情報処理学会論文誌 Vol.58 No.3 654–662 (Mar. 2017)

タスクスケジューリング問題における レディ状態の割当て削減による PDF/IHS の高速化

中村 あすか[1,a)] 富永 浩文[1,b)] 前川 仁孝[2,c)]

受付日 2016年5月31日, 採録日 2016年12月1日

概要：本論文は，タスクスケジューリング問題の厳密解法である PDF/IHS（Parallel Depth First/Implicit Heuristic Search）の探索ノード数を削減するアルゴリズムを提案する．PDF/IHS は，階層的挟み撃ち探索を用いた分枝限定法の並列探索アルゴリズムであり，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は探索ノード数の削減が必要となる．PDF/IHS 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，PDF/IHS 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，最適解が得られないことが保障できる部分問題を枝刈りする．提案するアルゴリズムは，分枝操作で割り当てられたタスクの処理時間の情報から探索する必要のない部分問題を判定する．評価の結果，提案手法は，PDF/IHS に対して最大約 96 倍高速に求解できることを確認した．

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，PDF/IHS，分枝限定法

A Speedup Method for PDF/IHS by Reducing Allocations of Idle Tasks in Task Scheduling Problem

Asuka Nakamura[1,a)] Hirobumi Tominaga[1,b)] Yoshitaka Maekawa[2,c)]

Received: May 31, 2016, Accepted: December 1, 2016

Abstract: This paper proposes a reduction algorithm of branching nodes for the task scheduling algorithm PDF/IHS (Parallel Depth First/Implicit Heuristic Search). The PDF/IHS is a parallel branch and bound method using HPAS (Hierarchical Pincers Attack Search). For solving large-scale task scheduling problems fast using the PDF/IHS, it is necessary to not only using parallel search algorithms but reducing branching nodes. The branching operation of the PDF/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 PDF/IHS may make subproblems assigned some unnecessary idle tasks. Therefore, for reduction of branching nodes of the PDF/IHS, the proposed method cuts some subproblems assigned idle tasks, whose schedule length is longer than others. The proposed algorithm determines unnecessity of searching subproblems using time of allocated task in the branching operation. As a result of evaluation, the speedup ratio of the proposal method compared with the PDF/IHS is about 96 times on the maximum.

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

[1] 千葉工業大学大学院情報科学研究科情報科学専攻
Graduate School of Information and Computer Science, Chiba Institute of Technology, Narashino, Chiba 275–0016, Japan
[2] 千葉工業大学情報科学部情報工学科
Department of Computer Science, Chiba Institute of Technology, Narashino, Chiba 275–0016, Japan
[a)] nakamura@mae.cs.it-chiba.ac.jp
[b)] tominaga@mae.cs.it-chiba.ac.jp
[c)] maekawa@mae.cs.it-chiba.ac.jp

1. はじめに

マルチプロセッサシステム上で実行するソフトウェアには，処理速度向上やシステムの稼働効率向上による省エネルギー化などが求められるため，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 [1]．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およびタスク間の先行制約形状が任意であるとき，プロセッサ間の通信時間が無視できるような場合でもスケジュールパターンが膨大となり，短時間で最適解を求解することが難しい [6], [7]．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 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（Depth First/Implicit Heuristic Search）[10]，および，その並列探索アルゴリズムである PDF/IHS（Parallel DF/IHS）[11] が提案されている．

DF/IHS 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な解法である 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．また，PDF/IHS は，階層的挟み撃ち探索を用いて DF/IHS の探索木を複数の PE（Processor Element）が階層的に左右から挟み撃つ形で探索する．タスクスケジューリング問題は，PDF/IHS 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．

筆者ら [12] は，DF/IHS の分枝操作に着目し，逐次探索アルゴリズムである DF/IHS がレディ状態を割り当てた部分問題の中には探索する必要のない部分問題が存在することを明らかにした．さらに，DF/IHS に対する探索ノード数削減手法を提案し，有効性を示した．本手法は，各部分問題のタスクを割り当てる時刻と割当て可能なタスクの処理時間を利用して枝刈りを行う．このため，DF/IHS と同じ分枝規則で探索を行う PDF/IHS に対しても有効に働くと考えられる．

