クリティカルチェーン法の枠組みにおいて効率的に資源を活用するためのスケジューリング方法

KOGA, Hiroki / 古賀, 裕紀

(開始ページ / Start Page)
1
(終了ページ / End Page)
42
(発行年 / Year)
2015-03-24
(学位授与年月日 / Date of Granted)
2015-03-24
(学位名 / Degree Name)
修士(工学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

2014 年度　修士論文

クリティカルチェーン法の枠組みにおいて効率的に資源を活用するためのスケジューリング方法

SCHEDULING METHODS TO UTILIZE RESOURCES EFFECTIVELY IN THE CRITICAL CHAIN METHOD

法政大学大学院 理工学研究科

システム工学専攻 経営系　修士課程

13R6209　古賀裕紀（こがひろき）

Hiroki KOGA

指導教員：五島洋行 教授

Abstract

We propose approximate methods and a strict method for resolving resource conflicts in the critical chain project management method. The critical chain project management method is a technique to shorten the duration time of a given project, and to decrease the delay rate in the overall project. In the critical chain project management method, the uncertainty of the duration time of each task and time buffers are assumed. This method consists of (1) identifying tasks of a project, (2) collection of time margins, (3) resolution of resource conflicts, (4) classifying tasks according to critical or non-critical tasks, and (5) insertion of time buffers. Effective approaches for four of the five processes already exist. For the remaining unresolved process (3) resolution of resource conflicts, an effective method has yet to be proposed. Hence, we develop three simple approximate solving methods, and improve these using a local search or a genetic algorithm. Methods based on the earliest and latest start times are used. The local search method performs a basic search by swapping the processing order of two arbitrary tasks. The genetic algorithm performs a search by crossing the three solutions of the three simple approximate solving methods. Through numerical experimentation, we found that these solving methods are practical. The average value of the solutions obtained by the three simple methods was improved up to approximately 18% if a local search is used for a 100-task project, while improved approximately 7% if a genetic algorithm is used. However, the computational time of the genetic algorithm is faster than that of the local search. The strict method could find the optimal solution if the number of tasks is smaller than 20.

目次

1 はじめに

1.1 プロジェクト管理手法の歴史

スケジューリングとは, タスクまたはジョブを実行するために時間軸上に資源を割り当てることである[1]. スケジューリングは，明確に認識はしていないとしても，いつの時代にも行われてきた. 例えば, 第一次大戦中に兵站用に開発されたガントチャートがある. 兵站とは戦闘地帯から後方の，軍の諸活動・機関・諸施設を総称したものである. ガントチャート(Gantt chart)は，時間軸上に割り当てられたタスクと資源をグラフで表現したものであり, スケジューリングのために用いられた最初のフォーマルなモデルであったといえる. その後，クリティカルパス法(critical path method: CPM)が続き，これらは現在でも広く使われている. 1950 年代になってスケジューリング問題を解析するためのモデルが提案され，普及し始めた. 解析モデル普及の理由はスケジューリングが重要であることと，スケジューリングに多くの時間を費やさなくてはならず，その時間を短縮するために必要であることにある. したがって，より速く正確なスケジューリングを可能とする解析モデルが求められる. またスケジューリング機能を利用するためには，実施されるタスクおよびそのために使われる資源は自ら設定する必要がある.

大規模な研究開発などのプロジェクトや生産システムなどのスケジュール管理には, ガントチャートが用いられることが多い. また，少しでも遅れたらプロジェクト全体が遅れる工程の集合であるクリティカルパスの把握には，PERT(Program Evaluation and Review Technique)などのアローダイヤグラムが利用されることが多い. 前者のガントチャートは，各工程がいつ実行されるかを明確に記述することができるため，人員や設備の割り当て計画，作業の進渉状況の把握に適している. しかし，工程の実行順序に関する制約関係，つまり先行関係は明確に知ることができないため，ある工程の遅れが後続の工程に与える影響を把握することが難しい. 一方で PERT は，第二次世界大戦中の米海軍のボラリス潜水艦発射弾道ミサイルの開発プロジェクトにおいて, 複雑かつ大規模な兵器開発のスケジュールを適切に管理するために考案された手法である. PERT はプロジェクトや生産システムなどを構成する作業の先行制約関係を表現するために, アローダイヤグラムと呼ばれるネットワーク図を用いる. また，プロジェクトを最短で完遂するために，クリティカルパスに着目する. クリティカルパスは，プロジェクトの開始点から終了点に至る最長経路である. したがって，PERT ではプロジェクトや生産システムなどの納期を短縮するために，クリティカルパス上の工程の作業時間を短縮する. PERT は，アローダイヤグラムによって作業の実行順序を明確に記述するため，クリティカルパスの把握が容易である. 一方で, どの資源をいつ使用するのかの把握が難しく，資源の割り当てや作業の進捗管理には適さないなどの問題がある.

この PERT によく似た手法がクリティカルパス法である. クリティカルパス法は 1957 年にデュポン社で，レミトンランド社のコンサルテーションの下に，化学プラントの保

全停止をスケジューリングするために開発された．PERT との違いは，プロジェクトの各工程処理時間の設定方法のみであり，クリティカルパスの把握が容易である点は同じである．

従来のプロジェクト管理では，クリティカルパス法や PERT を利用し，プロジェクトのグラフを作りグラフ内の最長経路であるクリティカルパスを求める．その後，プロジェクト内で時間的に余裕のない，ボトルネックとなるクリティカルパスに着目し，スケジューリングを行っている．しかし実際には，クリティカルパスよりも作業員や機械設備などの資源の不足や競合がボトルネックとなっていることが多い．資源が競合すると競合した工程を予定通りに処理することができないため，プロジェクトに遅れを引き起こす可能性が高い．また，プロジェクト内の各工程の作業が初期のスケジュール通りに進行することを前提としているため，作業者は各工程の作業を時間内に達成するために，あらかじめ余裕を持たせた作業時間を申告することが多い．その結果，無駄な余裕時間が多いという問題点がある．これらの問題を，本研究で扱うクリティカルチェーン法では解決することができる．

1.2　クリティカルチェーン法

クリティカルチェーン法[2]とは，Goldratt が提唱した制約理論[3]に基づくプロジェクト管理手法の一種である．この手法の目的は納期遅れの防止とメイクスパン，つまりプロジェクト全体の終了時刻の短縮を両立することである．特徴は大きく分けて二つあり，一つは資源競合を考慮に入れてスケジューリングを行うことである．もう一つは，工程が予定より遅れる可能性があることを前提とし，処理時間の不確実性を考慮に入れていることである．

クリティカルチェーン法では，資源競合を解決するために，プロジェクトの計画段階で資源競合を考慮に入れた上でクリティカルパスを求める．クリティカルチェーン法において，資源競合を考慮に入れたクリティカルパスのことをクリティカルチェーンと呼ぶ．また，処理時間の不確実性を考慮するために，各々の工程を 50%の確率で処理終了できる時間設定を行い，その分プロジェクトの決まった位置に仮想の工程であるバッファを，クリティカルチェーンを基準として挿入することで処理の遅れを吸収する．このことによって，申告された余裕を持たせた作業時間の余裕時間分を削り，バッファを挿入することによって適切な余裕時間を設定することができる．これらの特徴によって，クリティカルチェーン法は従来のプロジェクト管理手法よりも優れた結果を残している．

クリティカルチェーン法を利用して結果を出した実例を文献[4]より二つ挙げる．

一つ目は，ハリス社のウエハ工場の立ち上げである．米国のハリス社では航空機や軍事産業，放送機器向けの半導体を製造している．同社はマウンテントップ工場において世界初の 8 インチ・ウエハ生産を行う新工場の立ち上げにクリティカルチェーン法を採

用した．プロジェクトの要件は次のようなものであった．

- 世界初の 8 インチ・ウエハ工場の建設と立ち上げを行う
- 新しい材料を使用する
- 新しいオートメーション設備を導入する
- 新規工場を建設し，2 倍の生産能力を確保する
- 納期はできるだけ早く

以上の五つの要求を満たすため，ハリス社ではクリティカルチェーン法の採用を決め，Goldratt の指導を受けながらプロジェクトを推進した．その成果は目覚ましいものであり，以下のような結果となった．

① 工場の整地作業から最初のウエハ出荷までの期間を，従来の 36 ヶ月から 18 ヶ月に半減させた．余裕時間の没収をした上でスケジューリングを行った結果である．

② 製造ラインの立ち上げからフル生産 90%の稼働率までの立ち上げ期間を，従来の 13 ヶ月から 1 ヶ月に短縮した．これには，クリティカルチェーン法によるネットワーク作成手法を使うことにより，管理する作業の数を約 2000 から 120 へと激減させたことが，大きく寄与している．

二つ目は，バルフォア・ビューティー社の道路建設である．バルフォア・ビューティー社英国の鉄道建設会社で，主に道路建設を請け負っている．この事例で取り上げたプロジェクトの要件は，次の三つである

- 8km の道路設計と建設をする
- 3500 万ポンドの予算内に収める
- 124 週間の期間内に収める

契約後，計画を厳守するためにクリティカルチェーン法の適用を同社は決めた．従来あったスケジュールをクリティカルチェーン法で立て直したところ，124 週から 88.5 週へ短縮可能であることが明らかになった．さらに実際工事を実施してみると，計画の 88.5 週より 9.5 週も短い 79 週で完成した．つまり，プロジェクトに挿入したバッファを十分残して完成した．

以上の実例から，クリティカルチェーン法が従来のプロジェクト管理手法に比べて，納期短縮に優れた手法であるといえる．

1.3　先行研究と目的

クリティカルチェーン法の運用手順は，1．タスクの洗い出し，2．余裕時間の没収，3．資源競合の解消，4．工程の分類とネック工程の検出，5．時間バッファの挿入，の五つに分けることができ，各手順の詳細は 3 章で説明する．これらの手順の内，手順 3，4，5 は計算機によって計算が可能である．五島ら[5]では，このうちの 4，5 を max-plus 代数で定式化することに成功している．max-plus 代数とは，max 演算と+演算を基本演算とした代数系である．この代数系は，事象駆動システムの振る舞いを表現するのに適

している．事象駆動システムとは情報通信システムや経営システムなどのように，事象の生起によってシステム内の状態の変化が引き起こされるシステムのことである．プロジェクトはこの事象駆動システムに当たるため，クリティカルチェーン法の枠組みを max-plus 代数で表現することは有効である．また，max-plus 代数でシステムを表現し計算することによって，バッファの挿入の計算を簡易に表現し，かつ高速にネック工程の検出とバッファの挿入の計算を行える．

しかし，3 の資源競合の解消に関しては，バッファの挿入を考慮しない場合の資源競合の解消問題の定式化や近似解法が，文献[6][7]などで提案されているが，クリティカルチェーン法のための，バッファの挿入を考慮に入れた資源競合の解消問題に関する，具体的な近似解法の提案は行われていない．そこで本研究では，バッファの挿入を考慮に入れた資源競合の解消問題を解くための近似解法を提案する．また計算実験を行い，提案する近似解法が実用的かどうか検証する．加えて厳密解を求める手法も提案し，その計算時間と計算可能な工程数を調べ実用性を検証する．

1.4　論文構成

本論文は，全 7 章で構成される．

2 章では，max-plus 代数に関して，具体的な演算子の定義と具体例を示す．3 章では，クリティカルチェーン法の特徴や有用性を詳しく説明し，具体的な運用手順を説明する．4 章では，資源競合の解消のための近似解法と，プロジェクトの工程数が少ない場合に厳密解を求めるための手法を提案する．5 章では，提案手法による計算実験を行い考察し，提案手法の有用性を示す．6 章では，本論文のまとめを行う．

2 Max-plus 代数

max-plus 代数とは，max 演算と+演算を基本演算とした，事象駆動システムの振る舞いを表現するのに適している代数系である．事象駆動システムとは情報通信システムや経営システムなどのように，事象の生起によってシステム内の状態の変化が引き起こされるシステムのことである．プロジェクトも事象駆動システムに含めることができる．また本研究では，クリティカルチェーン法の枠組みの中のネック工程の検出とバッファの挿入の計算に，この max-plus 代数を採用する．これは max-plus 代数で表現し計算することによって，バッファの挿入の計算を簡易に表現し，かつ高速にネック工程の検出とバッファの挿入の計算を行えるからである．

本章では，この max-plus 代数の具体的な使い方について説明する．

2.1　事象駆動システムに有効である理由

事象駆動システムを表現する場合なぜ max-plus 代数が有効であるかを説明する．事象駆動システムの典型的な事象生起条件の一つに，「複数の事象の同期」がある．ある事象はそれを生起させるための先行制約をすべて満たせば，生起できるという意味である．工程の開始や完了を事象とみなせば，この条件は「先行する工程の処理が完了すれば後続工程の処理を開始することができる」，と言い換えることができる．この場合，先行工程の完了時刻を$x_1, x_2, \cdots, x_n$，後続工程の開始時刻をyとすると，

$$y = \max(x_1, x_2, \cdots, x_n) \tag{1}$$

となり，事象の生起時刻を max 演算で数式表現が可能となる．このことから，「複数の事象の同期」を max 演算で表現することが適していることが分かる．

また複数の事象生起に時間的な関係を導入すると，「事象生起後，一定時間が経過した後，別の事象が生起する」という条件も必要となる．この条件も工程の開始や完了を事象とみなせば，「開始した工程が一定時間経過して作業が完了する」と言い換えることができる．この条件は工程開始時刻をx，工程完了時刻をy，作業に必要な時間をdとすると，

$$y = x + d \tag{2}$$

となり事象生起時刻を+演算で表現できる．

以上のことから，max 演算および+演算が事象駆動システムの事象生起条件を数式表現することに有効であることが分かる．

2.2　演算記号の定義

スカラ演算の場合の演算子を定義する．実数全体を$\boldsymbol{R}$とし，$\boldsymbol{R}_{\max} = \boldsymbol{R} \cup \{-\infty, +\infty\}$，$x, y, z \in \boldsymbol{R}_{\max}$と定義する．max-plus 代数系では，加算$\oplus$と乗算$\otimes$を

$$x \oplus y = \max(x, y), \tag{3}$$

$$x \otimes y = x + y \tag{4}$$

と定義する．演算子の優先順位は，通常の代数系を同様に⊗は⊕よりも高いとする．

また，次の五つの公理が存在する．

・交換法則

$$\begin{aligned} x \oplus y &= y \oplus x, \\ x \otimes y &= y \otimes x. \end{aligned} \tag{5}$$

・結合法則

$$\begin{aligned} (x \oplus y) \oplus z &= x \oplus (y \oplus z), \\ (x \otimes y) \otimes z &= x \otimes (y \otimes z). \end{aligned} \tag{6}$$

