残余裕時間を可視化した max-plus 線形システム

廣田 良元(09X4085)　指導教員 五島 洋行

1. はじめに

本研究では，max-plus 線形システムを適用したクリティカルチェーン法において，残余裕時間を可視化することを検討する．クリティカルチェーン法は，各工程における作業の遅延を考慮した上でプロジェクトの所要時間を短縮し，かつ納期遅れを減らすためのプロジェクト管理手法である[1]．一方 max-plus 線形システムとは，max-plus 代数を用いてモデル化したシステムである．これら二つを組み合わせることで，研究開発のような大規模プロジェクトをスケジューリングすることができる．

2. Max-plus 線形システム

2.1　Max-plus 代数

Max-plus 代数[2]とは，$\mathcal{D}=\mathbb{R}\cup\{-\infty\}$において，max 演算と plus 演算を基本とする代数系である．以下のように演算子を定義する．

$$x\oplus y=\max\{x,y\} \tag{1}$$
$$x\otimes y=x+y \tag{2}$$
$$x\wedge y=\min\{x,y\} \tag{3}$$
$$x\odot y=-x+y \tag{4}$$

演算子$\oplus$，$\otimes$の単位元を，それぞれ$\varepsilon=-\infty$，$e=0$と定義する．行列での演算の定義は以下の通りである．行列$\boldsymbol{X},\boldsymbol{Y}\in\mathcal{D}^{m\times n}$に対し，

$$[\boldsymbol{X}\oplus\boldsymbol{Y}]_{ij}=[\boldsymbol{X}]_{ij}\oplus[\boldsymbol{Y}]_{ij} \tag{5}$$
$$[\boldsymbol{X}\wedge\boldsymbol{Y}]_{ij}=[\boldsymbol{X}]_{ij}\wedge[\boldsymbol{Y}]_{ij} \tag{6}$$

$\boldsymbol{X}\in\mathcal{D}^{m\times l}$，$\boldsymbol{Y}\in\mathcal{D}^{l\times n}$のとき，

$$[\boldsymbol{X}\otimes\boldsymbol{Y}]_{ij}=\max_{k=1,2,\dots,l}\{[\boldsymbol{X}]_{ik}\otimes[\boldsymbol{Y}]_{kj}\} \tag{7}$$
$$[\boldsymbol{X}\odot\boldsymbol{Y}]_{ij}=\min_{k=1,2,\dots,l}\{[\boldsymbol{X}]_{ik}\odot[\boldsymbol{Y}]_{kj}\} \tag{8}$$

2.2　Max-plus 線形表現

前節で定義した演算子をもとに，対象システムをモデリングする．工程i（$1\le i\le n$）の作業時間をd_iとし，システムの構造を表現するベクトル$\boldsymbol{d}$，行列$\boldsymbol{P},\boldsymbol{F}_0,\boldsymbol{B}_0$および$\boldsymbol{C}_0$を，以下のように定義する．

$$[\boldsymbol{d}]_i=d_i \tag{9}$$
$$\boldsymbol{P}=\mathrm{diag}(\boldsymbol{d}) \tag{10}$$

$$[\boldsymbol{F}_0]_{ij}=\begin{cases}e：工程i\text{が先行工程}j\text{を持つ}\\ \varepsilon：上記以外\end{cases} \tag{11}$$

$$[\boldsymbol{B}_0]_{ij}=\begin{cases}e：工程i\text{が外部入力}j\text{を持つ}\\ \varepsilon：上記以外\end{cases} \tag{12}$$

$$[\boldsymbol{C}_0]_{ij}=\begin{cases}e：工程j\text{が外部出力}i\text{を持つ}\\ \varepsilon：上記以外\end{cases} \tag{13}$$

また，kを初期状態からの事象の発生回数とし，各工程における作業開始時刻，作業終了時刻，外部入力からの入力時刻，作業終了時刻をそれぞれ$\boldsymbol{x}^-(k),\boldsymbol{x}^+(k),\boldsymbol{u}(k),\boldsymbol{y}(k)$とすると，各工程の最早終了時刻$\boldsymbol{x}_E^+(k)$，最早終了時刻$\boldsymbol{y}_E(k)$，最遅開始時刻$\boldsymbol{x}_L^-(k)$は以下のように与えられる．

$$\boldsymbol{x}_E^+(k)=(\boldsymbol{PF}_0)^*\boldsymbol{P}[\boldsymbol{x}^+(k-1)\oplus\boldsymbol{B}_0\boldsymbol{u}(k)] \tag{14}$$
$$\boldsymbol{y}_E(k)=\boldsymbol{C}_0\boldsymbol{x}_E^+(k) \tag{15}$$
$$\boldsymbol{x}_L^-(k)=[(\boldsymbol{PF}_0)^*\boldsymbol{P}]^T\odot[\boldsymbol{C}_0^T\odot\boldsymbol{B}_0\boldsymbol{y}_E(k)] \tag{16}$$

ただし，$(\boldsymbol{PF}_0)^*$は工程間の伝搬時間を表す．トータルフロート$\boldsymbol{\omega}(k)$は，以下のように与えられる．

$$[\boldsymbol{\omega}(k)]_i=(d_i\odot[\boldsymbol{x}_E^+(k)]_i)\odot[\boldsymbol{x}_L^-(k)]_i \tag{17}$$

クリティカルパスは以下の条件を満たす工程αの集合である．

$$\{\alpha|[\boldsymbol{\omega}(k)]_\alpha=0\} \tag{18}$$

3. クリティカルチェーン法

クリティカルチェーン法と従来の PERT との違いは，主に本章で述べる四つがある．

3.1　作業時間の見積もり

一度限りの作業を行うプロジェクト型業務の場合，作業時間のばらつきが正規分布することは稀であり，通常ベータ分布する．ベー

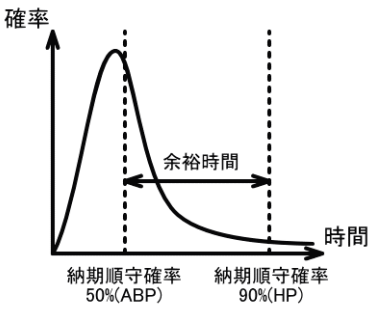

図 1　作業時間の分布

タ分布とは，図 1 のように，正規分布が右方向に長く伸びた確率分布である．作業が 90%の確率で終了する時間は HP，50%の確率で終了する時間は ABP と呼ばれ，ABP は HP の 1/3 倍で与えられる．作業をシミュレーションする時は，$Be(3,6)$に従う乱数を HP に掛けた値を実際の作業時間と考える．

3.2　リソース競合の解消

リソース競合とは，作業者や機械設備といったリソースが，同じ時間帯に複数の工程で使用される状況のことである．クリティカルチェーン法では，リソースは一度に一つの工程の作業しかできないという制約を考える．

3.3　時間バッファの挿入

ABP で作業時間を見積もることによるプロジェクト遅延防止のため，プロジェクトバッファとフィーディングバッファと呼ばれる二種類の余裕時間を挿入する．HP と ABP の差を余裕時間とし，前者はクリティカルパス上の，後者は非クリティカルパス上の工程の総余裕時間を半分にした値に設定する．

3.4　フィーバーチャート

フィーバーチャートとは，横軸にプロジェクトの進捗度，縦軸に消費したプロジェクトバッファの割合を割り当てたグラフであり，安全領域，注意領域，危険領域と呼ばれる三つの領域に分かれている．プロジェクトマネージャーは，プロジェクトバッファの消費状況がどの領域にあるかを確認することで，状況に応じた行動を取ればよい．

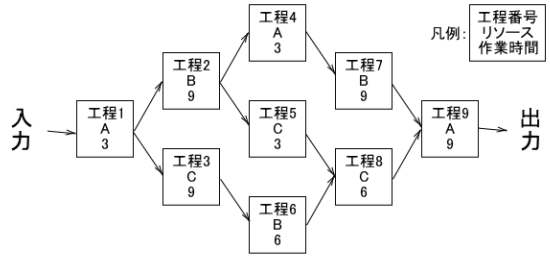

図 2　9 工程のプロジェクトの先行制約関係

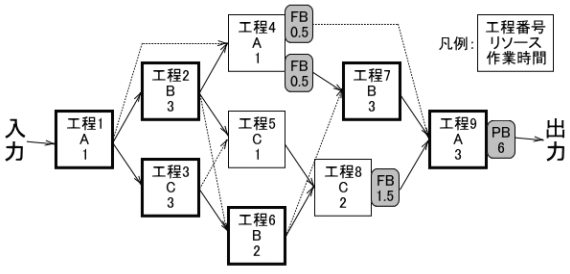

図 3　時間バッファ挿入後の先行制約関係

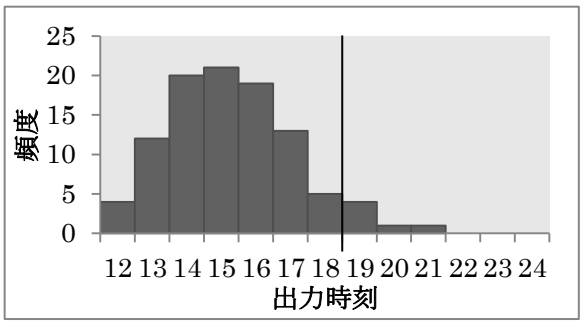

図 4　出力時刻のヒストグラム

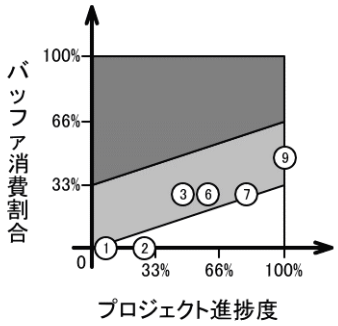

図 5　フィーバーチャート導出例

4. 数値実験

図 2 のようなプロジェクトを考える．このプロジェクトの作業時間の見積もりを HP から ABP に変更し，リソース競合を解消，さらに時間バッファを挿入した結果が図 3 である．このとき，最早終了時刻は 18 と求まる．

このプロジェクトでの作業を 100 回シミュレーションした．図 4 は計算された出力時刻のヒストグラムである．6 回が最早終了時刻 18 を上回ったものの，平均は 14.9 であり，13.1～16.0 に結果が集中したことから，この見積もりは妥当といえる．図 5 は導出したフィーバーチャートの一例である．このように，プロジェクト進捗度に合わせて残余裕時間を可視化できたことから，フィーバーチャートはスケジューリングの一手法として機能することが確認された．

参考文献

[1] 中野明:「エリヤフ・ゴールドラットの「制約理論」がわかる本」，株式会社秀和システム (2006)

[2] 雨宮孝，竹安数博，増田士朗:「新しい経営・経済数学」，株式会社中央経済社 (2004)

残余裕時間を可視化した max-plus 線形システム

指導教官　　五島洋行

法政大学　理工学部　経営システム工学科

経営数理工学研究室

2013 年 2 月

09X4085　　廣田良元

論文要旨

本論文では，max-plus 線形システムを適用したクリティカルチェーン法における，残余裕時間の可視化について検討する．

プロジェクトの管理手法の一つに，クリティカルチェーン法がある．これは，各工程における作業の遅延をあらかじめ念頭においた上でプロジェクトの所要時間を短縮し，かつ納期遅れを減らす手法であり，作業時間に不確実性があるプロジェクト型業務にも適用できる手法である．従来の PERT/CPM との違いは，1.作業時間の見積もり方法の変更，2.リソース競合の解消，3.時間バッファと呼ばれる余裕時間の挿入，4.フィーバーチャートによる進捗管理の四点が挙げられる．

一方 max-plus 線形システムとは，max 演算と plus 演算を基本とする max-plus 代数を用いてモデル化したシステムである．このシステムは，同一商品を繰り返し製造する生産システムなどの記述に適しており，複雑な先行制約関係を持つ大規模なシステムであっても扱うことができる．

先行研究では，クリティカルチェーン法の概念を max-plus 線形システムに適用し，作業時間に不確実性があり，かつ複雑な先行関係を持つ大規模なプロジェクトにおけるスケジューリングを可能にした．また，他の先行研究では，ベータ分布を用いて作業時間を確率的に変動させることにより，作業時間の見積もり方法が妥当であることを確認した．

本研究では，リソース競合が発生する場合および時間バッファの挿入を行った場合において，作業時間を確率的に変動させ，シミュレーションを行った．その結果，リソース競合が発生する場合でもシミュレーションを行うことができ，時間バッファの大きさが妥当であることを確認した．また，各工程の残余裕時間を可視化する，フィーバーチャートの導出を行うこともできた．

目次

第1章　序論

本論文では，max-plus 線形システムを適用したクリティカルチェーン法における，残余裕時間の可視化について検討する．まず，研究背景と先行研究について述べたのち，本研究の目的を明らかにする．

