千葉工業大学

博士学位論文

階層的挟み撃ち探索を用いた
並列分枝限定法の
探索ノード数削減による高速化に関する研究

平成30年 3月

中村 あすか

要旨

本論文は，階層的挟み撃ち探索を用いた並列分枝限定法を高速化するために，探索ノード数を削減する手法を提案し，その有効性を評価する．

分枝限定法は，NP 困難な組合せ最適化問題の厳密解を求めるための木探索アルゴリズムである．分枝限定法のアルゴリズムは，高い並列性を持つため，求解時間の短縮手法のひとつとして並列分枝限定法の研究が行われている．階層的挟み撃ち探索は，探索の早い段階に最適解を発見することで多くの探索ノードに枝刈りを行う並列探索アルゴリズムである．一方で，本手法は，問題規模が大きくなるほど探索ノード数の増加により求解に長い時間が必要になる．このため，大規模な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．

そこで，本論文では，代表的な組合せ最適化問題のひとつである巡回セールスマン問題および，分枝限定法における階層的挟み撃ち探索が最初に提案されたタスクスケジューリング問題において，階層的挟み撃ち探索を用いた並列分枝限定法の探索ノード数を削減し，高速化する手法を提案する．提案手法は，下界値を用いずに探索ノード数を削減できるため，探索の進行具合に関係なく高い効果が得られると期待できる．

以下に本論文の各章の概要を述べる．本論文は全 5 章より構成される．

まず，第 1 章「序論」では，本研究における背景および従来研究について述べ，提案手法の目的や位置づけを明らかにする．

2 章「分枝限定法」では，分枝限定法のアルゴリズムおよび，その高速化手法について述べる．分枝限定法は，分枝操作と限定操作を繰り返すことで組合せ最適化問題

の厳密解を求める木探索アルゴリズムである．本手法は，問題規模が大きくなるほど探索ノード数が増加するため，探索ノード数の削減が重要である．このため，並列探索による高速化が有効であることが知られている．並列探索アルゴリズムのひとつである階層的挟み撃ち探索は，探索木中の最適解の位置に関係なく高速に求解可能であり，分枝限定法においても高い高速化率が得られることが示されている．

3 章「無駄な待ち状態の割当て削減」では，タスクスケジューリング問題を解くために提案された分枝限定法のひとつである DF/IHS およびその並列探索アルゴリズム PDF/IHS の探索ノード数を削減する手法を提案する．DF/IHS および 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探索ノードとして生成されることがある．そこで，本章では，まず DF/IHS および PDF/IHS においてレディ状態を割り当てられた部分問題の中には探索する必要の無い部分問題が存在することを明らかにする．次に，DF/IHS および 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．評価の結果，提案手法は，提案手法を用いない探索アルゴリズムと比較して，DF/IHS において最大約 79 倍，PDF/IHS において最大約 96 倍高速に求解できることを確認した．

4 章「探索の重複領域削減」では，巡回セールスマン問題の求解において，階層的挟み撃ち探索を用いた分枝限定法の探索ノード数を削減する手法を提案する．分枝限定法における階層的挟み撃ち探索は，複数のプロセッサが左右から挟み撃つように探索するため，複数のプロセッサが同一のノードを探索する探索の重複領域が生じる．巡回セールスマン問題の求解で生成する探索木は，各探索ノードの分枝数が少ないため，スレーブプロセッサの割当領域に対する探索の重複領域の割合が大きくなる．そこで，本論文では，探索の重複領域を削減することで，階層的挟み撃ち探索の求解時間を短

縮する．本削減手法は，スレーブプロセッサの探索済領域をリーダプロセッサが探索しないように，スレーブプロセッサだけでなくリーダプロセッサも探索の重複を検出する．また，複数のスレーブプロセッサが同じノードを探索しないように，スレーブプロセッサの探索経路を反映した再割当を行う．評価の結果，提案手法は階層的挟み撃ち探索に対して相乗平均で約1.2倍，最大約2.1倍高速に求解できることを確認した．

最後に第5章「結論」では，提案手法と評価結果をまとめ，論文全体の総括をする．本論文は，巡回セールスマン問題とタスクスケジューリング問題において，階層的挟み撃ち探索を用いた分枝限定法の探索ノード数を削減する手法を提案した．

Summary

This paper proposes the method to reduce the number of the searching nodes for speed-up of the branch-and-bound method parallelized by the hierarchical pincers attack search(HPAS) and evaluates its method.

The branch-and-bound method is a tree search algorithm to solve the NP-hard combinatorial optimization problem(COP). The algorithm of the branch-and-bound method has the high parallelism, and its parallel search algorithm has been studied for speed-up. HPAS is a fast parallel tree search algorithm for cutting a lot of search nodes by early detection of the optimal solutions. Despite this, the branch-and-bound method using HPAS solving the large-scale problem takes a long time. For solving the large-scale problem fast, it is necessary to not only using parallel search algorithms but reducing branching nodes.

Therefore, this paper proposes the reducing method of the number of the searching nodes for speed-up solving the task scheduling problem and the traveling salesman problem. The proposed method reduces the number of the searching nodes without using the lower bounds and is expected that the high reduction effect is obtained.

This paper is composed of five sections as follows.

In first, section 1 mentions the summary of this study and the necessity of speed-up branch-and-bound method. This section mentions the aim and effects of the proposed method.

Section 2 describes the branch-and-bound method and its speed-up methods. The branch-and-bound method repeats the branching operation and the bounding operation and solves the exact solutions of COP. The branch-and-bound method searches a lot of searching nodes in solving the large-scale problem. This means that reducing the searching nodes of the branch-and-bound method is important. For speed-up the branch-and-bound method, the parallelization is the large effect. HPAS that is one of the parallel search algorithms can solve fast regardless of the position of the optimization and is shown the high speed-up ratio by the branch-and-bound method.

Section 3 describes the study of the depth-first/implicit heuristic search(DF/IHS) and its parallel search algorithm PDF/IHS to reduce the number of the searching nodes, in the task scheduling problem in which branch-and-bound method using HPAS is proposed first. The branching operation of DF/IHS and PDF/IHS makes one or more subproblems by enumerating all combinations of the allocatable tasks and idle tasks at the time when the schedule is undecided. For this reason, the branching operation may make subproblems assigned some unnecessary idle tasks. Therefore, firstly, the study of this section discusses that the unnecessary searching nodes are existing in the tree of DF/IHS and PDF/IHS or not . Secondly, the study of this section proposes the reduction method of a searching nodes of the allocated idle tasks. Therefore, for reduction of branching nodes of the DF/IHS and PDF/IHS, the method cuts a lot of subproblems assigned idle tasks whose execution time is longer than others. As a result of the evaluation, the speedup ratio of the proposed method compared with the DF/IHS is about 79 times on the maximum, with the PDF/IHS is about 96 times on the maximum.

Section 4 describes the study of the branch-and-bound method using HPAS to re-

duce the number of the searching nodes, on the traveling salesman problem. In the branch-and-bound method using HPAS, some processors search from the right and left sides of each subtree in a whole tree. Hence, some of the processors search the same search space. In the searching tree of traveling salesman problem consisting of a node connecting a few branches, overlapping search space occupies vast search area in search area allocated from a slave processor. Therefore, the proposed method speeds up HPAS by reducing overlapping search space. In this method, the leader processor detects overlapping the search area so that it does not search a node which searched slave processors, and allocates a node so that the slave processors do not search the same node. As a result of evaluations, the speedup ratio of the proposed method compared with the hierarchical pincers attack search is about 1.2 times on the average of geometric mean, and about 2.1 times on the maximum.

Finally, section 5 describes the conclusions of this paper. This paper proposed the reduction method of the number of the searching nodes of the branch-and-bound method using HPAS on the task scheduling problem and the traveling salesman problem.

目 次

図 目 次

表 目 次

第1章

序論

1.1 本研究の背景

組合せ最適化問題は，制約条件の中に組合せ的条件を持つ最適化問題であり，オペレーションズリサーチの分野およびその応用として，システム工学，コンピュータ科学などの理工科学や，数理経済学，社会学，心理学など，広範な分野において求解が行れている [1]．組合せ最適化問題の多くは，NP 困難な問題であり，多項式時間で求解可能なアルゴリズムが提案されていない [2]．つまり，組合せ最適化問題の最適解を求めるためには，与えられた問題の制約を満たすすべての解の中から目的関数の評価が最も良くなる解を選択する必要がある．組合せ最適化問題は，問題サイズが大きくなると解の個数が指数関数的に増加するため，最適解求解に必要な時間が問題サイズに応じて指数関数的に長くなる．このような理由から，組合せ最適化問題の求解では，近似解法を用いて一定時間計算を行い，評価の良い組合せを解とすることが多い [3][4].

近似解法は，組合せ最適化問題の解を実用的な時間で得ることができる．また，多くの近似解法は，求解対象の問題に合わせ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．一方で，多少の時間をかけてでも最適解を導出した方が結果的に良くなるという場面も多い．例えば，電力網の設計や，製品を製造する機器の動作パターンなど，組合せ最適化問題の求解によって得られた結果を長期間にわたって利用するような場合が挙げられる [5][6]．このような例では，解の利用期間が長

いほど近似解と最適解の誤差による損失が蓄積する．また，金額や大きさ，期間など，問題の扱う単位が大きい場合，近似解と最適解の誤差が小さいにもかかわらず，実際の損失は大きくなる．このため，扱う単位が大きな問題においても，最適解での求解が必要となる場合がある．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，厳密解の求解が必要となる場合がある．このように，組合せ最適化問題の最適解を求める必要性は依然として高く，組合せ最適化問題の最適解求解の高速化が求められている．

1.2 本研究の目的

組合せ最適化問題の最適解求解に有効な手法のひとつに，分枝限定法がある [7]．分枝限定法は，分枝操作と限定操作を繰り返し行うことで木探索をする手法である．本手法が生成する探索木は，すべての探索ノードが独立した組合せ最適化問題であり，先祖—子孫の関係にない探索ノードを異なる PE(Processor Element) で並列に探索することができる．このため，並列処理を用いた分枝限定法の高速化が行われている [8][9].