そこで本論文では，PDF/IHS 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．

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な n 個のタスクからなるタスク集合 $\mathbf{T} = \{T_1, T_2, \cdots, T_n\}$ 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 $\mathbf{P} = \{P_1, P_2, \cdots, P_m\}$ 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 [8]．本論文で扱う問題は，粒度が大きい処理をタスクとしてモデル化した問題であり，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，処理割込みが起こらないとする．

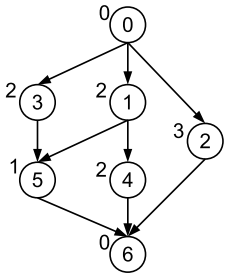


図 1 タスクグラフの例
Fig. 1 An example of task graph.

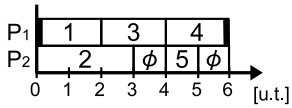


図 2 スケジュールの例
Fig. 2 An example of schedule.

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．本例では，タスク 1 からタスク 4 にエッジが伸びている．このため，タスク 4 の実行開始時刻は，タスク 1 の実行完了時刻以降である必要がある．ただし，プロセッサ間のデータ転送時間を 0 としているため，タスク 1 の実行完了時刻と同じ時刻にタスク 4 の実行を開始することができる．また，T_0 は先行タスクを持たない入口ノードであり，T_{n+1} は後続タスクを持たない出口ノードである．入口ノードと出口ノードは，先行制約のエッジをたどってすべてのノードに到達可能な処理時間 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 $V(G) = \mathbf{T}$ となる．以下では，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. PDF/IHS

PDF/IHS は，分枝限定法を元にした探索アルゴリズムである DF/IHS を階層的に挟み撃つ形で並列化したアルゴ

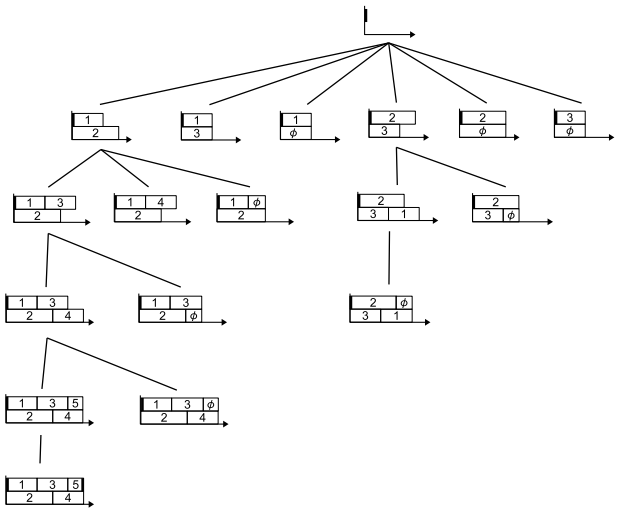


図 3 DF/IHS で生成する探索木の例
Fig. 3 An example of search tree by the DF/IHS.

リズムである [11]．このため，本手法は，複数の PE が分枝操作と限定操作を繰り返し行うことで最適解を求める．以下では，PDF/IHS の分枝操作と限定操作，および，並列探索アルゴリズムについて述べる．

3.1 分枝操作

PDF/IHS 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 PDF/IHS で探索する例を示す．本例のように，PDF/IHS が割当て可能なタスクが存在する場合にもレディ状態を割り当てる部分問題を生成するのは，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 [13]．

PDF/IHS 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スにおいて評価が高い部分問題を優先的に探索する．CP/MISF 法のヒューリスティックスは，クリティカルパス [14] が長く，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し，レディ状態 ϕ の割当て優先度を最も低く設定する．このようにプライオリティを設定することで，最適解が得られる可能性の高い部分問題を探索木の左側に集めることができる．

3.2 限定操作

PDF/IHS は，部分問題 π_a の下界値 $lb(\pi_a)$ を，式 (1) によって求める [15]．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．

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

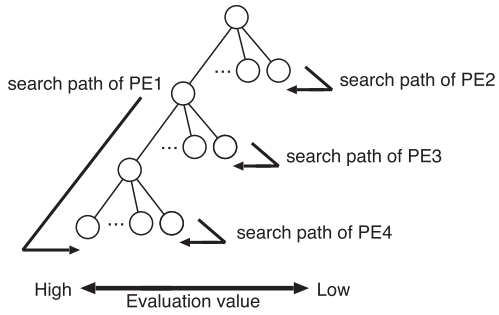


図 4 4PE による PDF/IHS
Fig. 4 PDF/IHS by 4 PEs.