・分配法則

$$x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z). \tag{7}$$

さらに，通常の代数系での 0 と 1 に相当するゼロ元と単位元をそれぞれ$\varepsilon(=-\infty), e(=0)$と表記すると，以下の三つの公理を満たす．

$$x \oplus \varepsilon = \varepsilon \oplus x = x, \tag{8}$$

$$x \otimes e = e \otimes x = x, \tag{9}$$

$$x \otimes \varepsilon = \varepsilon \otimes x = \varepsilon. \tag{10}$$

次に，演算子∧， ⊙を

$$x \wedge y = \min(x, y), \tag{11}$$

$$x \odot y = -x + y \tag{12}$$

と定義する．これらの演算子の優先順位の扱いは，∧は⊕，⊙は⊗と同等である．これらの演算子は，max 演算と+演算での表現が困難な箇所で利用することになる．

集合$\boldsymbol{R}_{\max}$に$+\infty$を含めているのは，式(12)において$x = \varepsilon$のときに$-x = +\infty$となり，必要であるからである．また，$x, y = \varepsilon$のとき式(12)は式(10)より，$x \odot y = -x + y = (+\infty) \otimes \varepsilon = \varepsilon$となる．

次に，行列演算はスカラ演算と演算子の定義が異なるため，行列演算の場合の演算子を定義する．ただし，演算子の優先順位や式(5)~(7)の公理はスカラ演算と同様である．また，行列の要素の演算の場合は式(3)，(4)と式(8)~(12)もスカラ演算と同様である．

$\boldsymbol{X}, \boldsymbol{Y} \in \boldsymbol{R}_{\max}^{m \times n}$，$\boldsymbol{Z} \in \boldsymbol{R}_{\max}^{n \times p}$とし，さらに

$$\oplus_{k=1}^{n} x_k = x_1 \oplus x_2 \oplus \cdots \oplus x_n, \tag{13}$$

$$\wedge_{k=1}^{n} x_k = x_1 \wedge x_2 \wedge \cdots \wedge x_n = \min(x_1, x_2, \cdots, x_n) \tag{14}$$

とすると，列演算はスカラ演算は

$$[\boldsymbol{X} \oplus \boldsymbol{Y}]_{ij} = [\boldsymbol{X}]_{ij} \oplus [\boldsymbol{Y}]_{ij}, \tag{15}$$

$$[\boldsymbol{X} \otimes \boldsymbol{Z}]_{ij} = \oplus_{l=1}^{n} ([\boldsymbol{X}]_{ij} \otimes [\boldsymbol{Z}]_{ij}) = \max_{l=1,\cdots,n} ([\boldsymbol{X}]_{il} + [\boldsymbol{Z}]_{lj}) \tag{16}$$

と定義する．また，

$$[\boldsymbol{X} \odot \boldsymbol{Z}]_{ij} = \wedge_{l=1}^{n} ([\boldsymbol{X}]_{ij} \odot [\boldsymbol{Z}]_{ij}) = \min_{l=1,\cdots,n}(-[\boldsymbol{X}]_{il} + [\boldsymbol{Z}]_{lj}), \tag{17}$$

$$\boldsymbol{X}^{\mathrm{T}} \backslash \boldsymbol{Z} = \boldsymbol{X} \odot \boldsymbol{Z} \tag{18}$$

とする．

2.3　演算記号の使用例

2.2 章で定義した演算記号の具体的な計算例を以下に示す．

$\mathbf{X} = \begin{bmatrix} e & \varepsilon & 1 \\ 2 & 3 & 4 \end{bmatrix}$, $\boldsymbol{Y} = \begin{bmatrix} -5 & 6 & -7 \\ 8 & -9 & 10 \end{bmatrix}$, $\boldsymbol{Z} = \begin{bmatrix} -11 & 12 \\ 13 & -14 \\ 15 & 16 \end{bmatrix}$とすると，

$$\boldsymbol{X} \oplus \boldsymbol{Y} = \begin{bmatrix} e & \varepsilon & 1 \\ 2 & 3 & 4 \end{bmatrix} \oplus \begin{bmatrix} -5 & 6 & -7 \\ 8 & -9 & 10 \end{bmatrix} = \begin{bmatrix} e \oplus -5 & \varepsilon \oplus 6 & 1 \oplus -7 \\ 2 \oplus 8 & 3 \oplus -9 & 4 \oplus 10 \end{bmatrix} = \begin{bmatrix} e & 6 & 1 \\ 8 & 3 & 10 \end{bmatrix}$$

$$\boldsymbol{X} \otimes \boldsymbol{Z} = \begin{bmatrix} e & \varepsilon & 1 \\ 2 & 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} -11 & 12 \\ 13 & -14 \\ 15 & 16 \end{bmatrix}$$

$$= \begin{bmatrix} e \otimes (-11) \oplus \varepsilon \otimes 13 \oplus 1 \otimes 15 & e \otimes 12 \oplus \varepsilon \otimes (-14) \oplus 1 \otimes 16 \\ 2 \otimes (-11) \oplus 3 \otimes 13 \oplus 4 \otimes 15 & 2 \otimes 12 \oplus 3 \otimes (-14) \oplus 4 \otimes 16 \end{bmatrix}$$

$$= \begin{bmatrix} -11 \oplus \varepsilon \oplus 16 & 12 \oplus \varepsilon \oplus 17 \\ -9 \oplus 16 \oplus 19 & 14 \oplus -11 \oplus 20 \end{bmatrix} = \begin{bmatrix} 16 & 17 \\ 19 & 20 \end{bmatrix}$$

$$\boldsymbol{X}^{\mathrm{T}} \backslash \boldsymbol{Z} = \boldsymbol{X} \odot \boldsymbol{Z} = \begin{bmatrix} e & \varepsilon & 1 \\ 2 & 3 & 4 \end{bmatrix} \odot \begin{bmatrix} -11 & 12 \\ 13 & -14 \\ 15 & 16 \end{bmatrix}$$

$$= \begin{bmatrix} -e \otimes (-11) \bigwedge -\varepsilon \otimes 13 \bigwedge -1 \otimes 15 & -e \otimes 12 \bigwedge -\varepsilon \otimes (-14) \bigwedge (-1) \otimes 16 \\ -2 \otimes (-11) \bigwedge (-3) \otimes 13 \bigwedge (-4) \otimes 15 & -2 \otimes 12 \bigwedge (-3) \otimes (-14) \bigwedge (-4) \otimes 16 \end{bmatrix}$$

$$= \begin{bmatrix} \min(-11, -\varepsilon, 14) & \min(12, -\varepsilon, 15) \\ \min(-13, 10, 11) & \min(10, -17, 12) \end{bmatrix} = \begin{bmatrix} -11 & 12 \\ -13 & -17 \end{bmatrix}$$

となる． ⊗や⊙の行列の演算において，$m \times n$行列と$n \times p$行列の演算結果は，一般の代数系の行列の積と同様に$m \times p$行列となる．

次に，実際のプロジェクトでは max-plus 代数がどのように利用可能なのかの簡単な具体例を挙げる．ここでは，プロジェクトの各工程の処理の終了時刻を計算していく手順を示す．図 1 はプロジェクトの具体例である．このプロジェクトで利用する文字を

- x_i: 工程iの終了時刻
- d_i: 工程 i の実行時間
- u_i: 外部入力 i からの作業開始時刻

と定義する．外部入力 i の開始時刻u_iは，プロジェクト全体の中で工程処理を開始する箇所が複数存在するときに，各外部入力で異なる時刻に作業を開始することを可能にす

る．

図 1 のプロジェクトの終了時刻はx_3であり，

$$x_1 = d_1 + u_1 = d_1 \otimes u_1 \tag{19}$$

$$x_2 = d_2 + u_2 = d_2 \otimes u_2 \tag{20}$$

$$x_3 = \max(x_1, x_2) + d_3 = (x_1 \oplus x_2) \otimes d_3 \tag{21}$$

の順に計算を行うことで，外部出力時刻を求めることができる．

この例題の終了時刻を具体的な数値で解く．$[d_1 \quad d_2 \quad d_3]^{\mathrm{T}} = [1 \quad 2 \quad 3]^{\mathrm{T}}$，$[u_1 \quad u_2]^{\mathrm{T}} = [0 \quad 0]^{\mathrm{T}}$とすると，

$$x_1 = d_1 + u_1 = 1 \otimes 0 = 1 \tag{22}$$

$$x_2 = d_2 + u_2 = 2 \otimes 0 = 2 \tag{23}$$

$$x_3 = \max(x_1, x_2) + d_3 = (1 \oplus 2) \otimes 3 = 5 \tag{24}$$

となり，プロジェクトの終了時刻は 5 となる．

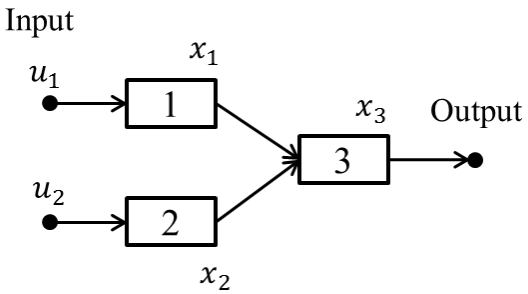

図 1: 2 入力 1 出力のプロジェクト

3 クリティカルチェーン法

本章では，クリティカルチェーン法の特徴を説明し，具体的な運用手順を説明する．

3.1 目的と特徴

クリティカルチェーン法の目的は納期遅れの防止とメイクスパン，つまりプロジェクト全体の終了時刻の短縮を両立することである．この手法の特徴は大きく分けて二つあり，一つは工程が予定より遅れる可能性があることを前提とし，処理時間の不確実性を考慮に入れていることである．そしてもう一つは，資源競合を考慮に入れていることである．

最初に，処理時間の不確実性を考慮に入れる方法について説明する．クリティカルチェーン法では，処理時間の不確実性を考慮に入れるために各々の工程を 50%の確率で処理終了できる時間設定を行い，その分図 2 のようにプロジェクトのいたるところに仮想の工程であるバッファを挿入することで各工程処理の遅れを吸収する．

バッファを挿入する理由は，各工程の作業に余裕を持たせたとしても，プロジェクトのスケジュールは遅れる可能性が高いからである．従来のクリティカル･パス法や PERT 等のプロジェクト管理手法では，プロジェクト内の各工程の作業が，初期のスケジュール通りに進行することを前提としている．このため，作業者は各工程の作業を時間内に達成するために，あらかじめ余裕を持たせた作業時間を申告することが多い．その申告時間は作業着手の先延ばしなどで時間を無駄に消費してしまう可能性が高い．その原因は以下の三つであると言われている[4].

- 学生症候群

提出期限が先のレポートに対して，多くの学生は提出期限の直前までそのレポートに着手しない．この現象を学生症候群と呼ぶ．プロジェクトの場合でも，作業者は「時間に余裕があるのですぐ着手する必要がない」と考えることがある．その結果，作業に想定以上の時間がかかってしまったとしても，すでに余裕時間を消費してしまっているので後れを吸収することができず，その結果プロジェクトに遅れが発生する．

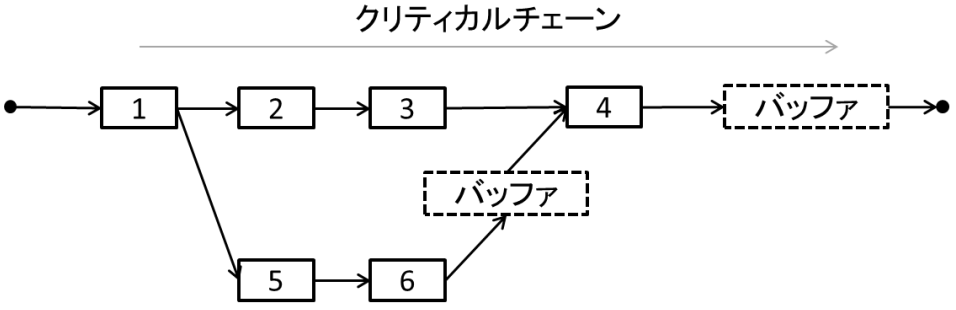

図 2: バッファを挿入したプロジェクトの例

● パーキンソンの法則

パーキンソンの法則とは, 用意された時間やカネはすべて消費されるという法則のことである．例えば，ある工程が早く終了しても，余った時間を必要のないことなどに消費し，最終的には事前のスケジュールの通りに次の工程へと引き渡す．しかし，次の工程の作業者の処理に遅れが生じた場合には, その遅れはプロジェクト全体の遅れにつながる可能性がある．このように，工程の処理が遅くなる被害は受けるが，工程が早く終わる恩恵は受けられない．

● マルチタスク

マルチタスクとは，一人の作業者が複数の工程を掛け持ちすることである．このような作業者は，各プロジェクトのマネージャからプレッシャーをかけられると，緊急でやらなければならない工程から片付けざるを得なくなり, 複数のプロジェクトの仕事を断片的に処理することになる．このように頻繁に処理工程の切り替えを行った場合，各工程の片付けや準備,「どこまで進んでいるのか」といったことを思い出すための時間が余計にかかり，複数のプロジェクトすべてが遅れる原因となる．

以上の原因より，工程の処理時間は大きく余裕を持って設定し，時間を無駄に消費している可能性が高く，その上で，工程処理が予定より遅れてしまうことが多いと考えられる．そこで，各工程の処理予定時間に余裕を持たせるよりも，バッファを挿入することによって時間に余裕を持たせた方が納期遅れの防止になると考えたのが, クリティカルチェーン法である．

次に，資源競合の考慮がなぜ必要なのかについて説明する．クリティカルチェーン法が発案される以前は, プロジェクト内で時間的に余裕のないボトルネックとなるクリティカルパスに着目しスケジューリングを行っていた．しかし実際には，作業員や機械設備などの資源の不足や競合がボトルネックとなっていることが多い. 資源が競合すると競合した工程を予定通りに処理することができないため, プロジェクトに後れを引き起こす可能性が高い. 資源が人の場合はマルチタスクによってプロジェクトに後れを引き起こす.

