日本オペレーションズ・リサーチ学会
2005年春季研究発表会

拡張型資源制約付プロジェクトスケジューリング問題に対するヒューリスティックな解法

申請中　東京農工大学　*草部　博輝　KUSAKABE Hiroaki
01606760　東京農工大学　宮代　隆平　MIYASHIRO Ryuhei
01401940　東京農工大学　中森　眞理雄　NAKAMORI Mario

1. はじめに

多くのスケジューリング問題を枠組の中でとらえられる汎用的なモデルとして，資源制約付プロジェクトスケジューリング問題 (*Resource Constrained Project Scheduling Problem* : *RCPSP*) が知られている[1]．これは，1つのプロジェクトにおいて，各アクティビティ間に先行順序が課せられており，アクティビティの実行には有限個の資源が必要になるという制約条件のもとで，プロジェクト期間を最も短くするように，各アクティビティの処理開始時刻を決定する問題である．また，プロジェクト期間中の使用可能資源数の変化や，アクティビティ処理中の要求資源数の変化を取り込んだモデルとして，*RCPSP*/τ がある[1]．しかしこれは，先行順序を課せられたアクティビティ間での処理開始のタイミングといった条件を取り込んではいない．

本稿では，*RCPSP*/τ に上記の条件を取り込んだ問題を，拡張型資源制約付プロジェクトスケジューリング問題 (*Extended Resource Constrained Project Scheduling Problem* : *ERCPSP*) として提案する．また，解法としてタブーサーチを実装し，その性能を評価する．

2. *ERCPSP*

ERCPSP は，一般的な *RCPSP*/τ に対して以下の時間的な条件を追加したものである．

先行順序を課されたアクティビティ間において，先行アクティビティの処理完了後，所定時間内に後続アクティビティを開始しなければならない猶予期間，もしくは所定時間内は処理を開始してはならない待機時間が課せられる（それぞれを，**猶予制約**，**待機制約**と呼ぶことにする）．

以上の条件の下で，メイクスパンの最小化を目的とする問題を，*ERCPSP* と定義する．*ERCPSP* は以下の 0-1 整数計画問題として定式化できる．

$$\min \quad z=\max_j\left(\sum_{t=0}^{T_{\max}-1} t x_{jt}+p_j\right) \qquad \cdots①$$

subject to

$$\sum_{t=0}^{T_{\max}-1} t x_{st} \le \sum_{t=0}^{T_{\max}-1} t x_{jt}+p_j+w_{js} \quad j=0,\ldots,n-1,\ s\in S_j \qquad \cdots②$$

$$\sum_{t=0}^{T_{\max}-1} t x_{st} \ge \sum_{t=0}^{T_{\max}-1} t x_{jt}+p_j+a_{js} \quad j=0,\ldots,n-1,\ s\in S_j \qquad \cdots③$$

$$\sum_{j=0}^{n-1}\sum_{u=0}^{\min(t,p_j-1)} d_{jru}x_{j(t-u)} \le l_{rt} \quad t=0,\ldots,T_{\max}-1,\ r=0,\ldots,m-1 \qquad \cdots④$$

$$\sum_{t=0}^{T_{\max}-1} x_{jt}=1 \quad j=0,\ldots,n-1 \qquad \cdots⑤$$

$$x_{jt}\in\{0,1\} \quad j=0,\ldots,n-1,\ t=0,\ldots,T_{\max}-1 \qquad \cdots⑥$$

n：アクティビティ数
m：再生型資源種類数
T_{max}：プロジェクト期間
p_j：アクティビティ j の処理時間．
S_j：アクティビティ j の後続アクティビティの集合．
w_{js}：アクティビティ j 完了後，$s\in S_j$ の処理開始の締切時間．s は j 完了後，w_{js} 単位時間以内に処理が開始される．
a_{js}：アクティビティ j 完了後，$s\in S_j$ の処理開始の待機時間．s は j 完了後，w_{js} 単位時間以内は処理を開始できない．
d_{rju}：アクティビティ j が，処理開始後 u 単位時間目に要求する再生型資源 r の量．
l_{rt}：時刻 t における再生型資源 r の使用可能量．
x_{jt}：アクティビティ j の処理開始時刻が t であれば 1，そうでなければ 0．

