CCPM法の枠組みにおける資源競合の解消方法：局所探索法と遺伝的アルゴリズムの活用

古賀, 裕紀 / GOTO, Hiroyuki / KOGA, Hiroki / 五島, 洋行

(出版者 / Publisher)
法政大学情報科学部・理工学部・生命科学部
(雑誌名 / Journal or Publication Title)
法政大学理系学部研究集報 / 法政大学理系学部研究集報
(巻 / Volume)
50
(開始ページ / Start Page)
7
(終了ページ / End Page)
12
(発行年 / Year)
2014-04
(URL)
https://doi.org/10.15002/00010556

CCPM法の枠組みにおける資源競合の解消方法 ～局所探索法と遺伝的アルゴリズムの活用～

Resolution of Resource Conflicts in the CCPM Framework ~Utilization of a Local Search Method or Genetic algorithm~

古賀 裕紀　五島 洋行　千葉 英史
Hiroki KOGA　Hiroyuki GOTO　Eishi CHIBA

法政大学大学院理工学研究科システム工学専攻修士課程

We propose approximate methods for resolving resource conflicts in the Critical Chain Project Management (CCPM) method. The CCPM method consists of five processes. There are effective approaches for four of the five processes. However, for the remaining unresolved process that resolves 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 genetic algorithm. Methods based on the earliest and latest start times are used. Through numerical experimentations, we found that the proposed methods are practical if the number of outputs is one.

***Key Words** : CCPM method, resource conflict, local search, genetic algorithm, max-plus algebra*

1．はじめに

本研究では，CCPM法の手順の一つである資源競合の解消を行うための近似解法を提案する．CCPM法とは，プロジェクト工程の実行時間の不確実性を考慮し，プロジェクトにかかる時間の短縮と遅延防止の両立を目的としたプロジェクト管理手法である．その運用手順は，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2章で説明する．文献1)では，手順4，5の二つをmax-plus線形方程式を用いることで，従来のCCPM法の計算に比べて簡素な線形方程式に定式化することに成功した．文献2)では，時間バッファを考慮しない資源競合の解消問題の定式化に成功した．しかし，時間バッファを考慮した資源競合の解消の具体的な解法は提案されていない．そこで本研究では，三つの初歩的な近似解法と，その近似解法を局所探索や遺伝アルゴリズムを用いて改善する手法を提案する．また計算実験を行い，提案する近似解法が実用的な近似解法かどうか検証する．

2．背景知識

（1）Max-plus代数

Max-plus代数[3]とは，max演算と+演算を基本演算とする代数系であり，生産システムやプロジェクト管理，交通システムなどの離散事象システムのモデリングに利用することができる．

実数全体を$\boldsymbol{R}$とし，$\boldsymbol{R}_{\max}=\boldsymbol{R}\cup\{-\infty\}$，$x,y\in\boldsymbol{R}_{\max}$と定義する．max-plus代数系では，加算$\oplus$と乗算$\otimes$を

$$x\oplus y=\max(x,y), \tag{1}$$

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

と定義する．演算子の優先順位は，通常の代数系を同様に$\otimes$は$\oplus$よりも高いとする．また，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$\varepsilon(=-\infty)$，$e(=0)$と表記し，

$$x\oplus\varepsilon=\varepsilon\oplus x=\varepsilon, \tag{3}$$

$$x\otimes e=e\otimes x=x \tag{4}$$

が成立する．また，

$$x\otimes\varepsilon=\varepsilon\otimes x=\varepsilon \tag{5}$$

が成り立つ．さらに，演算子\を次式のように定義する．

$$x\setminus y=-x+y. \tag{6}$$

（2）CCPM法の運用手順

まず，使用する文字や記号を定義する．

- n：工程数
- q：外部入力数
- p：外部出力数
- $\boldsymbol{F_0}\in\boldsymbol{R}_{\max}^{n\times n}$：$(j,i)$に枝がある場合は$[\boldsymbol{F_0}]_{ij}=e$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$i$に接続されているならば$[\boldsymbol{C_0}]_{ij}=e$，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($\boldsymbol{d}$)

● $u \in R^q$：各外部入力の開始時刻のベクトル

F_0, B_0, 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．

次に，CCPM法を運用するための五つの手順を説明する．最初の三つの手順は，タスクの洗い出し，余裕時間の没収，資源競合の解消である．

タスクの洗い出しでは，各工程の先行関係F_0と実行時間を設定し，余裕時間の没収では，各工程の実行時間を50%の確率で工程処理に間に合うように短縮する．その代わりに，時間バッファの挿入の際に実行時間の遅れを考慮に入れる．

