情報処理学会論文誌　コンピューティングシステム　Vol.11　No.2　17–26 (Aug. 2018)

タスクスケジューリング問題における探索ノード数削減によるPDF/IHS法の高速化

松瀬 弘明[1,a)]　中村 あすか[2,b)]　富永 浩文[1,c)]　前川 仁孝[3,d)]

受付日 2017年12月27日, 採録日 2018年4月20日

概要：本論文では，タスクスケジューリング問題を高速に解くために，レディ状態を削減したPDF/IHS（Parallelized Depth First/Implicit Heuristic Search）法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法は，階層的挟み撃ち探索を用いた並列探索アルゴリズムであり，実行可能なタスクをPE（Processing Element）に割り当てる組合せをすべて列挙することで分枝限定法の探索木を生成する．このため，問題規模が大きくなるほど探索ノード数が膨大になり，探索ノード数の削減が必要となる．PDF/IHS法の探索ノード数を削減するために，不必要なレディ状態を割当てた部分問題を枝刈りする手法が提案されている．不必要なレディ状態を割り当てた部分問題を枝刈りする法は，割り当てたタスク情報のみを用いて枝刈りするため，探索する必要のない部分問題のすべてを枝刈りできない．そこで本論文では，探索する必要のないすべての部分問題を枝刈りするために，探索済みノード情報を利用する．評価の結果，探索済みノードを利用した枝刈りをすることで，スレッド数2のとき相乗平均約1.39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，PDF/IHS法，分枝限定法，並列処理

A Speedup Method for PDF/IHS by Reducing of Branching Nodes in Task Scheduling Problems

Hiroaki Matsuse[1,a)]　Asuka Nakamura[2,b)]　Hirobumi Tominaga[1,c)]　Yoshitaka Maekawa[3,d)]

Received: December 27, 2017, Accepted: April 20, 2018

Abstract: This paper proposes a reduction algorithm of branching nodes for the PDF/IHS (Parallelized Depth First/Implicit Heuristic Search) using allocations of idle tasks method for solving task scheduling problems fast. The PDF/IHS is parallel search method using HPAS (Hierarchical Pincers Attack Search). The search method creates the search tree by enumerating all combinations of the allocatable tasks to PE (Processing Element). Therefore, it is necessary to reduce the number of search nodes to solve large-scale problems having many search nodes. We proposed the bounding method of the number of search nodes allocating unnecessary idle tasks to reduce search nodes of PDF/IHS. This method can not cut all of subproblems which do not need to search, because this method cuts subproblems using information of allocated tasks. Therefore, it is necessary to reduce all unnecessary nodes to perform further speedup. The proposed method reduces unnecessary subproblems by using searched subproblems. As a result of evaluation, the speedup ratio of the proposed method with the PDF/IHS is about 1.39 times on geometric average at 2 threads.

Keywords: task scheduling, PDF/IHS mehod, branch and bound method, parallel processing

1. はじめに

タスクスケジューリング問題は，任意の先行制約を持つタスク集合を複数台のプロセッサで並列実行するスケ

[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] 千葉工業大学専門研究員
Chiba Institute of Technology, Researcher, Narashino, Chiba 275–0016, Japan
[3] 千葉工業大学情報科学部情報工学科
Department of Computer Science, Faculty of Information and Computer Science, Chiba Institute of Technology, Narashino, Chiba 275–0016, Japan

[a)] matsuse@mae.cs.it-chiba.ac.jp
[b)] nakamura@mae.cs.it-chiba.ac.jp
[c)] tominaga@mae.cs.it-chiba.ac.jp
[d)] maekawa@mae.cs.it-chiba.ac.jp

ジュールのうち，スケジュール長が最も短いスケジュールを求める問題である [1], [2]．本問題は，タスクごとに処理時間の異なる問題や，プロセッサ集合がヘテロジニアスで処理能力にばらつきがある問題，先行制約を持つタスクが異なるプロセッサに割り当てられた際に通信遅延が生じる問題などと，用途に合わせたモデル化が行われている．一方，タスクスケジューリング問題は，多くの組合せ最適化問題と同様に問題サイズや問題の複雑さに応じて解の個数が指数関数的に増加するため，最適解の求解に必要な時間も問題サイズに応じて指数関数的に長くなる．また，スケジューリング時間が長くなると，スケジューリングを行うことによってスケジュール長を短縮できたとしても，スケジュールを含めた実行時間全体が長くなり，スケジューリングを行う意義がなくなる．このような理由から，タスクスケジューリング問題の求解においては，近似解法や時間打ち切りを用いた厳密解法を用いて，高速に得ることが可能なスケジュールを解とすることが一般的である [3]．

近似解法は，スケジューリング結果を実用的な時間で求めることができる．また，多くの近似解法は，スケジューリング対象に合わせたヒューリスティックスを用いてアルゴリズムが設計されているため，高い精度の解を得ることができる．さらに，動的にスケジュールを構築することで，実行時間が不確定なタスクや割込み処理などにも対応することができる [4], [5]．一方で，多少の時間をかけてでも静的に最適解を導出した方が結果的によくなるという場面も存在する．これは，得られたスケジュール結果の利用期間や利用回数が長いほど，近似解と最適解の誤差が小さいにもかかわらず，実際の損失が大きくなるためである．たとえば，クラウド型の計算機環境などでタスクの処理時間が数日単位となる場合は，単位時間が長い問題としてモデル化することができる．他にも，近似解法に対する評価を行う目的において，近似解と最適解の誤差を算出する必要が生じ，厳密解法による求解が必要となる場合がある．このように，スケジューリング問題の最適解を求める必要性は依存として高く，スケジューリング問題の最適解求解の高速化が求められている．