そこで，クリティカルチェーン法では資源競合を解決するために，プロジェクトの計画段階で競合を起こしている工程のスケジュールを変更する．これによって，クリティカルパスが競合を考慮に入れる前に比べて変化する場合がある. クリティカルチェーン法では, 資源競合を考慮に入れたクリティカルパスのことをクリティカルチェーンと呼ぶ．資源競合を解消する手法が本研究のテーマである．

クリティカルチェーン法には以上のような目的と特徴がある．3.2 章以降は，クリティカルチェーン法を具体的にプロジェクトにどう適用するのかについて説明する．

3.2　文字の定義

クリティカルチェーン法ついて説明する前に，主要な文字を定義する．これらは次章以降でも利用する．

- n：工程数
- q：外部入力数
- p：外部出力数
- $\boldsymbol{F}_0 \in \boldsymbol{R}_{\max}^{n\times n}$：$(j,i)$に先行関係がある場合は$[\boldsymbol{F}_0]_{ij} = e$で，枝がない場合は$\varepsilon$
- $\boldsymbol{B}_0 \in \boldsymbol{R}_{\max}^{n\times q}$：外部入力$j$につながる工程$i$が存在すれば$[\boldsymbol{B}_0]_{ij} = e$，それ以外の場合は$\varepsilon$
- $\boldsymbol{C}_0 \in \boldsymbol{R}_{\max}^{p\times n}$：工程$j$が外部出力$i$につながるならば$[\boldsymbol{C}_0]_{ij} = e$，それ以外の場合は$\varepsilon$
- $\boldsymbol{d} \in \boldsymbol{R}^n$：各工程の実行時間を表すベクトル
- $\boldsymbol{P}$：diag($\boldsymbol{d}$)
- $\boldsymbol{u} \in \boldsymbol{R}^q$：各外部入力の開始時刻のベクトル

$\boldsymbol{F}_0, \boldsymbol{B}_0, \boldsymbol{C}_0$はそれぞれ隣接行列，外部入力行列，外部出力行列と呼ばれる．

3.3　タスクの洗い出し

3.2 章で紹介した文字の内, 工程数n, 隣接行列$\boldsymbol{F}_0$, 外部入力行列$\boldsymbol{B}_0$, 外部出力行列$\boldsymbol{C}_0$, 外部入力の開始時刻$\boldsymbol{u}$を明確にする．各工程の実行時間$\boldsymbol{d}$に関しては，次節の「余裕時間の没収」という手順で設定する．また，次の「余裕時間の没収」の手順のために，余裕時間を持たせた各工程の実行時間をタスクの洗い出しで設定する．

具体的に, どのようにして設定するのかを, 図 3 を用いて具体例を挙げる. 工程数nは単純にグラフの工程数を見て，$n = 3$とする．隣接行列$\boldsymbol{F}_0$は，工程 1 と 2 が工程 3 との間に先行関係があるので，$[\boldsymbol{F}_0]_{31} = e$，$[\boldsymbol{F}_0]_{32} = e$としてそれ以外の$\boldsymbol{F}_0$の要素は$\varepsilon$となり，$\boldsymbol{F}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ e & e & \varepsilon \end{bmatrix}$となる．$\boldsymbol{B}_0$は工程 1 と 2 に外部入力が存在するので$\boldsymbol{B}_0 = \begin{bmatrix} e & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon \end{bmatrix}$，外部入力開始時刻はすべて 0 ならば$\boldsymbol{u} = [0 \quad 0]^{\mathrm{T}}$. $\boldsymbol{C}_0$は外部出力が工程 3 とつながっているので，$\boldsymbol{C}_0 = [\varepsilon \quad \varepsilon \quad e]$となる．余裕時間を持たせた各工程の実行時間は，この時点では設定する必要はない．

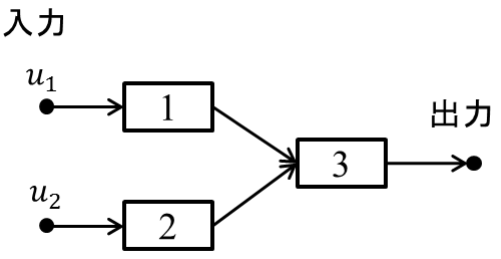

図 3: 2 入力 1 出力のプロジェクト

3.4　余裕時間の没収

クリティカルチェーン法では，プロジェクト全体の納期短縮を図ることを目的に，余裕時間をカットした実行時間を各工程に設定する．本章では何を基準として余裕時間を没収するかについて説明する．

製造ラインや事務処理などの日常的に繰り返される作業の場合，実行時間のばらつきは正規分布とすることが多い．この場合，図 4 のように中央値を基準として，最小値と最大値が左右の等距離にある．つまり，50%の確率で作業が終わるときの実行時間の見積り値は中央値となる．一方で，プロジェクト作業の実行時間が正規分布に従う例は稀で，ほとんどの作業の実行時間は図 5 のようなベータ分布に従う．その理由は，プロジェクトの作業は，日常の繰り返し作業とは異なり，毎回異なる作業の場合がほとんどであり，実行時間を推定するのが難しいからである．したがって，プロジェクト型業務のスケジューリングでは，実行時間の不確実性が大きいため，正規分布の仮定の下では実行時間の見積もりが適切に行えない可能性が高い．

作業担当者は 90%程度の確率で作業が終了するように実行時間を見積もることが多いため，それを踏まえた上でプロジェクトの作業の実行時間をベータ分布に従って，50%の確率で作業が終了するように実行時間$\boldsymbol{d}$を設定する．90%程度の確率で工程が終了する時間を HP(Highly Possible)とよび，50%程度で工程が終了する時間を ABP(Aggressive But Possible)と呼ぶ．本研究では，ABP は HP の 1/3 として実行時間$\boldsymbol{d}$を設定する．

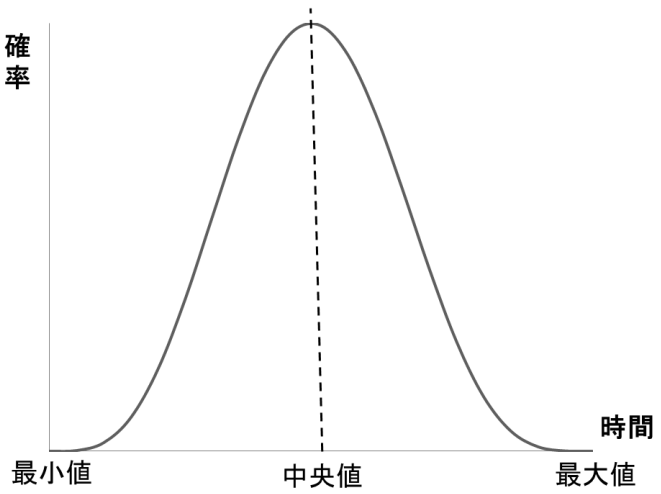

図 4: 正規分布の例

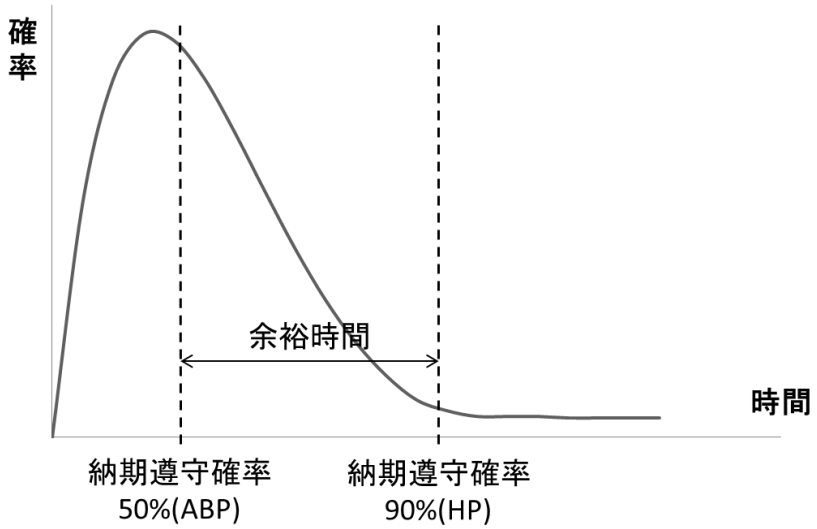

図 5: ベータ分布の例

3.5 資源競合の解消

3.1 章で説明したとおり，資源競合とは作業者や機械設備などの資源が，同時刻に複数工程を処理するようなスケジュールになっている状況のことである．資源競合が発生すると，競合が発生している工程は予定通りに処理を開始できないので，プロジェクトに遅れを起こす可能性が高い．資源競合の解消とは，複数の工程が一つの資源を同時に占有しないようにすることである．

競合の発生と解消の例を図 6 のような例題を用いて示す．工程 1, 4, 5 を資源 1 に，工程 2, 3 を資源 2 に割り当て，各工程の実行時間を工程 1 から順に 3, 4, 5, 2, 1 であるとすると，入力は以下のようになる．

$n = 5,$

$$\boldsymbol{F}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & e & \varepsilon \end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix} 3 \\ 4 \\ 5 \\ 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{r} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \\ 1 \end{bmatrix}.$$

このとき，工程 2 と工程 3 に同じ資源を割り当てているが，同時刻に実行が可能である．図 6 で資源競合を考慮に入れずにガントチャートを組むと， 図 7 のようになる．このガントチャートでは，同一資源を利用する工程 2，3 を時間 4 から 5 で同時に処理してしまっているため，資源競合が発生している．このような状態の場合，工程 2 の開始時刻を工程 3 が終了する時刻まで右にずらし，図 8 のようにすることによって資源競合を解消する．また，図 9 のようにガントチャートを各資源が処理する工程別に整理すると，より分かりやすくなるだろう．

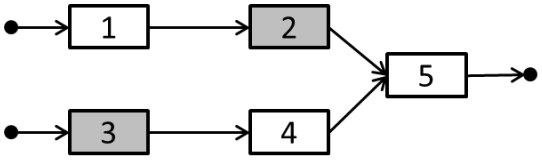

図 6: 資源競合解消前のプロジェクト

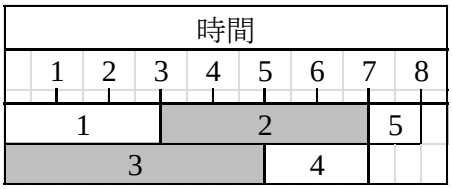

図 7: 資源競合の発生例

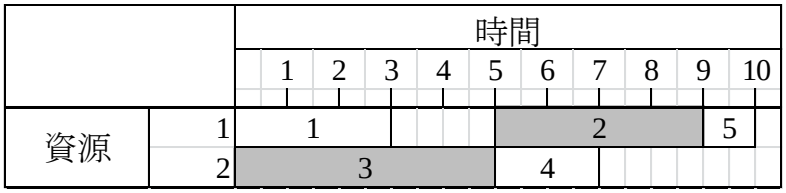

図 8: 資源競合解消後のガントチャート

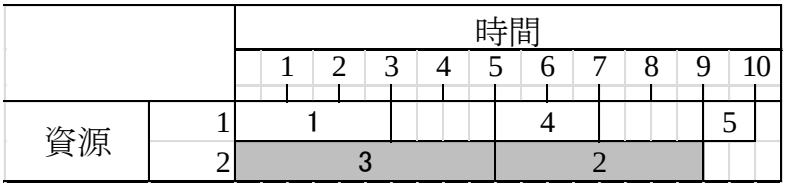

図 9: 資源競合解消後整理したガントチャート

3.6　ネック工程の検出

最早終了時刻と最遅終了時刻を求めることでネック工程を見つけ, ネック工程とそうでない工程とに分類する. ネック工程とは, その工程の実行時間が予定より少しでも遅れると, プロジェクト全体の終了時間に影響する工程のことである. Max-plus 代数による計算手法は文献[5]より引用する. 最早終了時刻$\boldsymbol{x}_E$と最遅終了時刻$(\boldsymbol{x}_L + \boldsymbol{d})$は次のようにして計算できる.

まず, 最早終了時刻は,

$$\boldsymbol{x}_E = \boldsymbol{A} \otimes \boldsymbol{B}_0 \otimes \boldsymbol{u} \tag{25}$$

で計算できる. ただし $\boldsymbol{A} = \boldsymbol{P} \otimes (\boldsymbol{F}_0 \otimes \boldsymbol{P})^*$であり, $\boldsymbol{X}^*$は任意の二工程間の最長経路に等しく,

$\boldsymbol{x}_{ij}^* = \{j$から$i$への最長距離(工程$j$から工程$i$へのパスが存在するなら), ε(それ以外)$\}$

となる性質がある.

また，プロジェクト全体の最早終了時刻は，

$$\boldsymbol{y}_E = \boldsymbol{C}_0 \otimes \boldsymbol{x}_E \tag{26}$$

で求められる．これらより，最遅開始時刻は，

$$\boldsymbol{x}_L = (\boldsymbol{C}_0 \otimes \boldsymbol{A}) \backslash \boldsymbol{y}_E \tag{27}$$

となる．$\boldsymbol{m} = (\boldsymbol{x}_L + \boldsymbol{d}) - \boldsymbol{x}_E$とすると，ネック工程とは$[\boldsymbol{m}]_i = 0$となる工程である．そこで，ネック工程の集合を$\alpha$，ネックでない工程の集合を$\beta$と表記すると，

$$\alpha = \{i \mid [\boldsymbol{m}]_i = 0\}：ネック工程 \tag{28}$$

$$\beta = \{i \mid [\boldsymbol{m}]_i > 0\}：ネックでない工程 \tag{29}$$

となる．集合αに属する工程の連なり，つまりクリティカルチェーンをα-chain，集合βに属する工程の連なりをβ-chain と呼ぶこととする．

また，時間バッファを挿入するために，$[\boldsymbol{a}]_i = \{e: \text{if } i \in \alpha, \varepsilon: \text{if } i \in \beta\}$，$[\boldsymbol{b}]_i = \{e: \text{if } i \in \beta, \varepsilon: \text{if } i \in \alpha\}$として，$\boldsymbol{P}_a = \text{diag}(\boldsymbol{a}) \otimes \boldsymbol{P}$，$\boldsymbol{P}_b = \text{diag}(\boldsymbol{b}) \otimes \boldsymbol{P}$というベクトルと行列を作成する．

3.7　時間バッファの挿入

時間バッファとは仮想的な工程のことである．クリティカルチェーン法では，各工程での実行時間の遅れを考慮して，プロジェクトの完了時刻に遅れを出さないためにバッファを挿入する．

バッファの挿入位置について説明する．納期遅れはある工程の遅れが，クリティカルパスに影響を与えた時に発生する．工程の遅れがクリティカルパスに影響を与える原因は，

・　クリティカルチェーン上の工程の遅れ
・　非クリティカルチェーン上の工程の遅れがクリティカルチェーンに伝搬

の二つに分けることができる．そこで，クリティカルチェーン法ではクリティカルチェーンの最後と，非クリティカルチェーン上の工程とクリティカルチェーンの工程が合流する場所にバッファを挿入する．

時間バッファとして次の二種類を挿入する．ただし，max-plus 代数による計算手法は文献[5]より引用している．

● FB（フィーディングバッファ）：

β-chain とα-chain が合流する前の工程の後に，β-chain の実行時間の合計値の半分を仮想的な工程として挿入する．図 10 は FB 挿入の例である．工程 i の実行時間をd_iとし，工程 1，2，3，6 をα-chain，工程 4，5 をβ-chain とすると，α-chain に合流するβ-chain の実行時間の合計値の半分である$(d_4 + d_5)/2$が FB の大きさである．