多くの並列分枝限定法は，限定操作の効率を高めるために，評価の良い探索ノードから探索することで最適解を早い段階に発見できるように設計されている [10]．このように設計された手法を用いると，評価の良い探索ノードに最適解が存在する問題では高速に求解することができるが，評価の悪い探索ノードに最適解が存在する問題では求解に時間がかかる．一方，並列部分問題探索法 [11] は，評価の悪い探索ノードは広い探索領域を持つという考え方に基づき広い探索領域から順にプロセッサに割り当てるために，評価の悪い探索ノードから優先的に探索するように設計されている．このように設計された手法を用いると，評価の悪い探索ノードに最適解が存在する問題の求解では逐次探索に対して高い高速化率を得ることができるが，逐次探索で高速に求解可能な評価の良い探索ノードに最適解が存在する問題の求解では探索に時間がか

かる場合がある．上記のように，並列分枝限定法は，評価の良い探索ノードに最適解が存在する問題と，評価の悪い探索ノードに最適解が存在する問題の両方を高速に求解することが難しい．

また，並列分枝限定法では，スレッドやプロセスを管理するために，同期処理のような，逐次分枝限定法のアルゴリズム中に存在しない処理を追加で行う必要がある．これらの処理を細かく行うことで，早期に探索した方が良いと判断した探索ノードを PE 間で共有したり，負荷分散が実現できる．このため，分枝操作で新たに探索ノードを生成するたびに PE 間で探索ノード情報を共有する並列分枝限定法が多く提案されている．一方，追加する処理による並列化オーバヘッドが大きくなると，並列化による処理時間短縮の効果が小さくなる [12]．分散メモリ環境のように PE 間でデータ共有するコストが大きい並列化環境では，部分木の単位で PE に処理を割り当てることが一般的である [13][14][15]．

共有メモリ環境において，低コストで動的負荷分散を実現し，最適解の探索木上の位置に関係なく高い高速化率を得ることができる並列探索手法のひとつとして，階層的挟み撃ち探索が提案されている [16]．階層的挟み撃ち探索は，評価の良い探索ノードに多くのプロセッサを割り当て，探索木の左右から挟み撃つように探索を行う．このため，評価の良い探索ノードに最適解が存在する場合だけでなく，評価の悪い探索ノードに最適解が存在する場合も高速に求解することができる．一方で，問題規模が大きくなるほど探索ノード数の増加により求解に長い時間が必要になるため，大規模な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．

そこで，本論文では，分枝限定法における階層的挟み撃ち探索が最初に提案された組合せ最適化問題であるタスクスケジューリング問題および，代表的な組合せ最適化問題のひとつである巡回セールスマン問題において，階層的挟み撃ち探索を用いた並列分枝限定法の探索ノード数を削減し，高速化する手法を提案する．

第2章

分枝限定法

2.1 はじめに

分枝限定法は，組合せ最適化問題 (COP:combinatiorial optimization problem) の厳密解を求める木探索アルゴリズムである．本手法は，大規模な問題ほど探索ノード数が多くなり，求解に時間がかかる．このため，問題やアーキテクチャの特性に合わせて多くの分枝限定法を高速化する手法が提案されている．また，探索木上の最適解の位置に関係なく高速に求解可能な並列分枝限定法のひとつとして，階層的挟み撃ち探索が提案されている．

以下では，まず，第 2.2 節で組合せ最適化問題の特性について述べる．次に，第 2.3 節で分枝限定法のアルゴリズムについて述べる．第 2.4 節と第 2.5 節で逐次および並列分枝限定法の高速化アルゴリズムについて述べ，第 2.6 節で階層的挟み撃ち探索のアルゴリズムについて述べる．最後に，第 2.7 節で，本章の総括を行う．

2.2 組合せ最適化問題

組合せ最適化問題は，最適化問題のひとつであり，組合せ的条件を含んだ制約の中から最も高い評価を持つ組合せを求める問題である．最適化問題は，数理計画問題と

も呼ばれ，一般的に式 (2–1) のように表される．

$$\begin{cases} min. & f(\mathbf{x}) \\ s.t. & \mathbf{x} \in \mathbf{F} \end{cases} \tag{2–1}$$

ただし，式 (2–1) 中の $min.$ は，目的関数 f が最小となる $\mathbf{x}$ の組合せを求めることを表す．最大化問題の場合は，目的関数に負の値を乗算し，$-f$ を最小化する問題として一般化できる．また，$s.t.$ は，subject to または such that の略であり，$\mathbf{x}$ が取りうる値に対する制約条件を表す．制約条件を満たす組合せ $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ を実行可能解または解，実行可能解 $\mathbf{x}$ の集合 F を実行可能領域または解空間という．つまり，式 (2–1) の最適解は，実行可能領域 $\mathbf{F}$ 中で目的関数 f を最小にする $\mathbf{x}$ であると言い換えることができる．

最適化問題は，数学的性質や問題の特性によって分類することができ，それぞれの特性に合わせた求解アルゴリズムが提案されている．最適化問題のうち，$\mathbf{F}$ や f が n 次元ベクトル空間 $\mathbb{R}^n$ によって定義され，変数ベクトル $\mathbf{x}$ が連続的な値をとる問題は，連続最適化問題と呼ばれる．また，そうでない問題は離散最適化問題と呼ばれる．

組合せ最適化問題は，最適化問題の制約条件の一部に，組合せ的な離散的条件が含まれた問題である．このため，組合せ最適化問題を数理モデルに定式化すると，整数計画問題や 0-1 計画問題のような離散最適化問題となることが多い．組合せ最適化問題は，目的関数 $f(\mathbf{x})$ を最小化する変数 $\mathbf{x}$ を求める問題として，一般的に式 (2–2) のように表される．

$$\begin{cases} min. & f(\mathbf{x}) \\ s.t. & \mathbf{x} \in \mathbf{S} \\ & \mathbf{S} \subset \mathbf{X} \end{cases} \tag{2–2}$$

ただし，$\mathbf{S}$ は，実行可能領域であり，$\mathbf{X}$ は，組合せ的条件によって定義された離散集合である．$\mathbf{X}$ を構成する要素は有限個であるため，$\mathbf{S}$ も有限個の要素を持つ離散集合である．

2.2.1　最適解求解の難しさ

最適化問題は，実行可能領域 $\mathbf{F}$ が n 次元ベクトル空間 $\mathbb{R}^n$ の部分集合であれば，勾配法のように微分を利用した解法を用いることで最適解を高速に求解することができる．一方，組合せ最適化問題のように実行可能領域 $\mathbf{F}$ が離散的条件を持つ場合，求解アルゴリズムに微分を用いることが難しい．このため，組合せ最適化問題の多くは，最適解を多項式時間で求めることが難しい．

解の候補が与えられたときに，それが解か否かの検証を多項式オーダで行える問題は，クラス NP(Nondeterministic Polynominal) に属する [4]．ただし，クラス P(Polinominal) に属する問題はクラス NP にも属するため，$P \subseteq NP$ が成り立つ [17]．一方，クラス NP に属する問題のうち，判定問題が NP 完全な問題を NP 困難という．NP 困難 (NP-hard) な問題は，クラス NP に属するかどうか不明なので，計算複雑度が NP 完全以上となる [4].

組合せ最適化問題の定義に当てはまる問題は，多数存在する．このため，組合わせ最適化問題の中には，最短経路問題 [2] のように多項式時間で求解可能な問題も存在する．一方で，多くの組合せ最適化問題は，ナップサック問題や，巡回セールスマン問題のように NP 困難な問題に分類される．

クラス P とクラス NP の定義から，もし NP 完全問題を解く多項式オーダのアルゴリズムが存在するならば，クラス NP の問題はクラス P に属することになる．$NP \neq P$ は証明されていない [2] が，これまでに NP 完全問題を解く多項式オーダのアルゴリズムは見出されていない [4][17]．また，NP 完全に属する問題は，「1 問でも多項式時間で

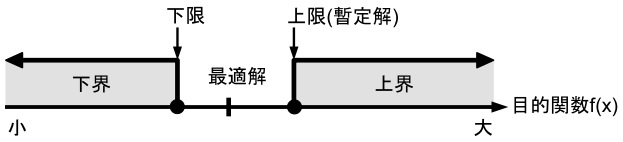

図 2–1 ： 数直線上における最小化問題の上界と下界

解けるアルゴリズムがあれば，クラス NP に属するすべての問題が多項式時間アルゴリズムが存在する」とされているため，今後も，多項式時間アルゴリズムは発見されないと考えられている [18]．このため，組合せ最適化問題においても，$P \neq NP$ という考えに基づいてアルゴリズムが考案されている [19].

2.2.2　最適解に対する上限と下限

組合せ最適化問題は，制約条件の一部の変数を固定したり，制約条件の式をいくつか取り除くことで制約条件を緩めると，別の組合せ最適化問題を作ることができる．この特性を用いることで，最適解の上限値や下限値を導出し，組合せ最適化問題の最適解が取りうる目的関数の値を推測することができる．

図 2–1 に，数直線上における最小化問題の上界と下界の例を示す．任意の要素 α に対して $\alpha \leq \beta$ が成り立つとき，β は α の上界である．一方，任意の要素 α に対して $\alpha \geq \beta$ が成り立つとき，β は α の下界である．また，上限は，最も目的関数 $f(\mathbf{x})$ の値が小さい上界であり，下限は，最も目的関数 $f(\mathbf{x})$ の値が大きな下界である．最小化問題では，下界は最適解より小さな値を取り，上界は最適解より大きな値を取る．このため，上界，および下界は，最適解に対して図 2–1 のような関係になる．下限値と上限値が等しいとき，(下限値) = (上限値) = (最適解) の関係が成り立つ．つまり，精度の良い上界と下界を算出できれば，それを基に組合せ最適化問題の最適解を算出することができる．

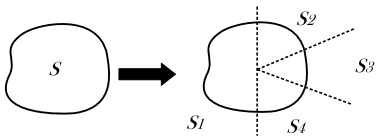

図 2–2 ： 部分問題の生成

部分問題を用いた上界値の導出

組合せ最適化問題は，NP 困難であるため，問題規模が大きな問題ほど求解に時間がかかる．一方，元問題の部分問題は，元問題よりも問題規模が小さいため，元問題よりも容易に求めることができる．図 2–2 に，部分問題の例を示す．図 2–2 は，部分列挙法 (Partial Enumeration Method) を用いて，元問題を 4 つの部分問題に分割する例である．図 2–2 中の S_i は，問題 P_0 のとりうる組合せ S を互いに排反に分割した部分問題の実行可能領域を表す．図 2–2 より，実行可能領域 S_1 の最適解は，問題 P_0 の実行可能解である．本例において S_2 の最適解が不明である場合，S_1 の最適解が S の最適解であるかどうかを判断することができない．しかしながら，S_1 の最適解が分かった段階で，S の最適解が S_1 の最適解よりも大きな値にならないことが分かる．このように，部分問題を解くことで，元問題の上界値を求めることができる．

また，部分列挙法は，場合分けを行い，式 (2–2) の集合 S を式 (2–3) のように n 個の部分集合 S_i に分割する手法である．

$$\begin{cases} S = S_1 \cup S_2 \cup \cdots \cup S_i \cup \cdots \cup S_n = \displaystyle\bigcup_{i=1}^{n} S_i \\ S_i \cap S_j = \phi \qquad (i = 1, 2, \cdots, n, \;\; j = 1, 2, \cdots, n, \;\; i \neq j) \end{cases} \tag{2–3}$$

式 (2–3) より，各 S_i の最小値 $f(\bar{s}_i)$ $(\bar{s}_i \in S_i)$ が求まっているとき，すべての $f(\bar{s})$ のうち最小値を与える $\bar{s}_i$ が，最適解 $\bar{s}$ となる．S_i を実効可能領域とする部分問題は，S の変数を固定することで生成されるため，問題 P_0 よりも規模が小さな組合せ最適化問題となり，問題 P_0 よりも容易に解くことができる．