1.1 研究背景

プロジェクトや生産システムのスケジュールを管理する手法の中に，CPM（Critical Path Method）や PERT（Program Evaluation and Review Technique）[1]がある．CPM とは，プロジェクトネットワーク上の，開始から終了までの最長経路であるクリティカルパス（Critical Path）を決定し，作業全体の戦略やスケジュールを立てる際に用いられる技法である．一方，PERT とは事象指向型のプロジェクトネットワーク図法の一つであり，計画やスケジューリングの作成に用いられる．これにより，米国海軍のポラリスミサイル計画の完成を，少なくとも二年早めることができたとされる[2]．なお，これら二つを合わせて PERT/CPM と呼ぶこともある．本論文では，以後 PERT/CPM を従来法と呼ぶことにする．

クリティカルチェーン法は，Goldratt が提唱した制約理論に基づき，各工程における作業の遅延を考慮した上でプロジェクトの所要時間を短縮し，かつ納期遅れを減らすためのプロジェクト管理手法である．クリティカルチェーン法と従来法の違いは，主に以下の四つがある．

1. 作業時間の見積もり方法の変更
2. リソース競合の解消
3. 時間バッファと呼ばれる余裕時間の挿入
4. フィーバーチャートによる進捗管理

1.について述べる．従来法ではあらかじめ余裕を持たせた作業時間で見積もっていたが，これには無駄な余裕時間が含まれていることがあり，さらにそれが原因となり作業に遅延が生じる．これに対しクリティカルチェーン法では，作業時間から余裕時間を削減することで遅延防止と納期短縮を図る．2.～4.は従来法にはなかった手法で，これらも遅延防止や納期短縮を目的としている．

このクリティカルチェーン法は，作業時間を事前に見積もることが困難なプロジェクト型業務への適用が容易である．しかし，スケジュール作成後に予定が変更されるなどといった，再スケジューリングが必要な場合への対応が難しい．また，多入力多出力に対応できず，複雑かつ大規模なプロジェクトには適さない手法である．

一方，max-plus 線形システムとは max 演算と plus 演算を基本とする max-plus 代数を

用いてモデル化したシステムである. 中でも, 生産システム, プロジェクト管理, 交通システムといった, 離散事象システムのモデリング方法の一つとして有効である. Max-plus 線形システムは, 同一商品を繰り返し製造する生産システムなどの記述に適し, 多入力多出力型のシステムや, 複雑な先行制約関係を持つ大規模なシステムであっても扱うことができる. その反面, 事前に設定した作業時間をもとにスケジュールを立てるため, 一度限りの作業が多い研究開発プロジェクト業務など, 作業時間を事前に見積もることが困難な場合には適さない.

1.2 先行研究

笠原ら[3]は, クリティカルチェーン法の max-plus 線形システムへの応用について検討された. 工程間の制約条件や作業時間を max-plus 線形システムで表現し, さらにクリティカルチェーン法の特徴の一つである作業時間の見積もり変更も行った. 具体的には, 作業が 90%の確率で終了する作業時間で見積もられていたものを, 50%の確率で終了する作業時間に変更した. 後者は, 前者の 1/3 倍で与えられることが多いため, 作業時間の見積もり時間を 1/3 倍にした. また, 時間バッファと呼ばれる余裕時間の挿入も行った. これにより, 従来法と比較して納期が 2/3 倍と短縮できることが確認された.

吉田[4]は, 笠原ら[3]から表現行列の定義を一部変更し, 時間バッファの挿入方法を単純化した. さらに, 多出力のプロジェクトにおいてもフィーバーチャートを導出できる方法を提案した. また, 笠原らの先行研究ではリソース競合を考慮しない場合を扱っていたが, 吉田の先行研究ではリソース競合を解消する手法も提案し, より納期遅れのリスクを軽減させた. 笠原らや吉田の研究により, 作業時間に不確実性があり, かつ複雑な先行関係を持つ大規模なプロジェクトにおいてもスケジューリングを可能にした.

白石ら[5]は, max-plus 線形システムを適用したクリティカルチェーン法において, 作業時間が確率的に変動することを想定し, 作業時間を 1/3 倍にすることで納期が 2/3 倍に短縮できるか検討された. そのために, 作業時間を 1/3 倍にするベータ分布を実現させ, それを用いて作業時間を確率的に変動させることで, より実際の状況を表現した. これに基づきシミュレーションを行ったところ, 従来法と比較して納期が 2/3 倍以下に短縮できていることが確認された.

1.3 本研究の目的

本研究では, 吉田[4]で有効性が確認された手法を用い, 白石ら[5]のように作業時間を確率的に変動させ, 作業のシミュレーションを行うことが目的である. 白石ら[5]では, 作業時間を 1/3 倍にするベータ分布を実現させ, シミュレーションを行うことで納期を 2/3 倍にできることが実証された. しかしこのシミュレーションは, リソース競合や, 時間バッファの消費については触れられていなかった. それを踏まえ, 本研究では簡単なプロジェクトを用いてシミュレーションを行い, リソース競合や時間バッファを考慮しても見積もり

が妥当であることを確認する. そして, 白石ら[5]では行われなかったフィーバーチャートへのプロットも行い, 残余裕時間を可視化する方法も検討する.

ただし, 吉田[4]では, リソース競合解消法や時間バッファ挿入など, 一部の手続きが複雑である. このため, 本研究では先行研究の一部の手続きを変更し, 簡略化する. これについては第 3 章で詳しく述べる.

1.4 論文の構成

本論文は, 全 5 章で構成される.

第 2 章では, 数学的準備として max-plus 代数について説明し, それを用いてシステムの振る舞いを線形な状態方程式で表現する方法を述べる.

第 3 章では, クリティカルチェーン法の概要を述べる. 具体的には, リソース競合の解消法, 作業時間の見積もり方, 時間バッファの挿入方法, およびフィーバーチャートの概念を記述する. また, max-plus 線形システムを用いた手法も述べる.

第 4 章では, 作成した簡単な例を用いてシミュレーションを行い, 第 3 章で述べる手法の有効性を確認する.

第 5 章では, 本論文のまとめを行い, 今後の課題についても述べる.

第2章　Max-plus 線形システム

本章では, max-plus線形システムについて述べる. 2.1では数学的準備として, max-plus代数の基本演算規則をまとめる. 2.2では, max-plus線形システムとその振る舞いについて記述する. そして2.3ではmax-plus線形システムに用いる行列を具体的に定義し, 最早終了時刻などの導出方法も述べる.

2.1 Max-plus 代数

Max-plus 代数[6]とは, $\mathcal{D} = \mathbb{R} \cup \{-\infty\}$において, max 演算と plus 演算を基本とする代数系である．ここで，$\mathbb{R}$は実数全体の集合である．$x, y \in \mathcal{D}$に対し，max 演算および plus 演算を以下のように定義する.

$$x \oplus y = \max\{x, y\} \tag{2.1}$$
$$x \otimes y = x + y \tag{2.2}$$

また, 以下の演算子も合わせて定義する.

$$x \wedge y = \min\{x, y\} \tag{2.3}$$
$$x \odot y = -x + y \tag{2.4}$$

これら四つの演算子を用いた簡単な計算例を以下に示す.

$$3 \oplus 5 = \max\{3,5\} = 5 \tag{2.5}$$
$$5 \otimes (-4) = 5 + (-4) = 1 \tag{2.6}$$
$$(-2) \wedge 4 = \min\{-2,4\} = -2 \tag{2.7}$$
$$2 \odot 6 = -2 + 6 = 4 \tag{2.8}$$

次に, 単位元を導入する. $\oplus$, $\otimes$および$\wedge$の単位元をそれぞれ$\varepsilon = -\infty$, $e = 0$および$T = \infty$と定義すれば, 式(2.1)～(2.3)より以下の等式が成り立つ.

$$x \oplus \varepsilon = \varepsilon \oplus x = x \tag{2.9}$$
$$x \otimes e = e \otimes x = x \tag{2.10}$$
$$x \wedge T = T \wedge x = x \tag{2.11}$$

上記単位元について, 以下のような公理を定義する.

$$\varepsilon \otimes T = T \otimes \varepsilon = \varepsilon \tag{2.12}$$

なお, ⊕演算子には逆元が存在しない. 例えば, 与えられた$x \in \mathcal{D}$に対し,

$$x \oplus y = y \oplus x = \varepsilon \tag{2.13}$$

を満たす$\mathrm{y} \in \mathcal{D}$が存在すれば, その$y$が$x$の逆元である. しかし, $x \neq \varepsilon$であれば $x \oplus y = y \oplus x \geq x > \varepsilon$となり, 式(2.13)を満たす$y$は存在しない. よって, ⊕演算子には逆元が存在しないことが言えた.

Max-plus 代数は, 通常の代数系にはない, 加法のべき等性と呼ばれる特徴をもつ. それは以下の等式で表現できるが, これは式(2.1)より成立することがわかる.

$$x \oplus x = x \tag{2.14}$$

ここで, 導入した演算子の優先度について述べる. 通常の代数系においては, 乗法および除法は, 加法および減法より計算の優先度が高い. それと同様に, ⊗演算子および⊙演算子は, ⊕演算子および⋀演算子より計算の優先度が高いものとする.

以上のように定義した max-plus 代数は, $x, y, z \in \mathcal{D}$に対し, 以下の代数的性質をもつ.

- 交換法則

$$x \oplus y = y \oplus x \tag{2.15}$$
$$x \otimes y = y \otimes x \tag{2.16}$$

- 結合法則

$$x \oplus (y \oplus z) = (x \oplus y) \oplus z \tag{2.17}$$
$$x \otimes (y \otimes z) = (x \otimes y) \otimes z \tag{2.18}$$

- 分配法則

$$x \otimes (y \oplus z) = x \otimes y \oplus x \otimes z \tag{2.19}$$

複数の数$x_k \in \mathcal{D}(k = 1,2, \dots, n)$に対する演算子は, 以下のように記述する.

$$\bigoplus_{k=1}^{n} x_k = x_1 \oplus x_2 \oplus \dots \oplus x_n = \max\{x_1, x_2, \dots, x_n\} \tag{2.20}$$

$$\bigotimes_{k=1}^{n} x_k = x_1 \otimes x_2 \otimes \dots \otimes x_n = x_1 + x_2 + \cdots + x_n \tag{2.21}$$

$$\bigwedge_{k=1}^{n} x_k = x_1 \wedge x_2 \wedge \ldots \wedge x_n = \min\{x_1, x_2, \ldots, x_n\} \tag{2.22}$$

なお, 式(2.21)において$x_1 = x_2 = \cdots = x_n = x$とおくと, 以下の等式が成り立つ.

$$\bigotimes_{k=1}^{n} x_k = n \times x \tag{2.23}$$

ただし, 式(2.23)の×は通常の代数系の乗法とする. これを max-plus 代数ではべき乗のように,

$$x^{\otimes n} = n \times x \tag{2.24}$$

と表現する. 計算の優先度も, ⊗演算子および⊙演算子より高いものとする.

これより, 行列やベクトルにおける演算について述べる. Max-plus 線形代数では, 行列やベクトルの積といった線形演算においても, 通常の演算系と類似した計算方法を与えている. 本論文において, 行列$\boldsymbol{X} = \mathcal{D}^{m \times n}$に対し, $[\boldsymbol{X}]_{ij}$は行列$\boldsymbol{X}$の第(i,j)成分を表し, $\boldsymbol{X}^T$は転置行列を表すものとする.

