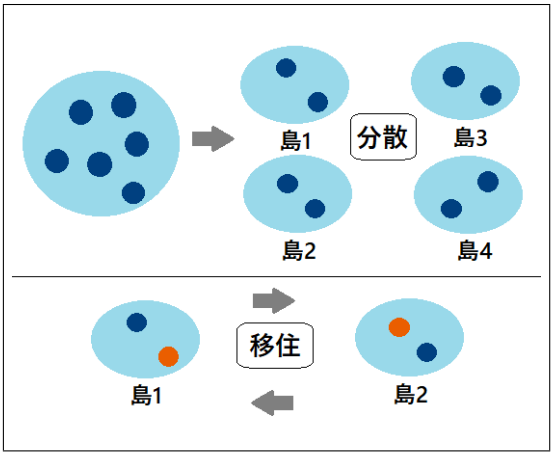

図 3.6: 並列分散 GA のイメージ

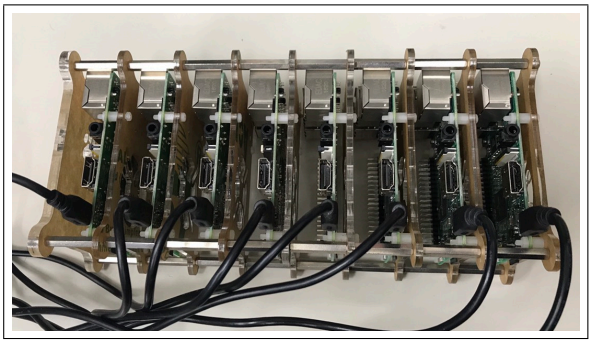

図 3.7: 実験に用いるラズベリーパイ 8 台

Raspberry Pi 3(Model B）を用いた並列分散環境の構築に必要なもの

1. Raspberry Pi3（ModelB）× 8
2. SD カード 16G 以上× 8
3. 8 ポート以上のハブ
4. SD カード 16G 以上× 8
5. USB 接続ができるキーボード
6. USB 接続ができるマウス
7. HDMI 接続ができるモニタ
8. AC アダプタ

Raspberry Pi 3（Model B）とは

Raspberry Pi とは，テレビ，キーボード，マウス，電源を繋ぐだけでで使用することができるコンパクトなコンピュータボードである．多種多様な製品があり，カメラや LCD ディスプレイモジュールなどの用途を実現することも可能である．

本研究で使用する Raspberry Pi 3（Model B）は，以前までのモデルからのアップグレードで，接続性が向上し，WLAN，Bluetooth，及び Bluetooth Low Energy (BLE) のすべてがオンボードで利用可能である．また，接続性の向上により，他のサポートと容易に通信ができ，IoT の設計にも最適とされている．

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 [17].

Raspberry Pi 3（Model B）の初期設定～並列環境の構築

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，それぞれの間でお互いに通信し，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．このとき，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」の二種類に分けられる．

手順 1 ：Raspbian のダウンロード

OS(Raspbian) を配布元からダウンロード・解凍し，Raspbian のディスクイメージファイルを入手する．入手したディスクイメージファイルを SD カードに焼く．問題なく SD カードに Raspbian を入れることができたら，各マシンに SD カードをセットし Raspberry Pi 3 を起動する．

手順 2 : IP アドレスを固定するためのファイル編集

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 アドレスを固定する．
設定ファイルを編集する際，必要であればテキストエディタをインストールする．

```
$ sudo apt-get install emacs
```

次に，各 Raspberry Pi のホスト名を変更する．

```
$ sudo emacs /etc/hostname
```

そして，固定 IP の設定のために 「/etc/dhcpcd.conf」を編集する．

```
$ sudo emacs /etc/dhcpcd.conf
```

ファイル内の「static ip __address=初期設定の IP アドレス」を，それぞれの Raspberry Pi 3 に固定する IP アドレスに書き換える．更に，ホスト名と固定した IP アドレスを一致させるために, 次のように編集する．

```
$ sudo emacs /etc/hosts
```

・ 192.168.0.60 master

・ 192.168.0.61 slave2

...

・ 192.168.0.67 slave8

各ファイルの編集が終わったら，全ての Raspberry Pi を再起動する．

手順 3：SSH 接続を有効にする

ネットワークを再起動し終わったら

$ ping ホスト名, ping IP アドレス

がそれぞれのノードに正常に通るか確認を行う．

$ sudo raspi-config

で ssh を enable に設定する．

$ ssh-keygen -t rsa

を実行し，SSH 鍵を作成する．更に，作成した公開鍵を全てのマシンに送信する．

$ ssh-copy-id master

$ ssh-copy-id slave2

...

$ ssh-copy-id slave8

上記の作業をすべてのマシン上で行う．ここまでの作業が完了したら，以降はマスターからリモートで各マシンに SSH 接続して設定を行う．

手順 4：mpich のインストール

次のコマンドで全てのマシンに mpich をインストールする．

$ sudo apt-get install mpich

mpich を用いたコンパイルと実行のコマンドは次のようになる．

$ sudo mpicc （ファイル名.c）

$ sudo mpirun ./（ファイル名）

複数台を同時に実行するときは

$ mpirun -np (並列数)(実行コマンド)

となる．

手順 5：NFS を用いた作業ディレクトリの共有

次に，NFS の設定を行う．NFS とは，分散ファイルシステムの一種で，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．
まず，マスターで NFS のインストールを行う．

$ apt-get install nfs-kernel-server

その後，/etc/exports ファイルを編集し以下の文を追記する．

・(公開したいディレクトリ) (どのマシンに公開するか)(公開モード)

・例/home/server/foo 192.168.1.200(rw, sync, no __ subtree __ check)

exports を編集したら以下のコマンドで再起動

$ /etc/init.d/nfs-kernel-server restart

マスターでの設定が完了したら，他のスレーブ 7 台の設定も行う．nfs-common のインストール，共有先のディレクトリ（/mpi1/test）の作成とマウント設定を以下のコマンドで行う．