資源競合の解消では，複数の工程が一つの資源を同時に占有する状況を除去する．本研究はこの部分を主に取り扱い，詳細は3章で説明する．

次に，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．最早終了時刻x_Eと最遅終了時刻$(x_L + d)$は次のようにして計算できる．

まず，最早終了時刻は，

$$x_E = A \otimes B_0 \otimes u \tag{7}$$

で計算できる．ただし$A = P \otimes (F_0 \otimes P)^*$であり，$X^*$は任意の二工程間の最長経路に等しい．また，プロジェクト全体の最早終了時刻は，

$$y_E = C_0 \otimes x_E \tag{8}$$

で求められる．これらより，最遅開始時刻は，

$$x_L = (C_0 \otimes A) \backslash y_E \tag{9}$$

となる．$m = (x_L + d) - x_E$とすると，ネック工程とは$[m]_i = 0$となる工程である．そこで，ネック工程の集合をα，ネックでない工程の集合をβと表記すると，

$$\alpha = \{i \mid [m]_i = 0\}：ネック工程 \tag{10}$$

$$\beta = \{i \mid [m]_i > 0\}：ネックでない工程 \tag{11}$$

となる．集合αに属する工程の連なりを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．

時間バッファの挿入の際，$[a]_i = \{e: \text{if } i \in \alpha, \varepsilon: \text{if } i \in \beta\}$，$[b]_i = \{e: \text{if } i \in \beta, \varepsilon: \text{if } i \in \alpha\}$として，$P_a = \text{diag}(a) \otimes P$，$P_b = \text{diag}(b) \otimes P$というベクトルと行列を作成する．

最後に，時間バッファを挿入する．時間バッファとは，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ように挿入する仮想的な工程のことである．時間バッファとして次の二種類を挿入する．

● 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．Fig. 1はFB挿入の例である．挿入するバッファの長さを表す行列をr_fとすると，r_fは下式を用いて計算できる．

$$r_f = [P_b \otimes (F_0 \otimes P_b)^* \otimes g]/2 \tag{12}$$

$$g = [e\ e\ \cdots e]^T \in R^n_{\max} \tag{13}$$

また，FBの追加後は隣接行列F_0をF_cに修正する必要がある．修正後は

$$F_c = F_0 \oplus \text{diag}(a) \otimes F_0 \otimes \text{diag}(r_f) \tag{14}$$

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．Fig. 2はPB挿入の例である．挿入するバッファの長さを含んだ行列をr_pとすると，r_pは次式のように計算できる．

$$r_p = [P_a \otimes (F_0 \otimes P_a)^* \otimes g]/2 \tag{15}$$

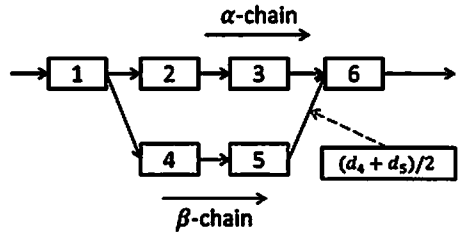

Fig. 1: Example of an insertion of a FB.

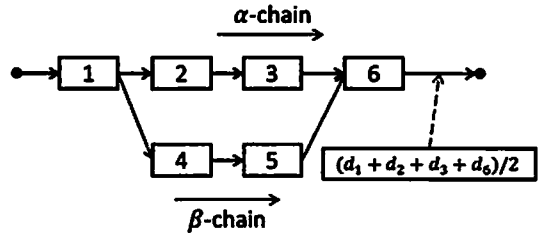

Fig. 2: Example of an insertion of a PB.

また，バッファの追加のために，出力行列を下式のように修正する．

$$C_c = C_0 \otimes [\text{diag}(r_p) \oplus \text{diag}(r_f)] \tag{16}$$

以上のようにして求めたF_cとC_cを用いて式(8)の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，$A = P \otimes (F_c \otimes P)^*$として$x_{cE} = A \otimes B_0 \otimes u$を求めたのち，最早終了時刻

$$y_{cE} = C_c \otimes x_{cE} \tag{17}$$

を求める．y_{cE}の要素の最大値をvとすると，vを本研究における評価値とする．

3．資源競合の解消問題

CCPM法の入力には，各工程に処理時間，工程間の先行関係，各工程を処理する資源の三つがある．資源は機械とも読み替えられる．また，一つの資源は同時に二つ以上の工程を処理できない．資源競合の解消問題とは，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方法を考える問題である．