$\boldsymbol{X}, \boldsymbol{Y} \in \mathcal{D}^{m \times n}$に対し, ⊕演算子および∧演算子を以下のように定義する.

$$[\boldsymbol{X} \oplus \boldsymbol{Y}]_{ij} = [\boldsymbol{X}]_{ij} \oplus [\boldsymbol{Y}]_{ij} \tag{2.25}$$
$$[\boldsymbol{X} \wedge \boldsymbol{Y}]_{ij} = [\boldsymbol{X}]_{ij} \wedge [\boldsymbol{Y}]_{ij} \tag{2.26}$$

また, $\boldsymbol{X} \in \mathcal{D}^{m \times l}$, $\boldsymbol{Y} \in \mathcal{D}^{l \times n}$に対し, ⊗演算子および⊙演算子を以下のように定義する.

$$[\boldsymbol{X} \otimes \boldsymbol{Y}]_{ij} = \bigoplus_{k=1}^{n} ([\boldsymbol{X}]_{ik} \otimes [\boldsymbol{Y}]_{kj}) = \max_{k=1,2,\ldots,l} \{[\boldsymbol{X}]_{ik} \otimes [\boldsymbol{Y}]_{kj}\} \tag{2.27}$$

$$[\boldsymbol{X} \odot \boldsymbol{Y}]_{ij} = \bigwedge_{k=1}^{n} ([\boldsymbol{X}]_{ik} \odot [\boldsymbol{Y}]_{kj}) = \min_{k=1,2,\ldots,l} \{[\boldsymbol{X}]_{ik} \odot [\boldsymbol{Y}]_{kj}\} \tag{2.28}$$

式(2.25)～(2.28)について, 簡単な計算例を以下に示す.

$$\begin{bmatrix}1\\3\end{bmatrix}\oplus\begin{bmatrix}4\\2\end{bmatrix}=\begin{bmatrix}1\oplus4\\3\oplus2\end{bmatrix}=\begin{bmatrix}4\\3\end{bmatrix} \tag{2.29}$$

$$[3\quad 1]\wedge[2\quad 1]=[3\wedge2\quad 1\wedge1]=[2\quad 1] \tag{2.30}$$

$$\begin{bmatrix}1&2\\3&4\end{bmatrix}\otimes\begin{bmatrix}2\\3\end{bmatrix}=\begin{bmatrix}1\otimes2\oplus2\otimes3\\3\otimes2\oplus4\otimes3\end{bmatrix}=\begin{bmatrix}5\\7\end{bmatrix} \tag{2.31}$$

$$\begin{bmatrix}1&3\\2&4\end{bmatrix}\odot\begin{bmatrix}4&5\\3&1\end{bmatrix}=\begin{bmatrix}1\odot4\wedge3\odot3&1\odot5\wedge3\odot1\\2\odot4\wedge4\odot3&2\odot5\wedge4\odot1\end{bmatrix}=\begin{bmatrix}0&-2\\-1&-3\end{bmatrix} \tag{2.32}$$

さらに，$\boldsymbol{x}\in\mathcal{D}^n$を対角成分に並べた行列$\boldsymbol{X}\in\mathcal{D}^{n\times n}$を以下のように定義する．

$$[\boldsymbol{X}]_{ij}=[\mathrm{diag}(\boldsymbol{x})]_{ij}=\begin{cases}[\boldsymbol{x}]_i : i=j\\ \varepsilon \quad : i\neq j\end{cases} \tag{2.33}$$

行列演算における単位元を，以下のように定義する．

$\boldsymbol{\varepsilon}$：すべての成分がε

$\boldsymbol{e}$：対角成分のみe，それ以外の成分はε

2.2 Max-plus 線形システムとは

Max-plus 線形システムは，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，計算機システムなどのように，時間の経過によってではなく，事象の生起によってシステム内の状態変化が引き起こされるシステムである．この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．すなわち，複数の事象がすべて生起することが，別の事象の生起条件となることがある．作業の開始や完了を事象とみなせば，「先行作業がすべて完了することが，後続する作業の開始条件になる」と言い換えることができる．先行作業の完了時刻を$x_1, x_2, \dots, x_n$，後続する作業の開始時刻をyとすると，この条件は以下のように表現できる．

$$y=\bigoplus_{k=1}^{n}x_k=\max\{x_1,x_2,\dots,x_n\} \tag{2.34}$$

このように，事象生起時刻を変数として記述すると，max 演算を用いて表現できる．そして，max 演算が，同期を伴う事象生起条件を記述するのに適していることがわかる．

次に，複数の事象の生起に，時間的な関係を導入する．このとき，「事象生起後，一定時間が経過した後に別の事象が生起する」という条件も必要になる．これも，作業の開始や完了を事象とみなせば，「開始した作業は，一定時間経過してから作業が完了

する」と言い換えることができる. そして, 作業開始時刻を$\boldsymbol{x}$, 作業完了時間を$\boldsymbol{y}$, 作業時間を$\boldsymbol{d}$とすれば, 以下のように書ける.

$$y = x \otimes d = x + d \tag{2.35}$$

この条件は, 事象生起時刻を変数として記述することで, plus 演算を用いて表現できる. 以上のことから, max 演算および plus 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のに有効であり, その二つの演算を基本とする max-plus 代数を用いるのに適していることがわかる.

式(2.34)および式(2.35)は, それぞれ一つの条件式で表すことができたが, モデル化すべきシステムの規模が大きくなると, その分条件式の数も多くなるため, 制約条件をベクトル化し, コンパクトな記述にすることが求められる. さらに, 式(2.34)および式(2.35)では, 一回の事象生起しか考えられていないが, システムの動的な振る舞いを表現するためには, 同じ条件において複数回の事象生起を想定する必要がある. したがって, 事象生起を表す変数は, 事象の生起回数を独立変数とする関数として表現する必要がある.

以上のことを考慮して, システムの振る舞いを数式表現したものが, max-plus 線形システムである. まず, 以下のように変数を設定する.

$\boldsymbol{x}(k) \in \mathcal{D}^n$: 状態変数
$\boldsymbol{u}(k) \in \mathcal{D}^p$: 入力変数
$\boldsymbol{y}(k) \in \mathcal{D}^q$: 出力変数
$\boldsymbol{A} \in \mathcal{D}^{n \times n}$: システム行列
$\boldsymbol{B} \in \mathcal{D}^{n \times p}$: 入力行列
$\boldsymbol{C} \in \mathcal{D}^{q \times n}$: 出力行列

kはイベントカウンタと呼ばれる変数であり, 初期状態からの事象の発生回数を表す. これらを用いると, max-plus 線形システムは以下のように表される.

$$\boldsymbol{x}(k) = \boldsymbol{A} \otimes \boldsymbol{x}(k-1) \oplus \boldsymbol{B} \otimes \boldsymbol{u}(k) \tag{2.36}$$
$$\boldsymbol{y}(k) = \boldsymbol{C} \otimes \boldsymbol{x}(k) \tag{2.37}$$

ここで, max-plus 線形システムを用いたモデリングの具体例を紹介する. 図 1 に示すような 2 入力 1 出力の生産システムを考える. この生産システムは 3 つの機械とベルトコンベアで構成されている. まず, 機械 1 と機械 2 にそれぞれ材料を投入し, 部品を作る. 次に, 出来上がった 2 種類の部品を機械 3 へ送り, 加工して製品にする. このシステムには, 以下の制約条件が課されているとする.

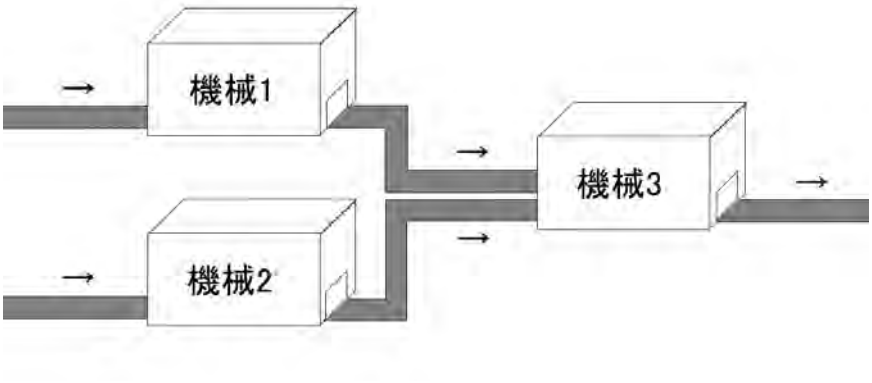

図 1　2 入力 1 出力の生産システム

- 機械 1 と機械 2 は，材料が投入されるまで作業を開始できない．
- 機械 3 は，機械 1 と機械 2 で作られる 2 種類の部品が揃わないと作業を開始できない．
- 各機械が作業を行っている時に材料や部品が送り込まれても，その作業が終わるまで次の作業は開始できない．

機械iにおける作業時間をd_iとし，どの機械も作業するための準備時間は0とする．また，材料や部品を運搬する時間は無視できるものとする．さらに，k番目の製品を作る時における，機械iへの材料の投入時刻を$u_i(k)$，機械iでの作業開始時刻を$x_i(k)$，製品の完成時刻を$y(k)$とする．

このとき，k番目の製品に関して，以下のような関係が成り立つ．

$$x_1(k) = x_1(k-1)\otimes d_1 \oplus u_1(k) = \max\{x_1(k-1)+d_1, u_1(k)\} \tag{2.38}$$

$$x_2(k) = x_2(k-1)\otimes d_2 \oplus u_2(k) = \max\{x_2(k-1)+d_2, u_2(k)\} \tag{2.39}$$

$$\begin{aligned} x_3(k) &= x_1(k)\otimes d_1 \oplus x_2(k)\otimes d_2 \oplus x_3(k-1)\otimes d_3 \\ &= \max\{x_1(k)+d_1, x_2(k)+d_2, x_3(k-1)+d_3\} \end{aligned} \tag{2.40}$$

$$y(k) = x_3(k)\otimes d_3 = x_3(k) + d_3 \tag{2.41}$$

これらの関係式を，行列やベクトルを用い，さらに$[x_1(k) \quad x_2(k) \quad x_3(k)]^T = \boldsymbol{x}(k)$，$[u_1(k) \quad u_2(k)]^T = \boldsymbol{u}(k)$と置き換えて整理すると，以下の通りに表される．

$$\boldsymbol{x}(k) = \begin{bmatrix} d_1 & \varepsilon & \varepsilon \\ d_2 & \varepsilon & \varepsilon \\ d_1^{\otimes 2} & d_2^{\otimes 2} & d_3 \end{bmatrix} \otimes \boldsymbol{x}(k-1) \oplus \begin{bmatrix} e & \varepsilon \\ \varepsilon & e \\ d_1 & d_2 \end{bmatrix} \otimes \boldsymbol{u}(k) \tag{2.42}$$

$$y(k) = [\varepsilon \quad \varepsilon \quad d_3] \otimes \boldsymbol{x}(k) \tag{2.43}$$

これらは式(2.36)および式(2.37)と同じ形であり，max-plus 線形システム表現できたことに

なる.

ここまで生産システムを例に挙げたが, 先行制約を有するプロジェクトのスケジューリング問題にも適用できる.

2.3 Max-plus 線形表現

前節の具体例では, 式変形によって max-plus 線形システム表現を得た. しかし, システム行列, 入力行列および出力行列の一般形についてまだ明確ではない. そこで本節では, これまで説明した先行制約と同期制約を有する離散事象システムを, max-plus 線形システム形式で表現する過程を記述する[4]. ただし, これ以降の数式はすべて max-plus 代数のものとし, ⊗演算子は混乱のない限り省略して記述する.

対象とするシステムに課される条件を, 以下のように設定する.

- 工程数はn, 外部からの入力数はp, 外部への出力数はqである.
- 工程i($1 \leq i \leq n$)の作業時間はd_iとする.
- 各ジョブは全ての工程を 1 回のみ通るものとし, どの工程も割り込みは不可とする.
- 外部入力がある工程は, 入力があるまで作業を開始できない.
- 先行工程を持つ工程は, 全ての先行工程が終わるまで作業を開始できない.
- 作業の準備時間はすべて0とする.

また, システムの構造を表現するベクトル$\boldsymbol{d}$, 行列$\boldsymbol{P}, \boldsymbol{F}_0, \boldsymbol{B}_0$および$\boldsymbol{C}_0$を以下のように定義する.

$$[\boldsymbol{d}]_i = d_i \tag{2.44}$$

$$\boldsymbol{P} = \mathrm{diag}(\boldsymbol{d}) \tag{2.45}$$

$$[\boldsymbol{F}_0]_{ij} = \begin{cases} e: \text{工程}i\text{が先行工程}j\text{を持つ} \\ \varepsilon: \text{工程}i\text{が先行工程}j\text{を持たない} \end{cases} \tag{2.46}$$

$$[\boldsymbol{B}_0]_{ij} = \begin{cases} e: \text{工程}i\text{が外部入力}j\text{を持つ} \\ \varepsilon: \text{工程}i\text{が外部入力}j\text{を持たない} \end{cases} \tag{2.47}$$

$$[\boldsymbol{C}_0]_{ij} = \begin{cases} e: \text{工程}j\text{が外部出力}i\text{を持つ} \\ \varepsilon: \text{工程}j\text{が外部出力}i\text{を持たない} \end{cases} \tag{2.48}$$

さらに, 以下のように記号を定義する.

