情報処理学会論文誌　プログラミング　Vol.7　No.1　1–9 (Jan. 2014)

タスクスケジューリング問題の厳密解求解における探索ノード数削減アルゴリズム

中村 あすか[1,a)]　前川 仁孝[1]

受付日 2013年7月4日, 採録日 2013年10月9日

概要：本論文は，分枝限定法を用いたタスクスケジューリング問題の厳密解法の 1 つである DF/IHS 法（Depth First/Implicit Heuristic Search method）の探索ノード数を削減するアルゴリズムを提案する．本手法を用いて大規模なタスクスケジューリング問題を高速に解くために，並列探索アルゴリズムが提案されているが，さらなる高速化を行うためには探索ノード数の削減が必要となる．DF/IHS 法の分枝操作は，スケジュールが未確定となる時刻に実行可能なタスクの処理またはレディ状態を割り当てる全組合せを部分問題として生成する．このため，不必要なレディ状態が割り当てられた部分問題が生成されることがある．そこで，本論文では，DF/IHS 法の探索ノード数を削減するために，レディ状態を割り当てる部分問題のうち，他の部分問題よりも短いスケジューリング長が得られない問題を探索することなしに判定し，探索木中に生成することを抑制する．提案するアルゴリズムは，同一の PE に割り当てられたタスクを融合することで，部分問題のタスクグラフを再定義する．次に，再定義したタスクグラフを比較することで，探索する必要のない部分問題の生成を中止する．本アルゴリズムは，暫定解や下界値を用いないため，探索の進行状況の影響を受けずに探索ノードを削減することができる．

キーワード：タスクスケジューリング，並列処理，組合せ最適化，DF/IHS 法，分枝限定法

A Reduction Algorithm of Branching Nodes for Solving Task Scheduling Problems

Asuka Nakamura[1,a)]　Yoshitaka Maekawa[1]

Received: July 4, 2013, Accepted: October 9, 2013

Abstract: This paper proposes a reduction algorithm of branching nodes for DF/IHS (Depth First/Implicit Heuristic Search method) which is one of the efficient algorithms for solving task scheduling problems using branch and bound method. For solving large-scale task scheduling problems fast using the DF/IHS, it is necessary to not only using parallel search algorithms but reducing branching nodes. The branching operation of the DF/IHS makes some subproblems by enumerating all combinations of the allocatable tasks and idle tasks at the time when the schedule is undecided. For this reason, the DF/IHS may make subproblems assigned some unnecessary idle tasks. Therefore, for reduction of branching nodes of the DF/IHS, the proposed method picks out some subproblems, whose schedule length is longer than others, from among subproblems assigned idle tasks without search. The proposed algorithm redefines task graph of subproblem assigned idle tasks in terms of task-fusion. Then it does not make unnecessary subproblems using the comparison of the redefined task graphs. Since the algorithm does not use upper and lower bounds, it is able to reduce branching nodes without effects of the search order.

Keywords: task scheduling problem, parallel processing, combinatorial optimization problem, DF/IHS, branch and bound method

1. はじめに

マルチコア CPU などのマルチプロセッサ環境の普及により，ソフトウェアの処理速度向上やシステムの稼働時間

[1] 千葉工業大学情報工学科
Department of Computer Science, Chiba Institute of Technology, Narashino, Chiba 275–0016, Japan
[a)] nakamura@mae.cs.it-chiba.ac.jp

の短縮による省エネルギー化などを目的としたプログラムの並列化が行われている [1]．このため，逐次のコードで記述されたソースプログラムを自動的に並列化するコンパイラに対して高い需要がある．並列化コンパイラにおいて，プログラム言語の構文からループ構造などを抽出して並列化するだけでは，ソースプログラム独自の特性などを活かすことが難しく，最適なコンパイル結果が得られるとは限らない [2], [3]．このため，目的コードを最適化するためには，タスクスケジューリング問題の求解が必要となる．

タスクスケジューリング問題は，各プロセッサがどのタスクをどのような順序で実行すれば実行時間が最小になるかを求める組合せ最適化問題である [4], [5]．本問題は，プロセッサ数やタスクの処理時間，プロセッサ間の通信時間，および，タスク間の先行制約の形状などが任意であるとき，スケジュールパターンが膨大になるため短時間で最適解を求解することが難しい [6], [7]．本論文では，タスクスケジューリング問題の中でも，粒度が大きい処理をタスクとしてモデル化した問題を扱う．本問題は，タスクの処理時間に対して並列処理のオーバヘッドが無視できるほど小さくなるが，このようなモデルにおいても（強）NP 困難になることが知られている [8]．このため，本問題の最適解求解には，分枝限定法 [9] による探索を行う必要がある．本問題の探索を効率良く行う手法として DF/IHS 法（Depth First/Implicit Heuristic Search）[10] が提案されている．

DF/IHS 法は，探索過程における枝刈りの効率を高めるために，ヒューリスティック解法である CP/MISF 法（Critical Path/Most Immediate Successors First）[10] のプライオリティリストを利用する探索アルゴリズムである．DF/IHS 法を用いることで効率良く探索することができるが，求解する問題の規模が大きくなるほど探索ノード数が多くなり探索時間が長くなる．DF/IHS 法による探索では，各部分問題から子問題を作成する際にタスクまたはレディ状態を割り当てるパターンをすべて列挙し，それぞれの割当てパターンの部分問題を生成する．このため，DF/IHS 法は，不必要なレディ状態が割り当てられた部分問題を生成することがある．そこで本論文では，DF/IHS 法において不必要なレディ状態が割り当てられた部分問題を判別し，探索過程で生成する部分問題数を削減する手法を提案する．

提案するアルゴリズムは，探索する必要のない部分問題を判別するために，探索木上の部分問題を再定義し，同一のプロセッサに割り当てられたタスクを融合して 1 つのタスクと見なす．これにより，再定義された部分問題において，タスクの割当て時刻のスケジュールが確定していないプロセッサにレディ状態を割り当てて生成した子問題のタスク割当て時刻を利用して，探索する必要のない部分問題の分枝を中止する．このように，提案するアルゴリズムは，暫定解や下界値を用いないため，探索の進行状況の影響を受けずに探索ノードを削減することができる．

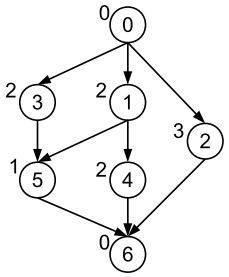

図 1　タスクグラフの例
Fig. 1　An example of task graph.