（1）競合発生の例

まず，使用する文字や記号を定義する．

● l：資源数

● s：各資源の処理工程数の最大値

● $r \in R^n$：工程iを資源jで処理するとき$[r]_i = j$

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

次に，この問題に必要な入力と出力を以下に示す．

入力：

● 工程数n

● 各工程の先行関係F_0

● 各工程の実行時間d

● 各工程に必要な資源を指定するベクトルr

出力：

- 資源競合が発生しない各資源の工程処理順番 S

具体例として Fig. 3 のような問題を考える．資源 1 に工程 1, 4, 5．資源 2 に工程 2, 3 を割り当て，各工程の実行時間を工程 1 から順に 3, 4, 5, 2, 1 だとすると，入力は以下のようになる．

$n=5$,

$$F_0=\begin{bmatrix}\varepsilon&\varepsilon&\varepsilon&\varepsilon&\varepsilon\\ e&\varepsilon&\varepsilon&\varepsilon&\varepsilon\\ \varepsilon&\varepsilon&\varepsilon&\varepsilon&\varepsilon\\ \varepsilon&\varepsilon&e&\varepsilon&\varepsilon\\ \varepsilon&e&\varepsilon&e&\varepsilon\end{bmatrix},\ d=\begin{bmatrix}3\\4\\5\\2\\1\end{bmatrix},\ r=\begin{bmatrix}1\\2\\2\\1\\1\end{bmatrix}$$

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，両者には先行関係がないため，同時刻に実行が可能である．このような場合に資源競合が発生する．

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．

資源競合を解消したプロジェクトの例が Fig. 4 である．資源 1 が工程 1，4，5．資源 2 が工程 3，2 の順に工程を処理する場合，Fig. 4 のプロジェクトの点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．Fig. 4 のように，新たに追加した先行関係は次の隣接行列F_0で表現でき，各資源の工程処理順番 S も次のように表現できる．

$$F_0=\begin{bmatrix}\varepsilon&\varepsilon&\varepsilon&\varepsilon&\varepsilon\\ e&\varepsilon&e&\varepsilon&\varepsilon\\ \varepsilon&\varepsilon&\varepsilon&\varepsilon&\varepsilon\\ e&\varepsilon&e&\varepsilon&\varepsilon\\ \varepsilon&e&\varepsilon&e&\varepsilon\end{bmatrix},\ S=\begin{bmatrix}1&4&5\\3&2&0\end{bmatrix}$$

また，FB と PB を挿入した解のガントチャートは Fig. 5 のようになる．

このようにして，資源ごとの作業の順番が決まる．この問題は NP 困難であると予想され，近似解法を考える必要がある．

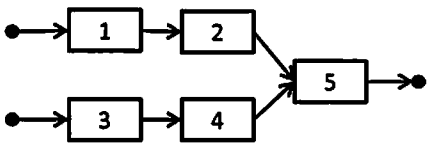

Fig. 3: Project before resolving resource conflicts.

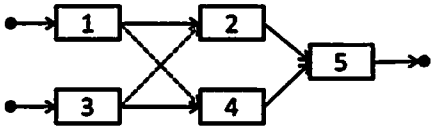

Fig. 4: Project after the resource conflicts have been resolved.

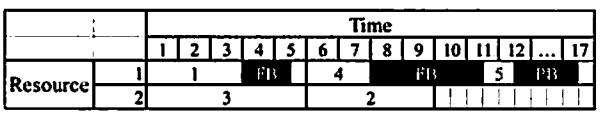

Fig. 5: Gantt chart of the solution.

（2）初歩的な近似解法

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B_0
- 外部出力行列C_0
- 各工程の先行関係F_0
- 各工程の実行時間d
- 各工程に必要な資源を指定するベクトルr

出力：

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

三つの近似解法は以下のとおりである．

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(10)を利用して

$$s_E=x_E-d \tag{18}$$

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s_Eを小さい順にソートし，その順番を各資源の工程処理の順番 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x_Lは式(9)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通りである．

1. 最早開始時刻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 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．

$$x_M=(s_E+x_L)/2 \tag{19}$$

アルゴリズムは前の二つと同様である．

（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．

本研究では，近傍 $NB_i(S)$を次のように定義する．

$NB_i(S)$ = {S' | S'は各資源の工程処理順番で，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 $NB_i(S)$を用いて，近傍 $NB(S)$を次のように定義する．