$ sudo apt-get install nfs-common

$ mkdir /home/mpi1/test

$ sudo mount –types nfs 192.168.0.60:/home/mpi1/test /home/mpi1/test

更に，/etc/fstab を編集して (共有したいディレクトリ) (共有先) を追記する．

例 192.168.0.60:/home/mpi1/test /home/mpi1/test nfs rw 0 0

以上の設定で，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ことができる．

手順 6：host ファイルの設置

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．

```
pi@master:/home/mpi1/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 Raspberry Pi 8 台を用いた Hello World の並列実行結果

表 3.1: 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

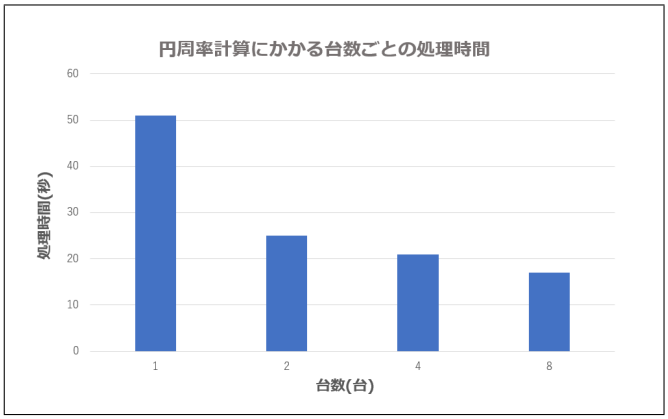

図 3.9: 円周率計算の処理時間グラフ

Raspberry Pi 3（Model B）と mpich を用いた並列分散処理の動作確認

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，Raspberry Pi 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：Hello World の並列処理

動作確認のため，ラズベリーパイ 8 台で「Hello 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 [17].

図 3.8 を見ると，問題なく 8 台分の「Hello 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 Hello 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るかの確認も行う（表 3.1，3.9 参照）．

表 3.2: TSP 問題の処理時間

台数	処理時間
1 台	320 秒
2 台	165 秒
4 台	71 秒
8 台	107 秒

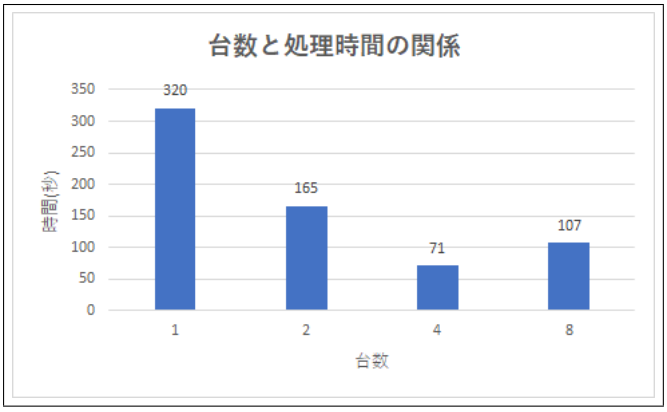

図 3.10: TSP 問題の処理時間のグラフ

表 3.1 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 3.9 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．

動作テスト 3：並列分散 GA による巡回セールスマン問題（TSP 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．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（表 3.2, 図 3.10 参照）[18]．MPI を用いた並列分散 GA のフローは次のようになる（図 3.11 参照）．

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

1. マップ：30 × 30
2. 都市数：50 都市
3. 遺伝子数：800 個
4. 世代数：50000 世代
5. 移住数：120 遺伝子 (15 %)
6. 端末数：1 台、2 台、4 台、8 台

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．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．

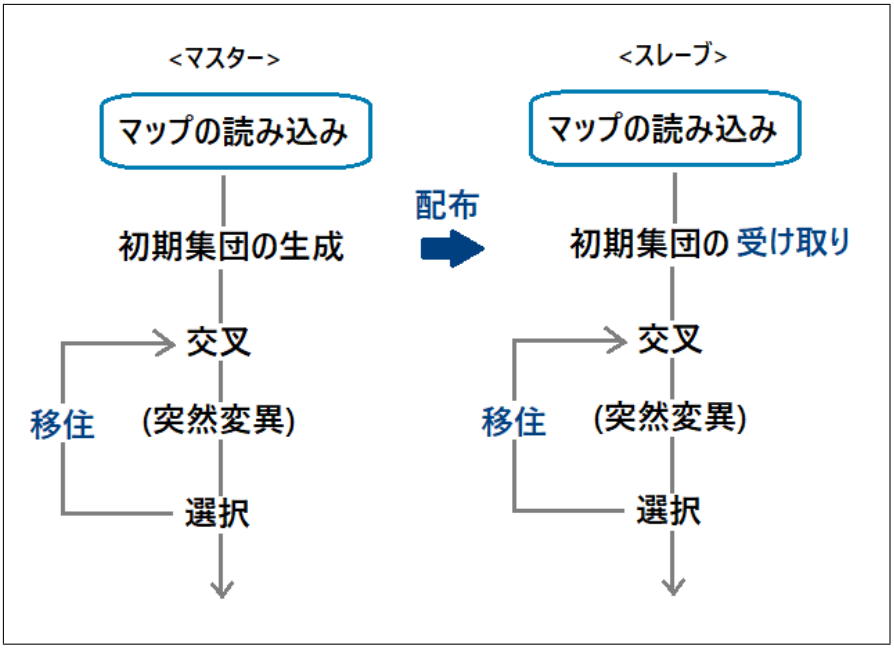

図 3.11: MPI による並列分散 GA のフロー

第4章

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，建築現場における資源の最適な配分による生産性の向上を目指す日程計画を考えた．

特に，提案モデルでは，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・費用などの不確定で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を考える（図 4.1 参照）．

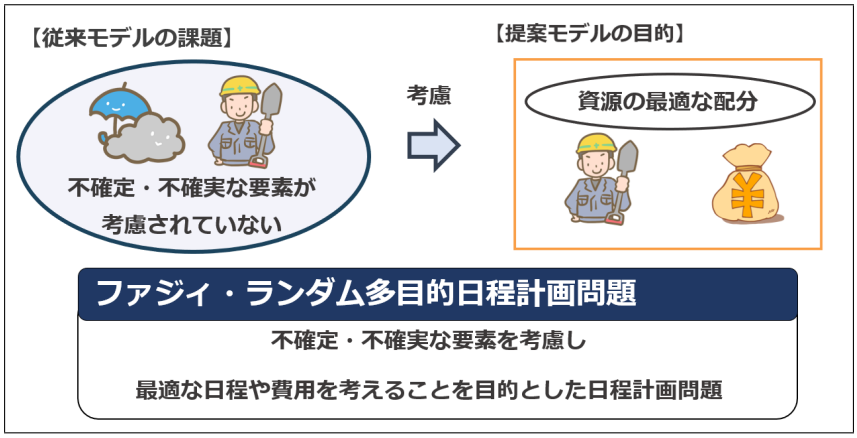

図 4.1: 提案モデルの特徴

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）．

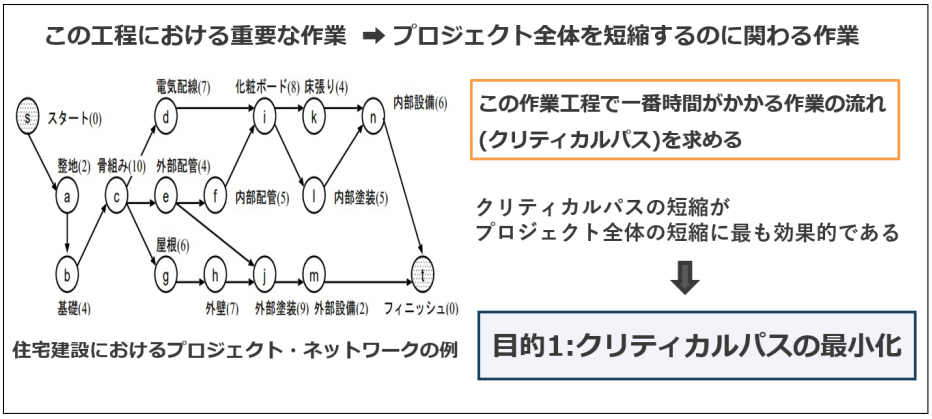

図 4.2: クリティカルパスの所要時間を最小化

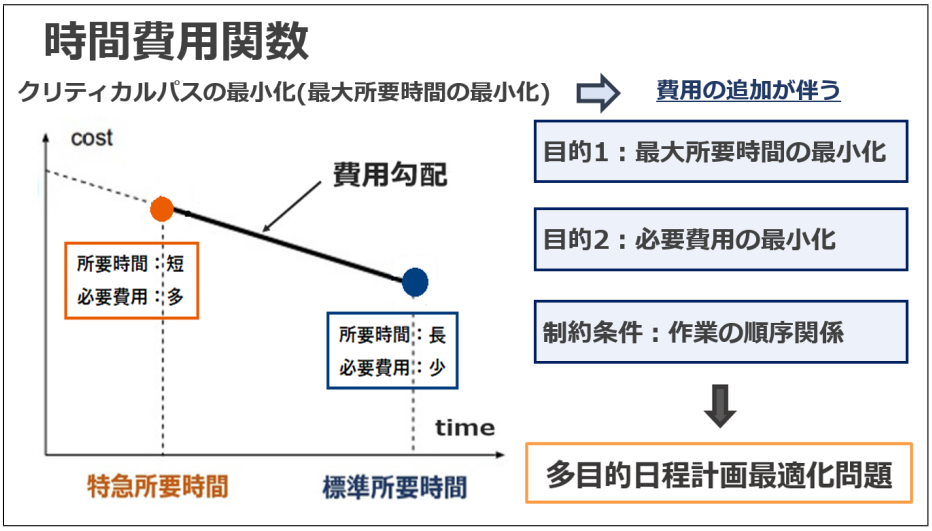

図 4.3: 時間費用関数と多目的最適化問題

目的関数 1 ：最大所要時間の最小化

このような，プロジェクトネットワークも活用し，全作業の順序関係を視覚的に把握する．そこから，工程における重要な作業，つまり，工程全体を短縮するのに関わる作業を求める．提案モデルでは，作業の所要時間の積算で所要時間が最大となる工程をクリティカルパスとし，このクリティカルパス上の作業を重点的に管理することで，工程全体の作業効率の向上を図る．したがって，提案モデルにおける目的関数の一つは，クリティカルパス上の作業所要時間の最小化（最大所要時間の最小化）とした．

目的関数 2：必要費用の最小化

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）．一般的に，ある作業の所要時間を短縮したいと考えたとき，多くの職人さんに依頼するなど，それに伴った費用がかかる．所要時間と費用の関係は，図 4.3 のような時間費用関数で表される．この時間費用関数は，費用をかけない代わり所要時間が長くなる標準所要時間と，その標準所要時間に対して費用をかけること

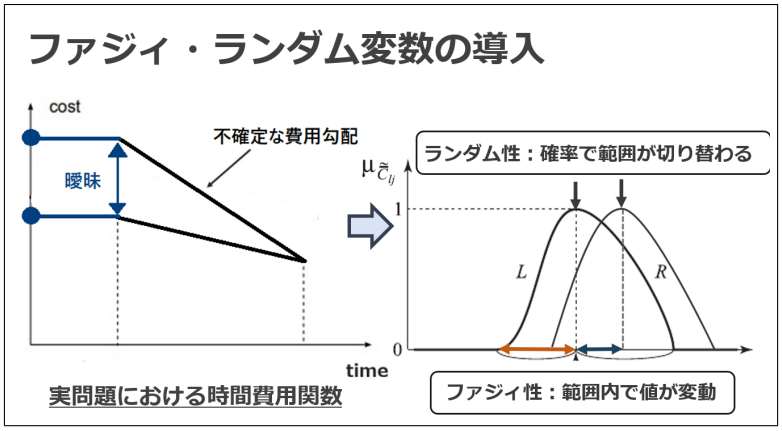

図 4.4: 実問題における時間費用関数

で所要時間をその分短縮することができる特急所要時間の関係を示している．このような，トレードオフの関係にある目的関数を扱う場合，節 3.2 で述べたように，最適解は一つではなくパレート最適フロントとして複数存在する．

ファイジィ・ランダム変数の導入

図 4.3 で説明した時間費用関数は，静的で確定的な値であることを前提として考えられている．(図 4.4 参照)．しかし，実問題を考える場合，所要時間や必要費用は不確定で不確実な要素であるため，図 4.3 の線形な時間費用関数では対応できない．そこで，実問題における時間費用関数は不確定な費用勾配を持つ関数であると考え，それを表現するためにファジィ・ランダム変数を導入する．

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．

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

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職人さん

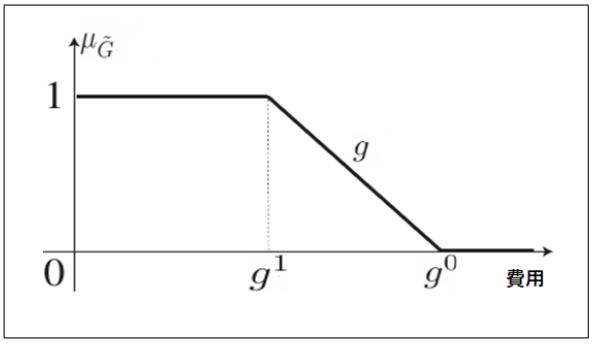

図 4.5: ファジィ目標 $\tilde{G}$ のメンバシップ関数 $\mu_{\tilde{G}}$

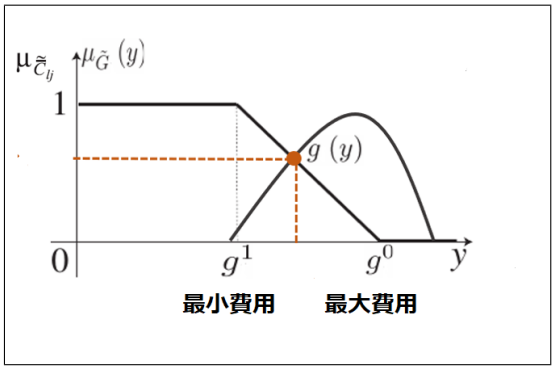

図 4.6: ファジィ目標の設定

（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 [19] [20] [21].

ファジィ目標 $\tilde{G}$ の導入

ファジィランダム変数を含む目的関数に対して，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).

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

ここで，g^0 は最悪値 (最高費用), g^1 は最良値 (最小費用) である．g は狭義単調減少連続関数である（図 4.6 参照）.

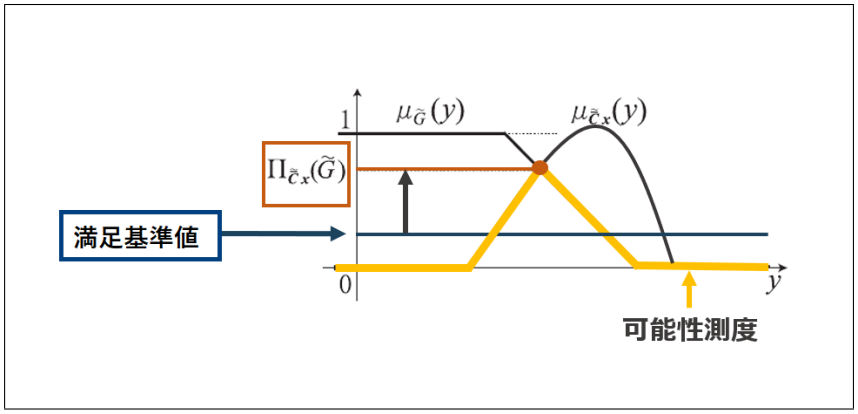

図 4.7: 満足基準値を用いた確率最大化モデル

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 4.7 参照）．

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