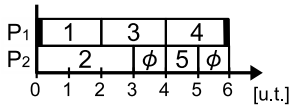

図 2　スケジュールの例
Fig. 2　An example of schedule.

2. タスクスケジューリング問題

タスクスケジューリング問題は，処理時間および先行制約が任意のタスク n 個からなるタスク集合 $T = \{T_1, T_2, \cdots, T_n\}$ を処理能力の等しい m 台のプロセッサからなるプロセッサ集合 $P = \{P_1, P_2, \cdots, P_m\}$ で並列処理するスケジュールのうち，スケジュール長が最も短くなるスケジュールを求める問題である．ただし，各プロセッサ間のデータ転送時間は無視できるほど小さく，処理割込みは起こらないとする．タスクをノード，先行制約をエッジとしたグラフは，タスクグラフと呼ばれる DAG（無サイクル有効グラフ）となる．**図 1** に，タスク数 $n = 5$ のタスクグラフの例を示す．図中では，ノード内の数値 i がタスク番号，ノード左上の数値がタスク i の処理時間 $time(i)$ を表す．また，入口ノード T_0 と出口ノード T_{n+1} は処理時間 0 のダミータスクである．本論文では，タスクグラフを G，そのノードの集合を $V(G)$，エッジの集合を $E(G)$ と表す．このように表記すると $V(G) = T$ となる．

本論文では，スケジュール結果をガントチャートで表す．**図 2** に，図 1 のスケジュール例を示す．本例では，処理時間が 0 であるダミータスクの割当ては太線で表している．また，図中の ϕ は，プロセッサにレディ状態を割り当てたことを示す．本例のスケジュールは，図 1 のすべての先行制約を満たしているため，スケジュール長 6 の実行可能解である．

3. DF/IHS 法

DF/IHS 法は，CP/MISF 法のヒューリスティックを用いて分枝操作を行う分枝限定法である．以下では，DF/IHS 法の木探索アルゴリズムおよび，下界値の導出手法について述べる．

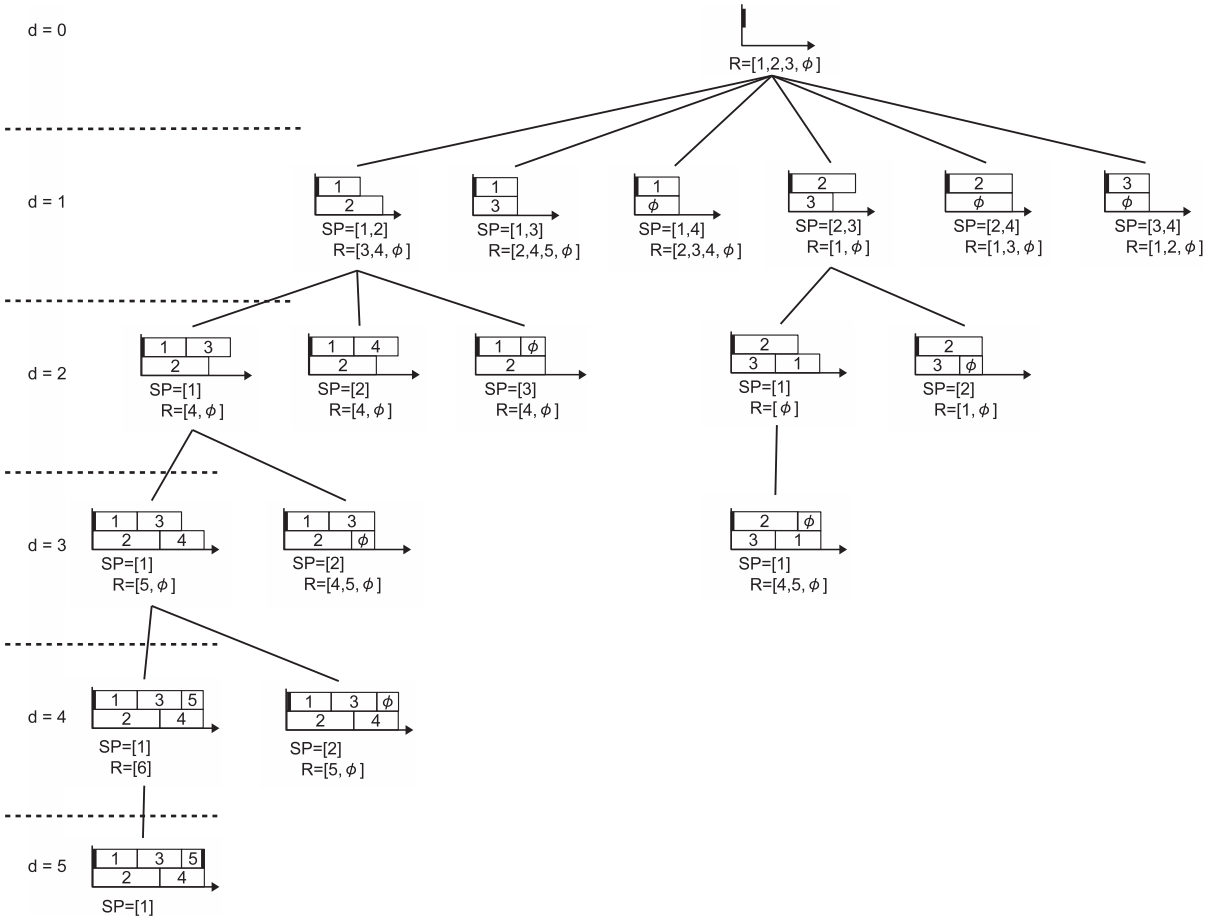

図 3　DF/IHS が生成する探索木の例
Fig. 3　An example of search tree by the DF/IHS.