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

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

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

ここで，m はタスク集合 $\mathbf{T}$ を実行するプロセッサの台数，$I(\pi_a)$ 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 [16] 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．

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

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

$$\overline{\tau}_j = lb_{cr}(\pi_a) - cp(j) - \min_{1 \leq i \leq m} t_{\pi_a}(P_i) \tag{7}$$

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

3.3 並列探索アルゴリズム

PDF/IHS は，階層的挟み撃ち探索を用いて木探索を行う．階層的挟み撃ち探索は，複数の PE（Processor Element）が階層的に左右から挟み撃つ形で探索する共有メモリ環境向けの並列探索手法である．本手法は，1 つのリーダ PE と複数のスレーブ PE が並列に深さ優先探索を行う．**図 4** に，階層的挟み撃ち探索における各 PE の振舞いを示す．図中の PE1 をリーダ PE，それ以外の PE をスレーブ PE とする．リーダ PE は，探索木左側から探索し，自身が探索中のノードから根ノードまでを結ぶ経路上のノードを待ち状態のスレーブ PE に割り当てる．スレーブ PE は，割り当てられたノードを根とする部分木を右側から探索する．このように，下界の評価が良いノードを多くの PE で探索しながら下界の評価の悪いノードを少数の PE で探索するため，下界の精度に影響されずに，早い段階で精度の

高いスケジューリングパターンを探索することができる．

本手法では，同一階層を左右から 2 つの PE で探索するため，同一ノードが 2 つの PE によって探索されることがある．これを，探索の重複と呼ぶ．同一ノードを複数回探索しても，得られる結果は変わらないため，探索の重複が生じたノードに対する 2 回目以降の探索結果は無駄になる．無駄な探索を防ぐために，探索の重複を検出するための操作をスレーブ PE が行う．スレーブ PE は，探索の重複を検出すると待ち状態になり，リーダ PE から処理の再割当てを受ける．

4. PDF/IHS の探索ノード数削減

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 [16], [17]．DF/IHS 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ス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 の分枝操作においては，割当て可能なタスクとレディ状態を割り当てるスケジュールパターンをすべて列挙して探索木を生成するため，探索する必要のない部分問題が生成されることがある．

筆者らは，DF/IHS で生成される部分問題を再定義することで探索する必要のない部分問題を判定できることを示し，その判定条件を用いた枝刈りの DF/IHS に対する有効性を示した [12]．本枝刈り手法は，DF/IHS に対してのみでなく，DF/IHS と同様の分枝規則を用いた並列探索アルゴリズムである PDF/IHS に対しても有効であると考えられる．そこで本論文では，PDF/IHS においても無駄な部分問題を削減する手法を提案する．

以降の節では，4.1 節で，文献 [12] で筆者らが提案した枝刈り可能な条件について述べ，4.2 節で，本論文で提案するアルゴリズムについて述べる．

4.1 再定義した部分問題を用いた枝刈り

DF/IHS の探索過程において生成される部分問題を再定義することで，探索する必要のない部分問題を判定することができる．再定義した部分問題のタスクグラフは，ダミータスクである T_0 以外で処理順序が確定したタスクを，割り当てられたプロセッサごとに 1 つのタスクとして扱う．**図 5** に，部分問題を再定義する例を示す．図中の網掛け部分は，複数のタスクを融合して生成したタスクである．図 5 のように，本手法では，必ず m 個のタスクが新たに生成される．新たに生成され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，$n+2$ から順に設定する．また，新たに生成されたタスクの先行タスクは T_0，後続タスクはプロセッサ P_i に最後に割り当てられたタスクとする．ただし，P_i に最後に割り当てられたタスクが ϕ の場合は，後続タスクを T_{n+1} とする．