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．

$$\left.\begin{array}{rl} \text{minimize} & \max\left\{\sum_{i=1}^{n}\sum_{j\in N_i}\sum_{k=1}^{w} t_{ijk}x_{ij}y_{ik}\right\} \\ \text{maximize} & \Pi_{\tilde{c}_{ik}x_{ij}y_{ik}}(\tilde{G}) \\ \text{subject to} & \\ & \sum_{i=1}^{n}\sum_{j\in N_i} x_{ij} - \sum_{i=1}^{n}\sum_{j\in N_i} x_{ji} = \left\{\begin{array}{l} 1, (i=s)\ , \\ 0, (i\neq s, l)\ , \\ -1, (i=l)\ . \end{array}\right. \\ & x_{ij}\in\{0,1\}, i=1,\ldots,n, j=1,\ldots,n; \\ & y_{ik}\in\{0,1\}, i=1,\ldots,n, k=1,\ldots,w; \end{array}\right\} \tag{4.4}$$

ここで，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．

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

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

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

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，

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

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

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

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．

$$
\left.
\begin{array}{rl}
\text{minimize} & \max\left\{\sum_{i=1}^{n}\sum_{j\in N_i} t_{ijk}x_{ij}y_{ik}\right\} \\
\text{maximize} & F\left(\frac{-\{d^1-L^*(h)\beta^1\}x_{ij}y_{ik}+\mu^*_{\tilde{G}}(h)}{\{d^2-L^*(h)\beta^2\}x_{ij}y_{ik}}\right) \\
\text{subject to} & \\
 & \sum_{i=1}^{n}\sum_{j\in N_i} x_{ij} - \sum_{i=1}^{n}\sum_{j\in N_i} x_{ji} = \left\{\begin{array}{l} 1, (i=s)\ , \\ 0, (i\neq s, l)\ , \\ -1, (i=l)\ . \end{array}\right. \\
 & x_{ij}\in\{0,1\}, i=1,\ldots,n, j=1,\ldots,n; \\
 & y_{ik}\in\{0,1\}, i=1,\ldots,n, k=1,\ldots,w;
\end{array}
\right\} \qquad (4.6)
$$

§ 4.3　提案手法のアルゴリズム

最後に，本研究で提案したモデルのパラメータ設定から解法までのアルゴリズムについてまとめる（図 4.8 参照）．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．

Step1：過去の作業情報の蓄積

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．

Step2：問題設定

作業情報からパラメータ設定に必要なデータを収集できたら，次に問題設定を行う．目的関数に最大所要時間の最小化と必要費用の最小化を設定し，制約条件として作業の順序関係を設定する．クリティカルパスの導出は節 2.1 で説明した方法を用いる．

Step3：作業履歴を活用したパラメータ設定

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のパラメータ $\bar{\beta_{ik}}$,$\bar{\gamma_{ik}}$ と中心値 $\bar{d_{ik}}$ を設定する．左の広がりパラメータ $\bar{\beta_{ik}}$ は作業履歴の中の最小費用を，右の広がりパラメータ $\bar{\gamma_{ik}}$ 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．

Step4：モデルの定式化と等価確定変換

ファイジィ・ランダム変数にパラメータを設定したら，提案モデルを解くための定式化と式の等価確定変換を行う．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 $\mu_{\tilde{G}}$ 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．

提案モデルを解ける形に等価確定変換したら，遺伝的アルゴリズムなどの近似解法を用いた解の導出が可能になる．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．

Step5：並列分散 GA を用いた解法

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．マスターで生成した初期集団を各スレーブ (7 台) へ MPISemd() 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．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．この操作を繰り返し解の探索を行う．

結果: 所要時間とそのときにかかる費用を見積もることができ，各工程に対する職人さんと費用の最適な配分を補助することができる．

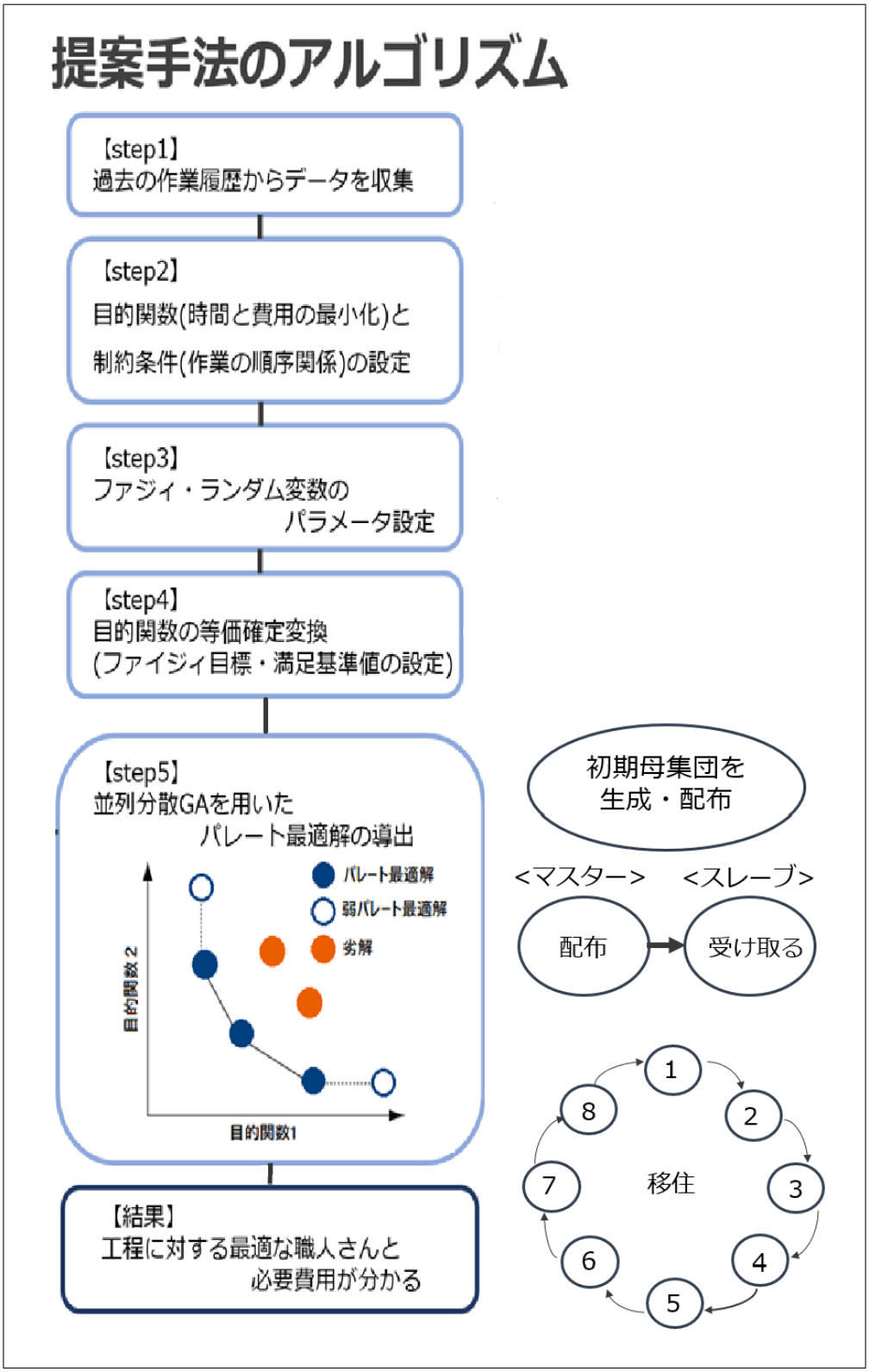

図 4.8: 提案手法の全体的なフロー

第5章

結論ならび今後の課題

本研究では，建築業界における資源の最適な配分による生産性の向上を目的とし，日程計画を考えた．そのなかで，不確定性・不確実性を含む要素に着目し，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った．更に，提案する問題に対しての定式化と等価確定変換，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，

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はいるが，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．よって，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. 閲覧日 2018,6,5.

[2] H. Kwakernaak, “Fuzzy random variable-l,” 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. 閲覧日 2018,12,20.

[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/. 閲覧日 2018,12,20.

[7] 木瀬 洋, “スケジューリング研究の過去・現在・未来”, スケジューリングシンポジウム 2000 講演論文集, 2000.

[8] 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-” 土木学会論文報告集, Vol. 182, pp. 41-58, 1970.

[9] “建築現場を IT 化する「ダンドリ」ツール 6 選”, https://mag.branu.jp /archives/2442. 閲覧日 2018,6,5.

[10] “現場コミュニケーションアプリ Kizuku”, https://www.ctx.co.jp/kizuku2pr/. 閲覧日 2018,6,5.