3.1　DF/IHS 法の木探索アルゴリズム

DF/IHS 法の分枝操作は，スケジュールが未設定となる時刻が最も早いプロセッサに対して，その時刻に実行可能なタスクの処理またはレディ状態を割り当てることで，部分問題 π_i（$i = 1, 2, \cdots$）を生成する．**図 3** に，図 1 のタスクグラフを DF/IHS 法で探索する例を示す．図 3 中の R は各部分問題において実行が可能なタスクを順番に並べた集合であり，R 中の ϕ はレディ状態を表す．R にレディ状態が含まれるのは，実行可能なタスクを割り当てるようなスケジューリングが最適解であるとは限らないためである [11]．また，SP は，R 中のどのタスクを割り当てるかを指定するポインタである．SP は，その部分問題において使用可能なプロセッサに R の何番目のタスクを割り当てるかを指定することで，レディ状態を含めたすべての割当てパターンの子問題を生成する．同じ部分問題から生成される子問題には同じ SP を持つ問題が存在しない．このため，根から順に深さ d の部分問題までの SP を $SP = [(SP_0), (SP_1), (SP_2), \cdots, (SP_d)]$ というように並べることで，任意の部分問題を指し示すことができる．たとえば図 3 において，$SP = [(1, 2), (1), (1), (1), (1)]$ の部分問題は，探索木上で最も左側の葉を指し示す．

DF/IHS 法は，精度の良い暫定解を用いた限定操作によって多くのノードを枝刈りするために，CP/MISF 法 [10] のヒューリスティックにおいて評価が高い部分問題を優先的に探索する．このため本手法は，以下のように R と SP を順序付けする．

R のタスクは，CP/MISF 法のヒューリスティックで高い割当てプライオリティを持つタスクほど先頭にくるように順序付けて格納する．CP/MISF 法のヒューリスティックは，クリティカルパス（CP）[12] の長いタスクほど割当てプライオリティを高く設定する．また，CP 長が同じタスク間においては直接の後続タスク数が多いタスクほど割当てプライオリティを高く設定する．ただし，レディ状態 ϕ は，割当て優先度を最も低く設定するため，つねに R の最後尾に格納する．本例の場合は，CP 長が大きいタスクほどタスク番号が小さいため，R をタスク番号順に記述するだけでよい．本例のように CP 長の順にタスク番号が割り振られていない問題においても，CP 長の順にタスク番号を振り直すような前処理を行うことで同様に探索ができる．

SP は，R の先頭にあるタスクを優先的に割り当てる子問題から順に生成する．**図 4** に，次に探索する部分問題の

```
set_sp(){
  if(直前に探索した部分問題から初めて子問題を生成する)
    for(i ← 0;i < 最大割当て数;i++){
      sp[i] ← i;
      return;
    }
  sp[] ← 直前に探索した部分問題の SP_d
  task ← 実行可能タスク数
  pe ← 割当て可能プロセッサ数
  i ← φ 以外で最後尾の sp 要素の配列番号
  sp[i] ← sp[i] + 1;
  while(sp[i] < task){
    if(i ≥ pe − 1)
      return;
    sp[i + 1] ← sp[i] + 1;
    i ← i + 1;
  }
}
```

図 4　SP 値設定の擬似コード
Fig. 4　Pseudo-code of calculating SP.

SP を生成する疑似コードを示す．図 4 のように SP を設定することで，最初に得られる実行可能解が CP/MISF 法の解と等しくなるため，探索初期に精度の良い暫定解を得ることができる．

3.2　下界値の計算

DF/IHS 法では，部分問題 π_a の下界値 $lb(\pi_a)$ を，式 (1) によって求める [13]．ただし，式 (1) の計算は，π_a が限定可能であると分かった時点，つまり，$lb(\pi_a) \geq$ (暫定解) が決定した時点で打ち切る．式 (1) 中の $lb_{cr}, lb_{div}, lb_{hu}$ は，式 (2)〜式 (4) により求める．

$$lb(\pi_a) = \max\{lb_{cr}(\pi_a), lb_{div}(\pi_a), lb_{hu}(\pi_a)\} \tag{1}$$

$$lb_{cr}(\pi_a) = \max_{i \in I(\pi_a)} cp(i) + \min_{1 \leq i \leq m} t_{\pi_a(Pi)} \tag{2}$$

$$lb_{div}(\pi_a) = \left\lceil \sum_{i \in I(\pi_a)} \frac{time(i)}{m} \right\rceil + \min_{1 \leq i \leq m} t_{\pi_a(Pi)} \tag{3}$$

$$lb_{hu}(\pi_a) = lb_{cr}(\pi_a) + \lceil q_{(\pi_a)} \rceil \tag{4}$$

ここで，$I(\pi_a)$ は π_a の未割当てタスク集合，$cp(i)$ は出口ノードからタスク T_i までの最長パス長，$t_{\pi_a}(P_i)$ はノード π_a においてプロセッサ P_i のスケジュールがまだ決まっていない時刻を表す．また，t_{hu} は，Fernández によって拡張された Hu の下界 [14] である．式中の $q_{(\pi_a)}$ は，式 (5)〜式 (8) のように負荷密度関数 $F(\overline{\tau}, t)$ から求める．

$$q(\pi_a) = \max_{0 \leq t_k \leq lb_{cr}(\pi_a) - t_0} \left(-t_k + \frac{1}{m} \int_0^{t_k} F(\overline{\tau}, t) dt \right) \tag{5}$$

$$F(\overline{\tau}, t) = \sum_{j \in I(\pi_a)} f(\overline{\tau}, t) \tag{6}$$

$$\overline{\tau} = lb_{cr}(\pi_a) - t_{pass}(i) - \min_{1 \leq i \leq m} t_{\pi_a(Pi)} \tag{7}$$

$$f(\overline{\tau}_j, t) = \begin{cases} 1, & for \quad t \in [\overline{\tau}, \overline{\tau} + time(j)] \\ 0, & otherwise \end{cases} \tag{8}$$