$\boldsymbol{x}^-(k) \in \mathcal{D}^n$: 各工程における作業開始時刻

$\boldsymbol{x}^{+}(k) \in \mathcal{D}^{n}$：各工程における作業終了時刻
$\boldsymbol{u}(k) \in \mathcal{D}^{p}$：外部入力からの入力時刻
$\boldsymbol{y}(k) \in \mathcal{D}^{q}$：作業終了時刻

このとき，各工程の最早終了時刻$\boldsymbol{x}_E^{+}(k)$は以下のように与えられる．

$$\boldsymbol{x}_E^{+}(k) = (\boldsymbol{PF}_0)^{*}\boldsymbol{P}[\boldsymbol{x}^{+}(k-1) \oplus \boldsymbol{B}_0\boldsymbol{u}(k)] \tag{2.49}$$

ただし，$(\boldsymbol{PF}_0)^{*}$は工程間の伝搬時間を表し，

$$(\boldsymbol{PF}_0)^{*} = \bigoplus_{i=0}^{l-1}(\boldsymbol{PF}_0)^{i} = \boldsymbol{e} \oplus \boldsymbol{PF}_0 \oplus (\boldsymbol{PF}_0)^{2} \oplus \ldots \oplus (\boldsymbol{PF}_0)^{l-1} \tag{2.50}$$

となる．lはシステムの先行制約条件に依存する定数であり，$(\boldsymbol{PF}_0)^{l} = \varepsilon$となる．そして，最早出力時刻$\boldsymbol{y}_E(k)$は以下のように与えられる．

$$\boldsymbol{y}_E(k) = \boldsymbol{C}_0\boldsymbol{x}_E^{+}(k) \tag{2.51}$$

最遅開始時刻は，最早時刻で作業を行った場合と同一の出力時刻が実現できる，最も遅い作業時刻と定義される．よって，最遅開始時刻$\boldsymbol{x}_L^{-}(k)$は以下のように与えられる．

$$\boldsymbol{x}_L^{-}(k) = [(\boldsymbol{PF}_0)^{*}\boldsymbol{P}]^{T} \odot [\boldsymbol{C}_0^{T} \odot \boldsymbol{B}_0\boldsymbol{y}_E(k)] \tag{2.52}$$

また，最遅入力時刻$\boldsymbol{u}_L(k)$は，以下のように与えられる．

$$\boldsymbol{u}_L(k) = \boldsymbol{B}_0^{T} \odot \boldsymbol{x}_L^{-}(k) \tag{2.53}$$

トータルフロート$\boldsymbol{\omega}(k)$は，工程が持つ余裕時間の総合計と定義されるため，以下のように与えられる．

$$[\boldsymbol{\omega}(k)]_i = (d_i \odot [\boldsymbol{x}_E^{+}(k)]_i) \odot [\boldsymbol{x}_L^{-}(k)]_i \tag{2.54}$$

そして，クリティカルパスは，トータルフロートが$\mathbf{0}$となる工程の集合であるから，以下の条件を満たす工程$\boldsymbol{i}$の集合$\boldsymbol{\alpha}$である．

$$\alpha = \{i | [\boldsymbol{\omega}(k)]_i = 0\} \tag{2.55}$$

なお，本研究では一回限りのプロジェクトしか扱わないため，今後はイベントカウンタkや一部の添え字を省略する．

第3章　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について記述する．3.1 では，クリティカルチェーン法の概要を，従来法との違いを含めて述べる．3.2～3.5 では，クリティカルチェーン法の具体的な特徴と，max-plus 線形システムを利用した手法を述べていく．

3.1 クリティカルチェーン法とは

クリティカルチェーン法とは，Goldratt が提唱した制約理論に基づき，各工程における作業の遅延をあらかじめ念頭においた上でプロジェクトの所要時間を短縮し，かつ納期遅れを減らすためのプロジェクト管理手法である．制約理論では，システムの中で一番弱い部分が制約条件であると考え，この制約条件がシステムのパフォーマンスを決定する，としている[7].

クリティカルチェーン法と従来法との違いは，主に四つある．一点目は，作業時間の見積もり方法である．従来法では，プロジェクト内の各工程の作業がスケジュール通りに進むことを前提としており，作業時間にはあらかじめ余裕時間が含まれているが，時間の無駄遣いも含まれている可能性が高い．これに対し，クリティカルチェーン法では余裕時間を考慮しない作業時間を設定する．これにより，プロジェクト全体の納期短縮を図っている．

二点目は，リソース競合の解消である．従来法では，余裕時間のない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，作業員や作業設備といったリソースが競合または不足することがボトルネックとなる場合もあり，それについては考慮されてなかった．そこでクリティカルチェーン法では，プロジェクト計画段階であらかじめリソース競合を解消する．なお，これによりクリティカルパスが変化する場合もある．

三点目は，時間バッファと呼ばれる余裕時間を挿入することである．一点目の違いとして述べたように，余裕時間を考慮しない作業時間を設定するため，従来法より余裕がなくなり，遅れが生じやすくなる．そこで，設定し直した作業時間に基づき，時間バッファの位置と長さを決め，納期遅れを防止する．時間バッファには様々な種類があるが，本論文では，外部出力の直前に挿入するプロジェクトバッファと，非クリティカルパスの工程の直後に挿入するフィーディングバッファの二種類のみを扱う．前者はクリティカルパス上の工程の遅れを吸収する時間バッファであり，後者は非クリティカルパス上の工程の遅れを吸収し，クリティカルパスへの遅れの伝搬を防ぐ時間バッファである．

四点目は，フィーバーチャートによる進捗状況の把握である．これは，横軸にプロジェクトの進捗度，縦軸に時間バッファの消費割合をとるグラフであり，安全領域，注意領域，危険領域と呼ばれる三つの領域に分けられている．これに，プロジェクトの進捗と同時に時間バッファの消費割合をプロットしていくことで，プロジェクトの進捗状況が一目でわかるようになる．

以降は，クリティカルチェーン法の詳細と，max-plus 線形システムを用いた手法を述べていく．

3.2 作業時間の見積もり

クリティカルチェーン法では，プロジェクト全体の所用時間を短縮することを目的に，作業時間の設定方法を変更している．本節では，まず作業時間の見積もり方法について述べたのち，作業のシミュレーションを行う際の作業時間の設定方法について述べる．

3.2.1 正規分布とベータ分布

生産ラインや事務処理といった日常的に繰り返される作業の場合，作業時間のばらつきは正規分布するとみなす場合が多い．ところが，一度限りの作業を行うプロジェクト型業務の場合，作業時間のばらつきが正規分布になることは稀である．グラフにすると，図 2 のように左右対称にはならず，右方向に長く伸びた形になることが知られている．これはベータ分布と呼ばれる確率分布である[8],[9]．従来法，特に PERT では，悲観値，楽観値，最頻値の三つの値から，最頻値に重みを置いた値で作業時間を見積もっている[8]．このため，プロジェクト型業務のスケジューリングにおいては，この方法で作業時間を適切に見積もることができない．

また，プロジェクト型業務では，作業担当者は作業時間を余裕込みで見積もり，作業が 90%の確率で終了する時間をプロジェクトマネージャに申告する場合が多い．これは，担当者がマネージャに申告した作業時間を厳守しなければならず，余裕時間のない作業時間を申請するのはリスクが高いために避けるからである．クリティカルチェーン法で

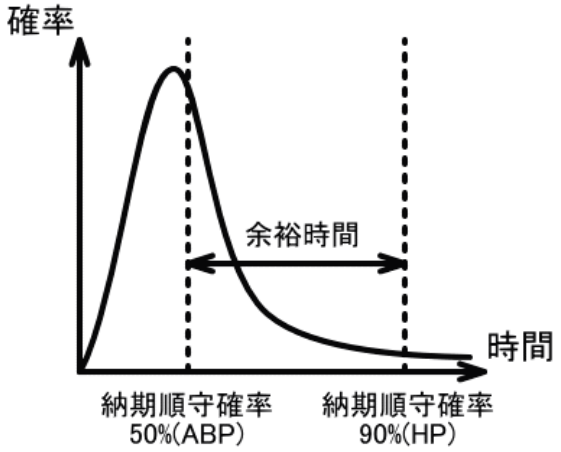

図 2　プロジェクト型業務における作業時間の確率分布

は，作業が 90%の確率で終了する作業時間を HP(Highly Possible)，50%の確率で終了する作業時間を ABP(Aggressive But Possible)と呼ぶ．作業時間の分布が図 2 のようなベータ分布に従うと仮定したとき，ABP の作業時間は HP の作業時間の 1/3 倍である．工程iにおける従来法による作業時間の見積もりを HP の作業時間としてd_i，ABP の作業時間をλ_iとすると，λ_iは以下の式で表される．

$$\lambda_i = d_i^{\otimes 1/3} \tag{3.1}$$

従来のスケジューリングでは，各工程の作業が時間内に終えることができるように，作業時間の見積もりに HP を用いていたと解釈できる．しかし，この見積もりには，作業着手の先延ばしといった無駄な余裕時間が含まれている可能性が高い．その原因の例を以下に挙げる[8],[9]．

- パーキンソンの法則

プロジェクトのある作業が予定より早く終了しても，余った時間を無駄に使い，予定通りの期日に終了させることである．これは，余った予算を会計年度の期末までに使い切ろうとすることとよく似ており，作業時間を何が何でも使い切ることである．

- 学生症候群

パーキンソンの法則とは逆に，納期ぎりぎりまで近づかないと作業を始めないことである．例えばレポート課題が出ると，多くの学生が最初のうちは作業せず，提出期限間際になってから作業を始める．このことから学生症候群という呼び名がついたとも言われる．所要時間に安全余裕を織り込んでいると，その余裕だけ作業開始が遅れる．場合によってはその安全余裕が納期遅れに結びつくこともある．

- マルチタスク

複数のプロジェクトを掛け持つことで，いわゆるエースと呼ばれるスキルの高い人に起きることである．マルチタスクを抱えると能率が下がる．例えば，前回どこまで作業したか把握するなど，アイドル時間が必要になる．マルチタスクを抱えた人は高負荷状態になり，いくら安全余裕を確保しても役に立たない場合が多いと言われている．

作業時間に安全余裕を加えても，先ほど述べた原因により作業が予定より遅れてしまうことが多々ある．このことから，クリティカルチェーン法では各工程の ABP により作業時間の設定を行う．

3.2.2 作業時間のシミュレーション手法

前項において，ABP の作業時間λ_iは式(3.1)のように表せると述べたが，このようにな

る作業はどのような確率分布をもつのか，裏づけの議論が不十分であった．具体的には，確率密度関数を$f(x)$とおくと，

$$\begin{cases} \int_0^{d_i} f(x)dx = 0.9 \\ \int_0^{\lambda_i} f(x)dx = 0.5 \\ \lambda_i = {d_i}^{\otimes 1/3} \end{cases} \tag{3.2}$$

となる分布が明らかにされていなかった．

ところで，ベータ分布の確率密度関数は，二つのパラメータを$\alpha, \beta > 0$とすると，以下のように表される[10]．

$$f(x) = \begin{cases} \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} : 0 < x < 1 \\ 0 \qquad\qquad : \text{その他} \end{cases} \tag{3.3}$$

これを$Be(\alpha,\beta)$と書くこともある．ここで，

$$B(\alpha,\beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx \tag{3.4}$$

であり，この関数はベータ関数といわれる．

白石ら[5]は，$\alpha = 3, \beta = 6$のとき，つまり$Be(3,6)$が式(3.2)を満たすことを，その分布に従う乱数を発生させることで実証した．そこで，本研究ではこれを用いてシミュレーションを行う手法を考える．

$Be(3,6)$に従う乱数をn個発生させ，i番目のものをp_iとする．このとき，工程iの実際にかかった作業時間μ_iを以下のように計算する．

$$\mu_i = {d_i}^{\otimes p_i} \tag{3.5}$$

3.3 リソース競合の解消

リソース競合とは，作業者や機械設備といったリソースが，同じ時間帯に複数の工程で使用される状況のことである．リソース競合の発生はプロジェクトの遅延を引き起こす．例えば，ある作業者が複数の工程を掛け持ちすると，前節で述べたマルチタスクを抱えることになり能率が下がり，プロジェクト遅延の原因になる．制約理論では，「リソースは一度に一つの工程の作業しかできない」という制約を考え，これを解消する手法を提案

している.

吉田[4]は，以下の手順でリソース競合を解消していた.

1. 同一時間帯に作業を行う工程の検出
2. 同一リソースである工程の検出
3. 上記いずれでも検出された工程においてリソース競合があると判定
4. リソース競合の解消

確かにこの手順によりリソース競合は解消されるが，リソース競合を検出するための計算手順が複雑である. さらに，リソース競合を解消しても，後続の工程で新たなリソース競合が発生する可能性があり，すべて解消できるまで何度も検出作業を行わなければならないことも問題点である. その煩雑さをなくすため，新たな手法を提案する.

3.3.1 基本例

リソース競合を解消する方法を提示する前に，まず，リソース競合が発生しないスケジューリングを考える. 例えば，以下のプロジェクト 1 ではリソース競合が発生しない.

プロジェクト 1
1. 工程数はnである.
2. 工程$i\,(1 \leq i \leq n)$のリソースはz_iだけである.

これは明らかにリソース競合が発生しない. なぜなら，2 つ以上の工程を掛け持ちするリソースがなく，どのリソースも同じ時間帯に複数の工程で使用される状況にはならないからである.

次に，以下のプロジェクト 2 ではリソース競合が発生しないことを説明する.