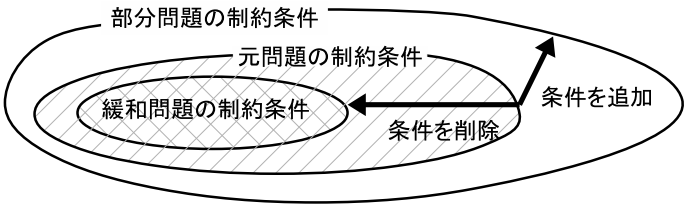

図 2–3 ： 組合せ最適化問題と緩和問題の制約条件

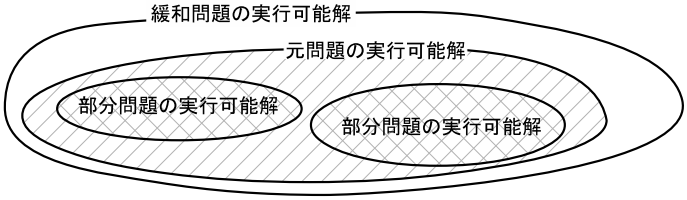

図 2–4 ： 組合せ最適化問題と緩和問題の実行可能解

緩和問題を用いた下界値の導出

組合せ最適化問題の制約条件を緩めた問題を緩和問題という．組合せ最適化問題は，ほとんどの問題が NP 困難であり，求解が難しい．一方で，元の組合せ最適化問題の制約条件を緩めることで生成された新たな最適化問題である緩和問題は，元問題よりも求解が容易になる場合がある．緩和問題は，式 (2–4) のように表すことができる．

$$\begin{cases} min. & f_s \\ s.t. & \mathbf{s} \in \mathbf{S}'_i \\ & \mathbf{S}_i \subseteq \mathbf{S}'_i \end{cases} \tag{2–4}$$

式 (2–4) より，関数 f_s の最小値を $f_{s'}$ とおくと，$f_{s'}$ が S_i の下界値となる．ただし，下界値を求める関数 f_s は $f(\mathbf{s})$ と同等または $f(\mathbf{s})$ よりも良い評価を得られるように定める．求解対象の組合せ最適化問題 P_0 の制約条件と，その緩和問題 P'_0 の制約条件の関係を図 2–3 に示す．また，求解対象の組合せ最適化問題 P_0 の解と，その緩和問題 P'_0 の解の関係を図 2–4 に示す．問題 P_0 の解は， 緩和問題 P'_0 の条件をすべて満たすため，

P_0' の解集合に含まれる．一方，緩和問題 P_0' の解は必ずしも問題 P_0 の条件を満たすわけではない．

式 (2–4) において問題 P_0 の条件を満たす下界 s' は，問題 P_0 の実行可能解である．また，図 2–4 より，問題 P_0 の解と P_0 の緩和問題 P_0' の解の間には包含関係が成り立つ．このため，問題 P_0 とその緩和問題 P_0' から得られる解には，以下の関係がそれぞれ成立する．

- 問題 P_0の 緩和問題 P_0' が実行可能解を持たない場合，P_0 は実行可能解を持たない．
- P_0' の最適解を $\overline{s'}$ とすると，$\overline{s'} \in S_0'$ ならば $\overline{s'}$ は (P_0) の最適解である．
- $f(s) = c_s$ とおき，緩和問題 P_0' の最適解を $\overline{s'}$，元問題 P_0 の最適解を $\bar{s}$ とすると，$\mathbf{c^t}_{\overline{\mathbf{s'}}} \leq \mathbf{c^t}_{\bar{\mathbf{s}}}$ が成り立つ．

2.3　分枝限定法のアルゴリズム

分枝限定法は，分枝操作と限定操作を繰り返し行うことで，組合せ最適化問題の最適解を求める手法である．分枝操作は，問題を場合分けし，解空間を分割することで，部分問題を生成する操作である．図 2–5 に，分枝限定法による問題の分割過程を示す．生成された部分問題のうち，まだ解かれていない部分問題である活性問題に対して分枝操作を繰り返し行うと，図 2–5 のように各部分問題を木構造で表すことができる．生成したすべての部分問題を解くことで元の問題の最適解が求められるため，図 2–5 のような木を探索することで与えられた問題を求解できる．探索中は，発見した実行可能解のうち最も評価の良い解を暫定解として記憶し，探索終了時に記憶している暫定解を最適解とする．限定操作は，分枝操作で生成した部分問題のうち，求解する必要の無い部分問題を活性部分問題から取り除く操作である．本操作では，各部分問題に

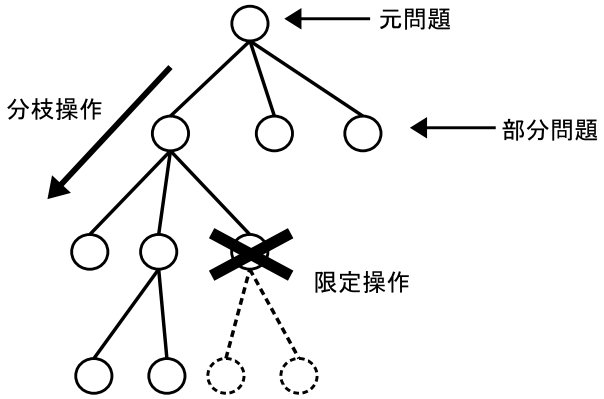

図 2–5 ： 分枝限定法による問題の分割

おいて緩和問題から下界値を求め，暫定解と比較することで最適解が存在するかどうか判定する．図 2–5 の例において，×印のついたノードの下界値よりも評価の良い暫定解が求まっているとする．このとき，×印のノードを根とする部分木内のノードで得られる実行可能解は，少なくとも×印のノードの下界値よりも評価の悪い解であると判断できる．このため，本例では破線部のノードを探索しない．

分枝限定法は，探索するノード数が少ないほど高速に求解することができる．このため，分枝限定法では，一般的に，部分問題の増加を防ぐために，各部分問題から生成する部分問題の個数が 2 または 3 になるように分枝規則を設定することが有効であると言われている [20]．また，限定操作の効率を上げることで，活性問題の数を減らすことができ，探索ノード数が少なくなる．

2.4　分枝限定法の高速化

分枝限定法は，多くの組合せ最適化問題に対して有効な探索アルゴリズムである．このため，本手法は，問題の特性やアーキテクチャなどに合わせて様々なアルゴリズムが提案されている [1][20]．

探索木の生成方法や下界値の算出方法が異なると，探索木の特性が変化する．このため，同一の問題に対する求解であってもアルゴリズムごとに探索効率は異なる．つまり，特定の問題に対して有効なアルゴリズムが，必ずしも他の問題でも有効であるとは限らない [21]．一方，分枝限定法におけるアルゴリズムの設定方針は，多くの問題で共通する．

以下では，分枝限定法の逐次アルゴリズムに対する設計方針について述べる．

2.4.1　分枝変数の選択

分枝操作を行う際に固定する変数を分枝変数という．組合せ最適化問題の中には，目的関数が複数の変数からなる問題が存在する．このような問題では，複数の変数の中から分枝変数を選択する必要がある．また，組合せ最適化問題の中には，整数計画問題や 0-1 計画問題など，複数のモデル化できる問題が多く存在する．このため，同一問題においても異なるモデル化を行うことで，異なる性質を持った変数が分枝変数となる場合がある．組合せ最適化問題では，同じ問題を求解する場合でも，選択した分枝変数により，生成される総部分問題数や，暫定解 (初期解) が求まるまでの分枝数，および，最適解を探索するまでの分枝数が変化する [21].

分枝限定法は，求解過程において生成される部分問題が多いと，探索領域が大きくなる．このため，生成される部分問題は少ない方が良い．また，分枝限定法では，暫定解を元に限定操作を行うので，暫定解が求まるまで限定操作を行うことができない．暫定解を早い段階で求めることができると，早い段階から限定操作ができるため，広い探索領域を削減できる可能性がある．さらに，限定操作を効率よく行うためには，最も評価の良い実行可能解である最適解を，早い段階で発見する必要がある．このように，同じ問題でも分枝変数によって探索の効率が異なるため，目的に合わせた分枝変数の選択が必要となる．

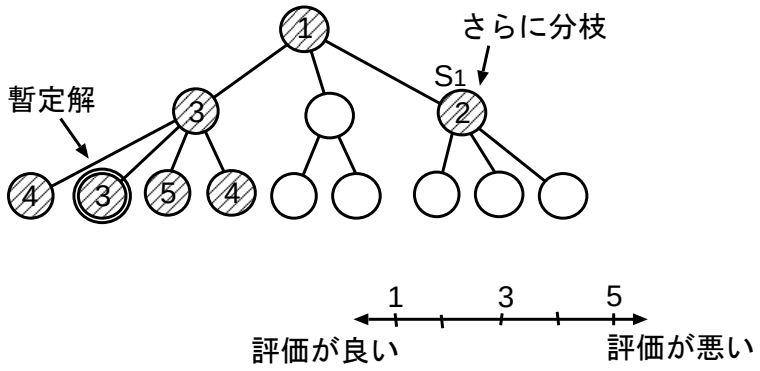

図 2–6 ： 探索済み領域と限定操作の効率 (暫定解 > 下界値)

2.4.2　木探索アルゴリズム

分枝限定法は，その時点で探索済みの領域内で最も評価の良い上界である暫定解を用いて限定操作を行う．このため，探索済み領域がどのように存在するかによって，あるノードに対する限定操作の結果が異なる．図 2–6 および図 2–7 に，探索済みの領域が限定操作に与える影響の例を示す．図 2–6，図 2–7 は，同じ探索木を異なる探索順序で探索中であり，それぞれ，新たに S_1 を探索するところである．図 2–6 では，暫定解が 3 であるため，ノード S_1 を分枝する．一方，図 2–7 では，暫定解が 1 であるため，ノード S_1 を分枝しない．図 2–7 の方が探索領域を多く削減できるため，この例では，図 2–7 の探索順序で探索した方が，S_1 の限定操作の効率が良いと言える．ただし，最適解がどのノードに存在するかは問題ごとに異なるため，すべての問題で同一の探索手法が有効であるとは限らない．

限定操作は，暫定解の評価が良いほど広い探索領域を削減する．このため，良い暫定解を早い段階で得ると，限定操作の効率が高まり，高速に求解することができる．最も良い上界値は最適解なので，分枝限定法を高速に求解するためには，最適解をなるべく早い段階で探索する必要がある．

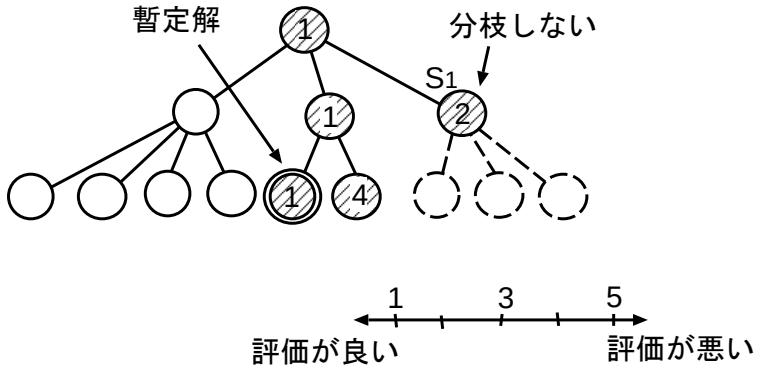

図 2–7 ： 探索済み領域と限定操作の効率 (暫定解 < 下界値)

部分問題の探索順序は，最良優先探索 (Best First Search)，深さ優先探索 (Depth First Search)，幅優先探索 (Breadth First Search) の 3 手法に大きく分類することができる [4]．以下に，これらの手法について述べる．

最良優先探索

最良優先探索は，ヒューリスティック関数を用いて次に探索する活性問題を定める探索手法である．本手法は，ノードプールを用いて実装することが一般的である．図 2–8 に，最良優先探索の例を示す．図中の灰色のノードは活性問題を表す．本手法は，ノードプール内のノードから最もヒューリスティック的な評価の最も高いノードを，次に探索するノードとして選択し，ノードプールから取り除く．また，ノードプールから取り除いた探索ノードから新たな探索ノードを生成すると，そのノードをノードプールに格納する．本例では，矢印で示すノードが選択され，点線のノードが新たに生成される．これらの操作を繰り返し行うことで，ヒューリスティック関数の評価の良いノードは最適解を持つ可能性が高いという考え方に基づき，ノードの探索木上の位置に関係なく，ヒューリスティック関数の評価の良いノードから探索を行うことができる．最良優先探索においてヒューリスティック関数に下界値を用いる場合は，最良下