4.　探索ノード数の削減

分枝限定法の探索時間を削減するためには，一般的に，限定操作および分枝操作においてそれぞれ探索ノード数を減らすための工夫が必要である [9]．このため，限定操作において多くの部分問題を枝刈りできるように，タスクスケジューリング問題の下界値の精度向上に関する研究が行われている [14], [15]．DF/IHS 法においても，探索ノード数を最小限に抑えるために，式 (1) のように下界値を求めるためのヒューリスティックを複数用いることで限定操作の効率を高めている．しかし，DF/IHS 法の分枝操作においては，実行可能解をすべて列挙するように探索木を生成するため，無駄な部分問題の生成を抑制しているとはいい難い．

無駄な部分問題の例として，図 3 において $SP = [(1, 4)]$ の部分問題をあげる．本部分問題は，P_1，P_2 ともに時刻 $2[u.t.]$ までのスケジュールまでが確定している．また，タスク T_1 のみが割当て済みであり，時刻 $2[u.t.]$ の段階で T_1 の先行制約のみ解決される．ここで，本部分問題と $SP = [(1, 3)]$ の部分問題を比較する．$SP = [(1, 3)]$ の部分問題は，スケジュールの確定している時刻が $SP = [(1, 4)]$ の部分問題と等しいにもかかわらず，T_1 だけでなく T_3 が割当て済みタスクであるため，時刻 $2[u.t.]$ の段階で T_1 と T_3 の先行制約が解決される．このため，$SP = [(1, 3)]$ の部分問題は，少なくとも $SP = [(1, 4)]$ の問題よりも精度の高い実行可能解が得られると判断できる．よって，$SP = [(1, 4)]$ の部分問題は探索する必要のない部分問題である．

DF/IHS 法の探索過程において無駄な部分問題を生成しても，限定操作によって枝刈りすることで，部分問題数を削減できる可能性がある．ただし，限定操作による枝刈りは，暫定解の精度に影響を受けるため，探索する必要がない部分問題でも枝刈りされない可能性がある．そこで，本論文では，生成された部分問題のうち探索する必要のない部分問題を，下界値や暫定解を用いずに判定し，枝刈りする．

以降では，まず，4.1 節と 4.2 節で部分問題を再定義し，探索する必要のない部分問題を一般化する．次に，4.3 節で提案するアルゴリズムについて述べる．

4.1　部分問題の再定義

本論文では，DF/IHS 法の探索過程において探索する必要のない部分問題を判別しやすくするために，部分問題を再定義する．本論文で再定義する部分問題は，同一プロ

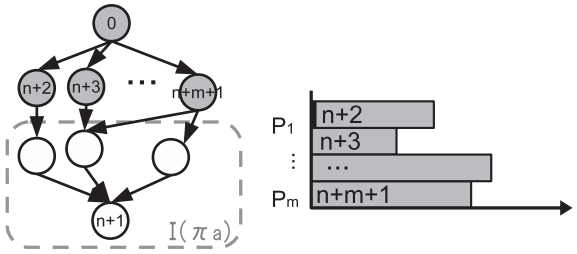

図 5　部分問題の例
Fig. 5　An example of subproblem.

セッサに割当てが確定している処理内容のうちダミータスクである入口ノード T_0 以外を，レディ状態を含め 1 つのタスクとして扱う．そして，元の部分問題を，タスクの融合によって生成されたタスクを各プロセッサの時刻 0 に割り当てた場合のスケジュールを求める問題に置き換える．このとき，新たなタスクとして再定義されるタスクは，DF/IHS 法の分枝規則によって，すべての先行制約を満たすように割当てが決められている．このため，割当て済みタスクを新たに定義されたタスクに置き換えた部分問題を探索しても，元の部分問題から最適解を探索することができる．また，同様の理由から，新たに定義された部分問題のタスクグラフから，元の部分問題で各プロセッサに最後に割り当てられたタスクと入口タスク以外の先行制約を取り除いても，元の部分問題の先行制約を守ることができる．

図 5 に，再定義した部分問題の例を示す．図 5 中の網掛け部分は，複数のタスクを融合して生成したタスクであることを表す．図 5 のように，複数のタスクを融合して生成したタスクのタスク番号は，元の部分問題のタスク番号と重複しないように，割り当てられたプロセッサ P_i のプロセッサ番号 i を用いて $n+1+i$ とする．タスク T_{n+1+i} の先行タスクは T_0，後続タスクは元の部分問題において P_i に最後に割り当てられたタスクの後続タスクである．

上記のように部分問題を再定義すると，各プロセッサに割り当てられたタスクがどの順番で実行されたかという情報を失ってしまう．しかし，DF/IHS 法は，どのようにタスクを割り当てることで現在探索中の部分問題が生成されたかという情報を SP に格納している．このため，暫定解が更新された際には，SP を根から葉までたどることで，生成されたスケジュールを知ることができる．

4.2　探索する必要のない部分問題

探索する必要のない部分問題は，4.1 節で再定義した部分問題を用いて一般化すると，定理 4.1 のように表すことができる．

定理 4.1　部分問題 π_a のタスクグラフである $G_{\pi_a}(V(\pi_a), E(\pi_a))$ と部分問題 π_b のタスクグラフである $G_{\pi_b}(V(\pi_b), E(\pi_b))$ において，$V(\pi_a) \subseteq V(\pi_b)$ かつ $E(\pi_a) \subseteq E(\pi_b)$，つまり，$G_{\pi_a}$ は G_{π_b} の部分グラフであるとする．このとき，部分問題 π_b は探索する必要のない部分問題である．