次に，FB の挿入を max-plus 代数を用いて表現する．挿入するバッファの大きさを表す行列を$\boldsymbol{r}_f$とすると，$\boldsymbol{r}_f$は下式を用いて計算できる．

$$\boldsymbol{r}_f = \left[\boldsymbol{P}_b \otimes (\boldsymbol{F}_0 \otimes \boldsymbol{P}_b)^* \otimes \boldsymbol{g}\right]/2, \tag{30}$$

$$\boldsymbol{g} = [e\ e\ \cdots e]^T \in \boldsymbol{R}_{\max}^n. \tag{31}$$

また，FB の追加後は隣接行列$\boldsymbol{F}_0$を$\boldsymbol{F}_c$に修正する必要がある．修正後は

$$\boldsymbol{F}_c = \boldsymbol{F}_0 \oplus \mathrm{diag}(\boldsymbol{a}) \otimes \boldsymbol{F}_0 \otimes \mathrm{diag}(\boldsymbol{r}_f) \tag{32}$$

となる．

- PB（プロジェクトバッファ）：

α-chain の実行時間の合計値の半分をプロジェクトの出力の直前に仮想的な工程として挿入する．図 11 は PB 挿入の例である．工程 1, 2, 3, 6 をα-chain, 工程 4, 5 をβ-chain とすると，α-chain の実行時間の合計値の半分である$(d_1 + d_2 + d_3 + d_6)/2$が PB の大きさである．

次に，PB の挿入を max-plus 代数で表現する．挿入するバッファの長さを含んだ行列を$\boldsymbol{r}_p$とすると，$\boldsymbol{r}_p$は次式のように計算できる．

$$\boldsymbol{r}_p = \left[\boldsymbol{P}_a \otimes (\boldsymbol{F}_0 \otimes \boldsymbol{P}_a)^* \otimes \boldsymbol{g}\right]/2 \tag{33}$$

また，出力行列を下式のように修正する．

$$\boldsymbol{C}_c = \boldsymbol{C}_0 \otimes [\mathrm{diag}(\boldsymbol{r}_p) \oplus \mathrm{diag}(\boldsymbol{r}_f)] \tag{34}$$

以上のようにして求めた$\boldsymbol{F}_c$と$\boldsymbol{C}_c$を用いて式(26)の$\boldsymbol{y}_E$を求めることで，各工程の実行時間の遅れを考慮したプロジェクト全体の最早終了時刻が求められる．具体的には，$\boldsymbol{A} = \boldsymbol{P} \otimes (\boldsymbol{F}_c \otimes \boldsymbol{P})^*$として$\boldsymbol{x}_{cE} = \boldsymbol{A} \otimes \boldsymbol{B}_0 \otimes \boldsymbol{u}$を求めたのち，最早終了時刻

$$\boldsymbol{y}_{cE} = \boldsymbol{C}_c \otimes \boldsymbol{x}_{cE} \tag{35}$$

を求める．$\boldsymbol{y}_{cE}$の要素の最大値を$C_{\max}$とすると，$C_{\max}$を本研究における評価値とする．

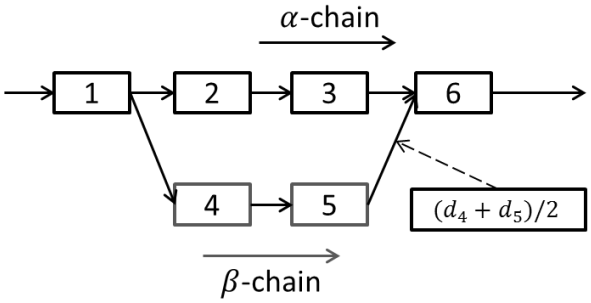

図 10: FBの挿入例

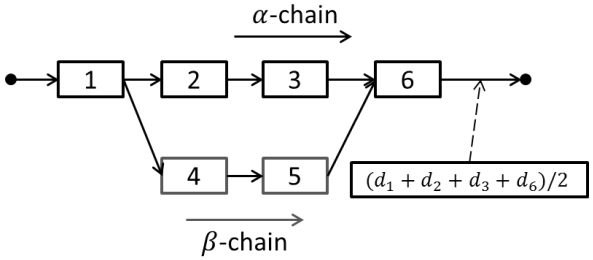

図 11: PBの挿入例

4 提案手法

本章では, 資源競合の解消のための近似解法や厳密解を求めるための手法を提案する.

4.1 資源競合の解消方法

まず，使用する文字や記号を 3.2 章に追加で定義する.

l：資源数

s：各資源の処理工程数の最大値

m：工程間の先行関係の数

$\boldsymbol{S} \in \boldsymbol{N}_{\max}^{l \times s}$：資源 i が j 番目に工程 k を処理するならば$[\boldsymbol{S}]_{ij} = k$ ，それ以外の場合は 0

ただし，$\boldsymbol{N}$ は自然数の集合である．次に，資源競合の解消問題に必要な入力と出力を以下に示す.

入力：

- n：工程数
- $\boldsymbol{F}_0 \in \boldsymbol{R}_{\max}^{n \times n}$：各工程の先行関係
- $\boldsymbol{d} \in \boldsymbol{N}^n$：各工程の実行時間
- $\boldsymbol{r} \in \boldsymbol{N}^n$：各工程に必要な資源を指定するベクトル

出力：

- $\boldsymbol{S}$：資源競合が発生しない各資源の工程処理順番

資源競合が発生する理由は，競合が発生する工程間に先行関係が存在しない結果，複数の工程を一つの資源で処理することが, プロジェクトの構造上可能になっているからである．したがって，同じ資源を割り当てている工程間に，工程処理順の先行関係を与えることによって競合を解消することができる.

例題を図 12 に示す．工程の実行時間などの入力は次のように設定する.

$n = 5,$

$$\boldsymbol{F}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & e & \varepsilon \end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix} 3 \\ 4 \\ 5 \\ 5 \\ 1 \end{bmatrix}, \quad \boldsymbol{r} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \\ 1 \end{bmatrix}.$$

資源競合を解消したプロジェクトの例が図 13 である．資源 1 が工程 1，4，5．資源 2 が工程 3，2 の順に処理する場合，図 13 のプロジェクトの破線のように先行関係を追加することによって，資源競合を解消する．新たに追加した先行関係は次の隣接行列$\boldsymbol{F}_0$で表現でき，各資源の工程処理順番 $\boldsymbol{S}$ も次のように表現できる.

$$\boldsymbol{F}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & e & \varepsilon \end{bmatrix}, \quad \boldsymbol{S} = \begin{bmatrix} 1 & 4 & 5 \\ 3 & 2 & 0 \end{bmatrix}$$

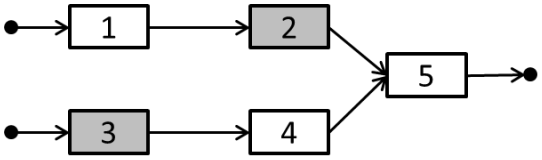

図 12: 資源競合解消前のプロジェクト

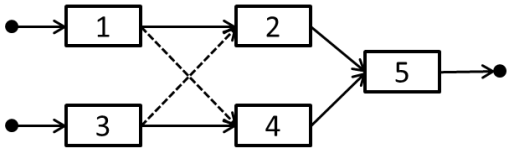

図 13: 資源競合解消後のプロジェクト

このようにして，資源ごとの作業順序が決まる．この問題は NP 完全であると予想され，最適解を求めるためには膨大な時間が必要であるため，近似解法を考える必要がある．

4.2 三つの簡素な近似解法

本研究で利用する初歩的な三つの近似解法について説明する．入力と出力は三つの近似解法で共通である．以下に入力と出力を示す．

入力：

- 工程数n
- 外部入力行列$\boldsymbol{B}_0 \in \boldsymbol{R}_{\max}^{n\times q}$
- 外部出力行列$\boldsymbol{C}_0 \in \boldsymbol{R}_{\max}^{p\times n}$
- 各工程の先行関係$\boldsymbol{F}_0\boldsymbol{R}_{\max}^{n\times n}$
- 各工程の実行時間$\boldsymbol{d} \in \boldsymbol{N}^n$
- 各工程に必要な資源を指定するベクトル$\boldsymbol{r} \in \boldsymbol{N}^n$

出力：

- 資源競合が発生しない，評価値 v がなるべく小さくなるような各資源の工程処理順番 $\boldsymbol{S}$

ただし，$\boldsymbol{N}$ は自然数の集合である．三つの近似解法は以下に示す．

A) 最早開始時刻法

最早開始時刻の早い順に各資源が工程を処理する解法である．最早開始時刻は式(25)を利用して

$$\boldsymbol{s}_E = \boldsymbol{x}_E - \boldsymbol{d} \tag{36}$$

で求めることができ，アルゴリズムは以下のとおりである．

1. 最早開始時刻$\boldsymbol{s}_E$を求める．

2. 各工程を利用する資源ごとに分ける.
3. 資源ごとに各工程の$\boldsymbol{s}_E$を小さい順にソートし，その順序を各資源の工程処理順序 $\boldsymbol{S}$ とする.

B)　最遅開始時刻法

最遅開始時刻の早い順に各資源が工程を処理する解法である．最遅開始時刻$\boldsymbol{x}_L$は式(27)で求めることができ，アルゴリズムは最早開始時刻法と同様で以下のとおりである.

1. 最早開始時刻$\boldsymbol{x}_L$を求める.
2. 各工程を利用する資源ごとに分ける.
3. 資源ごとに各工程の$\boldsymbol{x}_L$を小さい順にソートし，それを各資源の工程処理の順序 $\boldsymbol{S}$ とする.

C)　中間時刻法

最早開始時刻と最遅開始時刻の平均値$\boldsymbol{x}_M$が早い順に各資源が工程を処理する近似解法であり，式で表すと次式のようになる.

$$\boldsymbol{x}_M = (\boldsymbol{s}_E + \boldsymbol{x}_L)/2 \tag{37}$$

アルゴリズムは以下のとおりである.

1. 最早開始時刻$\boldsymbol{x}_M$を求める.
2. 各工程を利用する資源ごとに分ける.
3. 資源ごとに各工程の$\boldsymbol{x}_M$を小さい順にソートし，それを各資源の工程処理の順序 $\boldsymbol{S}$ とする.

図 12 を例として，三つの近似解法で資源競合を解消する．式(36)，(27)，(37)を用いて$\boldsymbol{s}_E$，$\boldsymbol{x}_L$，$\boldsymbol{x}_M$を求めると次のような結果になる.

$$\boldsymbol{s}_E = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 5 \\ 10 \end{bmatrix}, \boldsymbol{x}_L = \begin{bmatrix} 3 \\ 6 \\ 0 \\ 5 \\ 10 \end{bmatrix}, \boldsymbol{x}_M = \begin{bmatrix} 0+3 \\ 3+6 \\ 0+0 \\ 5+5 \\ 10+10 \end{bmatrix}/2 = \begin{bmatrix} 3/2 \\ 9/2 \\ 0 \\ 5 \\ 10 \end{bmatrix}.$$

次に, 各工程を利用する資源ごとに分ける. それを各資源の工程処理順序 $\boldsymbol{S}$ にすると,

$$\boldsymbol{S} = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 3 & 0 \end{bmatrix}$$

と表現できる.

次に，$\boldsymbol{s}_E$，$\boldsymbol{x}_L$，$\boldsymbol{x}_M$の早い順に処理順番をソートすると，三つの近似解法すべてで $\boldsymbol{S}$ が

$$\boldsymbol{S} = \begin{bmatrix} 1 & 4 & 5 \\ 3 & 2 & 0 \end{bmatrix}$$

と表現できる．この例題では，理解を容易にするためにプロジェクト構造を単純にし，工程数を少なくした結果三つの近似解法すべてで同じ解になった．しかし，問題によっては三つの近似解法でそれぞれ異なる解となることがあり, 工程数が多いほど別の解に

なりやすくなる．次章で紹介する局所探索法では，この三つの近似解法で導いた解のうち，最も良い解を初期解とする．

次に，計算量について考察する．この三つの近似解法は評価値を計算する際に，式(25)(30)(33)で任意の二工程間の最長経路の行列を計算する．そこで最もオーダーの高い計算を行っており，その計算量は$O(n(n+$先行関係の数$))$である．元々のプロジェクトの先行関係の数は m で，資源競合を解消するために追加する先行関係の数は資源数lを利用して$n-l$と表現でき，$b=n-l$とすると先行関係の数は$m+b$である．ただし $O(m) \le O(n^2)$，$O(b) \le O(n^2)$ である．したがって，三つの近似解法の計算量は $O\big(n(n+m+b)\big)$である．

4.3 局所探索法

局所探索法とは，ある初期解から出発して，解を改善していく手法である．また，同じ改善方法を何度も繰り返すことによって，解を改善していくことを基本とし，一般的な手順をまとめると次のようになる．

Step 0 : 初期解 x を定め，$k=1$ とする．

Step k : 解の近傍 $NB(x)$を探索し，現在の解よりも良い解があれば解をそれに更新する．その後，$k=k+1$ として step k に戻る．もし，より良い解がなければ終了．

4.3.1 近傍の定義

本研究では，近傍 $NB_i(\boldsymbol{S})$を次のように定義する．

$NB_i(\boldsymbol{S})$ = {$\boldsymbol{S}$' | $\boldsymbol{S}$'は各資源の工程処理順番で，$\boldsymbol{S}$ において資源 i の任意の二つの工程処理順番を入れ替えた解．}.

この $NB_i(\boldsymbol{S})$を用いて，近傍 $NB(\boldsymbol{S})$を次のように定義する

$NB(\boldsymbol{S}) = NB_1(\boldsymbol{S}) \cup NB_2(\boldsymbol{S}) \cup \cdots \cup NB_l(\boldsymbol{S})$.