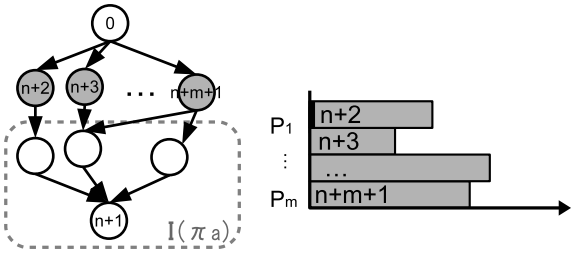


図 5 部分問題の例
Fig. 5 An example of subproblem.

上記のように再定義された部分問題を解くことで得られるスケジュールは，DF/IHS の分枝規則が元問題の制約条件を満たすようにタスクを割り当てることから，元問題の制約条件を満たしたスケジュールとなる．このため，DF/IHS において，部分問題を再定義された部分問題に置き換えて探索しても，元問題と同じ最適解を得ることができる．

ここで，部分問題 π_a のタスクグラフを G_{π_a}，G_{π_a} 中のタスク i の処理時間を $time_{\pi_a}(i)$ とおく．このとき，2 つの部分問題 π_a，π_b において，式 (9) の関係がすべて成り立つとき，部分問題 π_b を探索しても部分問題 π_a よりも短いスケジュール長を得ることはできない [12]．このため，DF/IHS の求解過程において π_b を枝刈りしても最適解を求めることができる．

$$
\begin{aligned}
&E(G_{\pi_a}) \subseteq E(G_{\pi_b}) \\
&V(G_{\pi_a}) \subseteq V(G_{\pi_b}) \\
&time_{\pi_a}(i) \leq time_{\pi_b}(i) \quad (n+2 \leq i \leq n+m+1) \quad (9)
\end{aligned}
$$

また，式 (9) を用いた枝刈りは，下界値を用いないため，探索の初期に精度の高い暫定解が得られる部分問題を枝刈りする場合がある．このような枝刈りによって暫定解の更新が遅れると，下界を用いた枝刈りの効率が低下し，探索時間が長くなる可能性がある．ただし，暫定解の精度が枝刈りに与える影響の大きな問題は，下界の精度が良く，DF/IHS の探索時間の短い問題がほとんどである．問題規模が大きな問題は一方で，下界の精度が悪く，下界を用いた枝刈りが起こりにくい問題では，DF/IHS の探索時間が長く，式 (9) を用いた枝刈りによって DF/IHS の探索時間が大きく短縮する [12]．DF/IHS は，問題規模の大きな問題ほど，下界の精度が悪く，探索時間が長くなる傾向がある．このため，式 (9) を用いた枝刈りは，問題規模の大きな問題ほど有効に働くと考えられる．

4.2 提案するアルゴリズム

PDF/IHS は DF/IHS と同じ分枝規則で探索木を生成す

るため，PDF/IHS でも 4.1 節の枝刈り判定を行うことができる．PDF/IHS では，リーダ PE とスレーブ PE の振舞いが異なるため，それぞれの PE でどのように枝刈り判定を行うかを検討する必要がある．ただし，PDF/IHS が生成する探索木から式 (9) の条件を満たすすべての部分問題を検出するためには，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．このようなデータ構造を用いることで，文献 [18] のように探索する部分問題数を大きく削減できるが，ハッシュテーブルを管理する処理を追加する必要が生じる．PDF/IHS 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においては高い効果を期待できるが，そうでない問題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 PDF/IHS 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，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．上記をふまえて，提案手法のリーダ PE とスレーブ PE の枝刈りの手順について述べる．