以下では，DF/IHS 法の探索過程において，定理 4.1 の π_b のように他の部分問題のタスクグラフを自身のタスクグラフが包含している部分問題を探索しなくても最適解を求解できることを示すために補題を設定し，証明する．

補題 4.1　G_{π_a} は G_{π_b} の部分グラフであるとする．このとき，G_{π_b} の実行可能解 $S(\pi_b)$ と同じ時刻に G_{π_a} のタスクを実行するスケジュール $S(\pi_a)$ があるとき，スケジュール $S(\pi_a)$ は G_{π_a} の実行可能解である．

証明 4.1　$E(\pi_a) \subseteq E(\pi_b)$ より，$E(\pi_b)$ の先行制約がすべて満たされているとき $E(\pi_a)$ もすべて満たされる．すべての先行制約を満たしたスケジュールは，タスクスケジューリング問題の実行可能解である．このため，補題 4.1 が成り立つ．

補題 4.2　G_{π_a} は G_{π_b} の部分グラフであるとする．部分問題 π の最適解のスケジュール長を $Best(\pi)$ とおいたとき，$Best(\pi_a) \leq Best(\pi_b)$ が成り立つ．

証明 4.2　背理法を用いる．$Best(\pi_a) > Best(\pi_b)$ と仮定する．このとき，補題 4.1 より，部分問題 π_a には $Best(\pi_b)$ と同一のスケジュール長を持つ実行可能解が存在するはずである．このため，$Best(\pi_a) > Best(\pi_b)$ が成り立たない．よって，補題 4.2 が成り立つ．

補題 4.2 より，G_{π_a} が G_{π_b} の部分グラフであるとき，$Best(\pi_a) \leq Best(\pi_b)$ であり，部分問題 π_b を探索しても π_a よりも短いスケジュール長が得られない．このため，DF/IHS 法の探索過程において π_b の探索を打ち切っても，DF/IHS 法の探索結果における最適性は失われない．

4.3　提案するアルゴリズム

4.1 節より，図 3 中の $SP = [(1,2),(1)]$ の部分問題と $SP = [(2,3),(1)]$ の部分問題は，部分問題を再定義すると同じタスクグラフで表すことができる．このため，定理 4.1 により，どちらか一方の部分問題のみ探索すればよく，片方の部分問題を枝刈りできる．このように，DF/IHS 法が探索過程で生成する部分問題の中には，定理 4.1 によって探索する必要がないと判定される問題が多くある．これらの部分問題をすべて発見するためには，ハッシュテーブルのようなデータ構造を用いて，探索過程で生成した部分問題をすべて記憶する必要がある．DF/IHS 法は，ハッシュテーブルを使わずに探索するため，ハッシュテーブルを追加すると，それを管理する処理を追加する必要が生じる．DF/IHS 法に新たな処理を追加すると，そのアルゴリズムがうまく働くような問題の求解においては高い効果を期待できるが，そうでない問題を求解する際には，追加した処理の分だけ探索時間が増加するというトレードオフが生じる．そこで，本論文では，なるべく DF/IHS 法に追加する処理が少なくなるようにアルゴリズムを設計し，SP を設

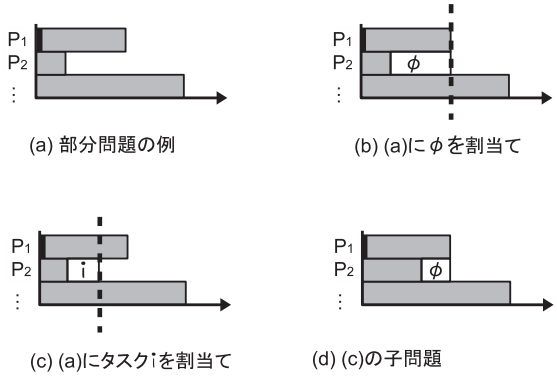

図 6　削減するスケジュールの例
Fig. 6　An example of reduced schedule.

定する際に，探索する必要がないことが容易に判定可能な部分問題のみを削減する．

図 6 に，本論文が探索を打切る部分問題の例を示す．ガントチャート中の点線は，タスクまたはレディ状態を割り当てることで生成された子問題のタスク割当て時刻を示す．図 6 (a) の部分問題は，スケジュールが確定している時刻が最も短いプロセッサに対して割当てが行われても，スケジュールが確定している時刻が最も短いプロセッサが変わらないようなタスク i がレディ状態であるとする．このとき，(a) の部分問題において，レディ状態を割り当てられた子問題が (b)，タスク i を割り当てられた子問題が (c) である．(b) の部分問題と，(c) の部分問題から生成される子問題である (d) の部分問題を，4.1 節のように再定義すると，(b) の部分問題のタスクグラフが (d) のタスクグラフの部分問題となる．このため，(c) の部分問題を探索することで (d) の部分問題も探索されるので，定理 4.1 より (b) の部分問題は探索する必要がないと判定できる．このように，提案するアルゴリズムは，あるタスクを割り当てることで子ノードの割当て時刻が早まるような割当てパターンが存在するときに，そのタスクが割り当てられずにレディ状態を割り当てられて生成された部分問題を探索木から取り除く．複数のプロセッサにタスクを割り当てる部分問題においても，SP の要素が確定するたびに 4.1 節の再定義を行い，同様の手順で探索木から取り除くことができる．

特定のタスクを連続して割り当て続けるように SP を設定する場合，子問題の生成は容易になるが，DF/IHS 法が探索に採用している CP/MISF 法のヒューリスティックを利用できなくなる．このため，提案するアルゴリズムでは，DF/IHS 法の探索順序を変えないようにする．