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

そして，この近傍 $NB(S)$を用いた二つの局所探索法を提案する．

- swap-best: 近傍 $NB(S)$を利用するアルゴリズム．まず，$NB_1(S)$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，$NB_2(S)$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，$NB(S)$で近似解が改善されたのなら，$NB_1(S)$からもう一度探索する．そうでなければ終了．
- swap-first: $NB_1(S)$から $NB_l(S)$まで順に探索し，最初に現在の近似解より良い評価値の解を見つけることができたら，その解に更新する．そして，$NB_1(S)$から探索をやり直す．$NB(S)$に改善可能な解がなくなったら終了．

さらにもう一つの近傍 $NB_{ij}(S)$を次のように定義する．

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

$$NB'(S) = \cup_{1 \le i \le l, 1 \le j \le l, i \ne j} NB_{ij}(S).$$

そして，この近傍 $NB'(S)$ を用いた二つの局所探索法を提案する．

- 2-swap-best: swap-best と似ており，$NB'(S)$を探索して現在の近似解 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．
- 2-swap-first: swap-first と似ており，$NB'(S)$を探索する中で，最初に見つけた更新可能な解に近似解 S を更新したのち，$NB'(S)$を探索し直す．更新可能な解が $NB'(S)$に存在しない場合は終了．

（4）遺伝アルゴリズム(Genetic Algorithm)による解法

本研究では，（2）の解法で求めた近似解の改善法として，遺伝アルゴリズムの考え方を利用した手法も提案する．以後，遺伝アルゴリズムのことはGAと呼ぶ．

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．このアルゴリズムでは，本章の（1）の入力に加えて以下の二つの入力を加える．

- 突然変異の確率 m_{ut}
- 終了条件 i

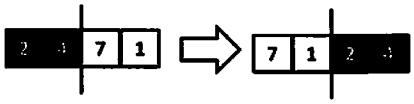

Fig. 6: Example of changing the execution order.

Fig. 7: Example of changing the execution order by mutation.

アルゴリズムを以下に示す．

アルゴリズム：

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，その中で最も良い評価値の解を暫定の近似解とする．
2. 暫定近似解の各資源の処理順番を，Fig. 6のようにランダムに区切り，できた二つのブロックを入れ替える．
3. 入れ替えの結果閉路ができた場合，資源の処理順を元に戻す．
4. 各資源に対して 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5へ，そうでないときは6へ進む．
5. ランダムで同じ資源内の二つの工程を選んで処理順番を Fig. 7 のように入れ替える．閉路ができた場合は元に戻す．
6. 2〜5で作った工程の処理順で，CCPM法の解 v を求める．
7. 暫定近似解よりも，2〜5による変化後の解の方が評価値 v が良い場合，変化後の解を暫定近似解とする．
8. 解の改善が 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2に戻る．

4．計算実験の結果

本章では3章で提案したアルゴリズムでの計算実験の結果を示す．計算実験の環境は以下のとおりである．

Machine: DELL Optiplex 390
CPU: Intel Core i5-2400 3.10GHz
OS: Microsoft Windows7 Professional 64bit
Memory: 4GB
Programming language: Octave3.6.4

（1）GAによる解法の適用例

Fig. 8のようなプロジェクトの例題を初歩的な近似解法三つを用いて解く．この例題は文献4)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．

$d = [5\ 10\ 15\ 10\ 20\ 15\ 10\ 10\ 5\ 15\ 10\ 15\ 5\ 20\ 5]^T$

$r = [4\ 5\ 3\ 1\ 4\ 1\ 4\ 2\ 1\ 2\ 3\ 1\ 4\ 2\ 3]^T$

例題を解いた結果，評価値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．最適解に一番近い最遅開始時刻法の近似解でも，最適化に比べて約31%大きいので良い解とはいえない．しかし，Fig. 10, Fig. 11の近似解のガントチャートと最適解のガントチャートを比較してみると，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いうことがわかる．そこで，局所探索や遺伝アルゴリズムで解を改善する．そのことによって，Fig. 8の最適解が120であるのに対して，m_{ut}=0.1，i=1000とした時のGAとswap-bestでは120という最適解を求めることができた．