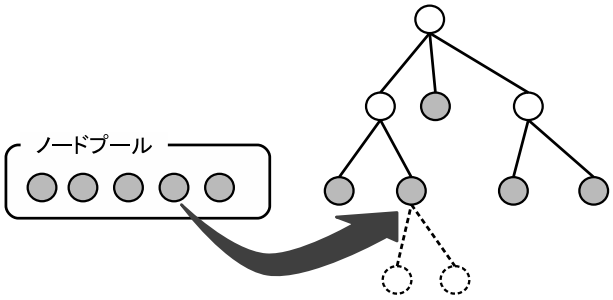

図 2–8 ： 最良優先探索

界探索とも呼ばれる．

ヒューリスティック関数の評価の精度が高い場合，最良優先探索は，深さ優先探索や幅優先探索より高速に最適解を発見できる．一方で，すべての未探索ノードをメモリ上に記憶する必要があるため，幅優先探索と同様にメモリ使用量の多いアルゴリズムである．

深さ優先探索

深さ優先探索は，探索木上で深さが深い位置にある活性問題から優先して探索する手法である．本手法は，スタックを用いて実装することが一般的である．図 2–9 に，深さ優先探索の例を示す．図中の矢印は，ノードを探索する順序を示す．深さ優先探索は，新しく生成した部分問題をスタックに格納し，スタックの上にある部分問題から順に分枝する．本手法は，探索木の深さ分の探索ノード情報を記憶することで実装可能なため，メモリ使用量を少なく抑えることができる．一方で，すべての枝を一度末端まで調べる必要があるため，探索木の最大深さが分からない問題において探索の効率が悪くなる．ただし，組合せ最適化問題で扱う解空間は有限個の集合からなるため最大探索深さも有限であり，解の無い空間を無限の深さまで探索することがない．また，多くの問題では，最大探索深さを探索前に把握することができる．

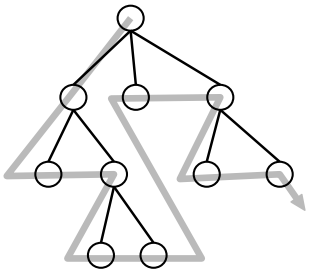

図 2–9 ： 深さ優先探索

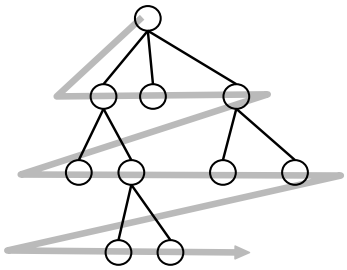

図 2–10 ： 幅優先探索

幅優先探索

幅優先探索は，探索木上で深さが浅い位置にある活性問題から優先して探索する手法である．本手法は，キューを用いて実装することが一般的である．図 2–10 に，幅優先探索の例を示す．図中の矢印は，ノードを探索する順序を表す．幅優先探索は，新しく生成した部分問題をキューに格納し，キューから取り出した問題を分枝する．本手法は同一深さのすべての探索ノードの情報をメモリに記憶する必要があるため，メモリの使用量が大きい．また，最も深さの浅い解を求める問題では，最初に見つけた解が最適解となる．

2.4.3　下界値の精度向上

あるノードに対する限定操作には，そのノードの下界値が用いられる．下界値の精度が良いほど，多くのノードを枝刈し，探索ノード数を削減することができる．

一般的に，精度の良い下界値を求めるためには，計算に時間をかける必要がある．図 2–11 に，探索過程の例を示す．図 2–11 中のノード A を根とする部分木には最適解が存在しないとする．また，網掛け部分は探索済みノードを表しており，ノード A 探索時において，ノード A を根とする部分木中には，暫定解よりも評価の良い上界を持つノードが存在しないとする．このとき，ノード A において，高い精度の下界値を得る

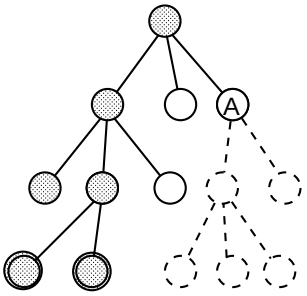

図 2–11 ： 下界値の精度と探索領域の大きさ

ことができれば，ノード A 以下のノードを探索する必要がなくなるため，探索領域を削減することができる．しかし，ノード A で下界値の計算に長い時間がかかる場合，ノード A を根とする部分木内の全てのノードを探索した方が速い可能性がある．このように，分枝限定法では，下界値の精度と下界値の導出時間におけるトレードオフを考慮して下界の評価関数を定める必要がある．

2.4.4　初期解の精度向上

最小化問題では，暫定解の値が小さいほど，限定操作の効率を上げることができる．このため，並列化により分枝限定法と改善法による求解を同時に行い，近似解法によって得られた解を初期解として扱うことで，暫定解の精度を高める手法が提案されている [21]．また，最も早い段階に求まる暫定解である初期解の精度を高めるために，分枝限定法による探索を行う前に，近似解法による求解を行う手法が提案されている [22]．このようなアルゴリズムの中でも，分枝限定法に切除平面法を併用した手法は分枝カット法と呼ばれ，高い効果が得られることが知られている [23][24][25]．

他にも，多くの分枝限定法のアルゴリズムでは，精度の良い解が初期解になるように，探索木上中のノードの位置を入れ替えたり，分枝規則および探索順序が決定され

ている [8][26]．このように設計されたアルゴリズムでは，求解時間が長い問題において探索を途中で打ち切る場合でも，精度の良い近似解が得られることが保証される．

2.5　並列分枝限定法

分枝限定法で生成された部分問題は，それぞれが独立な組み合わせ最適化問題として成立する．このため，各部分問題をそれぞれ独立に解くことが可能であり，異なるプロセッサで並列に求解することができる [9]．並列分枝限定法では，複数のプロセッサが同時に異なるノードの探索を行うので，逐次探索よりも早い段階で探索されるノードが存在する．このようなノードに最適解が存在する場合，早い段階から限定操作の効率を高めることができるため，高速に求解できる．一方，逐次探索において早い段階で最適解を発見できる問題の場合, 逐次探索で探索しないノードを，並列探索手法で探索する場合がある．このような場合，並列化によるプロセッサ間通信のオーバヘッドにより高速化率が低くなる．しかし，最適解を速い段階で発見できる問題は，逐次探索でも高速に求解できるため，高速化率が低い場合でも高速に求解することが可能である場合が多い．以下では，分枝限定法における並列化方針について述べる．

2.5.1　処理の分割方針

多くの並列分枝限定法では，1 つの探索木を複数のプロセッサで探索する．このため，処理を複数に分割し，それぞれの処理を各プロセッサに割り当てる．以下では，処理の分割単位，および，処理の分割を行うタイミングによる探索効率の違いについて述べる．

処理の分割単位

処理の分割単位として，各ノードの処理を複数プロセッサで行う分割方式や，各ノードの処理を別のプロセッサで行う分割方式が挙げられる．

各ノードの処理を複数のプロセッサで行う場合，処理の分割単位が細かくなる．このため，多くの場合，この様な分割方式は採用されない．ただし，複数のプロセッサが同一の演算を行うことで高速に求解することが可能なアーキテクチャや，メモリ容量などの制限があるアーキテクチャでは，ノード内の処理を分割する場合がある．

各ノードの処理を別のプロセッサで行う場合，探索木中の部分木をプロセッサに割り当てることで，複数のノードを一度に割り当てることができる．このとき，各プロセッサは，自身が分枝したノードをさらに分枝するので，割当て領域内の探索において他のプロセッサの分枝情報を必要とせず，独立して探索できる．また，各プロセッサに部分探索木を割り当てることで，各プロセッサ間で共有するデータの量を削減することができる．さらに，部分探索木は，根ノードを指定することで一意に定まる．このため，多くの並列分枝限定法は，ノードを指定することで，割当て領域に部分木を指定し，各プロセッサに探索領域を割り当てる．

静的な分割と動的な分割

求解中に動的に処理の分割を行うことで各プロセッサに割り当てられた処理の量を均等化する手法が提案されている．しかし，動的に処理の分割を行うと，探索以外の処理に時間をかけることとなる．このため，処理の再分割時間を削減するために，静的に処理の分割を行う手法も提案されている [11].

静的に探索領域の分割を行う場合，探索前から各プロセッサに割当て可能な領域が決定しているため，速い段階で各プロセッサに処理を割り当てることができる．しかし，本手法で分割された探索領域の位置情報は，探索木の形状に関係なく一定となる．

このため，探索木の形状に合わせた分割を行うことができない．

動的に探索領域の分割を行う場合，探索木の形状や，探索の進行に合わせて処理の割当てを行うことができる．特に，動的に分割した処理を動的に割当る手法では，各プロセッサが任意の探索領域を探索した結果を用いて処理の割り当てが行われる．このため，プロセッサに探索領域を割り当てるために，領域を分割するための処理が必要となる．動的分割手法を用いる並列探索手法でも，初期割当に静的分割を用いる場合がある．この場合，初期割当を高速に行い，探索開始時の並列化オーバヘッドを最小限に抑えることができる．

2.5.2　各プロセッサへの処理の割当方針

並列分枝限定法の求解過程において，待ち状態のプロセッサが生じると，並列処理の効率が落ちるため，求解時間が長くなる．つまり，複数のプロセッサが常に処理を行っている状態を保つことで，並列化の効率を高め，高速に求解することが可能となる．このため，高速に求解するためには負荷分散を行う必要がある．以下では，まず，処理の割当方針のうち，負荷分散に対する方針について述べる．次に，処理の割当順序に関する方針について述べる．

静的な割当と動的な割当

探索領域を各プロセッサに静的に割り当てた場合，動的に割り当てた場合よりプロセッサ間で共有するデータを少なく抑えることができる [11]．ただし，分枝限定法では，限定操作により探索領域の削減が行われる．このため，各プロセッサは，割り当てられた探索領域をすべて探索するまで，割り当てられた探索領域の広さを知ることができない．静的に広さが均等になるように探索領域を各プロセッサに割り当てても，限定操作により各プロセッサの探索領域の広さが不均等になる．このように，各プロセッ

サに静的に処理を割り当てた場合，各プロセッサごとの負荷が不均等になりやすい．

動的に各プロセッサに探索領域を割り当てた場合，探索状況に応じて各プロセッサに割り当てる処理を変えることができるため，各プロセッサの負荷を均等にすることができる [8]．しかし，動的な処理の割り当てを行うと，静的に処理を割り当てた場合よりも多くのデータをプロセッサ間で共有する必要がある．

処理の割り当てを頻繁に行うことで，より厳密に負荷を均等化することができるが，プロセッサ間で共有するデータは多くなる．一方，処理の割り当て回数を減らすことでプロセッサ間で共有するデータ量を減らすことができるが，各プロセッサの負荷が不均等になりやすくなる．このように処理の割り当てにおいて負荷分散とプロセッサ間で共有するデータの量はトレードオフの関係にある．どのような処理を優先して各プロセッサに割り当てるかによって負荷分散のオーバヘッドが変化する．