以上をふまえ，本論文で提案するアルゴリズムの疑似コードを**図 7** に示す．図 7 のコードでは，レディ状態を割り当てられずに生成された部分問題の探索は従来どおりに行う．一方，ϕ を含む割当てによって生成された部分問題は，生成時に，自身の割当て時間と親問題の割当て時間の差よりも処理時間が短い未割当ての実行可能タスクが存在するとき，探索を中止する．

```
set_sp(){
  sp[] ← 現在探索中の部分問題の SP
  task ← 実行可能タスク数
  pe ← 割当て可能プロセッサ数
  MinTime ← R 中のタスクの最小処理時間 + 割当て時刻
  if(pe = m and MinTime ≥ φ のみを割り当てたときの割当て時刻)
    MinTime ← φ のみを割り当てたときの割当て時刻
  if(直前に探索した部分問題から初めて子問題を生成する)
    for(i ← 0;i < 最大割当て数;i++){
      sp[i] ← i;
      return;
    }
  do{
    i ← φ 以外で最後尾の sp 要素の配列番号
    sp[i] ← sp[i] + 1;
    while(sp[i] < task){
      if(i ≥ pe − 1)
        return;
      sp[i + 1] ← sp[i] + 1;
      i ← i + 1;
    }
  }while(φ が割当てられている and 次の割当て時刻 ≥ MinTime)
}
```

図 7　最小処理時間のみを用いた SP 値設定の擬似コード
Fig. 7　Pseudo-code of calculating SP using minimal task processing time.

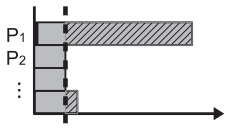

図 8　提案手法が効果的に働く部分問題の例
Fig. 8　An example of working efficiently by the proposed method.

上記のように，提案手法は，下界値を用いずに部分問題を枝刈りできる．このため，提案手法を用いることで，DF/IHS 法で下界値の精度が低い部分問題を枝刈りできる可能性がある．**図 8** に，提案手法が有効に働く部分問題の例を示す．DF/IHS 法の下界値を求めるヒューリスティックでは，図 8 中の斜線部のタスクが割当て済みの時刻にも未割当てタスクを割り当てるように計算する．このため，斜線部の時刻が長い部分問題ほど，式 (1) で高精度な下界値が求まりにくく，DF/IHS 法で枝刈りが起こりにくくなる．一方，提案手法では，SP を用いて ϕ を割り当てる子問題の包含関係を判定し，枝刈りできる．

5. 評価

提案手法の有効性を示すために，DF/IHS 法と提案手法で生成される部分問題数および探索時間を比較し，評価する．本評価で用いるタスクスケジューリング問題には，標準タスクグラフセット [16] の 50 タスクの問題 60 問を用い

る．以下の節では，まず，枝刈りによる部分問題の削減効果が最も高いと考えられる深さ 1 の部分問題の数について評価する．次に，求解過程で生成された部分問題数および探索時間を評価する．

5.1　深さが 1 の部分問題数の測定

本評価では，DF/IHS 法と提案手法が探索する深さ 1 の部分問題の数を比較する．**表 1** に，DF/IHS 法および提案手法による求解過程において生成される部分問題数を，根の部分問題で実行可能なタスク数による度数分布表で示す．表 1 の括弧内の数値は削減率である．削減率は，DF/IHS 法で生成する部分問題数を $n(S_{DF/IHS})$，提案手法で生成する部分問題数を $n(S_{proposed})$ とおいて，式 (9) で求める．

$$\text{削減率 } [\%] = \frac{n(S_{DF/IHS}) - n(S_{proposed})}{n(S_{DF/IHS})} \times 100 \quad (9)$$

表 1 より，プロセッサ数 m が大きいほど，削減率が高くなることが分かる．これは，提案手法が探索する必要があるかどうかを判定する部分問題が，レディ状態が割り当てられた部分問題だけだからである．同一の R を持つ部分問題において，プロセッサ数が m の SP のパターンはプロセッサ数が m 未満の SP をすべて含む．たとえば，R の要素数が 4 のとき，$m = 2$ の SP は $(0,1)$，$(0,2)$，$(0,3)$，$(1,2)$，$(1,3)$，$(2,3)$，$(3,3)$ となり，$m = 3$ の SP は $(0,1,2)$，$(0,1,3)$，$(0,2,3)$，$(0,3,3)$，$(1,2,3)$，$(1,3,3)$，$(2,3,3)$，$(3,3,3)$ となる．R の 3 番目の要素はレディ状態の割当てであり，$m = 2$ のときにはレディ状態を割り当てていない SP の組合せにも，$m = 3$ のときにはレディ状態が割り当てられる．このように，プロセッサ数が増えることで，レディ状態を割り当てる SP の割合が大きくなる．このため，提案手法は，プロセッサ数が多いほど多くの部分問題に対して探索する必要があるかどうかの判定を行う．これにより，プロセッサ数 m が大きいほど，高い削減率が得られた．

また，実行可能なタスク数が少ない問題において高い削減率が得られたが，度数分布表において，実行可能なタスク数と削減率に相関があるとはいい難い．このため，削減率が最も低い問題と最も高い問題に着目する．本評価において最も削減率が低い問題は，削減率が約 0.01%であった．本問題は，実行可能なタスク数が 16 で，実行可能なタスクの処理時間の最小値が 1，処理時間が 1 の実行可能タスクが 7 個の問題である．このため，処理時間が 1 のタスクが割り当てられる SP のパターンが多くなり，削減率が低くなったと考えられる．一方，本評価において最も削減率が高い問題は，削減率が約 50.00%であった．本問題は，実行可能なタスク数が 17 で，実行可能なタスクの処理時間の最小値が 1，処理時間が 1 の実行可能タスクが 1 個の問題である．このため，レディ状態が割り当てられた部分問題のうち，処理時間が 1 のタスクが同時に割り当てられていない SP のパターンが多くなる．これにより，高い削減率が得られたと考えられる．以上より，提案手法は，同一処理時間タスクが少ない問題ほど，高い削減率が得られると期待できる．