次に，近傍 NB_i の例を示す．あるプロジェクトの資源別の工程処理順番が図 14 である．このプロジェクトの$\boldsymbol{S}$は次のように表現する．

$$\boldsymbol{S} = \begin{bmatrix} 2 & 4 & 7 & 1 \\ 3 & 5 & 6 & 8 \end{bmatrix}$$

資源 1 の処理順番は 2，4，7，1 であり，この処理順番のうち，任意の 2 工程を入れ替えることによってできる各資源の工程処理順番 $\boldsymbol{S}$'は, 近傍 $NB_1(\boldsymbol{S})$の要素の一つである．例えば，工程 4 と 1 を入れ替えると図 15 のようになり，$\boldsymbol{S}'$は次のように表現する．

$$\boldsymbol{S}' = \begin{bmatrix} 2 & 1 & 7 & 4 \\ 3 & 5 & 6 & 8 \end{bmatrix}$$

このようにして，資源 1 の二つの工程を入れ替えて作ることのできるすべての$\boldsymbol{S}'$の集合が $NB_1(\boldsymbol{S})$である．また，資源 1 だけでなく資源 2 で $NB_2(\boldsymbol{S})$を求め $NB_1(\boldsymbol{S})$と和集合にすると，$NB(\boldsymbol{S})$は次式のように表現できる．

$$NB(\boldsymbol{S}) = NB_1(\boldsymbol{S}) \cup NB_2(\boldsymbol{S})$$

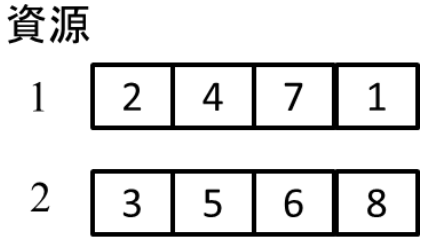

図 14: 2 資源の工程処理順番の例

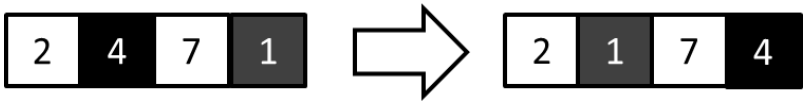

図 15: 2 工程入れ替えの例

さらに，もう一つの近傍 $NB_{ij}(\boldsymbol{S})$を次のように定義する．

$NB_{ij}(\boldsymbol{S}) = \{\boldsymbol{S}' \mid \boldsymbol{S}'$ は $\boldsymbol{S}$ において二つの資源 i と j それぞれで任意の二つの工程処理順番を入れ替えた解．ただし，資源 i もしくは j のみの二つの工程処理順番を入れ替えた場合を含む．

この $NB_{ij}(\boldsymbol{S})$を用いて，$NB'(\boldsymbol{S})$を次のように定義する．

$$NB'(\boldsymbol{S}) = \bigcup_{1\le i\le l-1, 1\le j\le l, i<j} NB_{ij}(\boldsymbol{S}).$$

次に，近傍 $NB_{ij}(\boldsymbol{S})$の具体例を $NB_i(\boldsymbol{S})$と同様に図 14 を用いて示す．このプロジェクトの $\boldsymbol{S}$は次のように表現する．

$$\boldsymbol{S} = \begin{bmatrix} 2 & 4 & 7 & 1 \\ 3 & 5 & 6 & 8 \end{bmatrix}$$

この処理順番 $\boldsymbol{S}$ のうち，各資源 1，2 の二工程を入れ替えることによってできる $\boldsymbol{S}'$は近傍 $NB_{12}(\boldsymbol{S})$の要素の一つである．例えば，資源 1 の工程 4，1 と資源 2 の 5，8 を入れ替えると図 16 のようになり，$\boldsymbol{S}'$は次のように表現する．

$$\boldsymbol{S}' = \begin{bmatrix} 2 & 1 & 7 & 4 \\ 3 & 8 & 6 & 5 \end{bmatrix}$$

このようにして，資源 1 と資源 2 のそれぞれで二つの工程を入れ替えて作ることのできる，すべての$\boldsymbol{S}'$の集合が $NB_{12}(\boldsymbol{S})$である．また，このプロジェクトは資源数 l が 2 であるため，

$$NB'(\boldsymbol{S}) = NB_{12}(\boldsymbol{S})$$

と表現できる．また，資源数 l が 3 のときは，

$$NB'(\boldsymbol{S}) = NB_{12}(\boldsymbol{S}) \cup NB_{13}(\boldsymbol{S}) \cup NB_{23}(\boldsymbol{S})$$

と表現できる．

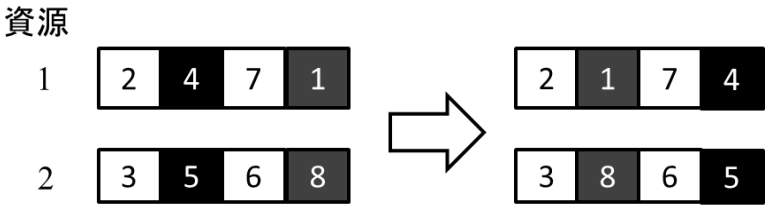

図 16: 二資源の入れ替えの例

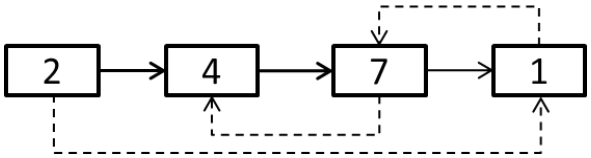

図 17: 閉路の発生例

近傍を求める際に注意することがある．4.2 章で述べたように，各資源の工程処理順 $\boldsymbol{S}$'に応じて先行関係を追加した結果, 閉路が出現した場合はその解 $\boldsymbol{S}$'は近傍の中には含まれない．その具体例を示す．図 17 の実線のような先行関係があるプロジェクトの一部に対して，2，1，7，4 と資源が工程を処理する場合，破線のような先行関係が作られる．この場合工程 1，7 で閉路が出現し，プロジェクトの先行関係に矛盾が生じるため，閉路内の工程はどの工程も処理を開始することができない．よって，閉路が出現したプロジェクトは近傍に含めない.

4.3.2 提案する局所探索法

この近傍 $NB(\boldsymbol{S})$を用いた二つの局所探索法を提案する.

● single-swap-best: 近傍 $NB(\boldsymbol{S})$を利用するアルゴリズム．まず，$NB_1(\boldsymbol{S})$の中で最も評価値の良い解が，現在の近似解よりも良い解であれば更新する．次に，$NB_2(\boldsymbol{S})$の中で，最も評価値の良い解に更新する．この手順を 1 から l までの資源に対して一つずつ順番に行う．もし，$NB(\boldsymbol{S})$で近似解が改善されたのなら，$NB_1(\boldsymbol{S})$からもう一度探索する．そうでなければ探索終了．具体的なアルゴリズムを以下に示す.

1. 4.2 章で紹介した三つの近似解法で解き, その中で最も良い解を初期解 $\boldsymbol{S}$ とする.
2. $i = 1$とする.
3. $NB_i(\boldsymbol{S})$を探索する.
4. $NB_i(\boldsymbol{S})$の中で最も良い解 $\boldsymbol{S}$'が現在の $\boldsymbol{S}$ よりも良い解ならば，$\boldsymbol{S}$'に $\boldsymbol{S}$ を更新する.
5. $i = i + 1$とする.
6. $i \leq l$ならば 3 へ戻る.
7. $\boldsymbol{S}$ が更新されたなら，2 へ戻る.
8. $\boldsymbol{S}$ が更新されなかったら終了.

● single-swap-first: $NB_1(\boldsymbol{S})$から $NB_l(\boldsymbol{S})$まで順に探索し，現在の近似解より良い評価値の最

初に見つけた解に更新する．そして，$NB_1(\boldsymbol{S})$から探索をやり直す．$NB(\boldsymbol{S})$に改善可能な解がなくなったら探索終了．具体的なアルゴリズムを以下に示す．

1. 4.2 章で紹介した近似解法三つで解き，その内で最も良い解を初期解 $\boldsymbol{S}$ とする．
2. $NB(\boldsymbol{S})$の中で最初に発見した $\boldsymbol{S}$ よりも良い解 $\boldsymbol{S}'$を $\boldsymbol{S}$ に更新する．
3. $\boldsymbol{S}$ が更新されたなら，2 へ戻る．
4. $\boldsymbol{S}$ が更新されなかったら終了．

これら二つの局所探索法は近傍 $NB(\boldsymbol{S})$を探索するのに$O(ls^2)$の計算量がかかる．s は 4 章で述べたとおり，各資源の処理工程数の最大値である．また，近傍すべての解を評価するために評価値 v を毎回求めており，オーダーは 4.2 章で説明したとおり $O\left(n(n+m+b)\right)$ である．したがって，二つの局所探索法の一ループ，つまり $NB(\boldsymbol{S})$を一回探索して評価する計算量のオーダーは$O(ls^2n(n+m+b))$である．

次に，この近傍 $NB'(\boldsymbol{S})$ を用いた二つの局所探索法を提案する．

- double-swap-best: single-swap-best と似ており，$NB'(\boldsymbol{S})$を探索して現在の近似解 $\boldsymbol{S}$ より良い解の中で最も評価値の良い解に更新する．これを更新できなくなるまで繰り返す．

具体的なアルゴリズムを以下に示す．

1. 4.2 章で紹介した近似解法三つで解き，その内で最も良い解を初期解 $\boldsymbol{S}$ とする．
2. $NB'(\boldsymbol{S})$の中で最も良い解 $\boldsymbol{S}'$が現在の $\boldsymbol{S}$ よりも良い解ならば，$\boldsymbol{S}'$に $\boldsymbol{S}$ を更新する．
3. $\boldsymbol{S}$ が更新されたなら，2 へ戻る．
4. $\boldsymbol{S}$ が更新されなかったら終了．

- double-swap-first: single-swap-first と似ており，$NB'(\boldsymbol{S})$を探索する中で，現在の近似解より良い評価値の，最初に見つけた解 $\boldsymbol{S}$ に更新したのち，$NB'(\boldsymbol{S})$を探索し直す．現在の近似解より良い解が $NB'(\boldsymbol{S})$に存在しない場合は探索終了．具体的なアルゴリズムを以下に示す．

1. 4.2 章で紹介した近似解法三つで解き，その内で最も良い解を初期解 $\boldsymbol{S}$ とする．
2. $NB'(\boldsymbol{S})$の中で最初に発見した $\boldsymbol{S}$ よりも良い解 $\boldsymbol{S}'$に $\boldsymbol{S}$ を更新する．
3. $\boldsymbol{S}$ が更新されたなら，2 へ戻る．
4. $\boldsymbol{S}$ が更新されなかったら終了．

これら二つの局所探索法は近傍 $NB'(\boldsymbol{S})$を探索するのに$O(l^2s^4)$の計算量がかかる．また，$NB(\boldsymbol{S})$近傍を利用した局所探索法と同様に，すべての解を評価するために評価値 v を毎回求めており，$O\left(n(n+m+b)\right)$の時間がかかる，一ループ当たりの計算量のオーダーは$O(l^2s^4n(n+m+b))$である．

4.4 遺伝アルゴリズム

4.2 章で提案した近似解法で求めた近似解の改善法として，遺伝アルゴリズム[9][10]の考え方を利用した解の改善も行う．以後，遺伝アルゴリズムのことを GA (Genetic

Algorithm)と略記する．本章では，GA の基本的な考え方と提案する GA による近似解法のアルゴリズムを示す．

GA とは，生物が環境に適応して進化していく過程を模倣したアルゴリズムである．以下に一般的な遺伝アルゴリズムの手順を示す．

1. 初期集団の作成：$t = 0$として，あらかじめ設定された数 N 個だけ個体を作成し，それを初期集団 $P(0)$とする．ただし本研究では，簡素な近似解法で求めた解を個体とし$N = 3$とする．
2. 評価：各個体の優秀さを一定の基準で調べる．本研究では，CCPM 法で解いた解の評価値である $C_{\max}$ を基準とする．
3. 選択：$P(t)$が複数個体ある場合は，各個体の評価を基準にして，次の交叉に利用する個体をあらかじめ設定した回数だけ二つ選択してペアを作り，これらの集合を$P'(t)$とする．本研究では $P(t)$の個体数は三つと少ないため，選択は順列の組み合わせで行う．
4. 交叉：選択したペアの個体集合$P'(t)$を利用して新たな個体の集合$P''(t)$を作る．本研究では二点交叉と呼ばれる方法で交叉を行う．
5. 突然変異：一定確率で交叉前や交叉後の個体の一部が変化する．本研究では交叉後の個体に突然変異を行う．
6. 生存選択：$P'(t)$と$P''(t)$の中から N 個選び，次の世代の集合 $P(t+1)$を作成する．
7. 終了条件：あらかじめ定められていた終了条件を満たしたら GA を終了する．満たしていなかったら 3 へ戻る．

本論文で利用される交叉の方法を説明する[10].

- 二点交叉

二つの工程処理順序において交叉点をランダムに二つ選び, 一つの工程処理順序の二つの交叉点の間の工程処理順序を, もう一つの工程処理順序に挿入する手法である．図 18 の工程処理順序を例として，本研究で行う交叉を具体的に説明する．まず，交叉点をランダムに選んだその結果，2，3 番目に処理する工程を入れ替えるとする．工程処理順序 x の処理順序は 1，2，3，4 とし，y の工程処理順序の 2，3 番目に処理する工程を挿入した場合，工程 3，1 を 2，3 番目に処理することは決まるので，残りの工程 2，4 は x と同様に 2，4 の順番で処理する．したがって交叉の結果，工程処理順序は 2，3，1，4 となる．

次に，本論文で利用される生存選択の方法について説明する．そのために，いくつかの生存選択の手法を簡潔に説明する[10].

- ルーレット選択

評価値の良い個体が次世代に残りやすくするように各個体の生存確率を設定し, 生存選択を行う手法である．評価値の高い個体ばかり生存しやすくなり，その結果解が十分に改善されないまま収束することがある．

- ランキング選択

評価値の良い順にランク付けをし, 各ランクの中でいくつ次世代に残すのかを決めておき，ランダムで選ぶ手法である．例えば，評価値の 1 から 3 番目に良い個体をランク A，3 から 6 番目の個体をランク B とする．ランク A から 2 個，ランク B から 1 個の個体を選ぶとしたら，ランク A の中からランダムに 2 個，ランク B の中からランダムに 1 個の個体を選ぶ.

- エリート選択

評価値の良い個体を一定数次世代に残す考え方である. 他の選択手法と組み合わせて利用する場合が多い.