ノード探索順序

第 2.4.2 項で述べたように，分枝限定法は，ノードの探索順序によって限定操作の効率が変化し，求解時間に影響を与える．多くの並列分枝限定法では，複数のプロセッサが同時に分枝を行うため，分枝限定法と異なる順序で探索が行われる．このため，分枝限定法において探索されるまでに時間がかかったノードを並列分枝限定法が早い段階に探索することがある．このような場合，限定操作の効率が向上し，並列分枝限定法が探索するノード数が分枝限定法よりも少なくなる．これにより，スーパーリニアスピードアップと呼ばれるプロセッサ台数倍以上の高速化率が得られる場合がある [9]．

多くの探索手法は，早い段階に最適解を発見できるように，最適解が存在する確率が高いと考えられる下界値の評価が高い領域を優先して割り当てる．しかし，このような割当方式を取ると，下界値の評価が低い領域に最適解が存在する場合の求解時間が長くなる．このため，逐次探索において求解に時間のかかる問題は並列探索でも求解に時間がかかることが多い．

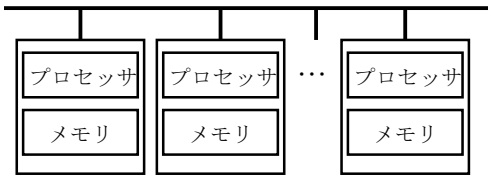

図 2–12 ： 分散メモリ環境

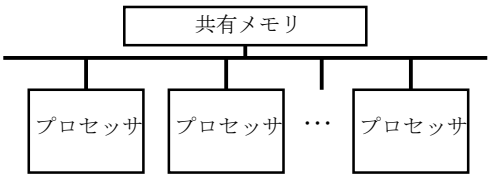

図 2–13 ： 共有メモリ環境

2.5.3　アーキテクチャに合わせたアルゴリズム設計方針

並列化環境は，プロセッサ間におけるメモリのアドレス空間共有方式によって，分散メモリ環境と共有メモリ環境の 2 種類に大きく分類することができる．図 2–12 に，分散メモリ環境の例を示し，図 2–13 に，共有メモリ環境の例を示す．分散メモリ環境は，各プロセッサが他のプロセッサが持つメモリに直接アクセスすることができない並列化環境である．一方，共有メモリ環境は，図 2–13 のように，すべてのプロセッサが全てのメモリのアドレス空間を共有する並列化環境である．以下に，各並列化環境における並列分枝限定法の一般的なアルゴリズム設計方針を述べる．

分散メモリ環境におけるアルゴリズム

並列分枝限定法は，各プロセッサの処理が独立しているため，各プロセッサが独自のメモリを参照して処理を行う分散メモリ環境に対する実装に向いている．一方で，分散メモリ環境における並列分枝限定法では，プロセッサ間でデータを共有するために，プロセッサ間通信が必要となる．このため，暫定解の更新や負荷分散のオーバヘッドが大きくなる．

分散メモリ環境における並列分枝限定法で高速に最適解を求解するためには，プロセッサ間の通信オーバヘッドを少なく抑える必要がある．そのため，分散メモリ環境における並列分枝限定法の多くは，通信を行う回数の削減や，非同期通信または 1 対

1 通信による通信時間の隠蔽を行う．このような並列分枝限定法のアルゴリズムとして，マスタ・スレーブ型をとる手法 [12][27] や隣接するプロセッサ間のみで負荷分散を行う手法 [13] などが提案されている．

共有メモリ環境におけるアルゴリズム

共有メモリ環境における並列分枝限定法は，各プロセッサが共通のメモリに直接アクセス可能である．共有メモリ環境における分枝限定法では，少ない通信オーバヘッドでプロセッサ間の通信を行うことができる．このことから，共有メモリ環境における並列分枝限定法では，それぞれのプロセッサの探索状況を頻繁に更新し，共有することで，再割当の効率を高める手法が有効であると言われている [8]．ただし，複数のプロセッサが同じデータを同時に更新するような場合，排他処理が必要となるためメモリアクセスのオーバヘッドが大きくなる．

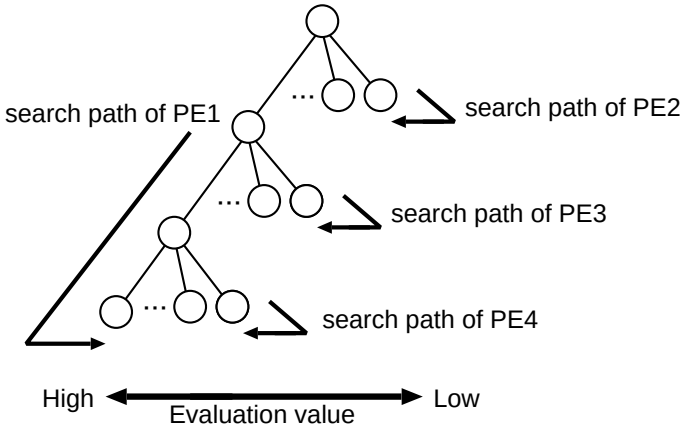

図 2–14 ： 4PE における階層的挟み撃ち探索

2.6　階層的挟み撃ち探索

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 [28][29]．本手法は，Prolog の探索において提案された手法であるが，分枝限定法の並列化においても高い有効性が確認されている [16][30]．分枝限定法における階層的挟み撃ち探索は，実行時間最小マルチプロセッサスケジューリング問題を解くための手法である DF/IHS 法 [26] の並列化手法として提案された．本並列探索手法は，1 つのリーダプロセッサと複数のスレーブプロセッサが並列に深さ優先探索を行う．図 2–14 に，4 つのプロセッサエレメント（PE）による分枝限定法における階層的挟み撃ち探索の振舞を示す．図中の PE1 はリーダプロセッサ，それ以外の PE はスレーブプロセッサとする．リーダプロセッサは，生成された子問題を探索木左側から探索し，待ち状態のスレーブプロセッサに探索経路上のノードを割り当てる．階層的挟み撃ち探索では，現在探索中のノードから根ノードまでの探索木上の経路を探索経路とし，探索経路をセレクションポインタ（SP）を用いて表す [16]．SP は，各ノードの探索木上における位置情報を表すポイン

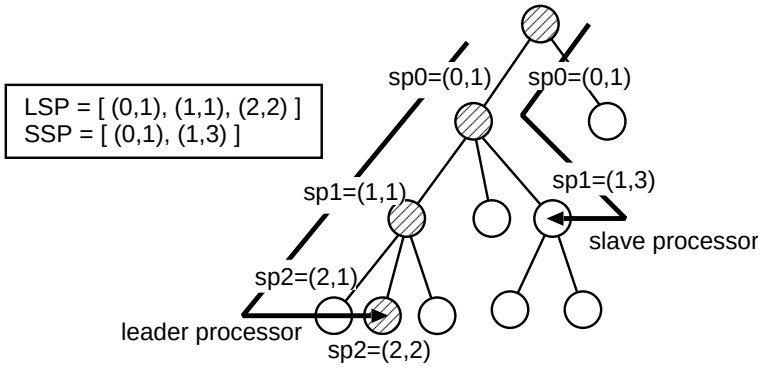

図 2–15 ： SP 値の例

タであり，ノードが探索木上で左から何番目のノードに位置するかを示す値である．根ノードから探索中ノードまでの経路上に存在するノードの SP を深さの浅い順に列挙したものを SP 値と呼ぶ．図 2–15 に，SP の例を示す．図中の斜線のノードは，リーダプロセッサの探索経路である．また，深さ i の SP を $sp_i =$（深さ, 枝番号）と表す．SP 値は，探索中ノードの深さ分の要素によって探索経路を記憶する．本論文では，リーダプロセッサの探索経路を表す SP 値を LSP，スレーブプロセッサの探索経路を表す SP 値を SSP とする．スレーブプロセッサは，割当ノードを根とする部分探索木を割当領域とし，割当領域を右側から深さ優先探索で探索する．

左右から探索するプロセッサが探索を続けることで探索済領域が重複することを，探索の重複という．探索の重複は，探索が重複したノードを，左右から探索するどちらのプロセッサが先に探索するかによって，スレーブプロセッサがあとから探索する場合と，リーダプロセッサがあとから探索する場合の 2 種類に分けることができる．階層的挟み撃ち探索では，どちらのタイプの探索の重複も，スレーブプロセッサが，自身の探索経路（$SSP = [ssp_1, ssp_2, \cdots]$）とリーダプロセッサの探索経路（$LSP = [lsp_1, lsp_2, \cdots]$）を比較することで検出する．深さ d のノードを割り当てられたスレーブプロセッサと

リーダプロセッサの間に探索の重複が生じると，深さ $1 \leq i \leq d$ において $ssp_i < lsp_i$ となるか，深さ $d+1$ において $ssp_{d+1} = lsp_{d+1}$ となる．本論文では，左右から探索するプロセッサの探索が重複したかどうか判定する操作を，探索の重複検出操作と呼ぶ．探索の重複検出操作は，SP 値の各要素を，深さの浅い方から順に $d+1$ 要素分比較するだけでよく，少ないコストで検出できる．スレーブプロセッサは，探索の重複を検出すると，リーダプロセッサに探索が重複したことを通知してから待ち状態になる．リーダプロセッサは，スレーブプロセッサから探索の重複が生じたという通知を受け取ると，そのスレーブプロセッサに割り当てた階層の探索が終了したことを知ることができる．さらに，リーダプロセッサは，待ち状態のスレーブプロセッサに LSP 上のノードを再割当する．このとき，広い探索領域を持つと考えられる深さの浅いノードを優先して割り当てるため，少ない再割当回数で，動的に負荷分散を行うことができる．また，階層的挟み撃ち探索は，限定操作の効率を上げるために，探索過程において，あるプロセッサが暫定解を更新すると，新たな暫定解をすぐにすべてのプロセッサで共有する．

本手法は，評価の良いノードが最適解である問題を求解する場合，評価の良いノードを多くのプロセッサで探索するため，早い段階で最適解を探索することができる．また，評価の悪いノードが最適解である問題を求解する場合でも，スレーブプロセッサは評価の悪いノードから探索するため，早い段階で最適解を探索することができる．このように，本手法は，最適解の探索木上の位置に関係なく高速に求解することが可能である [16]．

2.7　本章のまとめ

本章では，分枝限定法およびその高速化手法について述べた．多くの並列分枝限定法は，限定操作の効率の向上による探索ノード数削減や負荷分散を考慮して設計されているが，これらを両立して実現することは難しい．共有メモリ環境において，低コストで動的負荷分散を実現し，最適解の早期発見により限定操作で効率よく枝刈りすることができる並列探索手法のひとつとして，階層的挟み撃ち探索が提案されている．一方，階層的挟み撃ち探索を用いた並列分枝限定法においても，求解する問題の規模が大きくなるほど，探索する必要のある探索ノード数が増加し求解時間が長くなる．そこで，本論文では，階層的挟み撃ち探索を用いた並列分枝限定法を高速化するために，探索ノード数を削減する手法を提案する．

第3章

無駄な待ち状態の割当て削減

3.1 はじめに

本章では，タスクスケジューリング問題の厳密解法であるDF/IHS(Depth First/Implicit Heuristic Search)[26]およびその並列探索アルゴリズムPDF/IHS(Parallelized DF/IHS)の探索ノード数を削減する手法を提案し，その有効性を評価する．

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[31][32]．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[33][34]．本章では，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[35]．このため，本問題の最適解求解には，分枝限定法[1]による探索を行う必要がある．本問題の探索を効率よく行う手法としてDF/IHSおよびPDF/IHSが提案されている．