5.2　求解過程で探索する部分問題数の測定

本節では，DF/IHS 法と提案手法で 2，4，8，16 台のプロセッサへ割り当てるタスクスケジューリング問題を求解し，求解過程で生成される部分問題数を評価する．本評価の下界値の計算には，3.2 節の式 (1) を用いる．式 (9) を用いて削減率を求めた結果，240 問中，削減率が 0 の問題は 175 問（約 72.9%），削減率が正の問題は 63 問（約 26.3%），削減率が負の問題は 2 問（約 0.8%）であった．**図 9** に，削減率が正の値になった問題の削減率の分布を示す．図 9 の横軸は DF/IHS 法が生成した部分問題数，縦軸は削減率である．図 9 より，DF/IHS 法で部分問題が多く生成される問題ほど，削減率が高く，提案手法が有効に働くことが分かる．

表 1　DF/IHS 法と提案手法が生成する $d = 1$ の部分問題数 [個]
Table 1　Number of subproblems of $d = 1$ using the DF/IHS and the proposed method.

実行可能タスク数	問題数	DF/IHS 法				提案手法			
		$m = 2$	$m = 4$	$m = 8$	$m = 16$	$m = 2$	$m = 4$	$m = 8$	$m = 16$
3–5	7	13.0	24.6	25.3	25.3	9.4 (27.69)	13.1 (46.75)	13.1 (48.22)	13.1 (48.22)
6–10	19	37.1	184.5	345.0	345.9	30.8 (16.98)	134.8 (26.94)	247.6 (28.23)	248.4 (28.19)
11–15	18	90.0	1134.6	9632.1	12855.9	78.6 (12.67)	861.2 (24.10)	6244.1 (35.17)	8120.9 (36.83)
16–20	12	158.0	3492.8	84435.5	207492.8	143.5 (9.18)	2926.9 (16.20)	63247.3 (25.09)	147724.0 (28.81)
21–25	4	283.5	11801.0	1154464.5	17435441.0	262.3 (7.48)	10144.5 (14.04)	892235.8 (22.71)	12146605.5 (30.33)

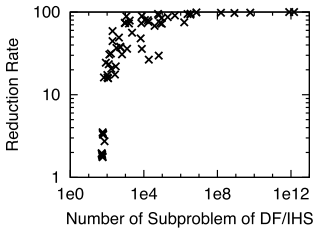

図 9　DF/IHS 法に対する提案手法の削減率
Fig. 9　Reduction rate of the proposed method with the DF/IHS.

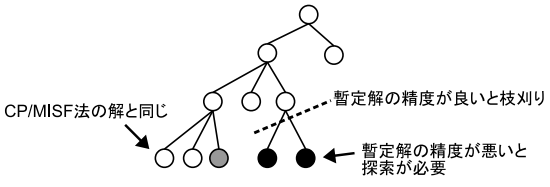

図 10　提案手法を用いることで部分問題数が増える例
Fig. 10　An example of increase of subproblems by the proposed method.

また，図 9 において，提案手法を用いることで部分問題数が増加した問題が 2 問あった．これは，提案手法によって，精度の高い上界値を持つ部分問題が枝刈りされたためであると考えられる．**図 10** に，提案手法を用いることで生成される部分問題数が増加する例を示す．図 10 は DF/IHS 法が生成する探索木であり，灰色の部分問題が初期解よりも精度の良い暫定解を与える部分問題であるとする．DF/IHS 法は，探索木左側から深さ優先探索を行うが，枝刈りの判定に下界を用いるため灰色の部分問題を枝刈りせずに探索し，暫定解を更新する．このため，本例の DF/IHS 法は，灰色の部分問題の暫定解を用いた枝刈りが行われ，点線の部分で枝刈りが起こり黒色の部分問題を探索する必要がなくなる．一方，提案手法では，最適解が存在しないと判断された部分問題は，どんなに精度の高い暫定解を持っていても探索が打ち切られる．図 10 において提案手法が灰色の部分問題の探索を打ち切ると，探索の初期に精度の良い暫定解を得ることができなくなる．これにより本例の提案手法は，点線の部分で枝刈りできずに黒色の部分問題を探索する必要が生じ，求解過程で生成する部分問題数が増加する．このように，DF/IHS 法の初期段階で探索された精度の高い暫定解を持つ部分問題を提案手法が枝刈りすると，暫定解が更新されるまでの間，下界値を用いた枝刈りの効率が落ちる．これにより，求解処理全体での探索する部分問題数が増えたと考えられる．ただし，提案手法は，DF/IHS 法と同様に CP/MISF 法の解を必ず初期解として探索するため，精度の高い初期解を用いて枝刈りし，多くの問題で高い削減率が得られたと考えられる．

5.3　探索時間の測定

本節では，5.2 節と同様に，DF/IHS 法と提案手法で 2, 4, 8, 16 台のプロセッサへ割り当てるタスクスケジューリング問題を求解し，探索時間を評価する．評価環境は，CPU が AMD Phenom II X6 1100T（3.3 GHz），メモリが 8 GB である．また，本評価では，高速化率を式 (10) で定義する．

$$\text{高速化率 [倍]} = \frac{\text{DF/IHS 法の探索時間}}{\text{提案手法の探索時間}} \tag{10}$$

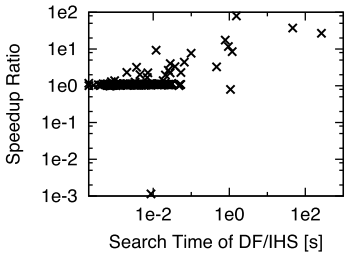

図 11　DF/IHS 法に対する提案手法の高速化率
Fig. 11　Speedup ratio of the proposed method with the DF/IHS.