本研究で扱うスケジューリング問題は，任意の処理時間を持つタスク集合を，処理能力が等しい複数のプロセッサで実行する問題であり，プロセッサ間のデータ転送時間を 0 と近似する．このような問題は，より複雑な条件を持つ問題の基礎研究として位置づけられる問題であるが，マルチコアプロセッサなどの SMP 型の並列化環境においてプロセッサ間の通信時間が無視できるほど長い処理時間のタスクを扱う際などに，利用することができる．また，共有メモリ環境において，先行制約を持つタスク間でデータの局所性が極端に大きい場合や極端に小さい場合も，どのプロセッサでタスクを実行してもデータ転送時間を一定と見なせるためデータ転送時間を 0 と見なすことができる．本問題の最適解を求解する有効な手法の 1 つとして，DF/IHS（Depth First/ Implicit Heuristic Search）法 [6] や，その並列探索アルゴリズムである PDF/IHS（Parallelized DF/IHS）法 [7] がある．

PDF/IHS 法は，探索ノード数が多くなるほど求解に時間を要するため，未割当てタスクのスケジュール長に対する下界値を算出し，限定操作を行うことで探索ノード数の削減を行う．一方で，PDF/IHS 法の分枝操作では，割当て可能なタスクをすべて列挙することで部分問題を生成する．割当て可能なタスクには，プロセッサが何も処理を行わない状態であるレディ状態も含まれるため，不必要にレディ状態を割り当てるような探索する必要のない部分問題も探索ノードとして生成する．このため，不必要にレディ状態を割り当てる部分問題の生成を抑制する手法が提案されている [8], [9]．以降では，不必要にレディ状態を割り当てる部分問題の生成を抑制する手法を PDF/IHS/idle 法と呼ぶ．

PDF/IHS/idle 法は，レディ状態を割り当てる部分問題を生成する際に，その状態を割り当てた時間内に実行を完了できるタスクがある場合を検出することで，枝刈りを行う．つまり，PDF/IHS/idle 法は，文献 [8] で示された枝刈可能な部分問題の多くを従来の PDF/IHS 法同様に枝刈りせずに探索する．このため，文献 [8] で示された探索する必要のない部分問題を PDF/IHS 法においてすべて削減した際の有効性は不明瞭である．

そこで，本論文では，文献 [8] で示された探索する必要のない部分問題をすべて枝刈りする手法である PDF/IHS/hash 法を提案し，その有効性を評価する．文献 [8] で示された探索する必要のない部分問題を枝刈りするためには，探索済みの部分問題の情報を記憶し，参照する操作を追加する必要がある．この操作をすべての部分問題に対して行うと追加する処理のオーバヘッドが大きくなるため，提案手法では，ハッシュを用いて参照する部分問題数の削減を行う．

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，n 個のタスクからなるタスク集合 $T = \{T_1, T_2, \cdots, T_n\}$ を能力の等しい m 台のプロセッサ集合 $PE = \{PE_1, PE_2, \cdots, PE_m\}$ 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いスケジュールを求める問題である [1]．本論文で扱う問題は，粒度の大きい処理をタスクとしてモデル化した問題であり，各プロセッサのデータ転送に必要な時間がタスクの処理時間に対して無視できるほど小さいため，データ転送時間を 0 とする．また，処理割り込みは起こらないとする．

タスクスケジューリング問題は，タスクをノード，先行制約をエッジとしたタスクグラフと呼ばれる DAG（無サイクル有効グラフ）で表すことができる．**図 1** に，タスクグラフの例を示す．図 1 では，ノード内の数値 i がタスク

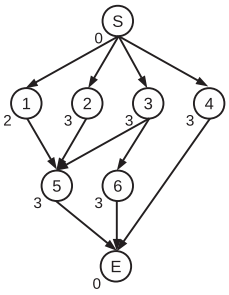

図 1　タスクグラフの例
Fig. 1　An example of task graph.

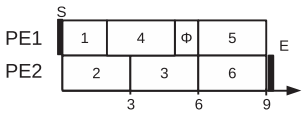

図 2　スケジュール結果の例
Fig. 2　An example of schedule.

番号，ノード左下の数値がタスク T_i の処理時間 $time(i)$，矢印が先行制約を表す．入口ノード S および出口ノード E は処理時間 0 のダミータスクである．このため，本例は，タスク数 $n = 6$ のタスク集合 $T = \{1, 2, 3, 4, 5, 6\}$ からなるタスクスケジューリング問題を表す．

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 中のタスクをプロセッサ台数 $m = 2$ で実行するスケジュールの例を示す．図中の太線は処理時間が 0 であるダミータスクの割当てを表し，ϕ は PE にレディ状態が割り当てられたことを表す．本例では，時刻が 5 から 6 までの間，PE_1 が図 1 中のタスクを実行せずにレディ状態となる．本例は，図 1 のすべての制約を満たし，かつ，スケジュール長が 9 のスケジュールである．

3. PDF/IHS 法

PDF/IHS 法は，CP/MISF 法 [6] のヒューリステックを分枝限定法に取り入れた手法である DF/IHS 法を複数のスレッドで挟み撃つ形で並列探索するアルゴリズムである [7]．分枝限定法は，分枝操作で探索木を生成し，限定操作で探索木の子ノードを枝刈りする．以降の節では，PDF/IHS 法の分枝操作と限定操作，および，並列探索について述べ，そのうえで PDF/IHS 法の探索ノード数を削減する手法である PDF/IHS/idle 法について述べる．