以上の手法のうち，本研究ではランキング選択を採用する．具体的な生存選択の方法は，一番評価値の良い個体をランク A，その他の個体をランク B としてランク A から 1 個，ランク B から 2 個次世代に残す．一番良い個体を必ず次世代に残すようにランク分けを行うことによって，エリート選択の考え方も採用している．

ランキング選択を採用する理由は，5 章の計算実験と同様の環境で数値実験を行った結果，ルーレット選択のように優れた個体が選ばれやすいようにするよりも，先程提案したランキング選択の方が良い解を得られやすかったためである. 原因は前述した通り，解が十分に改善されないまま収束するためだと考えられる．

本研究での近似解法のアルゴリズムでは，上記の GA の考え方を用いる．入出力とアルゴリズムは以下のとおりである．

入力：

- n：工程数
- $\boldsymbol{F}_0 \in \boldsymbol{R}_{\max}^{n\times n}$：各工程の先行関係
- $\boldsymbol{d} \in \boldsymbol{N}^n$：各工程の実行時間
- $\boldsymbol{r} \in \boldsymbol{N}^n$：各工程に必要な資源を指定するベクトル
- M_{ut}：突然変異の確率
- I：終了条件

出力：

- 評価値 v がなるべく小さくなるような各資源の工程処理の順番 S

アルゴリズムは次のとおりである．

1. 最早開始時刻法, 最遅開始時刻法, 中間時刻法の三つで資源競合の解消問題を解き，その三つの解を初期集団 $P(0)$とする．また，評価値の最小値 $y_{\min}$ とその時の処理順番 $\boldsymbol{S}$ を記録しておく．
2. $t = 1$とする．
3. $P(t)$の中の個体三つを a，b，c として，二つを選ぶ順列の組み合わせ 6 パターンすべてを集合$\boldsymbol{P}'(t)$とする．

4. $k = 1$とする．
5. $P'(t)$のうち一つの順列組合せを選び，図 18 のように処理順序 x と y を二点交叉させる．ただし x は選んだ順列の一つ目の個体の資源 k の処理順序，y は同じ順列上の二つ目の個体の資源 k の処理順序とする．
6. M_{ut} の確率で突然変異を行うか判定する．突然変異を行うなら 7 へ進み，そうでない場合は 8 へ進む．
7. 同じリソース内の二つの工程をランダムに選び，交叉後の処理順序を図 19 のように，ランダムで二工程選び処理順序を入れ替える．
8. 閉路ができた場合は資源 k の処理順番を x と同一にする．同一にしても閉路が発生したら，すべての資源の処理順番を x が所属する個体と同一にし，次世代の個体候補の集合$P''(t)$に加えて 10 へ進む．
9. $k = k + 1$として，$k \leq l$のとき 5 へ．そうでないときは 5〜8 で x と y を交叉等を行って作った解の個体を次世代の個体候補の集合$P''(t)$に加える．
10. $P'(t)$すべてで交叉するまで，4 へ戻る．図 20 のように，集合$P'(t)$の組み合わせで交叉して作った個体の集合$P''(t)$の個体数が 6 つになったら 11 へ進む．ただし，x←y は個体 x に個体 y を交叉してできる個体という意味である．
11. 6 つの個体が存在する$P''(t)$の中から最も評価値の良い個体を $P(t+1)$に加える．
12. 11 で加えた個体が $y_{\min}$ よりも評価値が小さい場合，$y_{\min}$ と処理順番 S を更新する．
13. 残りの$P''(t)$の中からランダムで 2 個選び, $P(t+1)$に加える. この時点で, 個体 a←b, b←a，b←c が$P''(t)$の中から選ばれているとしたら，$P(t+1)$の個体数は図 21 のように 3 個である．
14. $t = t + 1$とする．
15. $y_{\min}$ の更新が I 回連続で行われなかった場合は終了し, そうでなかったら 3 へ戻る．

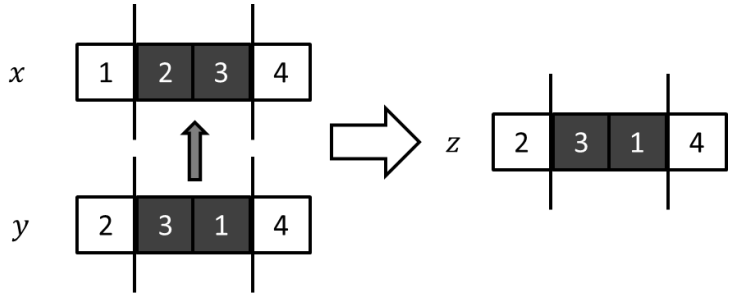

図 18: 処理順序 x と y の交叉の例

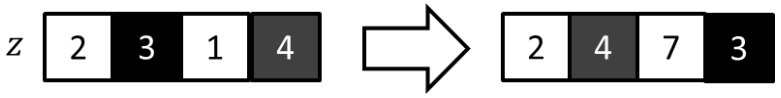

図 19: 処理順序 z の突然変異の例

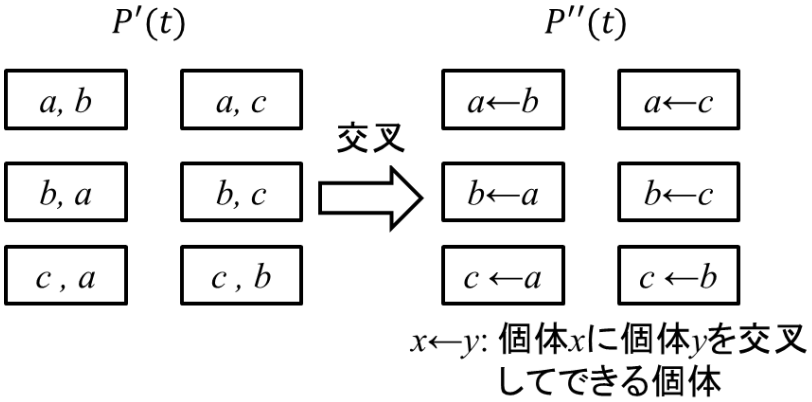

図 20: 順列組合せの集合 *P'*(*t*)の交叉後の個体の例

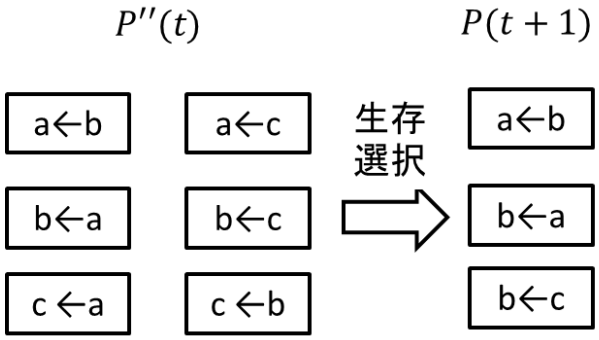

図 21: 集合 *P''*(*t*)の生存選択の例

4.5 厳密解の計算手法

5 章の計算実験では，近似解が最適解と比べてどの程度良い解が得られるかを確かめることによって，近似解法の有用性を示す．本節では，最適解の計算方法を提案する．

提案手法の基本的な考え方は，全数列挙つまりすべての解を調べてその中で最も良い解を採用とする考えである．全数列挙の場合は，各資源の工程処理順番の順列の組み合わせを，すべての資源で組み合わせることによって，解の候補を作成する．例えば，資源 1，2 の処理する工程数がそれぞれ 3，4 の場合は，それぞれの資源の工程処理順番の順列の組み合わせは 6，24 である．さらにこの二つの資源の工程処理順番を組み合わせると，解の候補の数は 6*24=144 通りである．提案手法ではこの全数列挙の中で，一つの資源のみの工程処理順番でプロジェクトに閉路が発生する場合は，その工程処理順番を解の候補から除外し，解の候補を減らすことによって計算時間を短縮する．この手法で先程の例の資源 1，2 の順列の組み合わせを半分にすることができれば，解の候補の数は 3*12=36 通りと 1/4 になる．このようにして，解の候補を減らし計算時間を短縮さ

せる．5 章の計算実験で利用する，20 工程のプロジェクトの厳密解を求める場合は，この手法を利用することによって，計算時間を全数列挙と比べて約 1/8000 にすることに成功した．

最適値を $y_{\min}$ とし，これをアルゴリズムの出力とすると，アルゴリズムは以下のとおりである．

資源 i の工程処理順番の順列の組み合わせの集合を P_i とする．

1. $i = 1$とする．
2. プロジェクトの先行関係$\boldsymbol{F}_0$に，P_i の内の一つの工程処理順番を先行関係に追加し$\boldsymbol{F}'_0$とする．
3. $\boldsymbol{F}'_0$に閉路が存在しない場合，2 で追加した工程処理順番を集合 P'_i に加える．
4. 2，3 を P_i の要素すべてで行う．
5. $i = i + 1$とする．
6. $i \leq l$ であれば 2 へ戻る．
7. l 個の集合 P_1 から P_l を利用し，l 個組の工程処理順番の組み合わせを可能な限り作り，それらの組み合わせの集合を Q とする．
8. プロジェクトの先行関係$\boldsymbol{F}_0$に，Q のうちから一つの抜き出した工程処理順番を先行関係に追加し$\boldsymbol{F}_1$とする．
9. 先行関係$\boldsymbol{F}_1$に閉路が存在しなければ，評価値$\boldsymbol{C}_{\max}$を求める．
10. もし，$y_{\min}$ よりも$\boldsymbol{C}_{\max}$が小さければ，$y_{\min}$ を更新する．また，工程処理順番 $\boldsymbol{S}$ も更新する．ただし，初めてこの手順を行う場合は$y_{\min} = \boldsymbol{C}_{\max}$とし，工程処理順番 $\boldsymbol{S}$ を更新する．
11. Q のすべての要素で 8〜10 を行う．

5 計算実験

本章では 5 章で提案した近似解法の計算実験を行う．

5.1 実験環境

計算実験の環境は以下のとおりである．

CPU: Intel Core i5-2400 3.20GHz
OS: Microsoft Windows7 Professional 64bit
Memory: 4GB
Programming language: MATLAB R2013b

また，入力する行列などは以下のとおりである．

- ケース数：100
- 資源の数：表 1 のように，資源数が 10 工程の時は 3，15 工程の時は 5，20，30，40 工程の時は 7，60，70 工程のときは 8，80 工程のときは 9
- 工程 i の利用資源 $[\boldsymbol{r}]_i$：[1, l]の離散一様乱数
- 工程 i の処理時間 $[\boldsymbol{d}]_i$：[5, 20]の離散一様乱数

厳密解は 20 工程まで求めることができた．

n	10	15	20	30, 40	60	80	100
l	3	5	7	7	8	9	10
厳密解	○			×			

表 1：資源の数

5.2 サンプルデータのグラフ構造

サンプルデータの構造，つまりプロジェクトの構造は一つのメインチェーンを決め，そこにいくつかのチェーンを接続する形状とする．具体例を図 22 に示す．

メインチェーンは実行時間の合計が最も大きいチェーンとし，そのチェーンに外部出力を一ヶ所作る．またその他のチェーンは，直接メインチェーンと接続されていなくても，メインチェーンとの先行関係が存在するようにランダムに接続する．

1. n 個の工程を作り，それぞれに利用資源$[\boldsymbol{r}]_i$と処理時間$[\boldsymbol{d}]_i$を設定する．
2. 図 23 のように，ランダムでいくつかのチェーンに区切る．
3. 区切ったチェーンごとに工程番号順の先行関係を作り，各チェーンの最小の工程番号の工程に外部入力を付ける．
4. 区切られたチェーンのうち，最も合計処理時間が大きいチェーンを選びメインチェーンとする．また，メインチェーンに外部出力を付ける．

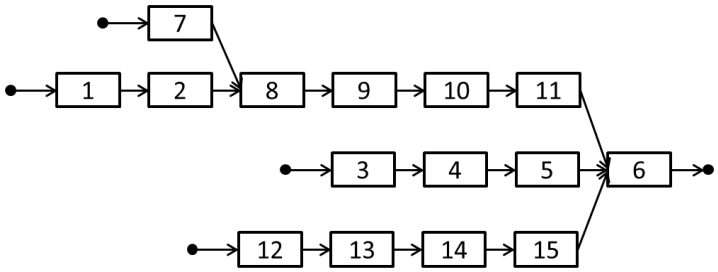

図 22：プロジェクトの構造の例

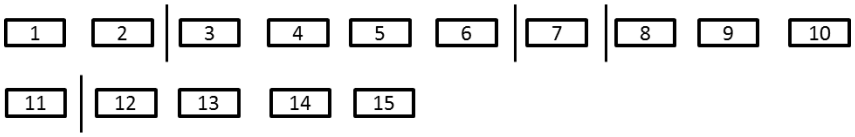

図 23：ランダムで区切る例

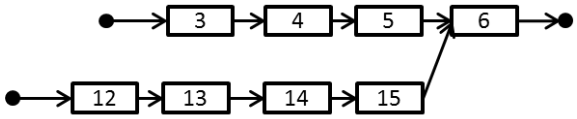

図 24：チェーンを繋ぐ例

5. メインチェーンを除く残りのチェーンのうち，一つをランダムで選び，図 24 のようにそのチェーンをメインチェーンの工程にランダムで一か所繋ぐ.
6. 選ばれていないチェーンのうち一つを選び，今まで選ばれてきたチェーンのうちどれかにランダムで繋ぐ.
7. 5 をすべてのチェーンを選ぶまで繰り返す.

5.3 実験結果

5.3.1 提案手法の適応例

まず，具体的な三つの近似解法で求めた近似解の改善例を示したのち，100 ケースのサンプルを用いる計算実験を行う.

図 25 のようなプロジェクトを簡素な近似解法三つを用いて解く．この例題は文献[2]から引用した．実行時間と資源の割り当ては以下のようにする.