PDF/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解法であるCP/MISF(Critical Path/Most Immediate Successors First) [26] 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．PDF/IHSを用いることで効率よ

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．PDF/IHS 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待ち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，PDF/IHS は，不必要な待ち状態が割り当てられた部分問題を生成することがある．そこで本章では，DF/IHS において不必要な待ち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 つのタスクとみなす．これにより，再定義された部分問題において，プロセッサに無駄な待ち状態を割り当てる部分問題を検出し，その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．

以降の節では，まず，第 3.2 節で本章で扱うタスクスケジューリング問題について述べる．次に，第 3.3 節でタスクスケジューリング問題の求解に有効な並列探索アルゴリズムのひとつである PDF/IHS の探索アルゴリズムについて述べ，第 3.4 節で PDF/IHS の探索木において探索する必要の無い探索ノードが存在することを示す．第 3.5 節では探索する必要の無い探索ノードの探索を打ち切ることで探索ノード数を削減する手法を提案し，第 3.6 節と第 3.7 節でその有効性を評価する．最後に，第 3.8 節で本章のまとめを行う．

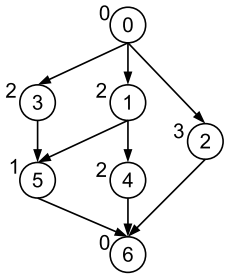

図 3–1 ： タスクグラフの例

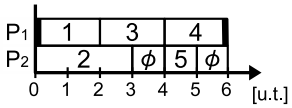

図 3–2 ： スケジュールの例

3.2　タスクスケジューリング問題

本章では，タスクスケジューリング問題を，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 [35] とおく．本章で扱う問題は，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．

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 3–1 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．

本章では，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 3–2 に，図 3–1 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，プロセッサに待ち状態を割り当てたことを示す．本例のスケジュールは，図 3–1 の全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．

3.3　分枝限定法によるタスクスケジューリング問題の求解

タスクスケジューリング問題の求解に有効な分枝限定法のアルゴリズムのひとつに，DF/IHS および PDF/IHS がある．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 を階層的挟み撃ち探索を用いて並列化した探索アルゴリズムである [16]．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．

DF/IHS は，CP/MISF 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．このため，以下では，まず，CP/MISF について述べる．次に，DF/IHS の分枝操作と限定操作について述べる．

3.3.1　CP/MISF

CP/MISF は，クリティカルパスを用いて作成したプライオリティリストを作成し，リストスケジューリングを行う近似解法である．本手法は，実行可能なタスクが複数存在する場合にクリティカルパスが長く，後続タスク数の多いタスクを優先的に割り当てることでというヒューリスティックに基づいてスケジューリングを行う．このため，CP/MISF は，クリティカルパス (CP)[36] 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．

リストスケジューリングは，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，実行可能なタスクのうちプライオリティの高いタスクから順に割り当てる手法である．スケジュールが未設定となる時刻に実行可能なタスクが無い場合は，待ち状態 ϕ を割り当てることで，割当て時点を更新する．タスク数 n，プロセッサ数 m のタスクスケジューリング問題の求解において本手法がタスクまたは待ち状態 ϕ を割り当てる回数は，最大でも mn 回である．このため，CP/MISF を用いることで，高速に精度の高い近似解を得ることができる．

3.3.2　分枝操作

DF/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待ち状態を割り当てることで，部分問題 $\pi_i(i = 1, 2, \cdots)$ を生成する．図 3–3 に，図 3–1 のタスクグラフを DF/IHS で探索する例を示す．図 3–3 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 は待ち状態を表す．R に待ち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 [37]．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，待ち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．例えば図 3–3 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

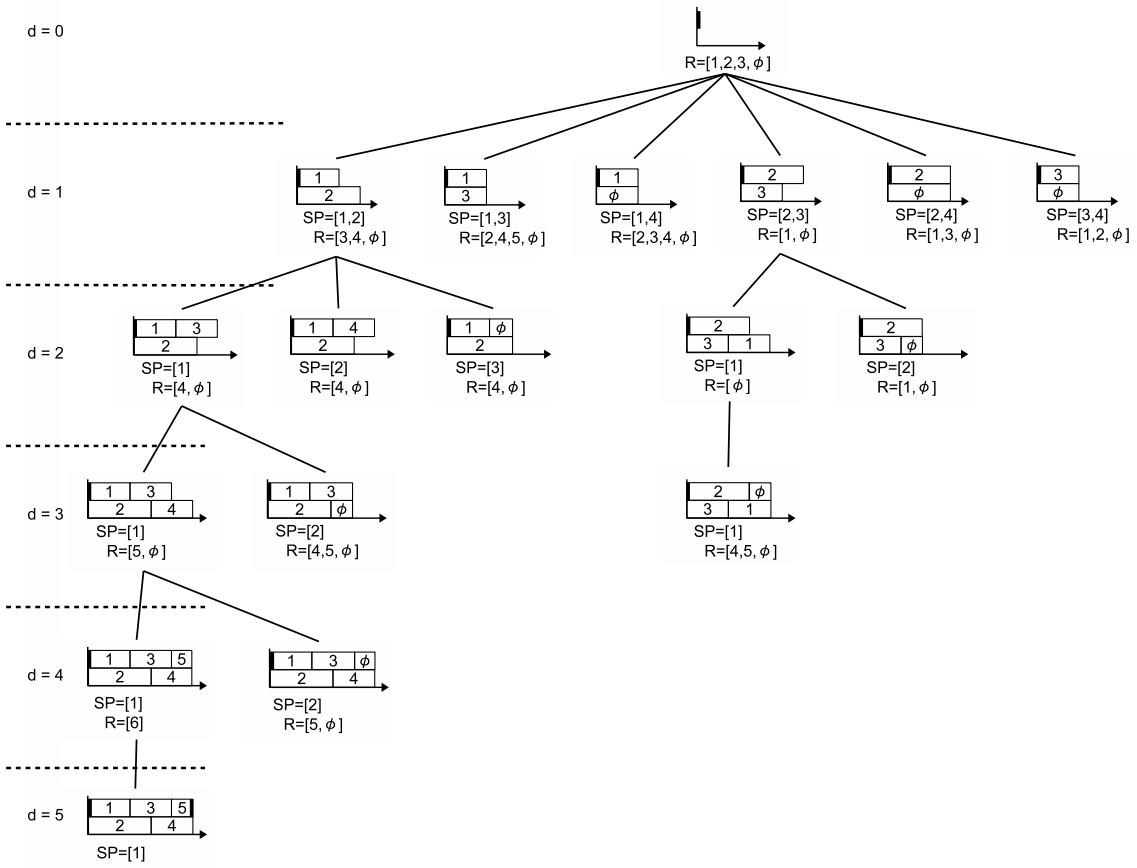

図 3–3 ： DF/IHS が生成する探索木の例

示す．本例の SP のように，DF/IHS が割当て可能なタスクが存在する場合にも待ち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 [37].

DF/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[26] 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 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．ただし，待ち状態 ϕ は，割当て優先度を最も低く設定するために，常に 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 3–4 に，次に探索する部分問題の SP を生成する擬似コードを示す．図 3–4 のように SP 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．このため，DF/IHS は，探索木の左側から深さ優先探索を行うことで，探索初期に精度の良い暫定解を得ることができる．また，DF/IHS で最初に得られる実行可能解は CP/MISF の解と等しくなる．

3.3.3　限定操作

PDF/IHS は，部分問題 π_a の下界値 $lb(\pi_a)$ を，式 (3–1) によって求める [38]．ただし，式 (3–1) 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 (3–1) 中の $lb_{cr}, lb_{div}, lb_{hu}$ は，式 (3–2)～式 (3–4) により求める．

$$lb(\pi_a) = \max\{lb_{cr}(\pi_a), lb_{div}(\pi_a), lb_{hu}(\pi_a)\} \tag{3–1}$$

$$lb_{cr}(\pi_a) = \max_{i \in I(\pi_a)} cp(i) + \min_{1 \leq i \leq m} t_{\pi_a(P_i)} \tag{3–2}$$