3.1　分枝操作

DF/IHS 法および PDF/IHS 法の分枝操作は，スケジュールが未設定となる最も早い時刻に対して，どのタスクを割り当てるかを列挙することで探索木を生成する．**図 3** に，図 1 のスケジュールを求める際に PDF/IHS 法が生成する探索木を示す．図 3 の R 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割当て可能なタスクの集合を表す．たとえば，本例の深さ 0 のノードでは，ダミータスクであるタスク S のみが割当て済みであるため，スケジュールが未設定となる最も早い時刻は時刻 0 であり，$R = \{1, 2, 3, 4, \phi\}$ となる．R にレディ状態 ϕ を含めるのは，レディ状態を含めたスケジュールが最適解になる場合があるためである [10], [11]．このため，根ノードでは，分枝操作で 10 個の子ノードを生成する．

また，図 3 の SP は，R 内の何番目のタスクを割り当てるかを示すポインタである．同じ SP を持つ部分問題は存在しないため，部分問題を $SP = [(SP_0, SP_1, \cdots, SP_d)]$ と表すことができる．例として，$SP = [(1, 2), (1), (1), (1), (1)]$ が示すノードは，図中で探索木の左下の深さ $d = 5$ のノードである．本手法は，精度の高い暫定解を優先して早い段階に探索するために，R をヒューリスティックスに基づく近似解法である CP/MISF 法 [6] のプライオリティリストの順番に並べ，SP を辞書的順序で指定する．

PDF/IHS 法の分枝操作は，R を PE に割り当てるすべての組合せを部分問題として生成するため，探索規模が大きくなるほど探索ノード数が膨大になり求解に時間を要する．このため，PDF/IHS 法は，探索ノード数を削減するために，限定操作で探索木の一部を枝刈りする．

3.2　下界を用いた限定操作

分枝限定法の限定操作では，分枝操作で生成した部分問題の下界値を算出し，下界値の値が暫定解の値以上である探索ノードを枝刈りする．DF/IHS 法および PDF/IHS 法でも同様に，下界値を用いて限定操作を行う．部分問題 π_a の下界値 $lb(\pi_a)$ は，式 (1) を利用して算出する [12]．

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

ここで，$lb_{cr}(\pi_a), lb_{div}(\pi_a), lb_{hu}(\pi_a)$ は，式 (2)，式 (3)，式 (4) からそれぞれ求める．

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

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

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

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，$t_{\pi_a}(PE_i)$ はノード π_a においてプロセッサ PE_i のスケジュールがまだ決まっていない時刻を表す．$lb_{cr}(\pi_a)$ はクリティカルパスを利用した下界値であり，$lb_{div}(\pi_a)$ は未割当てタスクの処理時間の和を利用した下界値である．また，$lb_{hu}(\pi_a)$ は，Fernández によって拡張された Hu の下界 [13], [14] である．式中の $q(\pi_a)$ は，スケジュール長がクリティカルパス長に比べてどの程度長くなるかを算出したヒューリスティックスであり，式 (5) から求めることができる．

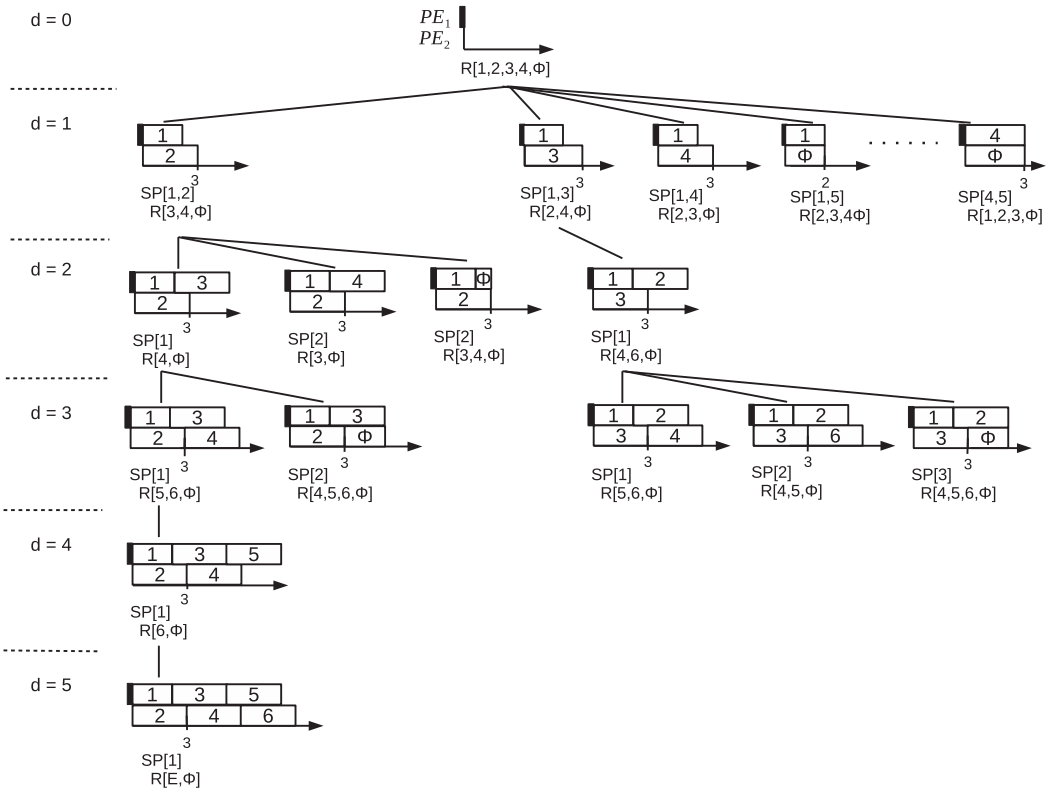

図 3　探索木の例
Fig. 3　An example of search tree.