この問題は，クラス NP-困難に属し，アクティビティの数が大きくなった場合，現実的な時間内に厳密解を得ることは難しい．

3. アルゴリズム

3. 1 初期解

初期解の構築には，*Single Pass Method* にタブーリストを併用した手法を用いる．

Single Pass Method を適用した結果，ディスパッチされなかったアクティビティがある場合，その原因は先行アクティビティにあると仮定し，その番号と開始時刻をタブーリストに記録する．先行アクティビティが複数ある場合は，その中から次の条件にしたがって1つだけ選択する．

1) 猶予制約が課せられている先行アクティビティを選択する．存在しない場合は，待機制約が課せられたものを選択する．
2) 猶予制約が課せられたアクティビティを選択した場合，その中から
（処理開始時刻）＋（処理時間）＋（猶予時間）
の値が最も大きいアクティビティを1つ選択する．待機制約が課せられたアクティビティを選択した場合は，その中から処理開始時刻が最も早いアクティビティを選択する．

タブーリストへの登録後，登録されたディスパッチを行わないように *Single Pass Method* の適用を繰り返す．終了条件は，実行可能解の取得，もしくは繰返し回数の規定値への到達である．この手法を，便宜上ルール 1 と呼ぶことにする．ルール 1 は，構築されたスケジュールと，ディスパッチされたアクティビティの順序を出力する．

ルール 1 で用いる優先度は，まずアクティビティ j ($j=0,1,\ldots,n-1$)が処理中に要求する全資源量，すなわち

$$\sum_{r=0}^{m-1}\sum_{t=0}^{p_j-1} d_{rju}$$

の昇順に優先度を与える．この優先度で例外的に1回のみ

アクティビティのディスパッチを試み，それ以降の繰返しでは，猶予制約を課せられた先行，後続アクティビティの総数の降順で優先度を与える．

3. 2 解の改善

ルール 1 から得られたスケジュールの改善を行う．ルール 1 は，1 つのディスパッチ順序に対して 1 つのスケジュールを出力するので，ディスパッチの順序に変更を加えたものを，得られたスケジュールの近傍と定義する．初期解の近傍から得られるディスパッチの順序を，ルール 1 に変更を加えたルール 2 に入力する．ルール 1 には，アクティビティの優先度を入力するが，ルール 2 にはディスパッチの順序を入力する．入力どおりの順序にディスパッチを行うことが不可能であった場合，直ちに実行不可能解を出力して終了する．

3. 3 近傍

本稿では，挿入近傍と 2swap 近傍の 2 つを提案する．

挿入近傍を用いる場合は，次のような操作を行う．まず，①アクティビティを 1 つ選択し，そのディスパッチ順序を変更する．②変更前と変更後の順位の間に含まれる他のアクティビティの順位は，適宜シフトして変更する．図 1 に例を示す．

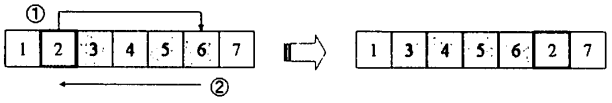

図 1 挿入近傍

2swap 近傍を用いる場合は，次のような操作を行う．まずアクティビティを 2 つ選択し，それらのディスパッチ順位を入れ替える．図 2 に例を示す．

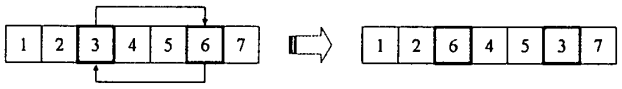

図 2 2swap近傍

3. 4 タブーサーチ

挿入近傍，2swap 近傍それぞれを用いて，タブーサーチ (*Tabu Search* : *TS*) を実装した．探索を行う際に用いた近傍によって，タブーリストの属性は異なるものを用いた．挿入近傍による *TS* に用いるタブーリストの属性は，アクティビティの番号と変更先のディスパッチ順位である．また，2swap 近傍による *TS* に用いたタブーリストの属性は，ディスパッチ順位を入れ替える 2 つのアクティビティの番号である．