提案手法のリーダ PE は，PDF/IHS のリーダ PE が DF/IHS で探索する PE と同じ振舞いをすることから，文献 [12] と同様の枝刈りを行う．**図 7** に，リーダ PE による SP 値更新処理の疑似コードを示す．図 7 のコードは，式 (9) を満たす部分問題のうち，実行可能タスク情報と割当て時刻情報のみから判定できる部分問題を枝刈りする．**図 6** に図 7 のコードで削減できる部分問題の例を示す．図中の網掛けの領域はスケジュールが確定したタスクであり，点線の時刻が割当て時刻である．本例の部分問題は，割当て時刻に実行可能なタスクが T_1，T_2，T_3，T_4 であり，各タスクの処理時間には $time(4) < time(1) = time(2) < time(3)$ という関係があるとする．この場合，T_4 の処理時間は ϕ が割り当てられた時間よりも短く，ϕ を割り当てた時刻に T_4 を割り当てる方が短いスケジュール長を得られる．図 7 のコードを用いることで，図 6 のように不必要な ϕ が割り当てられる部分問題は，do-while 文のループによって，下界値計算の前に SP 値が更新される．これにより，探索する必要のない部分問題が枝刈りできる．

提案手法のスレーブ PE は，PDF/IHS のスレーブ PE が DF/IHS で探索する PE とは異なる振舞いをするため，図 7 と同じ手順で SP 値を更新することができない．本論文では，スレーブ PE に枝刈りの判定処理を加えるにあたって，PDF/IHS の利点を活かすために，スレーブ PE への割当てや部分問題の探索順序を変更しない．一方，式 (9) の条件で枝刈りできなかった部分問題は，下界値を用いて枝刈りできる可能性がある．このため，提案手法で枝刈りができないことが判明した部分問題には，下界を用いた限定操作を行う．**図 8** に，スレーブ PE による SP 値更新処理の疑似コードを示す．図 8 に示すように，スレーブ PE では，探索の重複を検出する操作を，提案手法の枝刈りの判定を行い，枝刈りできないことが判明した直後に行う．これは，提案手法の枝刈りの判定に必要な時間が，探索の重複を判定する操作に必要な時間よりも短いためであ

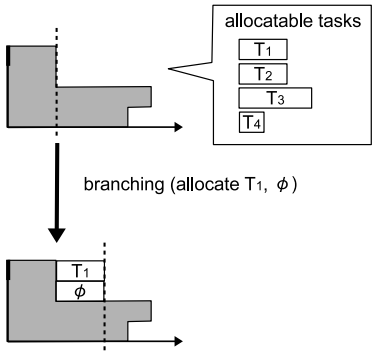


図 6 削減できる部分問題の例
Fig. 6 An example of unnecessary subproblem to search.

```
set_sp(){
  depth ← 探索中深さ
  sp[] ← 現在探索中の部分問題の SP
  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
  do{
    if(SP 値が更新できない)
      depth--;
    sp[] ←PDF/IHS のリーダ PE と同様に SP 値を更新;
  }while(φ が割り当てられている and 次の割当て時刻 ≥ MinTime)
  if(sp[] の下界 ≥ 暫定解)
    goto 枝刈り;
  goto 子ノード作成;
}
```

図 7 リーダ PE による SP 値更新処理の擬似コード
Fig. 7 Pseudo-code of calculating SP by leader PE.

```
set_sp(){
  depth ← 探索中深さ
  sp[] ← 現在探索中の部分問題の SP
  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
  do{
    if(SP 値が更新できない)
      depth--;
    sp[] ←PDF/IHS のスレーブ PE と同様に SP 値を更新;
  }while(φ が割り当てられている and 次の割当て時刻 ≥ MinTime)
  if(探索の重複を検出)
    goto 枝刈り;
  if(sp[] の下界 ≥ 暫定解)
    goto 枝刈り;
  goto 子ノード作成;
}
```

図 8 スレーブ PE による SP 値更新処理の擬似コード
Fig. 8 Pseudo-code of calculating SP by slave PE.