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

式 (5) では，負荷密度関数 $F(\overline{\tau}, t)$ を用いることで，プロセッサ台数が ∞ と仮定した際に各時刻に実行可能なタスクのうち，プロセッサ台数が m のときに同時に実行できない処理がどの程度存在するかを算出する．負荷密度関数 $F(\overline{\tau}, t)$ は，式 (6) から求める．

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

$\overline{\tau}_j$ は，未割当てタスクの集合 $I(\pi_a)$ のみをスケジューリングする際の最遅開始時刻であり，式 (7) から求める．

$$\overline{\tau}_j = lb_{cr}(\pi_a) - cp(j) - \min_{1 \le i \le m} t_{\pi_a(PE_i)} \quad (7)$$

また，式 (6) 中の $f(\overline{\tau}_j, t)$ は，式 (8) から求める．

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

$lb_{hu}(\pi_a)$ を求めるためには，式 (4)〜式 (8) の計算を行う必要がある．ただし，式 (1) の計算は，$lb(\pi_a) \ge$ (暫定解) となることが確定した時点で部分問題 π_a を枝刈りできると判断できる．このため，式 (1) を算出する際に，$lb_{cr}(\pi_a)$，$lb_{div}(\pi_a)$，$lb_{hu}(\pi_a)$ の順に算出することで，下界値を用いた限定操作を高速に行うことができる．

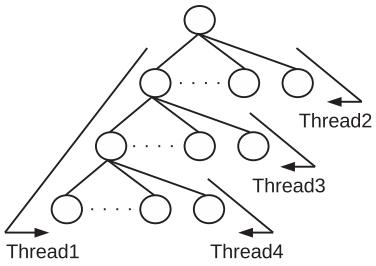

図 4　4 スレッドによる PDF/IHS 法
Fig. 4　PDF/IHS by 4 thtrads.

3.3　並列探索

PDF/IHS 法は，DF/IHS 法で生成した探索木に対して複数のスレッドを用いて挟み撃つ形で探索する．本手法は，最適解が得られる可能性の高い部分問題を多数のスレッドで探索し，そうでない部分問題を少数のスレッドで探索する．**図 4** に，4 スレッドによる PDF/IHS 法の探索例を示す．図 4 の Thread1 をマスタスレッド，他のスレッドをスレーブスレッドとし，各深さで挟み撃つ形で探索する．マスタスレッドは，左側から葉ノードまで深さ優先探索を行い，待ち状態のスレーブスレッドへ各深さのノードを動的に割り当てる．スレーブスレッドは，割り当てられたノードを根として右側から深さ優先探索を行う．PDF/IHS 法では，複数のスレッドが左右から探索するため，マスタスレッドとスレーブスレッドが同じノードを探索する場合がある．このような状態は探索の重複と呼ばれており，同じ

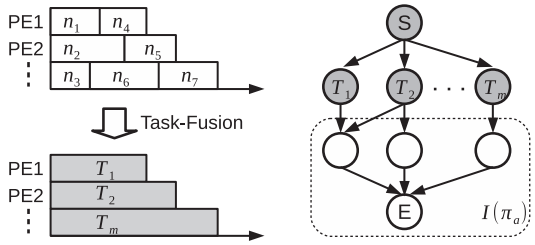

図 5　タスク融合の例
Fig. 5　An example of task-fusion.

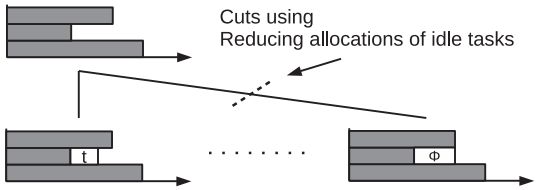

図 6　不要にレディ状態を割り当てた部分問題を削除する例
Fig. 6　A example of reducing allocations of idle tasks.

ノードを 2 回以上探索しても得られる結果が変わらないため，そのまま探索を続けると無駄な時間を要する．このため，スレーブスレッドは，探索の重複を検出すると，マスタスレッドに通知を行い，マスタスレッドから新しいノードの割当てを受ける．

3.4　PDF/IHS/idle 法による探索ノードの削減

PDF/IHS/idle 法は，部分問題ごとに割り当てたタスクを融合することで，不要なレディ状態を割り当てた部分問題を判定する [8]．**図 5** に，PDF/IHS/idle 法によるタスク融合の例を示す．図 5 中のガントチャートは，割当て済みの部分問題であり，灰色のノードがタスク融合によって生成されたタスクである．本例のように，タスク融合は，同一のプロセッサに割り当てられたタスクどうしで融合し，割当て済みタスクをプロセッサ数 m と同じ個数のタスクに置き換える．

PDF/IHS/idle 法では，部分問題生成時に割り当てるタスクを比較することで枝刈りする．**図 6** に，PDF/IHS/idle 法で枝刈りする部分問題の例を示す．本手法は，図 6 のように，部分問題生成時に割り当てたタスク t とレディ状態の処理時間を比較する．本例では，割り当てたタスク t がレディ状態の処理時間より短いため，レディ状態を割り当てた部分問題を枝刈りする．PDF/IHS/idle 法は，割当てタスクのみを比較するため，同じ部分問題から生成される部分問題のうち探索する必要のないレディ状態を割り当てた部分問題を枝刈りする．

4.　ハッシュを用いた PDF/IHS 法の探索ノード数の削減

文献 [9] では，探索済みの部分問題の割当て済みタスクの情報を参照することで，探索する必要のない部分問題を

図 7　部分問題を簡単化した例
Fig. 7　An example of simplified subproblem.

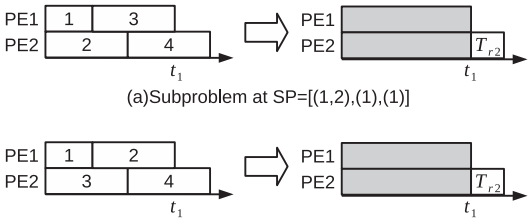

図 8　削減可能な部分問題の例
Fig. 8　An example of reduced subproblem.