プロジェクト 2
1. 工程数はnであり，外部からの入力は工程1のみに，外部への出力は工程nのみにある.
2. 工程$i\,(1 \leq i \leq n)$のリソースは，すべてz_1である.
3. 工程$j\,(2 \leq j \leq n)$の先行工程に，すべて工程$(j-1)$がある.

このプロジェクトの条件 3.より，工程nから先行工程を遡っていくと工程1に辿り着くことができる. このような先行関係を持つことを，今後「鎖状に先行関係を持つ」と表現する. 図 3 は，プロジェクト 2 を図示したものである.

このプロジェクトは，スケジューリング問題における一機械モデル[2]に帰着することができる. 2.は機械が一つしかないこと，1.および 3.は，一つの機械がn個あるジョブを処理

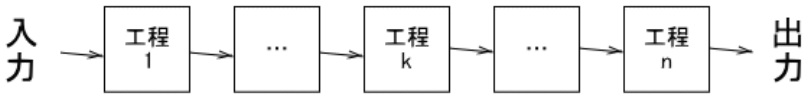

図 3　鎖状に先行関係を持つプロジェクト

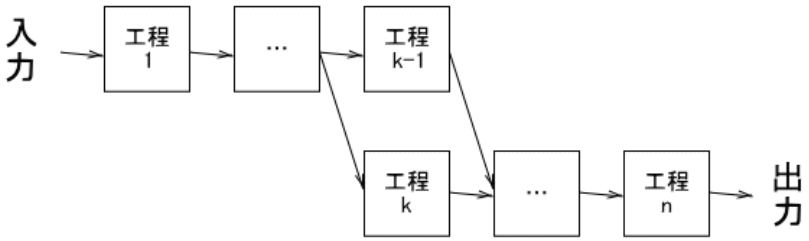

図 4　リソース競合が発生するプロジェクトの例

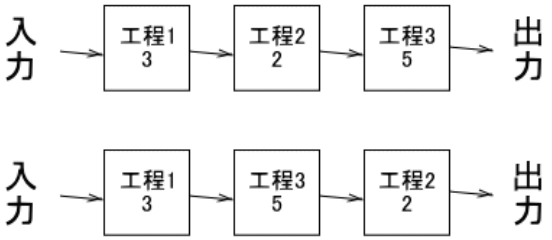

図 5　先行関係の入れ方の例

する順序の一つだと考えることができる. なお, 各ジョブの準備時間は 0, 各ジョブの納期は設定されてないものとする.

なお, このプロジェクトのうち, 条件 3.に一つでも当てはまらない工程が存在すると, リソース競合が発生する場合がある. 例えば工程k $(2 \leq k \leq n)$の先行工程に工程$(k-1)$がない場合, 図 4 のように表させる. このような場合, 工程$(k-1)$と工程kでリソース競合が発生する. 工程$(k-1)$と工程kは同一時間帯に作業する可能性があり, かつ工程$(k-1)$と工程kは同一リソースだからである.

プロジェクト 1 とプロジェクト 2 を合わせると,「リソース競合が発生しない必要十分条件は, 同一リソースの工程が鎖状に先行関係を持つことである」といえる. よって, この条件を満たすように先行関係を追加することでリソース競合を解消することができる.

ところで, この先行関係の追加方法は一通りではない. 例えば 1 入力 1 出力 3 工程のプロジェクトを一人で行うときを考える. 3 工程の作業順序は, $3! = 6$通り考えられるが, そのうち 2 通りを図 5 に示す. $\boldsymbol{d} = [3 \quad 2 \quad 5]^T$とするとき, いずれの場合でも所要時間は 10 と同じ値になるが, プロジェクトの規模が大きくなり先行関係が複雑化すると, 値が変わることも予想される. そこで, 全体の作業時間を短縮する最適な方法を考えることにする.

3.3.2　先行関係の追加

図 6 のようなプロジェクトを例とする. このプロジェクトの, リソース競合を解消したうえで実現できる最早出力時刻を求める手順を考える. まず, リソースを無視した場合を考える. 式(2.45)～(2.48)より, システムの構造を表現する行列$\boldsymbol{P}, \boldsymbol{F}_0, \boldsymbol{B}_0$および$\boldsymbol{C}_0$は以下のように求められる.

$$\boldsymbol{P} = \begin{bmatrix} 4 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 2 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 3 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 4 \end{bmatrix} \tag{3.6}$$

$$\boldsymbol{F}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & e & \varepsilon \end{bmatrix} \tag{3.7}$$

$$\boldsymbol{B}_0 = [e \quad \varepsilon \quad e \quad \varepsilon \quad \varepsilon]^T \tag{3.8}$$

$$\boldsymbol{C}_0 = [\varepsilon \quad e \quad \varepsilon \quad \varepsilon \quad e] \tag{3.9}$$

また, 式(2.49)～(2.54)より, 最早終了時刻, 最早出力時刻, 最遅開始時刻, 最遅入力時刻, トータルフロートは以下のように計算できる.

$$\boldsymbol{x}_E^+ = [4 \quad 7 \quad 2 \quad 5 \quad 9]^T \tag{3.10}$$

$$y_E = 9 \tag{3.11}$$

$$\boldsymbol{x}_L^- = [2 \quad 6 \quad 0 \quad 2 \quad 5]^T \tag{3.12}$$

$$u_L = 0 \tag{3.13}$$

$$\boldsymbol{\omega} = [2 \quad 2 \quad 0 \quad 0 \quad 0]^T \tag{3.14}$$

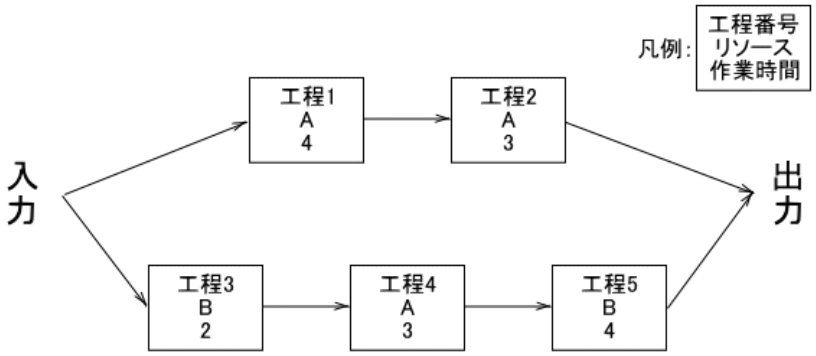

図 6　5 工程からなるプロジェクトの先行制約関係

式(2.55),(3.14)より，クリティカルパスは{3,4,5}であることがわかる．

ところが，実際にはリソース競合が発生するため，このプロジェクトは作業時間 9 で終わらせることができない．例えば，工程 1 と工程 3 を時刻 0 で同時に開始したときを考える．工程 3 は時刻 2 で終わるが，工程 1 は時刻 4 まで作業が終わらず，それより早く工程 4 を開始することができない．このため，工程 3 が終わったにもかかわらず工程 4 を開始できない状態になる．しかも工程 4 はクリティカルパスであるから，工程 4 が開始できない分プロジェクト全体にも遅れが発生する．その原因の一つとして，工程 4 が工程 1,2 と先行関係を持たないことが考えられる．

そこで，各リソースの工程が鎖状に先行関係を持つように先行関係を追加し，リソース競合を解消する．本研究では最早終了時刻の小さい工程から順に先行関係を追加することを考える．式(3.10)より，リソースが A の工程を最早終了時刻の小さい順に並べると，工程 1，工程 4，工程 2 となる．よって，工程 2 の先行工程に工程 4 を，工程 4 の先行工程に工程 1 を追加すればよい．

この例では，リソース B においてリソース競合することはないが，工程 5 の先行工程に工程 3 を追加しても問題はない．リソース競合が発生する工程に限り先行関係を追加する方法も考えられるが，この追加を許容することで，リソース競合の検出作業を省くことができる．

すべてのリソースについて先行関係を追加したことにより，リソース競合が解消された．解消後は図 7 のように表される．リソース競合解消のために追加した先行関係は，点線で示している．このプロジェクトの所要時間は 11 である．式(3.11)と比較すると 2 だけ所要時間が延びたが，これはリソース競合を解消したことに起因する．また，このときのクリティカルパスは{1,4,5}であり，リソース競合解消前のクリティカルパス{3,4,5}から変化したことがわかる．

リソース競合解消法を max-plus 代数で表現する．まず，リソースの数だけ以下のベクトルを設定する．

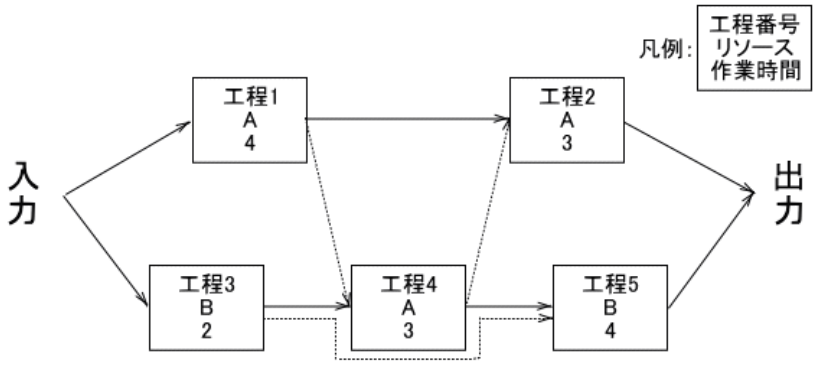

図 7　リソース競合解消後のプロジェクトの先行制約関係

$$[\boldsymbol{v}_m]_i = \begin{cases} e: \text{工程}i\text{がリソース}m\text{を持つ} \\ \varepsilon: \text{工程}i\text{がリソース}m\text{を持たない} \end{cases} \tag{3.15}$$

これを用いて，次のようなベクトルを計算する．

$$\boldsymbol{w}_j = \mathrm{diag}(\boldsymbol{v}_j)\boldsymbol{x}_E^+ \tag{3.16}$$

このベクトルにおいて，成分の値の小さい順に工程を結ぶように，先行関係を追加すればよい．ただし，成分がεの行は無視する．また，同じ値が出てきた場合は，工程番号の小さい方から順に工程を結ぶことにする．

この手法が有効であることを確認する．先の例において，$\boldsymbol{v}_A, \boldsymbol{w}_A, \boldsymbol{v}_B, \boldsymbol{w}_B$を計算すると，以下の通りに求められる．

$$\boldsymbol{v}_A = [e \quad e \quad \varepsilon \quad e \quad \varepsilon]^T \tag{3.17}$$
$$\boldsymbol{w}_A = [4 \quad 7 \quad \varepsilon \quad 5 \quad \varepsilon]^T \tag{3.18}$$
$$\boldsymbol{v}_B = [\varepsilon \quad \varepsilon \quad e \quad \varepsilon \quad e]^T \tag{3.19}$$
$$\boldsymbol{w}_B = [\varepsilon \quad \varepsilon \quad 2 \quad \varepsilon \quad 9]^T \tag{3.20}$$

式(3.18),(3.20)より，以下のように先行関係を追加すればよい．

- 工程 1→工程 4→工程 2 の順
- 工程 3→工程 5 の順

これらの先行関係を図 6 に追加すると図 7 のようになるため，この手法の有効性が確認できた．

先ほどの例ではすでにクリティカルパスおよびプロジェクト所要時間は算出されているが，念のため，先行関係を追加した後に max-plus 線形システムに当てはめることで成り立つかどうか確認する．行列$\boldsymbol{P}, \boldsymbol{B}_0, \boldsymbol{C}_0$は式(3.6),(3.8),(3.9)と同一である．しかし，先行関係を追加したため，行列$\boldsymbol{F}_0$を書き変える必要がある．そこで，新たに$\boldsymbol{F}_0{}'$とおき，式(2.46)に基づいて改めて計算すると，以下のように表される．

$$\boldsymbol{F}_0{}' = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & e & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & e & e & \varepsilon \end{bmatrix} \tag{3.21}$$

式(3.7)と式(3.21)を比較すると，(2,4),(4,1),(5,3)成分がεからeに置き換わっていることが

わかる. この$\boldsymbol{F}_0'$を$\boldsymbol{F}_0$の代わりに用いて, 最早終了時刻, 最早出力時刻, 最遅開始時刻, トータルフロートを再計算すると, 以下のようになる.

$$\boldsymbol{x}_E^+ = [4 \quad 10 \quad 2 \quad 7 \quad 11]^T \tag{3.22}$$

$$y_E = 11 \tag{3.23}$$

$$\boldsymbol{x}_L^- = [0 \quad 8 \quad 2 \quad 4 \quad 7]^T \tag{3.24}$$

$$\boldsymbol{\omega} = [0 \quad 1 \quad 2 \quad 0 \quad 0]^T \tag{3.25}$$

式(2.55),(3.25)より, クリティカルパスは{1,4,5}と求まる.