[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. K. Luhan （ljula. and M. M , Gupta ； On fuzzy stochastic optimization. Fuzzy Sets and Systems, 81, pp. 47- 55, 1996.

[13] M.Gen, R. Cheng, Genetic Algorithms, Engineering Design, John Wiley, Sons, 1997.

[14] 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”

[15] “Raspberry Pi とは”, https://furien.jp/columns/58/. 閲覧日 2018,11,6.

[16] “MPI の環境構築や基本コマンドのまとめ” ,https://qiita.com/kkk627/items/49c9c35301465f6780fa. 閲覧日 2018,11,6.

[17] “MPI を用いた並列プログラミングの概要” ,http://www.cenav.org/raspi4b/. 閲覧日 2018,11,8.

[18] “MPI 並列プログラミング” ,http://www.matlab.nitech.ac.jp/ kazu-k/mpi.html. 閲覧日 2018,11,8.

[19] Hideki Katagiri , “Interactive multiobjective fuzzy random linear programming: Maximization of possibility and probability”

[20] M. Sakawa, I. Nishizaki, H. Katagiri, Fuzzy Stochastic Multiobjective Programming, Springer, 2011.

[21] 椎名孝之, “確率計画法”, 朝倉書店，2015.

付録

A. 1　Hello World を並列実行するソースコード

Raspberry Pi 3 を 8 台で Hello World を並列実行するソースコード A.1 を示す.

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

Raspberry Pi 3 を 8 台で円周率計算を並列分散処理するソースコード A.2 を示す.

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 = 3.14159265358979323846264 3;
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並列分散GAで処理するソースコード

Raspberry Pi 3 を 8 台で巡回セールスマン問題を並列分散 GA で処理するソースコード A.3 を示す.

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(
        MPI_COMM_WORLD, &rank);
60  MPI_Comm_size(
        MPI_COMM_WORLD, &np);
61
62  double fast_rank[np], fast_all;
63  makelist();
64  int distribution[GENOM][NODE+1];
65
66 /* --- 親 (rank=0) --- */
67  if (rank == 0)
68    {
69      start = time(&t1); /* 開始時間取
          得 */
70      srand((unsigned)time(NULL));
71      func(); /* 母集団初期化 */
72
73      /* 母集団の配分 */
74        for (i = 1; i < np; i++) { //i:
            端末(rank)
75          for (j = 0; j < (GENOM); j
              ++) { //j: 遺伝子数
76            for (k = 0; k < NODE+1;
                k++) { //k: 経路
77              distribution[j][k] = result[
                    j].route[k];
78            }
79          }
80          MPI_Send(distribution, (
                GENOM)*(NODE+1),
                MPI_INT,i, 99,
                MPI_COMM_WORLD);
81        }
82    }
83
84 /* --- 子 (rank!=0) --- */
85  else
86    {
87      /* 母集団の受け取り */
88      MPI_Recv(distribution, (GENOM
            )*(NODE+1), MPI_INT, 0,
            99, MPI_COMM_WORLD, &
            status);
89      for (i = 0; i < (GENOM); i++)
            { //i: 遺伝子数
90        for (j = 0; j < NODE+1; j++)
              { //j: 経路
91          result[i].route[j] =
                distribution[i][j];
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          {
113            fast_all = fast;
114              for (j = 1; j < np; j++) {
115          //printf("%d\t%6.2f\n", j,
                 fast_all);
116                MPI_Recv(&fast_rank[j],
                       1, MPI_DOUBLE, j,
                       50,
                       MPI_COMM_WORLD
                       , &status);
117              if (fast_all > fast_rank[j])
118                fast_all = fast_rank[j];
119
120          //printf("%d\t%6.2f\n", j,
                 fast_rank[j]);
121              }
122
123   #if WRITEFILE
124        end = time(&t2); /* 終了時間取
             得 */
125            min = end - start;
126            sec = min / 60;
127            min -= sec * 60;
128
129            printf("%d\t%6.2f\n", i,
                   fast_all);
130        printf("exec time;;; %d:%02d\n
             ", sec, min);
131            //printf("%6.2f\n", fast_all
                 );
132   #endif
133        }
134
135        else
136          {
137            MPI_Send(&fast, 1,
                   MPI_DOUBLE, 0,50,
                   MPI_COMM_WORLD);
138      printf("%6.2f\n", fast);
139        }
140
141        if (np-1)/*0より大きいとき移住
             */
142          MPI_Barrier(
                 MPI_COMM_WORLD);
143           migrate(rank, np, &status); /*
                 移住 */
144      }
145    } //loop end
146   getchar();getchar();
147   #if WRITEFILE
148    //nanosleep(1000*rank*100);