$\boldsymbol{d} = [5\ 10\ 15\ 10\ 20\ 15\ 10\ 10\ 5\ 15\ 10\ 15\ 5\ 20\ 5]^{\mathrm{T}}$

$\boldsymbol{r} = [4\ 5\ 3\ 1\ 4\ 1\ 4\ 2\ 1\ 2\ 3\ 1\ 4\ 2\ 3]^{\mathrm{T}}$

例を解いた結果，評価値$C_{\max}$は最早開始時刻法では 182.5，最遅開始時刻法では 157.5，中間時刻法では 167.5 となった．最適解が 120 であるのに対して，最適解に最も近い近似解でも，最適解に比べて約 31%大きく良い解とはいえない．しかし，図 26 の近似解のガントチャートと，図 27 の最適解のガントチャートとを比較してみると，工程の処

理順番が異なる箇所は二箇所のみであり，そこを修正すれば最適解になるとわかる．そこで，局所探索や遺伝アルゴリズムで解を改善することを試みる．そのことによって，M_{ut}=0.1, I=200 としたときの GA と single-swap-best などの局所探索では 120 という最適解を求めることができた．また，M_{ut}=0.1，I=200 として GA で解いた場合，解の評価値は図 28 のように推移して改善される．ただし，横軸は解の改善の試行回数で，縦軸は評価値である．

5.3.2 近似解の比較

100 ケースのサンプルを用いる計算実験の近似解の平均値を表 2 に示す．表 2 の“Three methods”は，三つの近似解法で求めた解の中で最も良い解を採用した場合の平均値である．

また，解の改善率を表 3 に示す．解の改善率とは，“Three methods”で求めた解と比べて，局所探索法と GA によってどの程度改善されるかを，

改善率 = (“Three methods”で求めた評価値)/(改善後の評価値)

で計算し，求めた値である．例えば，改善率が 1.05 であった場合は解が 5%改善されていることになる．

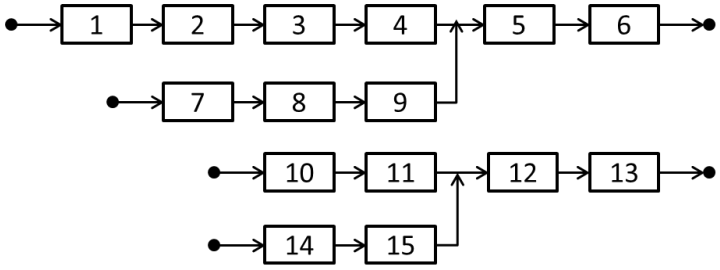

図 25: 15 工程のプロジェクトの例

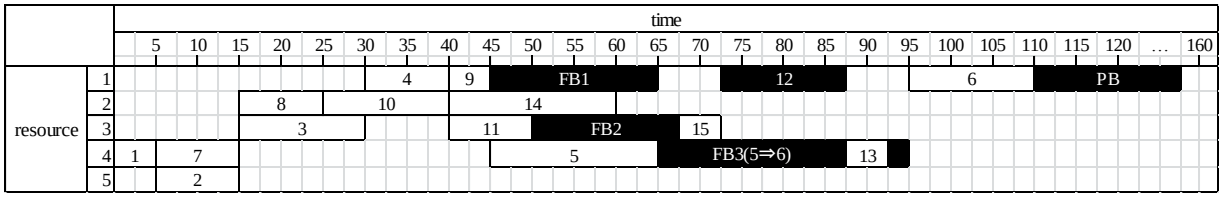

図 26：最遅開始時刻法で求めた近似解のガントチャート

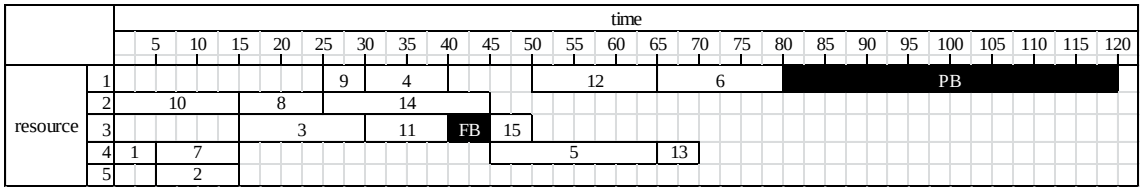

図 27：最適解のガントチャート

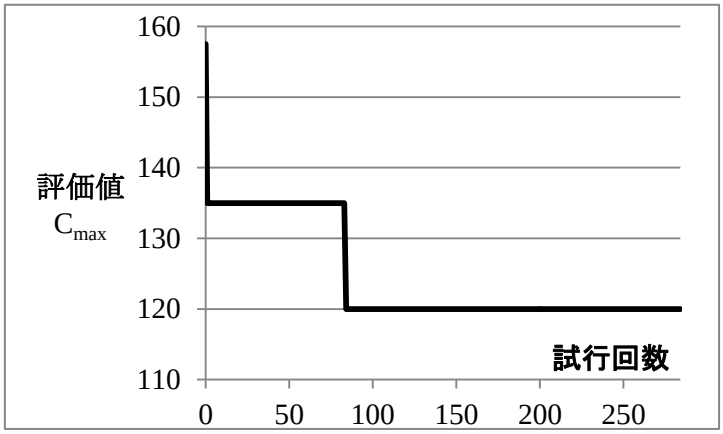

図 28：GA による評価値改善の推移

表 3 より，工程数が 100 の場合，“Three methods”の評価値の平均は 865 であり，double-swap-first を利用してこの評価値を改善すると網掛け部分の 888.2 となり，約 17.9%解が改善されているとわかる．GA の場合，突然変異率が高い方が少し良い解になる傾向があると表 2 よりわかる．*I*=100 で 80 工程の場合は，突然変異率が 0.1 のときよりも 0.2 のときの方が解の平均が悪いが，大きな違いはなく，他の工程数のほとんどの場合では 0.2 の場合の方が良い解が求まっている．一方 *I*=200 の場合は，60，80，100 工程の場合に突然変異率が 0.1 の方が良い解の平均値が得られているため，工程数が 50 以上の場合は突然変異率が 0.1 の方が 0.2 よりも良い可能性がある．また，*I* は 100 よりも 200 である方が解の平均値は良いが，50 工程までの範囲では 1～2%程度の違いしかない. 工程数が 100 の場合, M_{ut}=0.1, *I*=200 の GA によって求めた近似解は下線部の 979.9 となり，三つの初歩的な近似解法で求めた解と比べて約 6.8%改善されている．したがって，近似解は局所探索によって改善した方が，GA によって改善した解より良い解

表 2: 近似解の平均

使用手法		工程数						
I	M_{ut}	10	20	30	40	60	80	100
Three methods		156.2	262.6	377.6	492.5	672.4	865.0	1,046.8
single-swap-best		155.4	260.3	366.2	470.1	625.5	783.9	942.9
single-swap-first		155.2	260.4	366.3	470.5	623.3	779.4	925.9
double -swap-best		155.2	259.9	363.4	463.0	606.1	760.7	899.8
double -swap-first		155.2	259.9	364.8	464.1	606.2	753.7	888.2
100	0.1	155.2	259.6	365.5	474.6	642.6	818.5	992.7
100	0.2	155.3	259.4	365.7	468.8	639.0	823.7	986.9
200	0.1	155.3	259.0	364.5	466.7	634.4	810.9	979.9
200	0.2	155.2	259.3	363.3	464.9	634.9	814.2	984.2

表 3：解の改善率

使用手法		工程数						
I	M_{ut}	10	20	30	40	60	80	100
single-swap-best		1.006	1.009	1.031	1.048	1.075	1.104	1.110
single-swap-first		1.006	1.008	1.031	1.047	1.079	1.110	1.131
double -swap-best		1.006	1.010	1.039	1.064	1.109	1.137	1.163
double -swap-first		1.006	1.010	1.035	1.061	1.109	1.148	1.179
100	0.1	1.006	1.011	1.033	1.038	1.046	1.057	1.055
100	0.2	1.006	1.012	1.032	1.051	1.052	1.050	1.061
200	0.1	1.006	1.014	1.036	1.055	1.060	1.067	1.068
200	0.2	1.006	1.013	1.039	1.060	1.059	1.062	1.064

が得られる．ただし，30 工程以下の場合は GA の方が良い解が得られている．また，解の改善は 20 工程まででは 1%程度の改善しかされないため，3%以上の改善が見込める 30 工程以上の場合に特に有効であると考えられる．

5.3.3　計算時間の比較

平均計算時間を表 4 に示す．single-swap-best の計算時間は single-swap-first よりも若干遅く, かつ近似解に大きな違いはないため, single-swap-first の方が良い解法といえる．また，2-swap-first は 2-swap-best よりも計算時間が短く，求めることのできる近似解に大きな違いはないので，2-swap-first の方が良い解法といえる．したがって，資源競合の解消問題を局所探索で解く場合，近傍の中から最も良い解を見つけて改善せずに，現在の近似解より良い評価値の最初に見つけた解で改善した方が, 計算時間を短縮できるた

表 4：計算時間の平均(s)

使用手法		工程数						
I	M_{ut}	10	20	30	40	60	80	100
Three methods		0.02	0.03	0.06	0.10	0.25	0.44	0.66
single-swap-best		0.02	0.10	0.58	2.28	11.97	40.67	89.25
single-swap-first		0.02	0.10	0.46	1.69	9.86	34.65	87.26
double-swap-best		0.04	0.36	3.90	21.53	219.47	937.99	2,957.24
double-swap-first		0.02	0.28	2.37	10.96	100.34	452.10	1,598.32
100	0.1	1.68	6.12	16.43	26.11	68.41	122.04	183.54
100	0.2	1.86	6.40	15.22	30.79	63.44	114.35	171.22
200	0.1	3.46	11.05	33.29	55.64	135.78	243.92	350.83
200	0.2	3.40	11.77	33.67	64.92	128.34	215.74	326.66

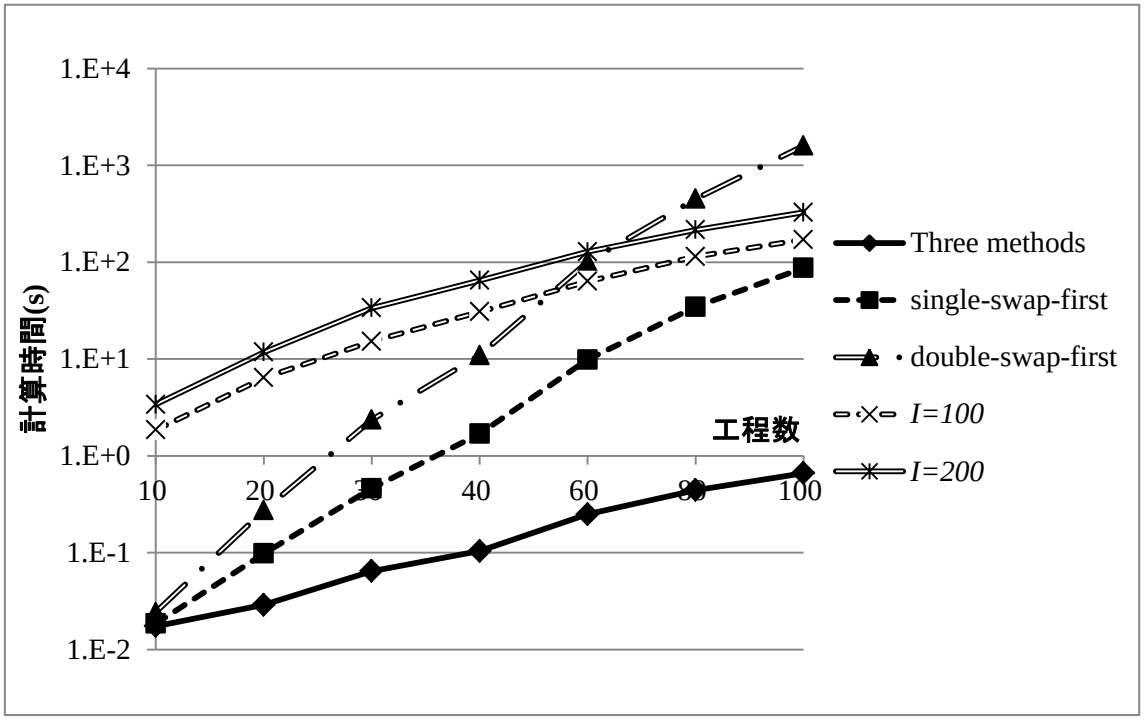

図 29：平均計算時間の対数グラフ

め良い解法といえる．GA を利用した解法では，工程数別に各突然変異率の計算時間を比べると，突然変異率を変えても計算時間は大きくは変わっていない．したがって GA の突然変異率は, 0.1 よりも 0.2 の方が良い場合が多いとわかる．また, *i* を 100 から 200 に増やすと，計算時間は約二倍になる．

図 29 は各提案手法の計算時間を対数グラフにしたものである．ただし，GA の計算時間は突然変異率 M_{ut} が 0.2, 試行回数 I の場合の計算時間の平均値である．局所探索の場合，計算時間は工程数が増えるにつれて著しく増加しているが，GA の場合は 10 工程増える毎に増加する計算時間が局所探索に比べて少ない．80 工程の場合，2-swap-first は GA の計算時間を上回っており，同様に他の局所探索のアルゴリズムも，工程数が増えていけば GA より時間がかかると考えられる．したがって，工程数が多いほど，局所探索よりも GA の方が良い計算時間で解を求めることができることが分かる．

また，表 5 に計算時間の増加比を示す．表中の比較する工程 A/B とは，「A 工程の計算時間と B 工程の計算時間の比較」という意味である．また，計算時間の増加比とは，A 工程の計算時間が B 工程の何倍になるのかを計算した数値である．表 5 より，局所探索の計算時間の増加率は工程数が増えるにつれて大きくなっており，GA の増加率は 3 から 5 倍辺りに落ち着いていることが確認できる．このことからも，入力プロジェクトの工程数が多い場合には, 計算時間は GA の方が局所探索より短くなると予想できる．また，GA の計算時間は工程数が 2 倍になると 3 から 5 倍となることから，計算量のオーダーはプロジェクトの工程数をnとすると$O(n^2)$に近いと考えられる．

表 5：計算時間の増加比

使用手法		比較する工程		
I	M_{ut}	20/10	40/20	80/40
Three methods		1.66	3.58	4.27
single-swap-best		4.41	21.74	17.87
single-swap-first		5.30	17.22	20.45
double-swap-best		9.53	60.11	43.57
double -swap-first		11.30	39.84	41.27
100	0.1	3.65	4.27	4.67
100	0.2	3.45	4.81	3.71
200	0.1	3.20	5.04	4.38
200	0.2	3.46	5.52	3.32

局所探索の増加率は 20/10 の場合と, 40/20, 80/40 の場合の増加率に大きな差がある．20/10 の場合を考慮しないとすると，single-swap-best と 2-swap-best の増加率は工程数が増えるにつれて減少しており, single-swap-first と 2-swap-first の増加率は増加している．増加率の変化は 2-swap-best を除いて大きな差ではないので，40/20 と 80/40 の比較のみで局所探索のオーダーを推測する．その場合，single-swap-best と single-swap-first は工程数が 2 倍になると計算時間が約 20 倍となるので，オーダーは$O(n^{4.5})$と推測できる．2-swap-best の場合は工程数が 2 倍になると計算時間が約 50 倍になると考え$O(n^{7})$と推測でき, 2-swap-first の場合は工程数が 2 倍になると計算時間が約 40 倍になると考え$O(n^{6.3})$と推測できる．

5.3.4 近似比

近似比は表 6 のとおりである．近似比とは，近似解を最適解で割った値であり，1 に近いほど最適解に近い良い解が求められている．表 6 を見ると，局所探索法と GA で求めた近似解は 20 工程の場合で 0.06%以下となっており，提案した近似解法はとても良い解を得ることができている．また，GA の方が局所探索よりも良い解が得られていることが分かる．

5.3.5 最適解を求める計算時間

最適解を求める計算時間は表 7 のとおりとなった．10，15 工程での計算時間は一つのサンプルに対する計算時間が最大 94 秒で平均計算時間も 3 秒とかかっておらず，十分実用的である．しかし，20 工程の場合は最大計算時間が約 20 時間かかるので一部のサンプルに対して実用的ではない．ただし，最小計算時間は 0.3 秒と小さいので，20 工程以下の場合は最適解を短時間で求められるかどうかを 10 秒～1 分程度試し，求められなかった場合は近似解法を利用することを考慮に入れて良いかもしれない．その際に