3.4 時間バッファの挿入

3.2 節で述べた通り, クリティカルチェーン法では各工程の作業時間を ABP で考える. ABP は作業が 50%の確率で完了する時間である. しかしこれは, 50%の確率で作業時間が ABP を超えることも意味する. つまり, 作業時間を ABP にすると, 50%の確率で遅れが発生してしまう. これによる納期遅れを防止するために, 時間バッファと呼ばれる余裕時間を挿入する.

従来法では余裕時間を含めて作業時間を見積もっているが, 3.2 節で述べたように, この余裕時間を無駄に浪費してしまう場合がある. さらに, すべての工程においてその見積もりを行っているため, 不要となった余裕時間を他工程の余裕時間に変更する点は考慮されていない. これに対しクリティカルチェーン法では, 各工程に含まれている余裕時間を除き, 時間バッファとして挿入することで, プロジェクト全体の遅れを防止する.

プロジェクトの遅延防止のために挿入する時間バッファは, プロジェクトバッファとフィーディングバッファの二種類がある. 前者はプロジェクト全体の遅れを吸収する時間バッファ, 後者はクリティカルパスへ遅れを伝搬させないための時間バッファである. 以降, この二種類の時間バッファの詳細を述べる.

3.4.1 プロジェクトバッファ

プロジェクトバッファ(Project Buffer)とは, プロジェクト全体の遅れを吸収するための時間バッファである. プロジェクトの遅れは, クリティカルパス上の工程の遅れにより発生するからである.

プロジェクトバッファの大きさは, クリティカルパス上の各工程の余裕時間の合計をそのまま割り当ててもよいが, これより小さくてもよい. 例えば, 余裕時間の合計を経験則に基づき半分にした時間を当てることがある[8]. 本研究では, それをさらに半分にした値をプロジェクトバッファの大きさとする. HP から ABP に変更したことで減らせる作業時間をすべての余裕時間とみなした場合, プロジェクトバッファの大きさは以下のように表せる.

$$(\mathrm{HP}-\mathrm{ABP})^{\otimes 1/4}=\mathrm{ABP}^{\otimes 1/2} \tag{3.26}$$

プロジェクトバッファは，外部出力の直前に挿入する．外部出力が複数ある場合は，その数だけ挿入する．

3.4.2 フィーディングバッファ

フィーディングバッファ（Feeding Buffer）とは，クリティカルパスへ遅れを伝搬させないための時間バッファであり，主に非クリティカルパス上の工程で発生した遅れを吸収することを目的に挿入する．このことから，フィーディングバッファは，非クリティカルパス上の工程からクリティカルパスへの合流点に設置する．フィーディングバッファの大きさは，プロジェクトバッファと同様に，非クリティカルパス上の工程の余裕時間の合計の 1/4 に設定する．

工程の先行関係によっては，フィーディングバッファを複数配置する場合もある．そのときは，非クリティカルパス上の工程の余裕時間を，複数のバッファで重複して考慮するのを防ぐ必要がある．

3.4.3 バッファの挿入方法

本項では，max-plus 代数を用いた，プロジェクトバッファとフィーディングバッファの挿入方法を述べる[11]．まず，クリティカルパスではない工程の集合βを定義する．

$$\beta=\{i|[\boldsymbol{\omega}(k)]_i>0\} \tag{3.27}$$

式(2.55)と式(3.27)より，以下の等式が成り立つ．

$$\alpha\cap\beta=\varphi \tag{3.28}$$
$$\alpha\cup\beta=\{1,2,\ldots.n\} \tag{3.29}$$

次に，以下のようにベクトル$\boldsymbol{a},\boldsymbol{b}$および行列$\boldsymbol{P}_a,\boldsymbol{P}_b$を定義する．

$$[\boldsymbol{a}]_i=\begin{cases}e:i\in\alpha\\ \varepsilon:i\in\beta\end{cases} \tag{3.30}$$

$$[\boldsymbol{b}]_i=\begin{cases}e:i\in\beta\\ \varepsilon:i\in\alpha\end{cases} \tag{3.31}$$

$$\boldsymbol{P}_a=\mathrm{diag}(\boldsymbol{a})\boldsymbol{P} \tag{3.32}$$

$$\boldsymbol{P}_b = \mathrm{diag}(\boldsymbol{b})\boldsymbol{P} \tag{3.33}$$

そして，n次元ベクトル$\boldsymbol{g}$を$\boldsymbol{g} = [e \quad e \quad \ldots \quad e]^T$と定め，以下のようにベクトル$\boldsymbol{r}_p, \boldsymbol{r}_f$を設定する．なお，式中に登場する，$(\boldsymbol{F}_0\boldsymbol{P}_a)^*, (\boldsymbol{F}_0\boldsymbol{P}_b)^*$の計算方法は，式(2.50)と同様である．

$$\boldsymbol{r}_p = (\boldsymbol{P}_a(\boldsymbol{F}_0\boldsymbol{P}_a)^*\boldsymbol{g})^{\otimes 1/2} \tag{3.34}$$
$$\boldsymbol{r}_f = (\boldsymbol{P}_b(\boldsymbol{F}_0\boldsymbol{P}_b)^*\boldsymbol{g})^{\otimes 1/2} \tag{3.35}$$

最後に，以下の行列$\boldsymbol{C}_c, \boldsymbol{F}_c$を計算する．

$$\boldsymbol{C}_c = \boldsymbol{C}_0(\mathrm{diag}(\boldsymbol{r}_p) \oplus \mathrm{diag}(\boldsymbol{r}_f)) \tag{3.36}$$
$$\boldsymbol{F}_c = \boldsymbol{F}_0 \oplus \mathrm{diag}(\boldsymbol{a})\boldsymbol{F}_0\mathrm{diag}(\boldsymbol{r}_f) \tag{3.37}$$

これらの行列を用いて，プロジェクトバッファとフィーディングバッファを挿入する．$[\boldsymbol{C}_c]_{ij} > e$のとき，大きさ$[\boldsymbol{C}_c]_{ij}$のプロジェクトバッファを出力$i$と工程$j$の間に挿入する．また，$[\boldsymbol{F}_c]_{ij} > e$のとき，大きさ$[\boldsymbol{F}_c]_{ij}$のフィーディングバッファを工程$j$と工程$i$の間に挿入する．

3.5 フィーバーチャート

これまで見積もり段階に設定することを中心に述べてきたが，本節ではプロジェクト実行段階に用いるフィーバーチャートについて述べる．

3.5.1 フィーバーチャートの概要

フィーバーチャートとは，横軸にプロジェクトの進捗度，縦軸に消費したプロジェクトバッファの割合を割り当てたグラフである．クリティカルチェーン法では，プロジェクトマネージャがこのフィーバーチャートを確認することで，プロジェクトが計画通りに進んでいるか，など進捗状況を管理する．図 8 に，フィーバーチャートの例を示す．このように，フィーバーチャートは三つの領域に分かれている．三つの領域はそれぞれ安全領域，注意領域，危険領域と呼ばれる．プロジェクトマネージャは，プロジェクトバッファの消費状況がフィーバーチャートのどの領域にあるかを確認することで，状況に応じた行動をとることができる．場合分けすると，以下のように行動をとればよい．

- 安全領域にある場合

プロジェクトの遅れを回避するための特別な行動はとらなくてよい．そのままプロジェクトの進捗を監視し続ける．

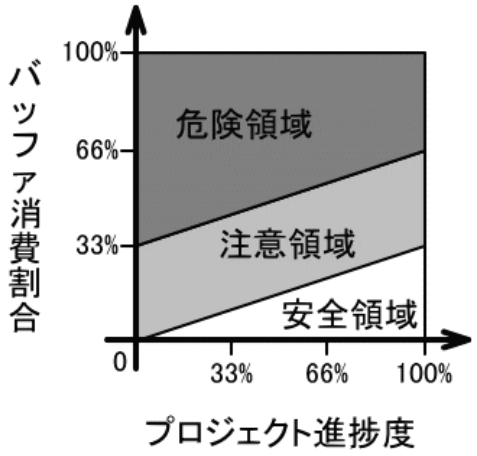

図 8　フィーバーチャート

- 注意領域にある場合

将来的にプロジェクトバッファの許容量を超えてしまう可能性が高いため，遅れの原因を調査し，さらなる遅れが発生しないための対策を立てる．

- 危険領域にある場合

遅延回避策を速やかに実行に移し，プロジェクトバッファの許容量が超えないようにする．

なお，フィーバーチャートにおける，安全領域と注意領域の境界線，および注意領域と危険領域の境界線を設ける指標は，まだ統一されていない．ここでは吉田[4]に倣い，以下のように境界線を設定する．

- 安全領域と注意領域の境界線：(0%,0%)と(100%,33%)を結ぶ直線
- 注意領域と危険領域の境界線：(0%,33%)と(100%,66%)を結ぶ直線

3.5.2　バッファ消費量の計算

本項では，フィーバーチャートへのプロットに必要なものを導出する方法を述べる．簡略のため，ここでは 1 出力の場合の数式を書くことにする．まず，フィーバーチャートの横軸に割り当てられる，プロジェクトの進捗度を求める．これは，

$$\frac{\text{プロジェクト開始から工程}i\text{の作業終了までの作業見積もり時間の総和}}{\text{クリティカルパス上の工程の作業見積もり時間の総和}} \tag{3.38}$$

という式で求めることができる. よって, 工程iの進捗度t_iは以下のように計算する.

$$t_i = \frac{\bigotimes_{k=1,k\in\alpha}^{n} \lambda_k}{\bigotimes_{k\in\alpha} \lambda_k} \tag{3.39}$$

次に, フィーバーチャートの縦軸に割り当てられる, プロジェクトバッファの消費割合を導出する. これは,

$$\frac{\text{工程}i\text{の作業が終了した時点でのプロジェクトバッファの累積消費量}}{\text{プロジェクトバッファの大きさ}} \tag{3.40}$$

という式で求めることができる. プロジェクトバッファの大きさは, 式(3.36)により導出できる. これをzとおく.

ここで, 工程iにおけるプロジェクトバッファの消費量を計算する. プロジェクトバッファの消費について, 以下の三通りに場合分けして考える.

- クリティカルパス上の工程を作業し, 見積もり時間λ_i以内に終了した場合

プロジェクトバッファは消費されない. このとき, プロジェクトバッファの消費量はeとする.

- クリティカルパス上の工程を作業し, 見積もり時間λ_iを超えて終了した場合

λ_iを超えた分だけプロジェクトバッファを消費することで, プロジェクトの遅れを吸収する. 超過した作業時間は$\lambda_i \odot \mu_i$と求められる. これをプロジェクトバッファの消費量とする.

- 非クリティカルパス上の工程を作業した場合

プロジェクトバッファの消費は考えない. 見積もり時間λ_iを超えて終了した場合, その遅れはフィーディングバッファで吸収されるものと考える.