149     printf("%2d> %6.2f: ", rank, result[
           champ].distance);
150     for (i = 0; i < NODE+1; i++) {
151       printf("%2d ", result[champ].route[
             i]);
152    }
153     puts("");
154
155     MPI_Barrier(MPI_COMM_WORLD)
           ;
156   #endif
157     if (rank == 0)
158      {
159        fast_all = fast;
160          for (i = 1; i < np; i++) {
161            MPI_Recv(&fast_rank[i], 1,
                   MPI_INT, i, 50,
                   MPI_COMM_WORLD, &
                   status);
162            if (fast_all > fast_rank[i])
163              fast_all = fast_rank[i];
164          }
165
166          printf("the shortest
                 distance: %.2f\n",
                 fast_all);
167
168
169   #if WRITEFILE
170        //nanosleep(1000*np*100);
171          // printf("exec time;;; %d:%02d
                 \n", sec, min);
172   #endif
173      }
174
175     else
176      {
177        MPI_Send(&fast, 1, MPI_INT, 0,
               50, MPI_COMM_WORLD);
178      }
179     MPI_Finalize(); /* MPI終了 */
180   }
181
182   /* func */
183   void func() {
184   #if DEBUG1
185    puts("--func");
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) +
                  1;
203         } while (town[go].flag); //do-
                while end
204         result[i].route[j] = go;
205         town[go].flag++;
206       } //for(j) end
207       calc_dist(&result[i]); /* 距離計算
              */
208     } //for(i) end
209
210
211  #if DEBUG1
212   puts("--func end");
213  #endif
214  }
215  /* calc_dist */
216  void calc_dist(RESULT *res) {
217   int i, now, go;
218
219     /* 経路が正しいかどうか確認 */
220  #if DEBUG2
221   double judge = 0, dist_judge = 0;
222   for (i = 1; i < NODE; i++) {
223     judge += res->route[i];
224     dist_judge += i;
225   }
226   if (judge != dist_judge) {
227     puts("route error!");
228     exit(1);
229   }
230  #endif
231
232   res->distance = 0;
233   now = 0;
234   for (i = 1; i < NODE+1; i++) {
235     go = res->route[i];
236     res->distance += town[now].dist[
            go];
237     now = go;
238   }
239  }
240
241  /* 選択*/
242  void tournament(int rank) {
243  #if DEBUG1
244   printf("--tournament %d\n", rank);
245  #endif
246   int i, n, x;
247
248   for (i = 0; i < GENOM/2; i++)
249     {
250       n = i;
251       x = GENOM-1-i;
252       if (result[n].distance > result[x].
              distance) {
253         n = x;
254         result[i] = result[n]; /* 代入 */
255       }
256       if (result[n].distance < result[
              champ].distance)
257         champ = n;
258     }
259
260   if (fast > result[champ].distance){
261     fast = result[champ].distance;
262  #if WRITEFILE==0
263     /* 記録表示 */
264     printf("%2d> %6.2f: ", rank, result
            [champ].distance);
265     for (i = 0; i < NODE+1; i++) {
266       printf("%2d ", result[champ].
              route[i]);
267     }
268     putchar('\n');
269  #endif
270   }
271
272   #if DEBUG1
273   printf("--tournament %d end\n",
          rank);
274  #endif
275  }
276
277  /* 部分交叉 */
278  void cross() {
279  #if DEBUG1
280   puts("--cross");
281  #endif
282   int i, j, k, l, a, b, x;
283   int tmp1, tmp2;
284   int flag1[NODE], flag2[NODE]; /*
          parent1,parent2用旗 */
285   RESULT parent1, parent2;
286
287   for (i = 0; i < GENOM/2; i+=2)
288     {
289       /* 交叉するペアを選択 */
290       a = rand() % (GENOM/2);
291       do {
292         b = rand() % (GENOM/2);
293       } while (a == b);
294       parent1 = result[a];
295       parent2 = result[b];
296
297        /* 旗初期化 */
298       for (j = 0; j < NODE; j+=2) {
299         flag1[j] = 1; flag1[j+1] = 1;
300         flag2[j] = 1; flag2[j+1] = 1;
301       }
```

```
302
303         /* 距離の短いルートは交叉しない
                ようにする */
304         for (j = 0; j < NODE; j++) {
305           if (town[parent1.route[j]].dist[
                  parent1.route[j+1]] < 2) {
306             flag1[j] = 0; flag1[j+1] = 0;
307           }
308           if (town[parent2.route[j]].dist[