る．また，提案手法の枝刈りを先に行うことで，スレーブ PE の SP 値が探索木の左側に進むため，探索の重複を早期に検出できると期待できる．

5. 評価

提案手法の有効性を示すために，PDF/IHS と提案手法で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．本評価では，標準タスクグラフセット [19] の 50 タスクの問題のうち 60 パターンのタスクグラフに対して，タスクを割り当てるプロセッサの台数 m を 2，4，8，16 とする合計 240 問のタスクスケジューリング問題を求解する．下界値の計算には，3.2 節の式 (1) を用いる．評価環境は，CPU が Intel Xeon E5-2687W v2，メモリが 64 GB である．また，本評価では，PDF/IHS に対する提案手法の高速化率を式 (10) で定義する．

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

ただし，探索時間は，CP/MISF に必要な前処理の時間や，すべてのプロセッサが探索を終了するまでの待ち時間を含んだ時間である．

図 9, **図 10**, **図 11**, **図 12**, **図 13** に，1PE, 2PE, 4PE, 8PE，16PE で PDF/IHS を行った際の提案手法の高速化率を示す．グラフの横軸は PDF/IHS の実行時間，縦軸は PDF/IHS に対する提案手法の高速化率である．ただし，図 9 は，1PE での実行であるため，DF/IHS と同様の探索を行う．

測定結果より，すべての PE 数で，PDF/IHS の探索に時間がかかる問題ほど高速化率が向上する傾向が確認できた．PDF/IHS で探索時間が短い問題の高速化率が低いのは，探索時間が短い問題ほど PDF/IHS が効率良く枝刈りし，無駄な探索ノードが存在しないためである．無駄な探索ノードが存在しない問題は，提案手法を用いても，新たに枝刈りできるノードが存在しないため，探索ノード数が減少せずに高速化率が低下した．このような問題を探索実行前に判別することは難しいが，PDF/IHS の求解時間が 0.001 秒以下であるため，提案手法を用いても短時間で求解できる．このため，このような問題による高速化率の低下が提案手法の有効性に与える影響はほとんどないと考えられる．そこで，PDF/IHS の探索時間ごとに分けて高速化率を導出する．

表 1 に，PDF/IHS の探索時間ごとの提案手法の高速化率を示す．表中の SD は，高速化率の標準偏差である．表 1 からもグラフと同様に，PDF/IHS で求解に時間がかか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．PDF/IHS で求解に時間のかかる問題は，求解時間の削減率に対して実際に短縮される時間が大きくなる．このため，提案手法による枝刈りの効果は大きい．

表 1 で PDF/IHS の求解時間が 1 秒以上のときに PE 数

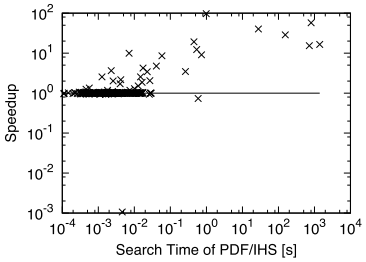


図 9 1PE の高速化率
Fig. 9 Speedup ratio of 1PE.

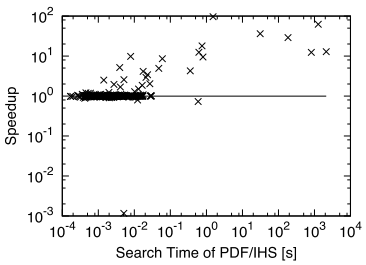


図 10 2PE の高速化率
Fig. 10 Speedup ratio of 2PE.

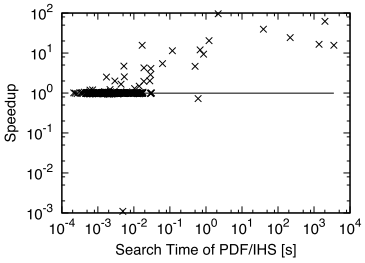


図 11 4PE の高速化率
Fig. 11 Speedup Ratio of 4PE.

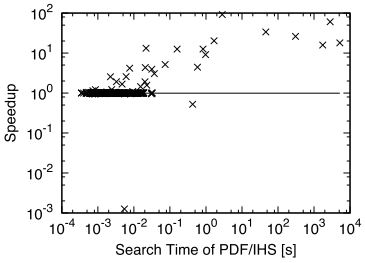


図 12 8PE の高速化率
Fig. 12 Speedup ratio of 8PE.