枝刈りできることが示されている．一方で，枝刈りできると示された部分問題が探索木上にどの程度存在し，それを削減することでどの程度の削減率が得られるのかが示されていない．そこで，本論文では，探索済みノードを用いて文献 [9] で探索する必要がないと示された部分問題をすべて枝刈りした際の効果を測定し，その有効性を評価する．

以下では，文献 [9] で探索する必要がないと示された部分問題をすべて枝刈りする手法を PDF/IHS/hash 法と呼ぶ．

4.1　枝刈りする部分問題

文献 [9] では，図 5 の簡単化を行った部分問題に対して式 (9)，式 (10)，式 (11) を用いることで，探索する必要のない部分問題を検出できることが示されている．

$$I(\pi_a) \subseteq I(\pi_b) \tag{9}$$

$$\min_{1\le i\le m} t_{\pi_a}(PE_i) \le \min_{1\le i\le m} t_{\pi_b}(PE_i) \tag{10}$$

$$time_{\pi_a}(ri) \le time_{\pi_b}(ri) \qquad (2 \le i \le m) \tag{11}$$

ここで，$time_{\pi_a}(ri)$ は，簡単化を行った部分問題 π_a のタスク T_{ri} の処理時間を表す．タスク T_{ri} の例を**図 7** に示す．図 7 では，スケジュールが未確定になる時刻が早い順に PE 番号を並べ替え，各プロセッサの割当て時刻以降の処理を新たなタスクとして定義してている．PE_1 は，割当て時刻の段階でスケジュールが未確定となるため，タスク T_{r1} が存在しない．ただし，部分問題 π_a において，すべてのプロセッサで割当て時刻にスケジュールが未確定になる場合は，$time_{\pi_a}(ri) = 0$ となる．部分問題ごとに割当て時刻や割手済みタスクなどが異なるため，$time_{\pi_a}(ri) = time_{\pi_b}(ri)$ とならない場合もある．

図 8 に，図 3 の部分問題のうち，式 (9)，式 (10)，式 (11) の条件をすべて満たす部分問題の例を示す．図 8 中の (a) が π_a，(b) が π_b であるとする．図 8 より，部分問題 π_a と π_b の未割当てタスクの集合は同じであるため，式 (9) を満たすことが分かる．同様に，式 (10) で示された部

分問題 π_a と π_b の割当て時刻や，式 (11) で示された割当て時刻以降の時刻でスケジュールが確定している時刻においても，それぞれの式を満たしていることが分かる．本例では，簡略化した後の部分問題はどちらも同じ問題であるため，どちらか片方の部分問題を探索することで，もう一方の部分問題でも同じスケジュール長のスケジュールを求めることが可能である．このため，どちらか片方の問題を枝刈りすることが可能である．また，簡略化した後の部分問題が同じ同じ問題にならない場合では，式 (9)，式 (10)，式 (11) のすべてを満たす部分門題 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のスケジュールを得ることができない．このように，式 (9)，式 (10)，式 (11) をすべて満たす部分問題 π_a，π_b があるとき，部分問題 π_b を枝刈りすることができる．

4.2 PDF/IHS/hash 法

PDF/IHS/hash 法は，式 (9)，式 (10)，式 (11) の条件をすべて満たす部分問題を枝刈りする．式 (9)，式 (10)，式 (11) の判定を行うためには，探索済み部分問題の割当て時刻や，$T_{r2}, T_{r3}, \cdots T_{rm}$ の処理時間，および，未割当てタスクのタスク番号を参照する必要がある．PDF/IHS の探索ノード数は問題規模に応じて大きくなるため，これらの情報をすべての探索済みノードの分だけ記憶することは，メモリ使用量的に現実的でない．また，同様に，特定の探索ノードに対して枝刈りができるかどうかを判定するために，それまでの探索過程で生成したすべての部分問題に対して式 (9)，式 (10)，式 (11) の比較を行うと，追加する処理のオーバヘッドが膨大になる．このため，本論文で提案する PDF/IHS/hash 法では，式 (9)，式 (10)，式 (11) の条件を満たす部分問題に対する枝刈りの精度を低下させることなく，記憶する探索ノード数および判定時に参照する探索ノード数を削減する．

PDF/IHS/hash 法は，記憶する探索ノード数を削減するために，式 (9) の包含関係と式 (10) の大小関係を利用する．たとえば，式 (9) が成り立つ部分問題 π_a と π_b が探索済みである際に，新たに探索した部分問題 π_c に対して $I(\pi_c) \subseteq I(\pi_b)$ が成り立つ場合，部分問題 π_c は枝刈り可能な部分問題であると判断できる．このような場合，式 (9) より $I(\pi_c) \subseteq I(\pi_a)$ も成り立つ．つまり，π_b を記憶せずに π_a のみを記憶していた場合でも，π_c は枝刈りできると判断することが可能であり，枝刈りの精度が低下しない．また，下界値で枝刈できる部分問題 π_d に対して式 (9)，式 (10)，式 (11) の条件を用いて枝刈り可能な部分問題 π_e が存在する場合，式 (10) より $lb_{cr}(\pi_d) \leq lb_{cr}(\pi_e)$ となる．このため，π_e は式 (9)，式 (10)，式 (11) の判定を行わない場合でも下界値を用いた限定操作で枝刈りできる．上記をふまえて，PDF/IHS/hash 法では，探索済みノードのうち，下界値を用いた限定操作や式 (9)，式 (10)，式 (11) を用いて枝刈りできると判断された探索ノードを記憶しないようにすることで，メモリ使用量や参照する探索ノード数を削減する．

```
cut(){
  /* PDF/IHS 法の枝刈り */
  if(暫定解 ≤ lb(π_b)){
    部分問題 π_b を枝刈り;
    return;
  }
  /* 提案手法で追加する枝刈り */
  key_{π_b} ← π_b の割当てタスク情報
  for(i = 0; key[i] ≠ NULL; i++){
    if(key[i] − (key[i]|key_{π_b}) == 0){
      while(T_time.T_time[j] ≠ NULL)
        if(∀_j T_time.T_time[j] ≤ t_{π_b(T_rj)}){
          部分問題 π_b を枝刈;
          return;
        }
        T_time.next ← T_time.next.next;
      }
    }
  }
  部分問題 π_b の情報をハッシュに格納;
}
```