                  parent2.route[j+1]] < 2) {
309             flag2[j] = 0; flag2[j+1] = 0;
310           }
311         }
312
313          /* 交叉 */
314         x = rand() % (NODE-1) + 1; /*
                切れ目 */
315         for (j = x; j < NODE; j++) {
316           tmp1 = parent1.route[j];
317           tmp2 = parent2.route[j];
318           if (flag1[j] && flag2[j]) {
319             for (k = 1; k < NODE; k
                    ++) {
320               if ((parent1.route[k] ==
                      tmp2) && k!=j &&
                      flag1[k]) {
321                 parent1.route[j] = tmp2;
                        flag1[j] = 0;
322                 parent1.route[k] = tmp1;
                        flag1[k] = 0;
323               }
324               if ((parent2.route[k] ==
                      tmp1) && k!=j &&
                      flag2[k]) {
325                 parent2.route[j] = tmp1;
                        flag2[j] = 0;
326                 parent2.route[k] = tmp2;
                        flag2[k] = 0;
327               }
328             } //for(k) end
329           } //if(j) end
330         } //for(j) end
331
332          /* 距離計算 */
333         calc_dist(&parent1);
334         calc_dist(&parent2);
335          l = GENOM/2+i;
336         result[l] = parent1;
337         result[l+1] = parent2;
338       } //for(i) end
339
340 #if DEBUG1
341       puts("--cross end");
342 #endif
343 }
344
345 /* 突然変異 --位置移動 */
346 void variation(int np) {
347 #if DEBUG1
348  puts("--variation");
349 #endif
350   int i, j, k, l;
351   int x, y, z, tmp;
352   RESULT parent;
353
354   /* 旗初期化 */
355   for (i = 0; i < GENOM; i++) {
356     result[i].flag = 0;
357   }
358
359   k = rand()%10+1;
360   for (i = 0; i < GENOM*k/100; i++)
361     {
362       do {
363         j = rand() % GENOM;
364       } while (result[j].flag);
365       result[j].flag = 1;
366       parent = result[j];
367
368       //l = rand()%15+1;
369       l = 1;
370       for (j = 0; j < l; j++) {
371         do {
372       x = rand() % (NODE-1) + 1;
373         } while (town[x-1].dist[x] < 2)
                ;
374       do {
375         y = rand() % (NODE-1) + 1;
376       } while (x == y || town[y].dist[y
              +1] < 2);
377
378       if (x > y) { /* x < y の関係にす
              る */
379         tmp = x;
380         x = y;
381         y = tmp;
382       }
383       /* 位置移動 */
384       tmp = parent.route[y];
385       for (z = y; z > x; z--) {
386         parent.route[z] = parent.route[z
                -1];
387       }
388       parent.route[x] = tmp;
389       }
390
391       calc_dist(&parent); /* 距離計算
              */
392       result[GENOM-i] = parent;
393     } //for(i) end
394
395 #if DEBUG1
396  puts("--variaion end");
397 #endif
398 }
399
400 /* 移住 */
401 void migrate(int rank, int np,
          MPI_Status *status) {
402 #if DEBUG1
403  printf("%d migrate\n", rank);
404 #endif
```

```
405  int i, j, b, x;
406  int flag[GENOM] = {0};
407  int a; /* 遺伝子数 x MIG_RATE[%]
         */
408  int immigrant[a][NODE+1];
409  a = GENOM*MIG_RATE/100;
410
411  /* 移住者を決める */
412  for (i = 0; i < a; i++) {
413    do {
414      x = rand() % GENOM;
415    } while (flag[x]);
416    flag[x] = 1;
417    for (j = 0; j < NODE+1; j++) {
418      immigrant[i][j] = result[x].route[j];
419    }
420  }
421  b = a * (NODE+1); /* 送信するデー
         タの数 */
422  /* 偶数rank 送信->受信 */
423  if (rank%2 == 0)
424    {
425      /* 送信 */
426      MPI_Send(immigrant, b,
             MPI_INT, rank+1,
427                  10,
                       MPI_COMM_WORLD
                       );
428      /* 受信 */
429      if (rank == 0) {
430        MPI_Recv(immigrant, b,
               MPI_INT, np-1,
431                    20,
                         MPI_COMM_WORLD
                         , status);
432      }
433      else {
434        MPI_Recv(immigrant, b,
               MPI_INT, rank-1,
435                    20,
                         MPI_COMM_WORLD
                         , status);
436      }
437      /* 受信データ格納 */