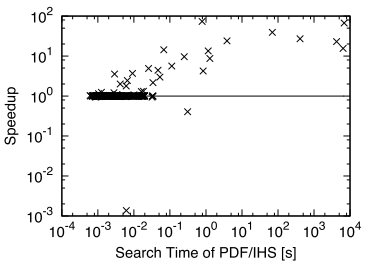


図 13 16PE の高速化率
Fig. 13 Speedup ratio of 16PE.

表 1 PDF/IHS の探索時間ごとの高速化率
Table 1 Speedup ratio for search time of PDF/IHS.

PE		count	ave	min	max	SD
1	$0.00 \leq time < 0.01$	200	1.083	0.941	9.715	1.372
	$0.01 \leq time < 1.00$	35	1.654	0.735	95.749	3.208
	$1.00 \leq time$	5	3.398	0.001	57.213	57.425
	total	240	1.180			2.202
2	$0.00 \leq time < 0.01$	199	1.087	0.901	9.594	1.388
	$0.01 \leq time < 1.00$	36	1.637	0.735	96.079	3.166
	$1.00 \leq time$	5	3.322	0.001	55.399	54.828
	total	240	1.183			2.201
4	$0.00 \leq time < 0.01$	193	1.066	0.937	15.387	1.341
	$0.01 \leq time < 1.00$	42	1.730	0.735	93.536	3.126
	$1.00 \leq time$	5	3.342	0.001	59.228	55.305
	total	240	1.188			2.215
8	$0.00 \leq time < 0.01$	190	1.058	0.962	13.741	1.331
	$0.01 \leq time < 1.00$	44	1.566	0.529	95.561	2.760
	$1.00 \leq time$	6	5.224	0.001	56.976	41.738
	total	240	1.183			2.172
16	$0.00 \leq time < 0.01$	188	1.029	0.730	13.543	1.346
	$0.01 \leq time < 1.00$	46	1.554	0.405	70.012	2.695
	$1.00 \leq time$	6	5.691	0.001	66.622	42.602
	total	240	1.162			2.183

表 2 提案手法を用いることによる探索ノード数の増減
Table 2 The number of subproblems using proposed method.

PE	increased	unchanged	decreased
1	2	175	63
2	17	149	74
4	31	129	80
8	42	108	90
16	79	68	93

が多くなるほど高速化率が向上する理由を調べるために，PDF/IHS と提案手法が探索した部分問題数を測定する．**表 2** に，提案手法を用いたことによって探索した部分問題数が増減した問題の数を示す．表 2 より，どの PE 数でも，部分問題数を削減できなかった問題数よりも削減できた問題数の方が多い．このため，提案手法は，多くの問題で探索する部分問題数を削減し，高い高速化率が得られたと考えられる．

また，PE 数の増加により，探索する部分問題数が増加した問題数が増えるのは，4.1 節で示したように，PDF/IHS が早い段階に探索する精度の良い解を提案手法が枝刈りしたためであると考えられる．下界を用いた枝刈りの精度が悪い状態で，最適解が得られるまでの間の探索を多くのプロセッサで行ったため，探索ノード数が増えたと考えられる．一方，PE 数の増加により，探索する部分問題数が減少した問題数が増えるのは，提案手法によって最適解が得られるまでの時間が短縮し，探索効率が向上したためであると考えられる．

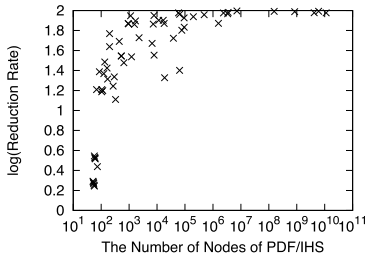


図 14 1PE の削減率
Fig. 14 Reduction rate of 1PE.

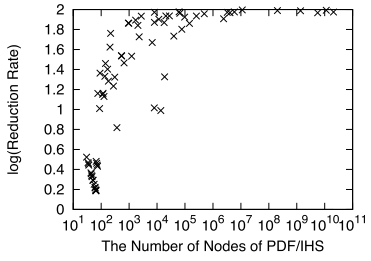


図 15 2PE の削減率
Fig. 15 Reduction rate of 2PE.