図 9　枝刈り判定の疑似コード
Fig. 9　Pseudocode of reduction search nodes.

他にも，PDF/IHS/hash 法は，枝刈りできるかを判定する際に参照する探索ノード数を削減するために，探索済みノードの情報を格納するデータ構造としてハッシュを利用し，式 (9) による包含判定を実行する回数を削減する．本論文では，ハッシュの単純な実装方法であるチェイン法を採用し，ハッシュキーに割当て済みタスク情報を格納したビット配列を用いる．**図 9** に，本実装で枝刈りする疑似コードを示す．図 9 中の $key[]$，key_{π_b} は，部分問題の割当て済みタスク情報を格納したビット配列である．ビット配列は，タスク i が割当て済みであれば i ビット目を 1，そうでないなら 0 とする．T_time は，部分問題情報を格納するための連結リストである．$T\_time.T\_time[i]$ に $t_{\pi_b}(T_{ri})$ を格納し，$T\_time.next$ に連結するリスト要素を格納する．このように，提案手法で追加した枝刈りは，図 9 のように，下界値を用いず割当て済みタスク情報を利用して判定する．

PDF/IHS/hash 法は，PDF/IHS 法や PDF/IHS/idle 法よりも多くの探索ノードを枝刈りできる．一方で，PDF/IHS/hash 法で枝刈りする探索ノードは，文献 [9] において，枝刈しても探索結果の最適性が保障される探索ノードである．このため，PDF/IHS/hash 法による枝刈りを行っても，タスクスケジューリング問題の厳密解を求解することが可能である．

5. 評価

文献 [8] で示された探索する必要のない部分問題を，すべて枝刈りすることによる効果を調べるために，提案手法である PDF/IHS/hash 手法の有効性を評価し，PDF/IHS 法と提案手法の探索ノード数および探索時間を測定する．本評価は，標準タスクグラフセット [15] のタスク数 50 の問題 60 問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 と変化させた，計 240 問を解く．評価環境は，CPU がハイパースレッディングで 16 スレッド実行可能な 8 コア CPU である Intel Xeon E5-2687W v2 を 2 個，メモリが 64 GB である．

以降の節では，探索ノード数と実行時間を測定し，評価する．

5.1 探索ノード数の測定

本評価では，探索済みノードを利用した枝刈りの有効性を確認するために，PDF/IHS 法と PDF/IHS/hash 法の探索ノード数を測定し，式 (12) から削減率を求める．

$$\text{削減率 [\%]} = \left(1 - \frac{\text{PDF/IHS/hash 法の探索ノード数}}{\text{PDF/IHS 法の探索ノード数}}\right) \times 100 \tag{12}$$

図 10 に，スレッド数を 1, 2, 4, 8, 16 と変化させて測定した探索ノード数の削減率を示す．図 10 の横軸は PDF/IHS 法の探索ノード数，縦軸は削減率である．図 10 より，探索ノード数が多い問題ほど削減率が高いことが確認できる．

高い削減率を得られた問題は，PDF/IHS 法が探索する必要のない部分問題を枝刈りせずに探索した問題である．つまり，これらの問題では，従来の PDF/IHS 法における下界を用いた限定操作において，探索する必要のない部分問題の多くを枝刈りできていないと言い換えることができる．PDF/IHS 法では，多くの分枝限定法を用いたアルゴリズムと同様に，探索ノード数が多くなるほど探索時間も長くなる．このため，PDF/IHS 法の探索に時間のかかる問題ほど，提案手法を用いることで高い高速化を見込むことができると考えられる．

一方，高い削減率を得られなかった問題は，PDF/IHS 法の下界を用いた限定操作が効率良く働いた問題である．これらの問題は，図 10 より PDF/IHS 法による探索ノード数が少ないことから，PDF/IHS 法の下界を用いた枝刈りが探索初期から有効に働いた問題であると考えられる．このため，提案手法を用いても新たに枝刈り可能な探索ノードが少なく，高い削減率が得られなかったと考えられる．また，探索ノード数が増加した問題が，少数であるが，存在する．これらの問題で探索ノード数が増加したのは，提案手法が初期解よりも精度の良い暫定解が得られる部分問題を枝刈りしたためである．これにより，暫定解の精度が悪くなり下界値を用いた枝刈りの効率が低下するため，探索ノード数が増加した．ただし，図 10 より，増加した探索ノード数および増加後の探索ノード数は少ないため，このような問題においても提案手法は高速に求解できると考えられる．

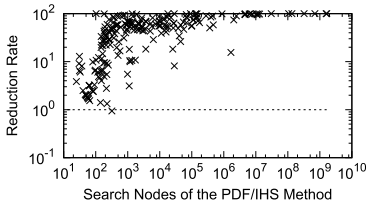

図 10　探索ノード数の削減率
Fig. 10　Reduction ratio of search nodes.