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

$$lb_{hu}(\pi_a) = lb_{cr}(\pi_a) + \lceil q_{(\pi_a)} \rceil \tag{3–4}$$

```
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 3–4 ： SP 値設定の擬似コード

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 [39] である．式中の $q_{(\pi_a)}$ は，式 (3–5)〜式 (3–8) のように負荷密度関数 $F(\overline{\tau}, t)$ から求める．

$$q(\pi_a) = \max_{0 \le t_k \le lb_{cr}(\pi_a) - t_0} \left(-t_k + \frac{1}{m} \int_0^{t_k} F(\overline{\tau}, t) dt \right) \tag{3–5}$$

$$F(\overline{\tau}, t) = \sum_{j \in I(\pi_a)} f(\overline{\tau}_j, t) \tag{3–6}$$

$$\overline{\tau}_j = lb_{cr}(\pi_a) - cp(j) - \min_{1 \leq i \leq m} t_{\pi_a(P_i)} \tag{3–7}$$

$$f(\overline{\tau}_j, t) = \begin{cases} 1, & for \quad t \in [\overline{\tau}, \overline{\tau} + time(j)] \\ 0, & otherwise \end{cases} \tag{3–8}$$

3.4　無駄な待ち状態の検出

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 [1]．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 [39][40]．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 (3–1) 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 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，探索する必要のない部分問題が生成されることがある．

無駄な部分問題の例として，図 3–3 において $SP = [(1, 4)]$ の部分問題を挙げる．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．

以降では，まず部分問題を再定義し，その上で探索する必要のない部分問題を一般化する．

3.4.1　部分問題の再定義

本節では，DF/IHS 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．本節で再定義する部分問題は，同一プロ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，待ち状態を含め 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 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 3–5 に，再定義した部分問題の例を示す．図 3–5 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 3–5 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 $n+i+1$ とする．タスク T_{n+i+1} 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が消失する．しかし，DF/IHS 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 を根から葉まで辿ることで，生成されたスケジュールを知ることができる．

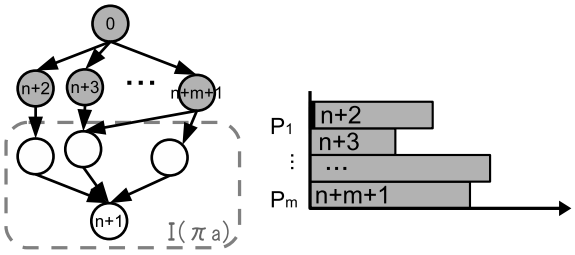

図 3–5 ： 再定義された部分問題の例

3.4.2　探索する必要の無い部分問題

探索する必要のない部分問題は，第 3.4.1 項で再定義した部分問題を用いて一般化すると，定理 3.4.1 のように表すことができる．

定理 3.4.1. 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 の探索過程において，定理 3.4.1 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 3.4.1. 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 3.4.1. $E(\pi_a) \subseteq E(\pi_b)$ 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 3.4.1 が成り立つ．

補題 3.4.2. 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 3.4.2. 背理法を用いる．$Best(\pi_a) > Best(\pi_b)$ と仮定する．このとき，補題 3.4.1 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 3.4.2 が成り立つ．

補題 3.4.2 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 の探索過程において π_b の探索を打ち切っても，DF/IHS の探索結果における最適性は失われない．

3.5　探索ノード数の削減

第 3.4 節より，図 3–3 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 3.4.1 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 が探索過程で生成する部分問題の中には，定理 3.4.1 によって探索する必要が無いと判定される問題が多くある．

DF/IHS 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．一方で，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，探索する必要が無い部分問題でも枝刈りされない可能性がある．そこで，本章では，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，第 3.4 節の定理を利用して下界値や暫定解を用いずに判定し，枝刈りする．また，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．そこで本節では，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．

3.5.1　DF/IHS における無駄な待ち状態の削減

第 3.4 節で示した探索する必要の無い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し，定理 3.4.1 の条件を満たすかどうかを調べる必要がある．DF/IHS 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 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，本章では，なるべく DF/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，SP を設定する際に，探索する必要が無いことが容易に判定可能な部分問題のみを削減する．

図 3–6 に，本章の提案手法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，タスクまたは待ち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 3–6 の部分問題 (a) 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り当てが行われても，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 が待ち状態であるとする．このとき，部分問題 (a) においてタスク i を割り当てられた子問題が (b)，待ち状態を割り当てられた子問題が (c) である．また，部分問題 (b) に待ち状態を割り当てた子問題が (d) である．タスク i の実行時間より，部分問題 (c) と部分問題 (d) は割当て時刻の等しい部分問題である．本例の部分問題 (c) と，部分問題 (d) を第 3.4 節のように再定義すると，部分問題 (d) のタスクグラフは (c) のタスクグラフの部分問題となる．つまり，部分問

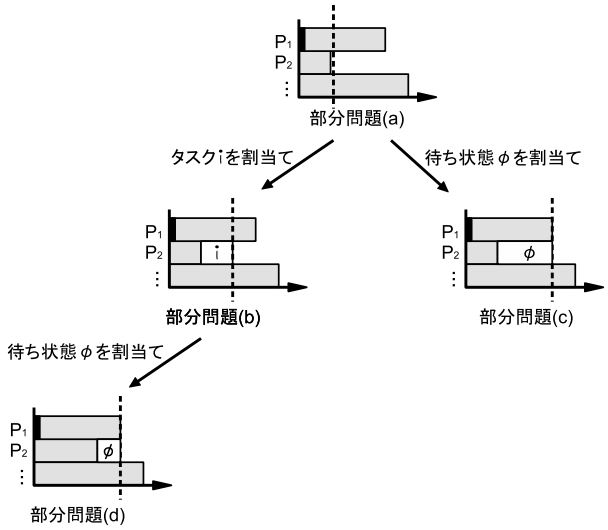

図 3–6 ： 削減するスケジュールの例

題 (c) を探索しても部分問題 (d) よりも短いスケジュールが求まらない．このため，定理 3.4.1 より部分問題 (c) と部分問題 (d) を比較すると，部分問題 (c) は探索する必要が無い部分問題であると判定できる．

図 3–6 の部分問題 (b) と部分問題 (d) は探索木上で親子関係にあるため，部分問題 (b) を探索することで (d) の部分問題も探索される．つまり，部分問題 (b) を探索することで部分問題 (c) を探索する必要が無くなる．このため，図 3–6 では，部分問題 (a) を分枝する段階で，部分問題 (c) と同様に待ち状態が割り当てられた部分問題を探索する必要の無い部分問題であると判別できる．ただし，部分問題 (b) が枝刈りされた場合は，部分問題 (d) が探索されなくなる．このような場合においても，部分問題 (d) および部分問題 (c) は枝刈りされた部分問題 (b) よりも短いスケジュールを求めることができないため，DF/IHS で求まる最適解に影響を与えない．

上記を踏まえて，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

当てられずに待ち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第 3.4 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 が探索に採用している CP/MISF 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 の探索順序を変えないようにする．

以上を踏まえ，本章で提案するアルゴリズムの擬似コードを図 3–7 に示す．図 3–7 のコードでは，待ち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通りに行う．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．

上記のように，無駄な待ち状態の割当て削減手法は，下界値を用いずに部分問題を枝刈りできる．このため，本手法を用いることで，DF/IHS 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 3–8 に，本手法が有効に働く部分問題の例を示す．DF/IHS の下界値を求めるヒューリスティックでは，図 3–8 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 (3–1) で高精度な下界値が求まりにくく，DF/IHS で枝刈りが起こりにくくなる．一方，無駄な待ち状態の割当てを削減する手法は，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．

```
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 3–7 ： 最小処理時間のみを用いた SP 値設定の擬似コード

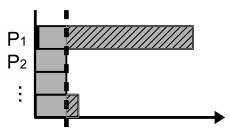

図 3–8 ： 無駄な待ち状態の割当ての削減が効果的に働く部分問題の例

3.5.2　PDF/IHS における無駄な待ち状態の削減

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 でも第 3.4 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 の振る舞いが異なるため，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第 3.4 節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 [41] 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，DF/IHS と同様に，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，本章では，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，探索する必要が無いことが容易に判定可能な部分問題のみを削減する．上記を踏まえて，無駄な待ち状態の割当てを削減する手法のリーダ PE とスレーブ PE の枝刈りの手順について述べる．

無駄な待ち状態の割当てを削減する手法において，リーダ PE は，PDF/IHS のリーダ PE が DF/IHS で探索する PE と同じ振る舞いをすることから，第 3.5.1 項と同様の枝刈りを行う．図 3–10 に，リーダ PE による SP 値更新処理の擬似コードを示す．図 3–10 のコードは，第 3.4 節の条件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 3–9 に図 3–10 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

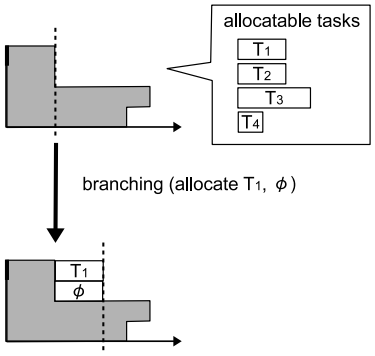

図 3–9 ： 削減できる部分問題の例

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 3–10 のコードを用いることで，図 3–9 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，探索する必要の無い部分問題が枝刈りできる．

無駄な待ち状態の割当てを削減する手法において，スレーブ PE は，PDF/IHS のスレーブ PE が DF/IHS で探索する PE とは異なる振る舞いをするため，図 3–10 と同じ手順で SP 値を更新することができない．本章では，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，本手法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，無駄な待ち状態の割当てを削減しても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 3–11 に，スレーブ PE による SP 値更新処理の擬似コードを示す．図 3–11 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，無駄な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

```
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 3–10 ： リーダ PE による SP 値更新処理の擬似コード

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．

3.6　DF/IHS に対する提案手法の評価

第 3.5 節で提案した無駄な待ち状態の割当てを削減する手法の有効性を示すために，本手法を適用した DF/IHS および PDF/IHS を用いてタスクスケジューリング問題を求解し，探索ノード数および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 [42] 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

```
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 3–11 ： スレーブ PE による SP 値更新処理の擬似コード

問題を求解する．下界値の計算には，第 3.3 節の式（3–1）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．

表 3–1 ： DF/IHS が生成する $d = 1$ の探索ノード数 [個]

実行可能タスク数	問題数	$m = 2$	$m = 4$	$m = 8$	$m = 16$
3–5	7	13.0	24.6	25.3	25.3
6–10	19	37.1	184.5	345.0	345.9
11–15	18	90.0	1134.6	9632.1	12855.9
16–20	12	158.0	3492.8	84435.5	207492.8
21–25	4	283.5	11801.0	1154464.5	17435441.0

表 3–2 ： 無駄な待ち状態の割当て削減手法が生成する $d = 1$ の探索ノード数 [個]

実行可能タスク数	問題数	$m = 2$	$m = 4$	$m = 8$	$m = 16$
3–5	7	9.4	13.1	13.1	13.1
6–10	19	30.8	134.8	247.6	248.4
11–15	18	78.6	861.2	6244.1	8120.9
16–20	12	143.5	2926.9	63247.3	147724.0
21–25	4	262.3	10144.5	892235.8	12146605.5

3.6.1　深さ 1 の探索ノードの削減率

本評価では，無駄な待ち状態の割当てを削減した手法および DF/IHS が探索する深さ 1 の部分問題の数を比較する． 表 3–1，表 3–2，表 3–3 に，DF/IHS および無駄な待ち状態の割当て削減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 3–3 の削減率は，DF/IHS で生成する部分問題数を $n(S_{DF/IHS})$，無駄な待ち状態の割当て削減手法で生成する部分問題数を $n(S_{proposed})$ とおいて，式 (3–9) で求める．

$$\text{削減率 [\%]} = \frac{n(S_{DF/IHS}) - n(S_{proposed})}{n(S_{DF/IHS})} \times 100 \tag{3–9}$$

表 3–3 ： $d = 1$ の探索ノードの削減率 [%]

実行可能タスク数	問題数	$m = 2$	$m = 4$	$m = 8$	$m = 16$
3–5	7	27.69	46.75	48.22	48.22
6–10	19	16.98	26.94	28.23	28.19
11–15	18	12.67	24.10	35.17	36.83
16–20	12	9.18	16.20	25.09	28.81
21–25	4	7.48	14.04	22.71	30.33

表 3–3 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，無駄な待ち状態の割当てを削減する手法で探索する必要があるかどうかを判定する部分問題が，待ち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．例えば，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 番目の要素は待ち状態の割当てであり，$m = 2$ のときには待ち状態を割り当てていない SP の組合せにも，$m = 3$ のときには待ち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，待ち状態を割り当てる SP の割合が大きくなる．このため，本手法は，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言い難い．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，待ち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，無駄な待ち状態の割当てを削減する手法は，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．

3.6.2　求解過程で探索する探索ノード数の測定

本節では，DF/IHS および無駄な待ち状態の割当てを削減する手法を用いて $2, 4, 8, 16$ 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，第 3.3 節の式 (3–1) を用いる．式 (3–9) 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 3–12 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 3–12 の横軸は DF/IHS が生成した部分問題数，縦軸は削減率である．図 3–12 より，DF/IHS で部分問題が多く生成される問題ほど，削減率が高く，無駄な待ち状態の割当ての削減が有効に働くことが分かる．

また，図 3–12 において，本章の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，本章の提案手法によって，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 3–13 に，本章の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 3–13 は DF/IHS が生成する探索木で

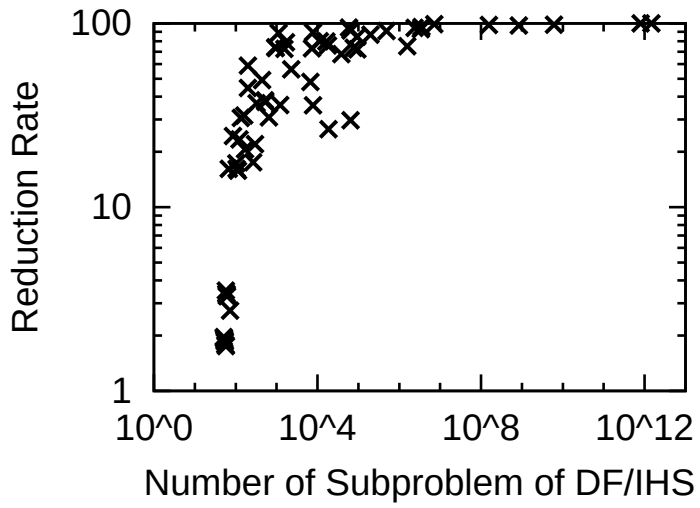

図 3–12 ： DF/IHS に対する無駄な待ち状態の割当て削減手法の削減率

あり，灰色の部分問題が初期解よりも精度のよい暫定解を与える部分問題であるとする．DF/IHS 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 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無くなる．一方，本章の提案手法では，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 3–13 において本章の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．これにより本例では，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 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本章の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，本章の提案手法は，DF/IHS