最後に，**図 14**，**図 15**，**図 16**，**図 17**，**図 18** に提案手法を用いることで得られる探索ノード数の削減率を示す．削減率は，式 (11) で定義する．

$$\text{削減率 [\%]} = \left(1 - \frac{\text{提案手法の探索ノード数}}{\text{PDF/IHS の探索ノード数}}\right) \times 100 \quad (11)$$

また，グラフの縦軸は削減率の常用対数である．図 14 か

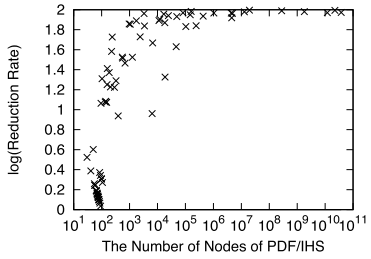


図 16 4PE の削減率
Fig. 16 Reduction rate of 4PE.

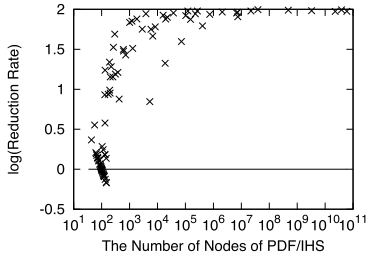


図 17 8PE の削減率
Fig. 17 Reduction rate of 8PE.

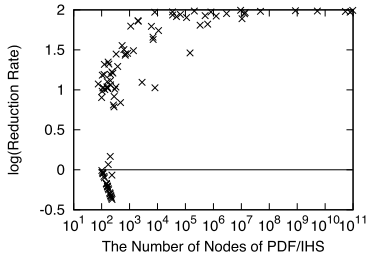


図 18 16PE の削減率
Fig. 18 Reduction rate of 16PE.

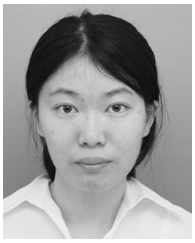
ら図 18 より，探索ノード数の削減率にも図 9 から図 13 と同様の傾向が表れていることが分かる．

6. おわりに

本論文では，PDF/IHS の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枝刈りを行う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，PDF/IHS に対して最大約 96 倍高速化することが確認できた．また，提案手法は，PE 数が多いほど高い高速化率が得られるため，大規模な問題ほど高い有効性が得られることが確認できた．

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] A.V. エイホ，R. セシィ，J.D. ウルマン，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 Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. Comput.*, Vol.C-33, No.11, pp.1023–1029 (1984).
[11] 笠原博徳, 伊藤 敦, 田中久充, 伊藤敬介：実行時間最小マルチプロセッサスケジューリング問題に対する並列最適化アルゴリズム，電子情報通信学会論文誌 D-I，Vol.J74-D1, No.11, pp.755–764 (1991).
[12] 中村あすか，前川仁孝：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，情報処理学会論文誌プログラミング（PRO），Vol.7, No.1, pp.1–9 (2014).
[13] Ramamoorthy, C., Chandy, K. and Gonzalez, M.: Optimal Scheduling Strategies in a Multiprocessor System, *IEEE Trans. Comput.*, Vol.C-21, No.2, pp.137–146 (1972).
[14] Coffman, E.: *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons (1976).
[15] 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).
[16] Fernández, E. and Bussell, B.: Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules, *IEEE Trans. Comput.*, Vol.C-22, No.8, pp.745–751 (1973).
[17] 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).
[18] 松瀬弘明，中村あすか，富永浩文，前川仁孝：タスクスケジューリング問題における DF/IHS 法のハッシュテーブルを用いた探索ノード数削減，情報処理学会第 78 回（平成 28 年）全国大会講演論文集，Vol.2016, No.1, pp.197–198 (2016).
[19] Standard Task Graph Set (STG), available from ⟨http://www.kasahara.elec.waseda.ac.jp/schedule/⟩ (accessed 2016-05-31).



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．

富永 浩文 （学生会員）

1984 年生．2007 年千葉工業大学情報科学部情報工学科卒業．2009 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．2010 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，GPU 等のアクセラレータを用いた各種アプリケーション高速化の研究に従事．

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．