5.2 実行時間の測定

5.1 節において提案手法を用いることで多くのノードを削減できることが確認できたが，ハッシュを用いた枝刈りの判定には時間がかかる．このため，本評価では，PDF/IHS 法と提案手法の実行時間を測定し，式 (13) から高速化率を求める．

$$\text{高速化率 [倍]} = \frac{\text{PDF/IHS 法の実行時間}}{\text{PDF/IHS/hash 法の実行時間}} \tag{13}$$

図 11，**図 12**，**図 13**，**図 14**，**図 15** に，スレッド数 1,

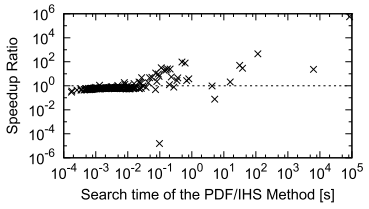

図 11　1 スレッドの高速化率
Fig. 11　Speedup ratio of 1 threads.

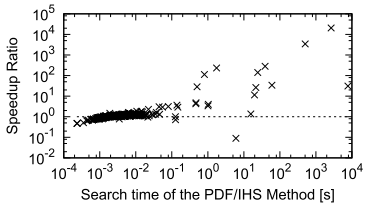

図 12　2 スレッドの高速化率
Fig. 12　Speedup ratio of 2 threads.

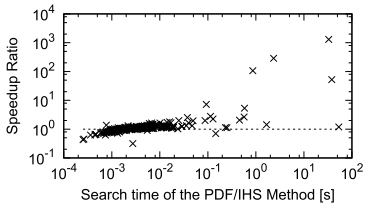

図 13　4 スレッドの高速化率
Fig. 13　Speedup ratio of 4 threads.

表 1　PDF/IHS 法の実行時間別の提案手法の高速化率
Table 1　Speedup of proposed method for search time of PDF/IHS.

Thread	Range of search time t [s]	Number of problems	ave.	min.	max.
1	$t < 0.01$	183	0.58	0.29	1.84
	$0.01 \leq t < 1.00$	43	1.17	0.48	93.98
	$1.00 \leq t$	14	54.37	0.07	553132.13
2	$t < 0.01$	170	0.78	0.47	1.61
	$0.01 \leq t < 1.00$	54	11.20	0.71	112.59
	$1.00 \leq t$	13	10.15	0.08	232.93
4	$t < 0.01$	174	0.93	0.31	1.64
	$0.01 \leq t < 1.00$	58	1.32	0.70	106.64
	$1.00 \leq t$	6	24.82	1.19	1304.77
8	$t < 0.01$	177	0.92	0.38	1.85
	$0.01 \leq t < 1.00$	50	1.47	0.51	6.99
	$1.00 \leq t$	10	40.81	5.55	1187.37
16	$t < 0.01$	171	1.04	0.65	2.03
	$0.01 \leq t < 1.00$	55	1.30	0.71	5.32
	$1.00 \leq t$	11	22.11	2.22	317.96

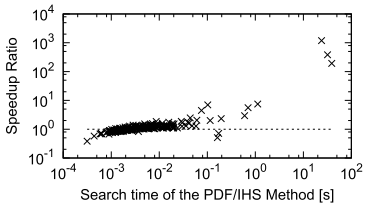

図 14　8 スレッドの高速化率
Fig. 14　Speedup ratio of 8 threads.

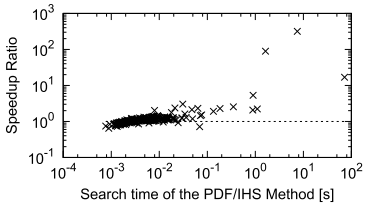

図 15　16 スレッドの高速化率
Fig. 15　Speedup ratio of 16 threads.

2，4，8，16 と変化させた PDF/IHS 法に対する提案手法の高速化率を示す．図 11，図 12，図 13，図 14，図 15 の横軸は PDF/IHS 法の実行時間，縦軸は高速化率である．図 11 から図 15 より，PDF/IHS 法の求解に時間を要する問題ほどハッシュを用いた枝刈り手法が有効に働くことが確認できる．これは，図 10 と同じ傾向であり，探索ノード数を削減するほど高速化が見込めることが確認できる．また，高速化率の相乗平均が最大となるのは，スレッド数が 2 のときであり，約 1.39 倍である．

表 1 に，PDF/IHS 法の求解時間ごとの提案手法の高速化率の相乗平均，最小値，最大値を示す．表 1 より，求解時間が短い問題ほど高速化率の相乗平均が低く，求解時間が長い問題ほど高速化率の相乗平均が高いことが分かる．本研究で実装したハッシュは単純な構造であるが，PDF/IHS 法が求解に時間を要する問題では，提案手法の枝刈りが有効に働くことで高い高速化率が得られたことが分かる．

一方，表 1 から，PDF/IHS 法が短い時間で求解できる問題において高い高速化率が得られていないことが分かる．これらの問題は，PDF/IHS 法で多くの問題を枝刈りした問題であり，PDF/IHS/hash 法を用いても新たに枝刈りできる部分問題が少ない．このため，追加した処理によるオーバヘッドの分だけ処理時間が増加したと考えらえる．また，提案手法で追加した処理は，ハッシュ内のデータを参照し，チェイン法で連結された 1 つ 1 つのデータに対して枝刈り条件を満たしているかの判定を行う．ハッシュに連結するリストが長いほど枝刈りの判定に時間を要するため，高速化率が低下したと考えられる．

また，スレッド数が 1 または 2 においては，PDF/IHS 法の実行時間が 1.00 秒以上の問題においても高い高速化率を得られない問題が存在する．スレッド数 1 で最も低い高速化率である 0.07 倍の問題は探索ノードの削減率が 73%であり，スレッド数 2 で最も低い高速化率である 0.08 倍の問題は探索ノードの削減数が 80%である．これらの問題においても高い削減率が得られていることから，ハッシュを利用した枝刈りの判定に時間を要したことが原因で高速化率が低下したことが分かる．