よって, プロジェクトバッファの消費量δ_iは以下のように計算できる.

$$\delta_i = \begin{cases} (\lambda_i \odot \mu_i) \oplus e : i \in \alpha \\ e \qquad\quad : i \in \beta \end{cases} \tag{3.41}$$

これを用いると, 工程iが終了した時点でのプロジェクトバッファの累積消費量σ_iは以下のように求められる.

$$\sigma_i = \bigotimes_{k=1}^{i} \delta_k \tag{3.42}$$

以上より，工程iが終了した時点でのプロジェクトバッファの消費割合r_iは，以下のように求めることができる．

$$r_i = \frac{\sigma_i}{z} \tag{3.43}$$

ただし，非クリティカルパス上の工程については，プロジェクトバッファの消費を考えないため，フィーバーチャートへのプロットの対象から外す．

第4章　数値実験

本章では，第 3 章で述べた手法を用い，簡単なプロジェクトを例にとって作業のシミュレーションを行う．具体的には，簡単なプロジェクトに対し，作業時間の見積もり，リソース競合の解消，時間バッファの挿入を行った後，実際の作業時間を$Be(3,6)$に従う乱数を用いることで作業のシミュレーションを行う．

図 9 のようなプロジェクトを考える．なお，図中の作業時間は HP であるとする．式(2.45)～(2.48)より, システムの構造を表現するベクトル$\boldsymbol{d}$, 行列$\boldsymbol{P}, \boldsymbol{F}_0, \boldsymbol{B}_0$および$\boldsymbol{C}_0$は以下のように求められる.

$$\boldsymbol{d} = [3 \quad 9 \quad 9 \quad 3 \quad 3 \quad 6 \quad 9 \quad 6 \quad 9]^{\boldsymbol{T}} \tag{4.1}$$

$$\boldsymbol{P} = \begin{bmatrix} 3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 9 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 9 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 6 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 9 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 6 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 9 \end{bmatrix} \tag{4.2}$$

$$\boldsymbol{F}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & e & e & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & e & e & \varepsilon \end{bmatrix} \tag{4.3}$$

$$\boldsymbol{B}_0 = [e \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon]^{\boldsymbol{T}} \tag{4.4}$$

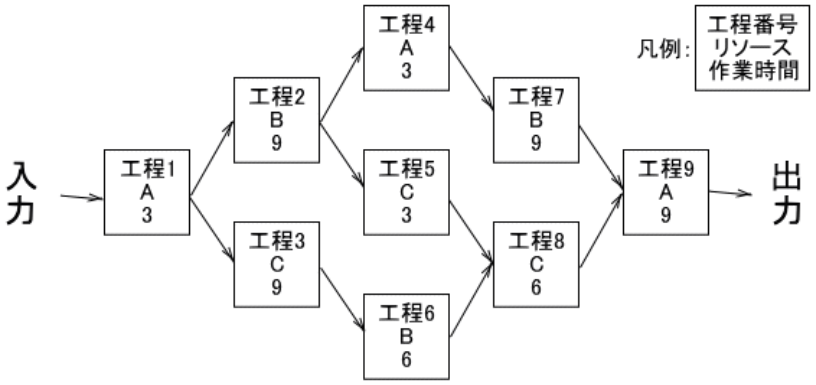

図 9　数値実験に用いるプロジェクトの先行制約関係

$$\boldsymbol{C}_0 = [\varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad e] \tag{4.5}$$

4.1 作業時間の見積もり

まず, 作業時間の見積もりを HP から ABP に変更する. 式(3.1)より, ベクトル$\boldsymbol{\lambda}$は以下のように求まる.

$$\boldsymbol{\lambda} = [1 \quad 3 \quad 3 \quad 1 \quad 1 \quad 2 \quad 3 \quad 2 \quad 3]^{\boldsymbol{T}} \tag{4.6}$$

見積もりを変更したため, $\boldsymbol{P}$も以下のように変更する. 変更後の行列を$\boldsymbol{P'}$とおく.

$$\boldsymbol{P'} = \begin{bmatrix} 1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 2 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 3 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 2 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & 3 \end{bmatrix} \tag{4.7}$$

4.2 リソース競合の解消

次に, リソース競合を解消する. そのために, この状態で作業した場合の最早終了時刻$\boldsymbol{x}_E^+$を求める.

$$\boldsymbol{x}_E^+ = [1 \quad 4 \quad 4 \quad 5 \quad 5 \quad 6 \quad 8 \quad 8 \quad 11]^T \tag{4.8}$$

そして, 式(3.15)～(3.16)を用いてベクトル$\boldsymbol{w_A}, \boldsymbol{w_B}, \boldsymbol{w_C}$を計算する.

$$\boldsymbol{w_A} = [1 \quad \varepsilon \quad \varepsilon \quad 5 \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad 11]^T \tag{4.9}$$
$$\boldsymbol{w_B} = [\varepsilon \quad 4 \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad 6 \quad 8 \quad \varepsilon \quad \varepsilon]^{\boldsymbol{T}} \tag{4.10}$$
$$\boldsymbol{w_C} = [\varepsilon \quad \varepsilon \quad 4 \quad \varepsilon \quad 5 \quad \varepsilon \quad \varepsilon \quad 8 \quad \varepsilon]^{\boldsymbol{T}} \tag{4.11}$$

式(4.9)～(4.11)より, 以下のように先行関係を追加すればよい.

- 工程 1→工程 4→工程 9 の順
- 工程 2→工程 6→工程 7 の順
- 工程 3→工程 5→工程 8 の順

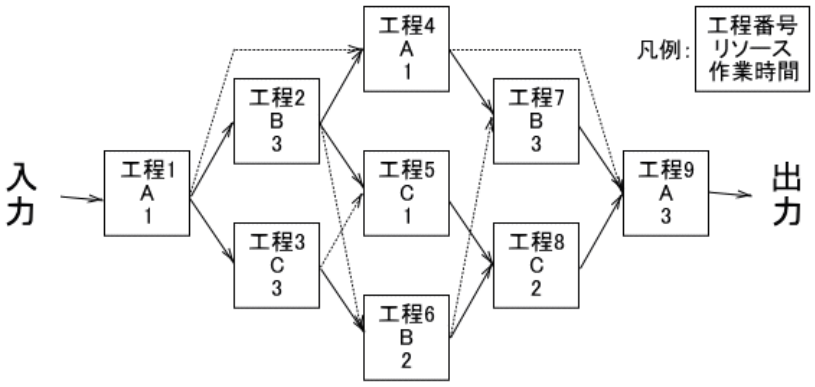

図 10　リソース競合解消後のプロジェクトの先行制約関係

ただし，工程 5 から工程 8 への先行関係は初期条件に含まれているため，これに関しては実際には何もしない．先行関係を追加すると，図 10 のように表される．先行関係を追加したため，$\boldsymbol{F}_0$も以下のように変更する．変更後の行列を$\boldsymbol{F}_0{}'$とおく．

$$\boldsymbol{F}_0{}' = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & e & \varepsilon & e & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & e & e & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & e & \varepsilon & \varepsilon & e & e & \varepsilon \end{bmatrix} \tag{4.12}$$

4.3 クリティカルパスの導出

図 10 のプロジェクトのクリティカルパスを求める．リソース競合を解消したため，最早終了時刻を計算し直す．その結果，以下の通りになる．

$$\boldsymbol{x}_E^+ = [1 \quad 4 \quad 4 \quad 5 \quad 5 \quad 6 \quad 9 \quad 8 \quad 12]^T \tag{4.13}$$

最早出力時刻，最遅開始時刻，トータルフロートは，式(2.51),(2.52),(2.54)より以下のように求まる．

$$y_E = 12 \tag{4.14}$$

$$\boldsymbol{x}_L^- = [0 \quad 1 \quad 1 \quad 5 \quad 6 \quad 4 \quad 6 \quad 7 \quad 9]^T \tag{4.15}$$

$$\boldsymbol{\omega} = [0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 0 \quad 0 \quad 1 \quad 0]^T \tag{4.16}$$

式(2.55)と式(3.27)より，集合α, βは以下のように求まる．

$$\alpha = \{1,2,3,6,7,9\} \tag{4.17}$$
$$\beta = \{4,5,8\} \tag{4.18}$$

4.4 時間バッファの挿入

図 10 のプロジェクトに対し，プロジェクトバッファとフィーディングバッファを挿入する．式(3.30)～(3.35)より，ベクトル$\boldsymbol{r}_p, \boldsymbol{r}_f$は以下のように求まる．

$$\boldsymbol{r}_p = [0.5 \quad 2 \quad 2 \quad \varepsilon \quad \varepsilon \quad 3 \quad 4.5 \quad \varepsilon \quad 6]^T \tag{4.19}$$
$$\boldsymbol{r}_f = [\varepsilon \quad \varepsilon \quad \varepsilon \quad 0.5 \quad 0.5 \quad \varepsilon \quad \varepsilon \quad 1.5 \quad \varepsilon]^T \tag{4.20}$$

次に，行列$\boldsymbol{C}_c$を計算すると以下のように求まる．

$$\boldsymbol{C}_c = [\varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad \varepsilon \quad 6] \tag{4.21}$$

よって，大きさ 6 のプロジェクトバッファを工程 9 の直後に挿入すればよい．さらに，行列$\boldsymbol{F}_c$を計算すると以下のように求まる．

$$\boldsymbol{F}_c = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ e & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 0.5 & \varepsilon & e & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & e & e & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 0.5 & \varepsilon & \varepsilon & e & 1.5 & \varepsilon \end{bmatrix} \tag{4.22}$$

よって，フィーディングバッファは以下のように挿入すればよい．

- 工程 4 と工程 7 の間，大きさ 0.5
- 工程 4 と工程 9 の間，大きさ 0.5
- 工程 8 と工程 9 の間，大きさ 1.5

時間バッファ挿入後のプロジェクトは，図 11 のように表される．なお，太い四角で強調した工程は，クリティカルパス上にあることを示す．

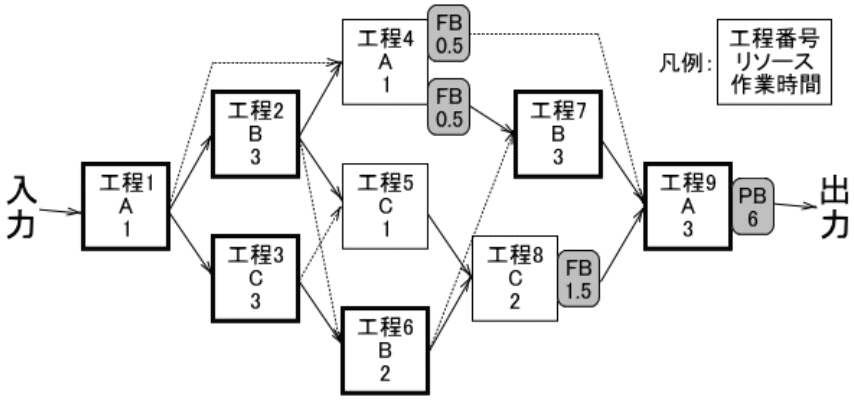

図 11　時間バッファ挿入後のプロジェクトの先行制約関係

4.5 作業のシミュレーション

図 11 のプロジェクトにおける作業のシミュレーションを行う. シミュレーションにより, 前節までの見積もりの妥当性を確認し, さらにフィーバーチャートを導出する.

このシミュレーションは, 以下のような条件を設定して行った.

- 入力時刻は 0 とする.
- 作業時間は式(3.5)により, 9 工程すべて算出し, 小数第 2 位以下を切り上げる.
- 非クリティカルパス上の工程は, 作業時間が見積もり時間を超過したとしても, 遅れはすべてフィーディングバッファで吸収できるものとする. フィーディングバッファで吸収しきれなかった遅れがあったとしても無視する.
- 作業時間が見積もり時間より少なかった場合, その差分を別工程の作業時間やプロジェクトバッファに振り替えることはできないものとする.
- プロジェクトバッファを全て消費した場合でも, 作業は打ち切らず最後まで行うものとする.
- 出力時刻は, 最早出力時刻の見積もり 12 と, プロジェクトバッファの消費量の和とする.

また, 最早出力時刻の見積もり 12 と, プロジェクトバッファ 6 の合計値 18 をこのプロジェクトの納期とする.

このシミュレーションを 100 回行った. まず, シミュレーションの一例を紹介する. 式(3.5)より, 実際の作業時間$\boldsymbol{\mu}$は以下のように求まった.

$$\boldsymbol{\mu} = [0.7 \quad 3.0 \quad 4.7 \quad 0.5 \quad 1.3 \quad 1.2 \quad 2.9 \quad 1.3 \quad 4.1]^T \tag{4.23}$$

式(3.41),(3.42)より, プロジェクトバッファの消費量$\boldsymbol{\delta}$および累積消費量$\boldsymbol{\sigma}$は, 以下のように求まった.

$$\boldsymbol{\delta} = [0.0 \quad 0.0 \quad 1.7 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 1.1]^T \tag{4.24}$$
$$\boldsymbol{\sigma} = [0.0 \quad 0.0 \quad 1.7 \quad 1.7 \quad 1.7 \quad 1.7 \quad 1.7 \quad 1.7 \quad 2.8]^T \tag{4.25}$$

$\sigma_9 = 2.8$より, このときの出力時刻は 14.8 となった. よって, 納期までにプロジェクトを終了することができた.

この例のフィーバーチャートを導出する. 式(3.39), (3.43)より, プロジェクトの進捗度$\boldsymbol{t}$およびプロジェクトバッファの消費割合$\boldsymbol{r}$は, 以下のように求まる. 各成分は小数第 3 位を四捨五入している.

$$\boldsymbol{t} = [0.07 \quad 0.27 \quad 0.47 \quad 0.47 \quad 0.47 \quad 0.60 \quad 0.80 \quad 0.80 \quad 1.00]^T \tag{4.26}$$
$$\boldsymbol{r} = [0.00 \quad 0.00 \quad 0.28 \quad 0.28 \quad 0.28 \quad 0.28 \quad 0.28 \quad 0.28 \quad 0.47]^T \tag{4.27}$$

これらの指標を用いてフィーバーチャートを導出すると, 図 12 のように表される. なお, プロットされた点内にある数字は, 工程番号を示す.