438      for (i = 0; i < a; i++) {
439        do {
440          x = rand()%(GENOM/2)+(
                 GENOM/2);
441        } while (flag[x]);
442        flag[x] = 1;
443        for (j = 0; j < NODE+1; j++)
               {
444          result[x].route[j] = immigrant
                 [i][j];
445        }
446        calc_dist(&result[x]);
447      }
448    }
449  /* 奇数rank 受信->送信 */
450    else
451      {
452      /* 受信 */
453      MPI_Recv(immigrant, b,
             MPI_INT, rank-1,
454                  10,
                       MPI_COMM_WORLD
                       , status);
455      /* 受信データ格納 */
456      for (i = 0; i < a; i++) {
457        do {
458          x = rand()%(GENOM/2)
                 +(GENOM/2);
459        } while (flag[x]);
460        flag[x] = 1;
461        for (j = 0; j < NODE+1; j
               ++) {
462          result[x].route[j] =
                 immigrant[i][j];
463        }
464        calc_dist(&result[x]);
465      }
466      /* 送信 */
467      if (rank == np-1) {
468        MPI_Send(immigrant, b,
               MPI_INT, 0,
469                    20,
                         MPI_COMM_WORLD
                         );
470      }
471      else {
472        MPI_Send(immigrant, b,
               MPI_INT, rank+1,
473                    20,
                         MPI_COMM_WORLD
                         );
474      }
475    }
476  /* 移住者を決める */
477  for (i = 0; i < a; i++) {
478    do {
479      x = rand() % GENOM;
480    } while (flag[x]);
481    flag[x] = 1;
482    for (j = 0; j < NODE+1; j++) {
483      immigrant[i][j] = result[x].route[j];
484    }
485  }
486  /* 偶数rank 受信->送信 */
487  if (rank%2 == 0)
488    {
489      /* 受信 */
490      MPI_Recv(immigrant, b,
             MPI_INT, rank+1,
491                  30,
                       MPI_COMM_WORLD
                       , status);
492      /* 受信データ格納 */
493      for (i = 0; i < a; i++) {
494        do {
495          x = rand()%(GENOM/2)+(
                 GENOM/2);
496        } while (flag[x]);
```

```
497         flag[x] = 1;
498         for (j = 0; j < NODE+1; j++)
                {
499           result[x].route[j] = immigrant
                [i][j];
500         }
501         calc_dist(&result[x]);
502       }
503       /* 送信 */
504       if (rank == 0) {
505         MPI_Send(immigrant, b,
              MPI_INT, np-1,
506                   40,
                        MPI_COMM_WORLD
                        );
507       }
508       else {
509         MPI_Send(immigrant, b,
              MPI_INT, rank-1,
510                   40,
                        MPI_COMM_WORLD
                        );
511       }
512     }
513   /* 奇数rank 送信->受信 */
514   else
515     {
516       /* 送信 */
517       MPI_Send(immigrant, b,
            MPI_INT, rank-1,
518                 30,
                      MPI_COMM_WORLD
                      );
519       /* 受信 */
520       if (rank == np-1) {
521         MPI_Recv(immigrant, b,
              MPI_INT, 0,
522                   40,
                        MPI_COMM_WORLD
                        , status);
523       }
524       else {
525         MPI_Recv(immigrant, b,
              MPI_INT, rank+1,
526                   40,
                        MPI_COMM_WORLD
                        , status);
527       }
528       /* 受信データ格納 */
529       for (i = 0; i < a; i++) {
530         do {
531           x = rand()%(GENOM/2)+(
                GENOM/2);
532         } while (flag[x]);
533         flag[x] = 1;
534         for (j = 0; j < NODE+1; j++)
                {
535           result[x].route[j] = immigrant
                [i][j];
536         }
537         calc_dist(&result[x]);
538       }
539     }
540   tournament(rank);
541 #if DEBUG1
542   printf("%d migrateend\n", rank);
543 #endif
544 }
545
546
547
548 /* makelist */
549
550 void makelist() {
551   int i, j;
552   double list[NODE][NODE] = {
553  0.00, 7.21, 6.71, 21.47, 10.30, 12.00,
       2.00, 13.60, 15.13, 23.19, 16.00,
       13.34, 13.89, 12.08, 15.03, 19.00,