表 6：近似比の平均

使用手法		工程数		
I	M_{ut}	10	15	20
Three methods		1.00644	1.00684	1.01382
single-swap-best		1.00090	1.00367	1.00500
single-swap-first		1.00010	1.00239	1.00546
double-swap-best		1.00003	1.00239	1.00332
double-swap-first		1.00010	1.00239	1.00357
100	0.1	1.00006	1.00179	1.00247
100	0.2	1.00045	1.00114	1.00149
200	0.1	1.00039	1.00142	1.00000
200	0.2	1.00000	1.00074	1.00097

表 7：最適解を求める手法の平均計算時間

	工程数		
	10	15	20
平均時間	0.08	2.44	759.82
最大時間	1.26	94.47	73,519.93
最小時間	0.02	0.02	0.03

利用する近似解法は表 6 の近似比より，20 工程の場合に最も良い解を得られていた I=200，M_{ut} =0.1 の GA が良いと考えられる．

5.3.6　改善率と計算時間の頻度

本節では，各提案手法の解の改善率や計算時間にどの程度のばらつきがあるかを調べる．

図 30 に 20 工程の場合の厳密解，GA，局所探索の計算時間のヒストグラムを示す．ただし，GA は一番良い平均解を得られた I=200，M_{ut}=0.1 の場合であり，局所探索は double-swap-best である．計算時間が 20 秒の場合は，そのひとつ前の横軸の数値 10 秒～20 秒の間の計算時間であることを示す．図 30 より，GA の計算時間は 10～20 秒に集中していることが分かる．また，double-swap-best は 0.01 秒～1 秒に集中している．厳密解を求める計算時間は 10 秒以下であることが多いが，10,000～100,000 秒近くかかる場合もある．このことから，厳密解を求める計算時間は入力データによって大きく異なり不安定であるが，局所探索や GA の計算時間は比較的安定しているといえる．

図 31 に 100 工程の場合の解の改善率の頻度を示す．ただし，GA は一番良い平均解

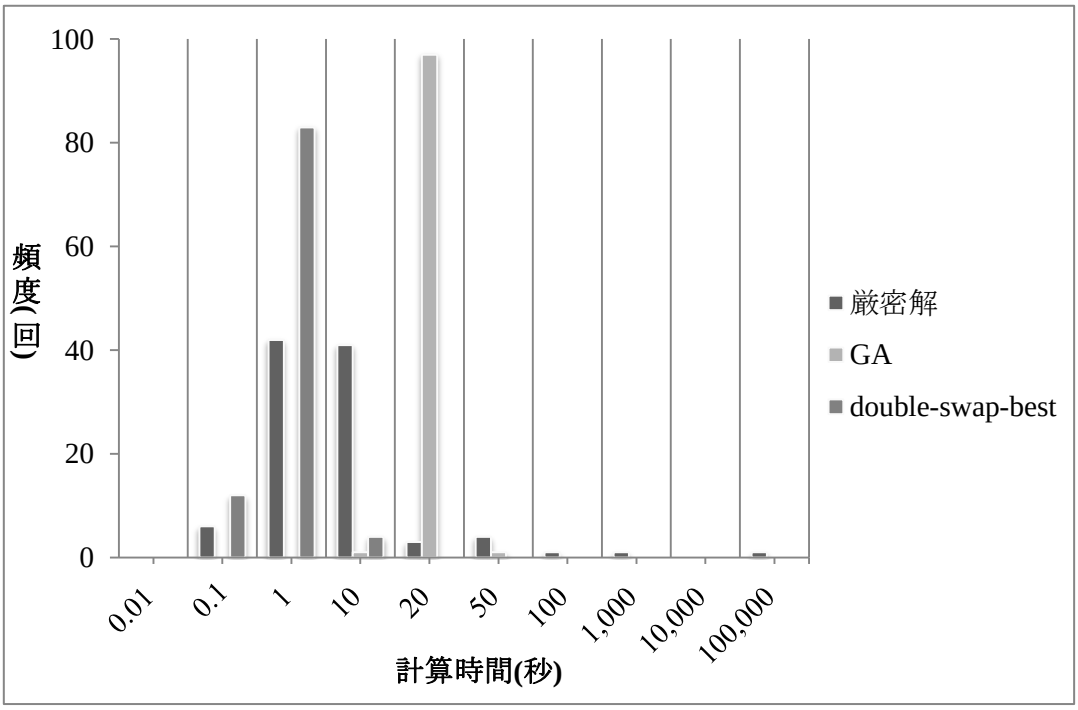

図 30：20 工程の場合の計算時間のヒストグラム

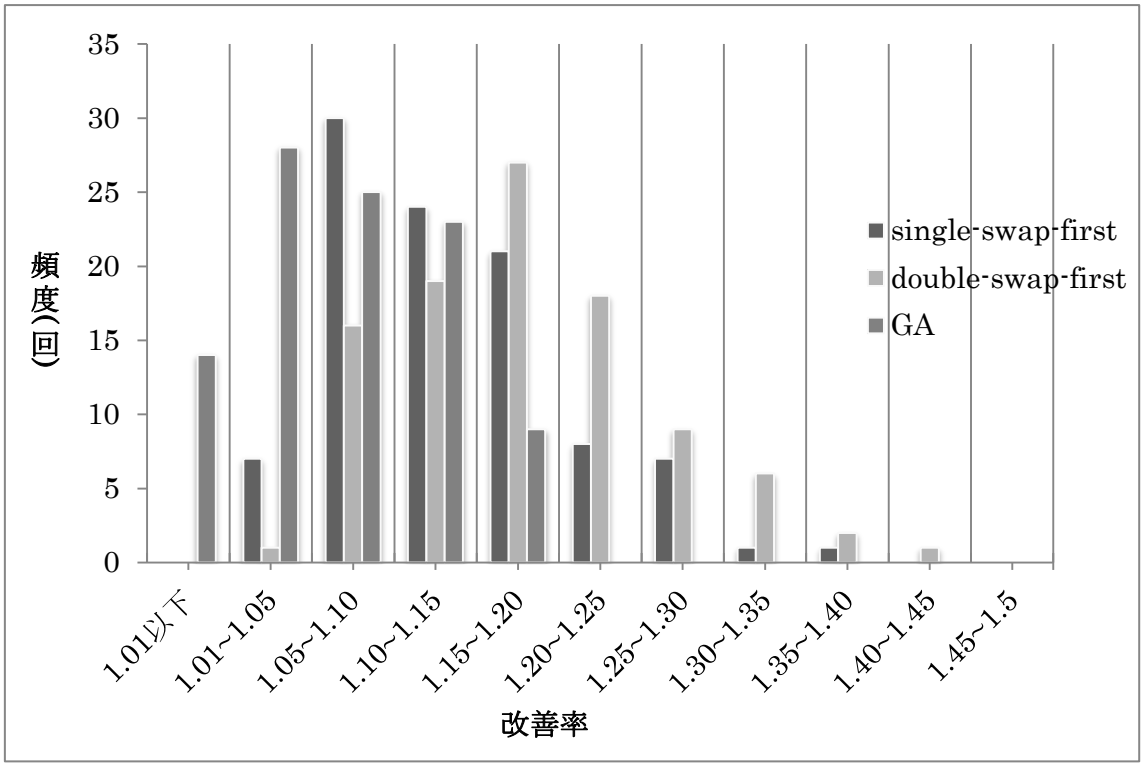

図 31：100 工程の場合の改善率のヒストグラム

を得られた I=200，M_{ut}=0.1 とする．図 31 より，最も改善率の頻度が大きいのは single-swap-first の場合は 1.05～1.1, double-swap-first の場合は 1.15～1.2 であることがわかる. single-swap-first と double-swap-first の改善率は平均を取ると 13.1 と 17.9 であるので，single-swap-first の改善率の平均値と最頻値には違いがある．また，single-swap-first と double-swap-first は改善率が 1.4 近くと, 大きく改善できる場合があることが分かる．GA の改善率は 1.01 以下の場合が 14 ケースあるが，局所探索では 0 ケースである．したがって, 局所探索と比べて GA ではほとんど解を改善できてない場合があることが分かる．20 工程以下の場合，GA は最も有効な近似解法であったが，工程数が大きい場合は有効でない場合も多い.

5.3.7　実験結果のまとめ

今節では，計算実験から分かった各工程数の場合において有効な手法をまとめる.

プロジェクトの工程数が 1～20 工程の場合は, 遺伝アルゴリズムと厳密解を求める手法が，良い解を求めることができるので有効である．ただし，厳密解を求める手法の場合は解を求めるのに 1 日近くかかる場合があるので注意する必要がある.

21～60 工程の場合は，計算時間と評価値どちらも GA よりも優れている single-swap-first か double-swap-first が有効である．より短い計算時間で近似解を求める必要がある場合は single-swap-first を利用し，single-swap-first と比べて 5～10 倍程度計算時間をかけてでもより良い解を求めたい場合は double-swap-first を利用すると良い.

61 工程以上の場合は，GA よりも double-swap-first の計算時間が長くなる．そこで，計算時間の都合で double-swap-first よりも GA の方が良い場合がある．ただし 100 工程までの場合では, GA よりも single-swap-first の方が近似解と計算時間が優れているため有効ではない．101 工程以上の場合，double-swap-first は実用的な計算時間で解を求めることができない.

表 8：工程ごとの有効な手法

	工程数		
	1~20	21~100	101~
有効な手法	GA(I=100 もしくは 200，M_{ut}=0.2)	single-swap-first	single-swap-first
	厳密解を求める手法	double-swap-first	GA(I=100 もしくは 200，M_{ut}=0.2)

6 おわりに

本研究では, クリティカルチェーン法の手順の一つである資源競合の解消するための近似解法と，工程数が少ない場合のみ利用できる厳密解を求める手法を提案した．提案した手法は，三つの簡素な近似解法と，得られた近似解を局所探索法もしくは遺伝アルゴリズムによって改善する手法である．これらの手法でどの程度厳密解に近い値を求めることができるかを，計算実験によって調べた．また，全数列挙を改良した最適解を求める手法も提案し，その計算時間を調べた．その結果，1 出力のプロジェクトの問題のうち，厳密解を求めることのできた 20 工程以下を解いた場合には，最適解に近い近似解が得られることを確認できた．また，20 工程以下の場合には GA の方が良い解が得られ, 30 工程以上の場合には局所探索の方が GA に比べて良い解が得られた．ただし，局所探索の計算時間は扱う工程数が大きくなるにつれて大幅に増えた．したがって，扱うプロジェクトが大きい場合は計算時間が局所探索よりも増加しにくい, 遺伝アルゴリズムを利用した解法を採用した方が良い場合もある．厳密解を求める手法は，15 工程以下の場合は実用的な計算時間で最適解を求めることができたが，20 工程の場合は実用的な計算時間で解を求められないことがあった．

今後の課題としては, 本研究で提案した手法の改良や整数計画問題への帰着などがある．

参考文献

[1] 森村英典，刀根薫，伊理正夫：「経営科学ＯＲ用語大辞典」，株式会社朝倉書店，1999.

[2] L. P. Leach: Critical Chain Project Management Second Edition, ARTECH HOUSE, New York, 1998.

[3] Goldratt, E., M.: Theory of Constraints: And how it should be implemented. North River Pr, 1990.

[4] 中島秀隆，津曲公二：PM プロジェクト・マネジメント　クリティカルチェーン，日本能率協会マネジメントセンター，2003.

[5] H. Goto, N. T. N. Truc, and H. Takahashi: Simple representation of the critical chain project management framework in a max-plus linear form, Journal of Control, Measurement, and System Integration, vol. 6, no. 5, pp. 341–344, 2013.

[6] 野々部宏司，茨木俊秀：資源制約付きスケジューリング問題の定式化と近似解法，数理解析研究所講究録，vol. 1120，pp.88–97，1999.

[7] N. Maculan, S. C. S. Porto, C. C. Ribeiro, C. C. de Souza: A new formulation for scheduling unrelated processors under precedence constraints, RAIRO Operations Research , vol. 33, no. 1, pp. 87–92, 1999.

[8] 雨宮孝，竹安数博，増田士朗：新しい経営・経済数学，中央経済社, 2004.

[9] 永田裕一：多点探索型アルゴリズムの基礎と最前線，オペレーションズ・リサーチ，vol.58, no.12，pp.708–712，2013.

[10] 伊庭斉志：遺伝アルゴリズムの基礎，オーム社，1996.

謝辞

本研究を行うにあたり，法政大学大学院 理工学研究科 システム工学専攻 五島洋行教授にはお忙しい中，親身に御支援とご指導を賜りました．本研究を進行，論文を執筆することができたのは，五島洋行 教授のおかげです．深謝申し上げます．

また，本研究を行うにあたり，多くの助言をくださった法政大学大学院 理工学研究科 システム工学専攻 千葉英史専任講師に心より感謝申し上げます．

本論文の副審を引き受けて下さった法政大学大学院 理工学研究科 システム工学専攻の木村光宏教授にも心より感謝申し上げます．

研究業績一覧

・紀要

[1] 古賀 裕紀，五島 洋行，千葉 英史，”CCPM 法の枠組みにおける資源競合の解消方法”，法政大学理系学部研究集報, ISSN: 2188-8507, vol.50, pp.7-12, 2014.

・国際会議

[2] Hiroki KOGA, Hiroyuki GOTO, and Eishi CHIBA, "Performance Evaluation of the Saving Method and the Optimal Partitioning Method for a Vehicle Routing Problem", Asia Pacific Industrial Engineering & Management Systems Conference 2013 (APIEMS 2013), In Cebu, Paper ID 1237 (11 pages), 2013.

[3] Hiroki KOGA，Hiroyuki GOTO，and Eishi CHIBA, "Resolution of Resource Conflicts in the CCPM Framework Using a Local Search Method", The IEEE International Conference on Industrial Engineering and Engineering Management (IEEM '2014), In Malaysia, Paper ID 206 (5 pages), 2014.

・口頭発表

[4] 古賀 裕紀，五島 洋行，千葉 英史，豊田 一裕，”CCPM 法の枠組みにおける遺伝的アルゴリズムを利用した資源競合の解消法”，計測自動制御学会 第 55 回離散事象システム研究会，pp.35-40，2014.

[5] 古賀 裕紀，五島 洋行，千葉 英史，”CCPM 法の枠組みにおける資源競合の解消方法 ～遺伝的アルゴリズムの活用～”，計測自動制御学会 システム・情報部門 学術講演会 2014, pp.641-646, 2014.