この例において, 工程 1,2,6,7 ではプロジェクトバッファを消費せず, ABP による作業見積もり時間λ_iまでに作業を終えることができた. このため, 工程 1,2 ではプロジェクトバッファ消費割合は 0 であり, いずれの点も安全領域にある. また, 工程 6,7 ではプロジェクトバッファ消費割合が工程 3 と同じであり, これら 3 点は横軸と平行に並んでいる. 工程 3 でのプロジェクトバッファ消費により, これら 3 点は注意領域にある. 実際に作業する際, プロジェクトマネージャは工程 3 が終了した時点でのフィーバーチャートを確認した後, 工程 3 で発生した遅れの原因を調査し, さらなる遅れが発生しないための対策を立てるべきである. 最終工程である工程 9 でもプロジェクトバッファが消費されたが, 危険領域に入ることなく注意領域でプロジェクトを終了している. しかし, この工程でも遅れが発生したため, その原因を調査すべきである. このように, フィーバーチャートはプロジェ

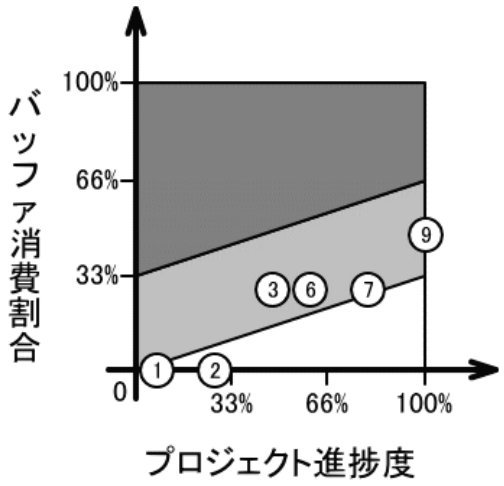

図 12　フィーバーチャート導出例 1

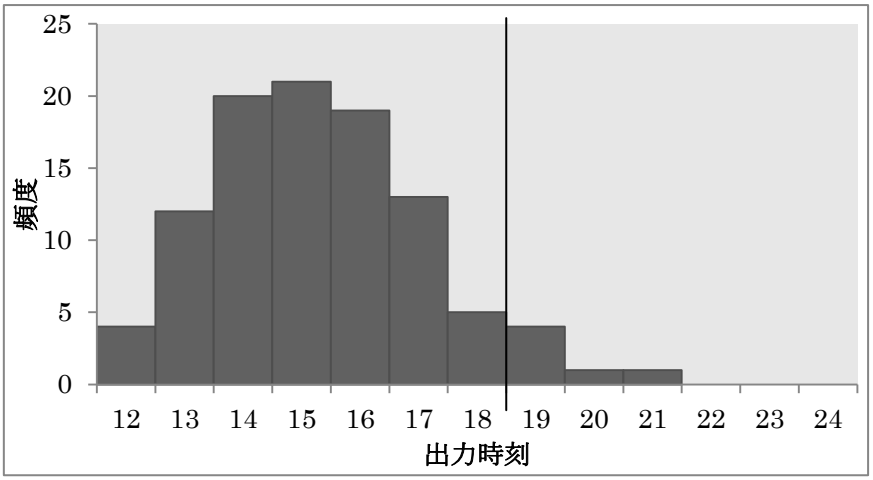

図 13　出力時刻のヒストグラム

クトの進捗状況および残余裕時間の割合を確認できる有効な手法である.

この例のように, プロジェクトを計 100 回シミュレーションした. その結果のうち, 出力時刻をヒストグラムにすると, 図 13 のようになった. シミュレーションの結果, 最大の出力時刻は 20.8 であり, 納期とした 18 を上回る結果が 6 回出た. しかし, 出力時刻の平均は 14.9 であるほか, 13.1～16.0 に集中しており, 100 回中 57 回の出力時刻が 15 以下であるため, 作業時間やプロジェクトバッファの見積もりは妥当といえる.

以下, 特徴のある結果が出たケースについて詳細を述べ, 考察する. まず, 出力時刻が 12 の場合である. このとき, プロジェクトバッファが全く消費されてないことになる. この出力時刻は今回のシミュレーションで考えられる最小のケースであり, これは図 13 の通り, 100 回中 4 回記録された. そのときの作業時間とフィーバーチャートの一例を示す. 実際の作業時間$\boldsymbol{\mu}$は, 以下のように計算された.

$$\boldsymbol{\mu} = [0.9 \quad 1.3 \quad 2.1 \quad 1.6 \quad 1.2 \quad 1.7 \quad 2.4 \quad 2.5 \quad 2.8]^T \tag{4.28}$$

そして, この場合のプロジェクトバッファの消費割合$\boldsymbol{r}$を求めると, 以下の通りになる.

$$\boldsymbol{r} = [0.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00]^T \tag{4.29}$$

これと式(4.26)を用いてフィーバーチャートを作成すると, 図 14 のように表される. 先述した通り, これはプロジェクトバッファが全く消費されないケースであるが, そのことが図 14 からも読みとれる. 当然, プロジェクトが進んでも, フィーバーチャート上では常に安全領域に点がプロットされる. しかし, ABP による作業見積もり時間と実際の作業時間に極端な差はなく, 工程によってはこの二つが近い値を取っている. 例えば工程 1 における差は 0.1, 工程 6 では 0.3, 工程 9 では 0.2 となっている. よって, ABP による作業見積もり時間が大きいとはいえず, むしろ妥当といえる.

ところで, ABP とは作業達成確率が 50%の作業時間である. このことから, 6 工程ある

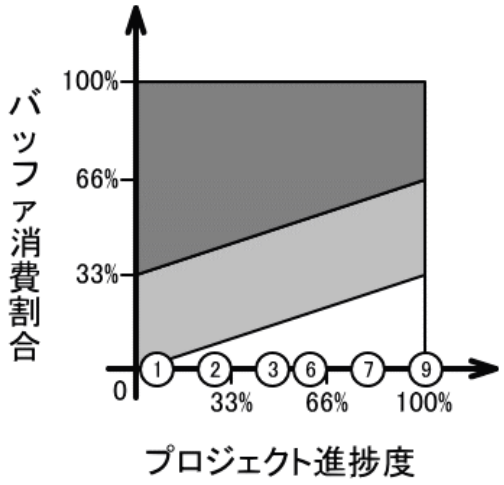

図 14　フィーバーチャート導出例 2

プロジェクトの全工程を ABP までに作業を終了する確率は，$(1/2)^6 = 1/64$である．つまり，今回のように 100 回シミュレーションした場合，このような結果は理論上でも 1,2 回出ることがいえる．

さらに，出力時刻が 18 を超えた時の詳細を述べ，考察する．ここでは出力時刻が 18.8 のときを例に挙げ，実際の作業時間とフィーバーチャートを示す．実際の作業時間$\boldsymbol{\mu}$と，プロジェクトバッファの消費割合$\boldsymbol{r}$は，以下のように求まった．

$$\boldsymbol{\mu} = [0.8 \quad 3.0 \quad 5.8 \quad 0.8 \quad 1.0 \quad 3.3 \quad 3.6 \quad 2.2 \quad 5.1]^T \tag{4.30}$$

$$\boldsymbol{r} = [0.00 \quad 0.00 \quad 0.47 \quad 0.47 \quad 0.47 \quad 0.68 \quad 0.78 \quad 0.78 \quad 1.13]^T \tag{4.31}$$

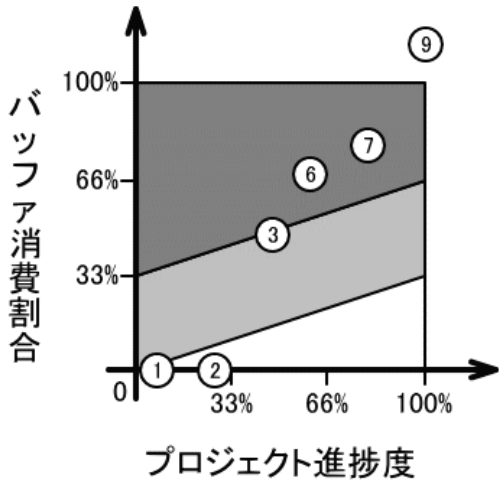

図 15　フィーバーチャート導出例 3

このときのフィーバーチャートは，図 15 のように表される．工程 1,2 についてはプロジェクトバッファの消費がなく，点は安全領域にある．しかし，工程 3 ではプロジェクトバッファを多く消費したため，点は注意領域にプロットされ，危険領域にも迫った．さらに，その後の工程 6,7 でもプロジェクトバッファを消費し，危険領域に点がプロットされた．最終工程ではプロジェクトバッファの累積消費量が見積もりを超え，出力時刻は納期時刻より後になった．

危険領域に点がプロットされた場合，遅延回避策を速やかに実行する必要がある．しかし，このシミュレーションでは作業時間を最初に一括で設定したため，遅延回避策は実行不可能であった．実際には遅延回避策が実行され，出力時刻が 18 から前になるよう努力するであろう．

工程 3 では，作業見積もり時間を 2.8 も超過したため，これだけでプロジェクトバッファの約半分が消費された．これが納期遅れになった大きな原因とも考えることができる．このシミュレーションでは，工程の作業終了後に点のプロット作業を行ったが，例えば一日ごとに消費状況をプロットすることでプロジェクトバッファを管理することで，遅延を短くすることができるかもしれない．

第5章　結論

本論文では，max-plus 線形システムを適用したクリティカルチェーン法における，残余裕時間の可視化について検討した．そのために，先行研究で定式化された手法を用いるだけではなく，シミュレーションを行うための式を設定した．具体的には，実際の作業時間の設定と，プロジェクトバッファの消費量の計算方法が挙げられる．前者は，HP の作業時間に，$Be(3,6)$に従って発生させた乱数を掛けたものを実際の作業時間と設定した．後者は，クリティカルパス上の工程を作業した時間が ABP の作業時間を超えなかった場合と超えた場合，そして非クリティカルパス上の工程を作業した場合の三つに場合分けし，工程ごとのプロジェクトバッファ消費量を計算しやすくした．これらを用いることで作業のシミュレーションを可能にした．さらに，プロジェクトバッファの累積消費量の割合を縦軸，プロジェクトの進捗度を横軸にとったフィーバーチャートに，工程の進捗に合わせて点をプロットすることで，プロジェクトバッファの消費状況がどのようになっているのか一目でわかるようにした．これにより残余裕時間を可視化することができ，フィーバーチャートはシミュレーションにおいても機能することが確認された．

また，本研究では，リソース競合の解消法を一から検討し直した．これは，先行研究におけるリソース競合解消法の手続きが複雑であったことに起因する．先行研究では，リソース競合を検出するための行列が数種類設定されており，しかもリソース競合が解消されるまで計算を続ける必要があった．一方本研究では，同一リソースの工程を鎖状に先行関係を持つように先行関係を追加するという手法を提案した．この手法では，各リソースに対し二種類のベクトルを計算するだけでよく，計算量が大幅に削減できた．また，リソース競合が発生する場合においてもシミュレーションを行うことができた．

今後の課題として，フィーバーチャートにおける安全領域と注意領域の境界線，および注意領域と危険領域の境界線を設ける最適の指標を求めることが挙げられる．本研究では，作業時間を一斉に割り当てシミュレーションを行ったが，実際にはフィーバーチャートを確認することで後続工程の作業時間に影響が出ると考えられる．その場合，フィーバーチャート内の二つの境界線の位置も重要になる．どのように境界線を引けば，最も遅延を発生しにくくさせるか，あるいは最も納期を短くすることができるのか明らかになれば，より実用性が高まると考えられる．

参考文献

[1] 加藤豊, 小沢正典:「OR の基礎 AHP から最適化まで」, 実教出版株式会社 (1998)

[2] 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)

[3] 笠原統徳, 高橋弘毅, 五島洋行, “クリティカル・チェーン法の max-plus 線形システムへの応用”, 日本オペレーションズ・リサーチ学会 2008 年秋季研究発表会アブストラクト集, pp.38-39 (2008)

[4] 吉田翔太郎, “作業時間の不確実性を考慮した max-plus 線形システム”, 長岡技術科学大学　工学研究科　修士論文 (2012)

[5] 白石高庸, 増田士朗, “作業時間が確率的に変動する場合の Max-plus 線形システムを用いたクリティカルチェーン法”, 第 51 回離散事象システム研究会講演論文集, pp.73-76 (2012)

[6] 雨宮孝, 竹安数博, 増田士朗:「新しい経営・経済数学」, 株式会社中央経済社 (2004)

[7] 中野明:「エリヤフ・ゴールドラットの「制約理論」がわかる本」, 株式会社秀和システム (2006)

[8] 中嶋秀隆, 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)

[9] 村上悟, 井川伸治:「最速で開発し最短で納めるプロジェクト・マネジメント」, 株式会社中経出版 (2002)

[10] 白旗慎吾:「Minerva ベイシック・エコノミクス 統計学」, ミネルヴァ書房 (2008)

[11] Goto, H., Nguyen Thi Ngoc Truc, Takahashi, H. : Simple Representation of the Critical Chain Project Management Framework in a Max-Plus Linear Form, 第 52 回離散事象システム研究会講演論文集, pp.5-8 (2013)