5.3　PDF/IHS/idle 法に対する有効性の評価

本評価は，容易に検出可能な部分問題のみを枝刈りする PDF/IHS/idle 法に対する，式 (9)，式 (10)，式 (11) の条件を満たす部分問題をすべて枝刈りする PDF/IHS/hash 法の有効性を調べる．**表 2** に，PDF/IHS 法に対する各手法の高速化率の相乗平均および最大値を示す．表 2 より，最大値はすべてのスレッド数で PDF/IHS/hash 法の方が高いが，相乗平均は PDF/IHS/idle 法の法が高いパター

表 2　相乗平均
Table 2　The geometric average of Speedup.

Thread	PDF/IHS/idle		PDF/IHS/hash	
	ave.	max.	ave.	max.
1	1.18	95.75	0.86	553132.13
2	1.18	96.08	1.39	232.93
4	1.19	93.54	1.10	1304.77
8	1.18	95.56	1.20	1187.37
16	1.16	70.01	1.26	317.96

ンが存在する．PDF/IHS/idle 法に比べて PDF/IHS/hash 法の相乗平均が低い場合があるのは，本評価で求解した問題のタスク数が 50 と小さく，PDF/IHS 法の下界値による枝刈りが効果的に働く問題が多いためであると考えられる．一方，高速化率の最大値は，すべてのスレッド数で PDF/IHS/hash 法の方が高い．このため，PDF/IHS/hash 法は PDF/IHS 法だけでなく PDF/IHS/idle 法に対しても優位な手法であるといえる．

6. おわりに

本論文では，PDF/IHS 法において探索する必要のない部分問題をすべて削減する手法を提案し，その有効性を評価した．評価の結果，単純なハッシュを用いた実装において，相乗平均 1.39 倍の高速化率が得られることを確認した．

また，提案手法は，追加した枝刈りのオーバヘッドが大きく，PDF/IHS 法が高速に求解できる問題において高い高速化率が得られにくいことが確認できた．このため，今後の課題の 1 つとして，提案手法で追加した枝刈りによるオーバヘッドを削減するために，文献 [9] の枝刈り手法と併用することや，ハッシュを改良することが考えられる．他にも，提案手法は，探索済みノードの情報を利用することで枝刈りを行うため，データ転送時間を考慮したタスクスケジューリング問題においても高い効果が期待できると考えられる．このため，データ転送時間を考慮したタスクスケジューリング問題に対する提案手法の有効性の評価も今後の課題である．

参考文献

[1] El-Rewini, H., Ali, H.H. and Lewis, T.: Task Scheduling in Multiprocessing Systems, *Computer*, Vol.28, No.12, pp.27–37 (1995).
[2] Land, A.H. and Doin, A.G.: An Automatic Method of Solving Discrete Programming Problems, *Econometrica*, Vol.28, No.3, pp.497–520 (1960).
[3] Sirisha, D. and Vijayakumari, G.: Exploring the Efficacy of Branch and Bound Strategy for Scheduling Workflows on Heterogeneous Computing Systems, *Proc. 6th International Conference on Advances in Computing and Communications*, pp.315–323 (2016).
[4] Pathan, R., Voudouris, P. and Stenström, P.: Scheduling Parallel Real-Time Recurrent Tasks on Multicore Platforms, *IEEE Trans. Parallel and Distributed Systems* (*TPDS*), Vol.29, No.4, pp.915–928 (2017).
[5] Rho, J., Azumi, T., Nakagawa, M., Sato, K. and Nishio, N.: Scheduling Parallel and Distributed Processing for Automotive Data Stream Management System, Vol.109, pp.286–300 (2017).
[6] Kasahara, H. and Narita, S.: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. Computers*, Vol.C-33, No.11, pp.1023–1029 (1984).
[7] 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，Vol.J74-D1, No.11, pp.755–764 (1991).
[8] 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).
[9] 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レディ状態の割当て削減による PDF/IHS の高速化，情報処理学会論文誌，Vol.58, No.3, pp.654–662 (2017).
[10] 笠原博徳：並列処理技術，技術評論社 (1991).
[11] Ramamoorthy, C.V., Chandy, K.M. and Gonzalez, M.J.: Optimal Scheduling Strategies in a Multiprocessor System, *IEEE Trans. Computers*, Vol.C-21, No.2, pp.137–146 (1972).
[12] 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，情報処理学会論文誌，Vol.43, No.4, pp.936–947 (2002).
[13] Hu, T.C.: Parallel Sequencing and Assembly Line Problems, *Oper. Res.*, Vol.9, No.6, pp.841–848 (1961).
[14] Fernandez, E.B. and Bussell, B.: Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules, *IEEE Trans. Computers*, Vol.C-22, No.8, pp.745–751 (1973).
[15] Standard Task Graph Set (online), available from ⟨http://ww.kasahara.elec.waseda.ac.jp/schedule/⟩ (accessed 2018-03-21).

松瀬 弘明

1993 年生．2018 年千葉工業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．主として並列探索に関する研究に従事．

中村 あすか　（正会員）

1986 年生．2017 年千葉工業大学大学院情報科学研究科情報科学専攻博士後期課程満期退学．2017 年同大学専門研究員．2018 年同大学特別研究員，現在に至る．博士（工学）．主として並列処理や探索アルゴリズムに関する研究に従事．

富永 浩文　（正会員）

1984 年生．2009 年千葉工業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．2018 年同大学院情報科学研究科情報科学専攻博士後期課程満期退学．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．1992 年千葉工業大学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．