       9.06, 19.10, 11.40, 7.07, 15.30,
       21.10, 28.16, 13.89, 14.32, 17.46,
       12.04, 14.14, 20.40, 22.83, 20.00,
       7.28, 5.00, 19.31, 12.04, 9.22,
       5.39, 2.00, 27.51, 4.12, 28.64,
       28.46, 17.03, 13.15, 26.68, 14.14,
       27.29, 4.47, 14.04, 11.18,
554  7.21, 0.00, 9.22, 21.93, 5.10, 18.44,
       5.66, 15.65, 11.70, 21.02, 13.42,
       7.07, 18.25, 7.07, 13.04, 16.16,
       5.83, 21.93, 7.62, 13.93, 21.02,
       14.87, 23.09, 6.71, 9.00, 17.69,
       15.52, 12.81, 18.87, 23.35, 13.42,
       8.54, 4.12, 14.04, 10.63, 15.13,
       8.06, 8.49, 22.20, 10.44, 22.80,
       27.46, 9.90, 19.92, 25.06, 17.09,
       25.08, 6.32, 11.18, 8.06,
555  6.71, 9.22, 0.00, 14.76, 8.54, 10.82,
       7.81, 7.07, 10.30, 17.00, 10.44,
       16.28, 9.06, 9.43, 9.22, 13.34,
       13.89, 13.04, 7.81, 8.06, 12.37,
       18.44, 23.71, 15.03, 11.40, 10.77,
       6.32, 8.06, 14.04, 16.12, 18.03,
       1.41, 10.00, 15.62, 6.32, 7.21,
       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,
556 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,
557 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,

558 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,

559 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,

560 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,

561 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,

562 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,

563 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,

564 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,

565 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,

566 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,

567 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,

568 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,

569 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,
570 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,
571 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,
572 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,
573 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,
574 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,
575 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,
576 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,
577 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,
578 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,
579 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,
580 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,
581 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,

582 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,

583 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,

584 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,

585 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,

586 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,

587 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,

588 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,

589 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,

590 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,

591 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,

592 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,

593 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,

594 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,
595     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,
596     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,
597     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,
598     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,
599     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,
600     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,
601     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,
602     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
603   };
604
605
606
607
608   for (i = 0; i < NODE; i++) {
609     for (j = 0; j < NODE; j++) {
610       town[i].dist[j] = list[i][j];
611     }
612   }
613 }
```