（2）1出力のサンプルによる計算実験

サンプルデータとして10，20，30，40，50工程を100個ずつ用意し，CCPM法によって求めた v を解として，3．

章で紹介した近似解法を利用して計算実験を行った．入力する行列やグラフ構造の設定は以下のとおりである．

資源数が10工程の時は3，15工程の時は5，20，30，40，50工程の時は7とする．

- 各工程を処理する資源$[r]_i$は[1, l]の一様乱数．
- 工程の処理時間$[d]_i$は[5, 20]の一様乱数．
- 10〜20工程のサンプルのみ，各資源の処理工程数の上限は5．
- プロジェクト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Fig. 9に示す．
- 外部出力は一ヶ所のみ．

近似解の平均値はTable 1に示す．Table 1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．

Table 1より工程数が50の場合，"Three methods"の評価値は537.5であり，2-swap-firstを利用してこの評価値を改善すると488.61となり，約10%改善されている．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わかる．iが500で40工程の場合は，突然変異率が0.1よりも0.2の方が解の平均が悪いが，大きな違いはなく，他の工程数の場合では0.2の場合の方が良い解が求まっているため，突然変異率が高い方が少し良い解になると言えるであろう．また，iが500よりも1000の方が平均解は良いが，50工程までの範囲では大きな違いはない．GAによって三つの近似解法による近似解は，三つの初歩的な近似解法で求めた解と比べて1%以上改善されている．近似解は局所探索によって改善した方が，GAによる改善解より良い解が得られる．

平均計算時間をTable 2に示す．swap-bestの計算時間はswap-firstと大きな違いはなく，かつ近似解はswap-bestのほうが良いため，swap-best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．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，0.05と0.1よりも0.2の場合の方がより良い解を求めることができ，計算時間も大きく変わらないので，0.05や0.1よりも0.2の方が良いとわかる．また，iを500から1000に増やすと，計算時間は約二倍になる．

Fig. 12は各アルゴリズムの計算時間を対数グラフにしたものである．ただし，GAの計算時間は試行回数iのとき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．50工程の場合，2-swap-firstはGAの計算時間と大きな差がなくなっており，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思われる．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．

近似比はTable 3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．Table 3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0.05%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．

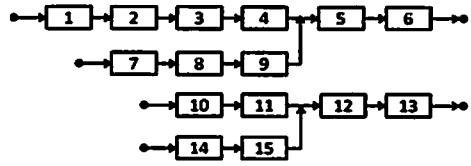

Fig. 8: Large example with fifteen tasks.

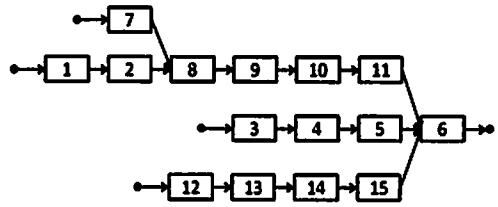

Fig. 9: Example of a graph for computational experiments.

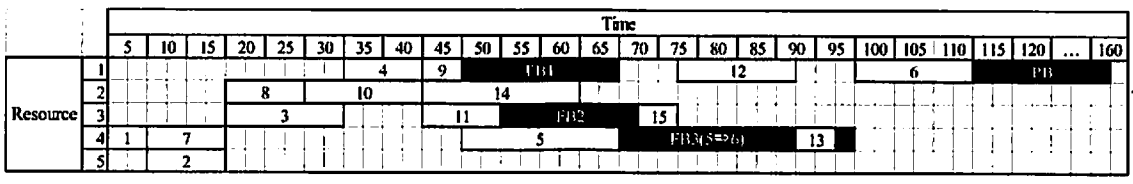

Fig. 10: Gantt chart of the solution solved by the latest start time method.

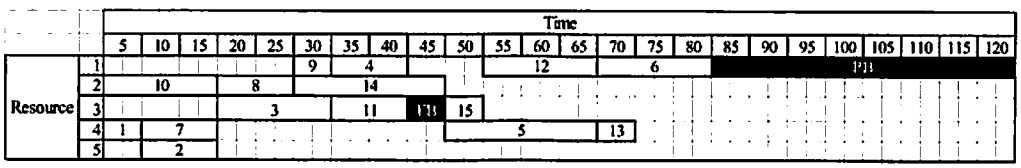

Fig. 11: Gantt chart of the optimal solution.

Table 1: Average values of the solutions.