図 11 に，DF/IHS 法に対する提案手法の高速化率を示す．図 11 の横軸は DF/IHS 法の探索時間，縦軸は高速化率である．図 11 より，高速化率が 1 以上となった問題は 236 問（約 98.3%），1 未満となった問題は 4 問（約 1.7%）であることが分かる．このため，提案手法を用いることで，多くの問題の探索時間を短縮できることが確認できた．また，提案手法を用いることで，最大約 79.3 倍，相乗平均で約 1.26 倍高速に探索することが確認できた．

図 11 において高速化率が 1 未満の問題 4 問のうち，2 問の高速化率は約 0.99 倍であり DF/IHS 法と提案手法にほとんど差がなかったが，残り 2 問の高速化率は約 0.80 倍と約 0.001 倍であった．特に高速化率が低かった 2 問題は，5.2 節の測定において部分問題数が増加し，削減率が負の値になった問題であった．しかし，これらの問題は DF/IHS 法での探索時間 0.1 秒未満と短いため，提案手法でも短い時間で探索することが可能である．また，5.2 節の評価において提案手法を用いても DF/IHS 法と部分問題数が変わらなかった 175 問のうち，173 問は高速化率が 1 より大きかった．これは，提案手法が，式 (4) による下界値の計算よりも高速に枝刈りの判定をできるためである．

6.　おわりに

本論文では，探索する部分問題数を削減するために，暫定解や下界値を用いずに DF/IHS 法で生成される無駄な部分問題を削減するアルゴリズムを提案し，有効性を評価した．評価の結果，提案手法を用いることで，DF/IHS 法が生成する部分問題数を最大約半分に削減できることが確認できた．また，提案手法は，DF/IHS 法に比べて，探索時間が最大約 79.3 倍，相乗平均で約 1.26 倍高速化することが確認できた．

本手法は，暫定解や下界値を用いずに探索する部分問題数を削減できるため，探索順序に関係なく部分問題数を削減することができる．このため，DF/IHS 法自身を並列に実行する際には，同時に探索を行うスレッドやプロセス間で暫定解を共有するためのコストの削減などが期待できる．また，並列化コンパイラにおけるタスクスケジューリングに本手法を用いることで，最適化された目的コードをさらに高速に生成できるようになることが期待できる．

参考文献

[1] 中田育男，渡邊　坦：21 世紀のコンパイラ道しるべ—COINS をベースにして：概要，情報処理，Vol.47, No.4, pp.425–436 (2002).
[2] エイホ，A.V.，セシィ，R.，ウルマン，J.D.（著），原田賢一（訳）：コンパイラ II—原理・技法・ツール，サイエンス社 (1990).
[3] 中田育男：コンパイラの構成と最適化，朝倉書店 (1999).
[4] Land, A. and Doig, A.: An Automatic Method of Solving Discrete Programming Problems, *Econometrica*, Vol.28, No.3, pp.497–520 (1960).
[5] El-Rewini, H., Ali, H. and Lewis, T.: Task scheduling in multiprocessing systems, *IEEE Computer*, Vol.28, No.12, pp.27–37 (1995).
[6] Rashtbar, S., Isazadeh, A. and Khanly, L.: A new hybrid approach for multiprocessor system scheduling with genetic algorithm and tabu search (HGTS), *2010 3rd International Conference on Information Sciences and Interaction Sciences* (*ICIS*), pp.626–631 (2010).
[7] 宇都宮雅彦，塩田隆二，甲斐宗徳：通信を考慮したタスクスケジューリング問題の探索解法のための高速化手法，情報科学技術フォーラム講演論文集，Vol.9, No.1, pp.233–237 (2010).
[8] 笠原博徳：並列処理技術，技術評論社 (1991).
[9] 茨木俊秀：組合せ最適化—分枝限定法を中心として，産業図書 (1983).
[10] Kasahara, H. and Narita, S.: Practical Multiprocessor Scheduliing Algorithms for Efficient Parallel Processing, *IEEE Trans. Comput.*, No.11, pp.1023–1029 (1984).
[11] Ramamoorthy, C., Chandy, K. and Gonzalez, M.: Optimal Scheduling Strategies in a Multiprocessor System, *IEEE Trans. Comput.*, No.2, pp.137–146 (1972).
[12] Coffman, E.: *Computer and Job-Shop Scheduling Tehory*, John Wiley & Sons (1976).
[13] 飛田高雄，笠原博徳：標準タスクグラフセットを用いた実行時間最小マルチプロセッサスケジューリングアルゴリズムの性能評価，情報処理学会論文誌，Vol.43, No.4, pp.936–947 (2002).
[14] Fernández, E. and Bussell, B.: Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules, *IEEE Trans. Comput.*, No.8, pp.745–751 (1973).
[15] 藤田　聡，益川正如，田頭茂明：マルチプロセッサスケジューリング問題に対する分枝限定解法の下界の改良に基づく高速化について，並列処理シンポジウム（JSPP2002），pp.289–296 (2002).
[16] Standard Task Graph Set (STG), available from ⟨http://www.kasahara.elec.waseda.ac.jp/schedule/⟩ (accessed 2013-06-30).

中村 あすか　（学生会員）

1986 年生．2009 年千葉工業大学情報科学部情報工学科卒業．2011 年同大学大学院情報科学研究科情報科学専攻博士前期課程修了．同年同大学院情報科学研究科情報科学専攻博士後期課程入学．主として，並列探索に関する研究に従事．

前川 仁孝　（正会員）

1967 年生．1990 年早稲田大学理工学部電気工学科卒業．1992 年同大学大学院理工学研究科電気工学専攻修士課程修了．1993 年日本学術振興会特別研究員．1994 年早稲田大学理工学部助手．1998 年千葉工業大学情報工学科講師．2002 年同大学助教授，2011 年同大学教授，現在に至る．博士（工学）．主として，各種アプリケーションの並列処理の研究に従事．