また，この *TS* では，近傍内にある解のうち，目的関数値が最も小さいものに移動する．最小値をとる解が複数存在する場合は，その中で最も早く発見された解に移動する．移動した解を初期解として，終了条件を満たすまで繰り返し探索を実行する．アルゴリズムの終了条件は，次のように判定する．近傍内の探索を行う際，終了条件の判定のため，改善解に移動できなかった回数を数える．この回数が累計で 100 回に達すると探索を終了し，探索中に発見された最もよいスケジュールを出力して終了する．

4 数値実験

PSPLIB に公開されている *RCPSP* のベンチマーク問題 200 問（アクティビティ数：10～40，資源種類数：1～6）から *ERCPSP* 用にランダムに生成した問題を用いて数値実験を行った．ILOG CPLEX と，本稿で提案する *TS* から出力された計算結果を，CPU 時間および目的関数値の 2 点で比較を行う．なお，ILOG CPLEX の計算時間は，1 問につき最大 50000 秒の制限で打ち切っている．そのため，CPU 時間が制限を超えている場合，必ずしも最適解を出力したとは限らない．なお，200 問中，実行可能解が存在しないことが保証されたのは，11 問であった．数値実験の結果を，表 1 に示す．実験を行ったアルゴリズムは，ルール 1，挿入近傍による *TS* (ins. *TS*)，2swap 近傍による *TS* (2swap *TS*)，ins. *TS* の実行後，2swap *TS* を実行したもの (ins.→2swap *TS*)，2swap *TS* の実行後，ins. *TS* を実行したもの (2swap *TS*→ins.) の 5 種である．

表 1 解の精度による比較

	実行可能	最適	gap①[%]	gap②[%]
ルール 1	187	51	11.50	11.78
ins. *TS*	188	110	3.69	4.32
2swap *TS*	188	123	2.52	3.07
ins.→2swap *TS*	188	136	1.66	2.30
2swap→ins. *TS*	188	132	1.84	2.42

"実行可能"欄は，各アルゴリズムによって得られた実行可能解数を，"最適"欄は，各アルゴリズムが最適値を得たインスタンス数を示す．"gap①"欄は，比較可能な全インスタンスにおける，ILOG CPLEX と各アルゴリズムが算出した値との相対値の平均を，また"gap②"欄は，ILOG CPLEX が最適性を保証したインスタンスにおける，最適値と各アルゴリズムが算出した値の相対値の平均を示す．

この結果，2 種の近傍を用いた ins.→2swap *TS* から，精度の高い実行可能解を得られることが分かった．

次に，最も良い精度が得られた ins.→2swap *TS* と，ILOG CPLEX の平均実行時間を比較する．それぞれの結果を，アクティビティ数によって分けて表 2 に示した．ins.→2swap *TS* は，ILOG CPLEX に比べ，高速に動作することが分かる．

表 2 実行時間の比較

	ILOG CPLEX	ins.? 2swap *TS*
n=10	17.81	1.38
n=20	193.65	16.18
n=30	6410.31	85.86
n=40	8543.76	109.39

5 終わりに

本稿において，我々は *ERCPSP* の定式化，およびそれに対するヒューリスティクスな解法を提案し，比較的小規模なインスタンスにおいて数値実験を試みた．本アルゴリズムが出力する解の，精度に関わる制約条件の調査，アルゴリズム中に用いた各種パラメータの設定，および大規模な問題に対する数値実験，以上の 3 点を今後の課題とする．

参考文献

[1]S. Harthman : *Project Scheduling under Limited Resources Models, Methods, and Applications* (Springer-Verlag Berlin, Heidelberg, 1999).