Type of the method		Number of Tasks				
i	m_{ut}	10	20	30	40	50
Three methods		156.17	255.48	344.53	438.49	537.82
swap-best		155.34	253.39	330.09	409.43	499.19
swap-first		155.47	253.31	330.08	410.41	501.68
2-swap-best		155.33	252.70	327.47	401.58	488.85
2-swap-first		155.48	252.77	328.36	400.59	488.61
500	0.05	155.55	253.21	339.81	432.42	532.30
500	0.1	155.21	253.01	339.07	431.51	532.74
500	0.2	155.17	252.68	337.36	433.94	531.65
1,000	0.05	155.20	252.94	337.32	433.79	532.14
1,000	0.1	155.17	252.68	337.41	432.08	530.73
1,000	0.2	155.17	252.54	335.83	431.20	530.72

Table 2: Average computation time in second.

Type of the method		Number of Tasks				
i	m_{ut}	10	20	30	40	50
Three methods		0.2	0.5	1.2	2.2	3.4
swap-best		0.3	1.9	14.3	57.5	123.9
swap-first		0.3	1.8	11.4	52.1	103.8
2-swap-best		0.5	8.1	106.7	698.6	2,059.5
2-swap-first		0.5	7.1	71.3	460.3	1,258.5
500	0.05	24.1	96.8	214.6	390.1	628.4
500	0.1	23.6	100.7	219.5	375.8	641.5
500	0.2	22.4	107.8	238.4	383.1	624.6
1,000	0.05	54.7	201.3	468.1	757.6	1,240.7
1,000	0.1	55.2	216.3	507.0	766.4	1,226.2
1,000	0.2	47.0	193.7	505.0	730.9	1,232.6

Table 3: Approximation ratios.

Type of the method		Number of Tasks		
i	m_{ut}	10	15	20
Three methods		1.0064	1.0108	1.0126
swap-best		1.0011	1.0033	1.0055
swap-first		1.0019	1.0033	1.0040
2-swap-best		1.0010	1.0004	1.0016
2-swap-first		1.0020	1.0002	1.0019
500	0.05	1.0024	1.0044	1.0036
500	0.1	1.0002	1.0012	1.0029
500	0.2	1.0000	1.0011	1.0015
1,000	0.05	1.0002	1.0027	1.0026
1,000	0.1	1.0000	1.0010	1.0015
1,000	0.2	1.0000	1.0000	1.0010

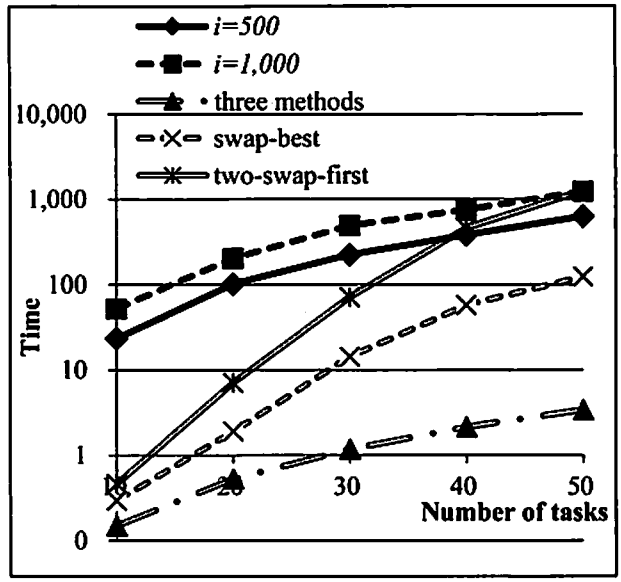

Fig. 12: Average computation times.

5．おわりに

本研究では，CCPM 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を提案した．提案した手法は，三つの初歩的な近似解法と，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最適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．その結果，サンプルとして用意した1出力のプロジェクトを入力とした問題を解いた場合には，最適解に近い近似解を得ることができた．また，20 工程以下の場合には GA の方が良い解が得られ，局所探索の方が GA に比べて 30 工程以上の場合には良い解が得られた．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加した．したがって，扱うプロジェクトが 50 工程より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であろう．

今後の課題としては，本研究で提案した局所探索以外の探索法の提案と，整数計画問題への帰着などがある．

参考文献

1) 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.

2) 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.

3) F. Baccelli, G. Cohen, G. J. Olsder, J.-P. Quadrat, Synchronization and Linearity: An Algebra for Discrete Event Systems, John Wiley & Sons, 1993.

4) L. P. Leach: Critical Chain Project Management, Second Edition, Section 6.3, Artech House, New York, p.133, 2004.