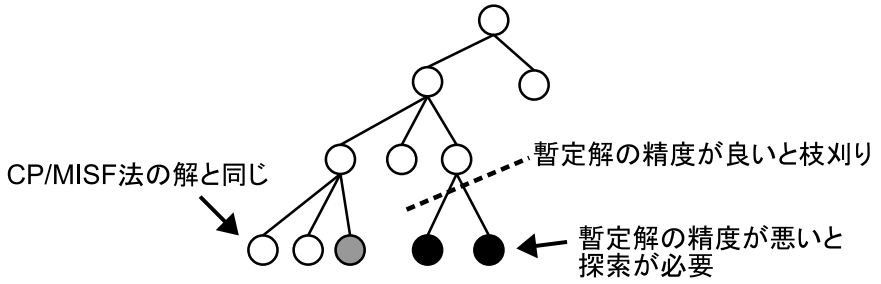

図 3–13 ： 無駄な待ち状態の割当て削減により部分問題数が増える例

と同様に CP/MISF 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．

3.6.3　探索時間の測定

本節では，第 3.6.2 項と同様に，DF/IHS と本章の提案手法で 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(3.3GHz)，メモリが 8GB である．また，本評価では，高速化率を式 (3–10) で定義する．

$$\text{高速化率 [倍]} = \frac{\text{DF/IHS の探索時間}}{\text{本章の提案手法の探索時間}} \tag{3–10}$$

図 3–14 に，DF/IHS に対する本章の提案手法の高速化率を示す．図 3–14 の横軸は DF/IHS の探索時間，縦軸は高速化率である．図 3–14 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，本章の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，本章の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．

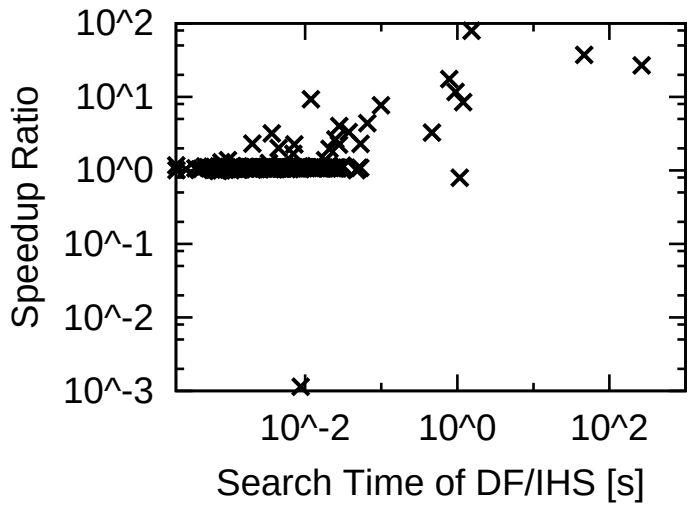

図 3–14 ： DF/IHS に対する無駄な待ち状態の割当て削減手法の高速化率

図 3–14 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 と本章の提案手法にほとんど差が無かったが，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，第 3.6.2 項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 での探索時間 0.1 秒未満と短いため，本章の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，第 3.6.2 項の評価において本章の提案手法を用いても DF/IHS と部分問題数が変わらなかった 175 問の内，173 問は高速化率が 1 より大きかった．これは，本章の提案手法が，式 (3–4) 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．

3.7　PDF/IHSに対する無駄な待ち状態の割当て削減手法の評価

無駄な待ち状態の割当てを削減する手法の有効性を示すために，本手法と PDF/IHS 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．評価環境は，CPU が Intel Xeon E5-2687W v2，メモリが 64GB である．また，本評価では，PDF/IHS に対する本章の提案手法の高速化率を式（3–11）で定義する．

$$\text{高速化率 [倍]} = \frac{\text{PDF/IHS の探索時間}}{\text{無駄な待ち状態の割当て削減手法の探索時間}} \tag{3–11}$$

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．

3.7.1　探索時間の測定

図 3–15 から図 3–19 に，1PE, 2PE, 4PE, 8PE, 16PE で PDF/IHS を行った際の本章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 に対する本章の提案手法の高速化率である．ただし，図 3–15 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 が効率よく枝刈りし，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，本章の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，本章の提案手法を用いても短時間で求解でき

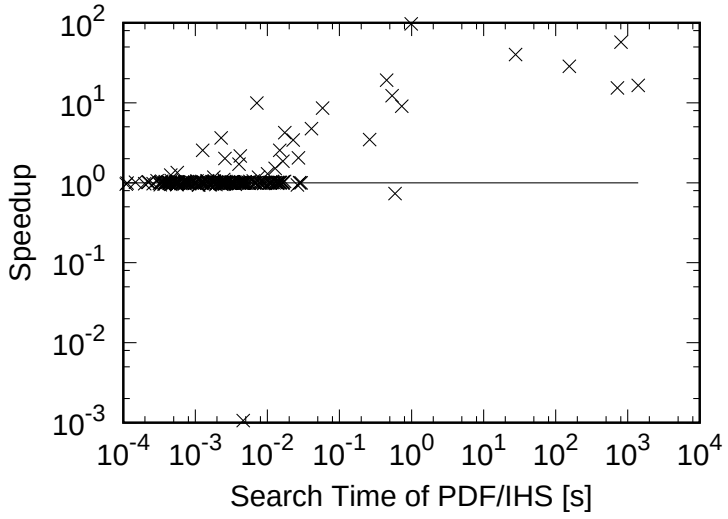

図 3–15 ： 1PE の高速化率

る．このため，このような問題による高速化率の低下が本章の提案手法の有効性に与える影響はほとんど無いと考えられる．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 3–4 に，PDF/IHS の探索時間ごとの本章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 3–4 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，本章の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，本章の提案手法による枝刈りの効果は大きい．

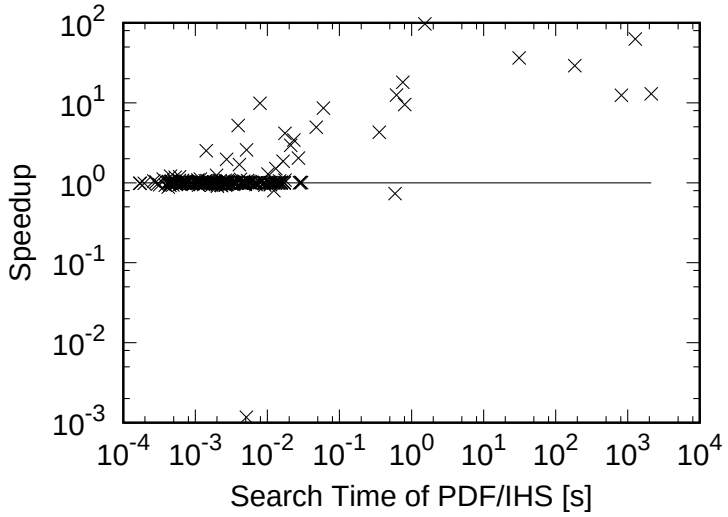

図 3–16 ： 2PE の高速化率

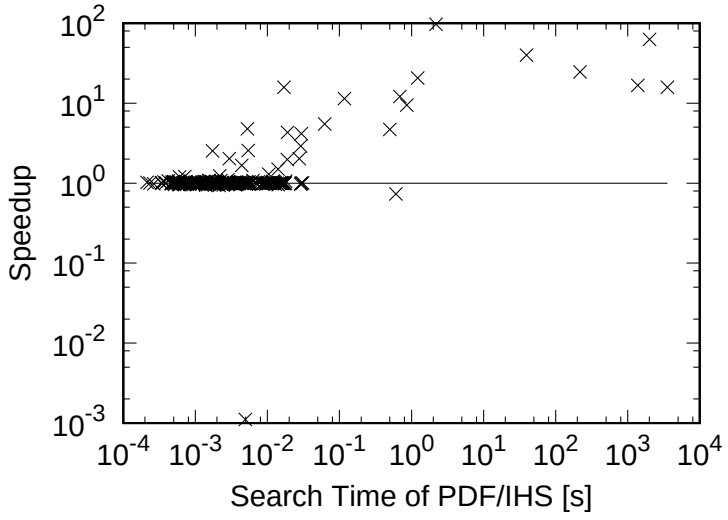

図 3–17 ： 4PE の高速化率

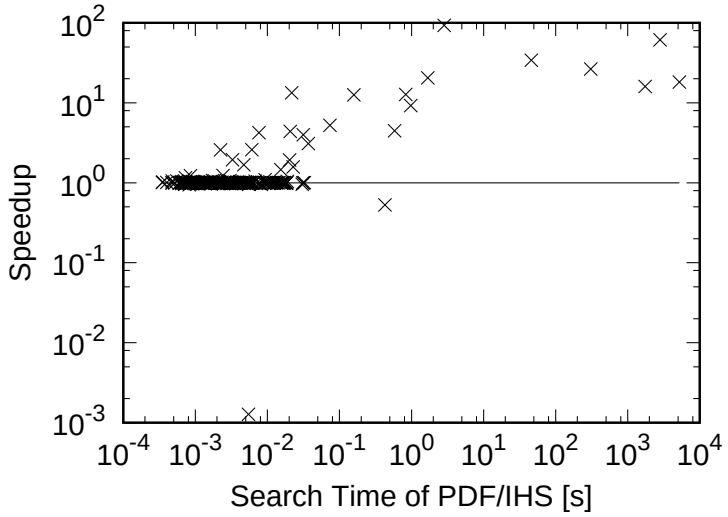

図 3–18 ： 8PE の高速化率

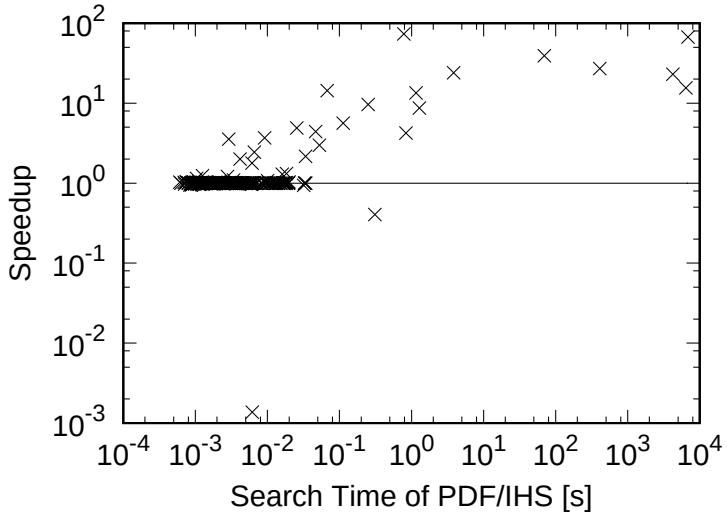

図 3–19 ： 16PE の高速化率

表 3–4 ： PDF/IHS の探索時間ごとの高速化率

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 3–5 ： 無駄な待ち状態の割当て削減手法を用いることによる探索ノード数の増減

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

3.7.2　部分問題数の測定

表 3–4 で PDF/IHS の求解時間が 1 秒以上の時に PE 数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 と本章の提案手法が探索した部分問題数を測定する．表 3–5 に，本章の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 3–5 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，本章の提案手法は，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える．

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，第 3.6.2 項で示したように，PDF/IHS が早い段階に探索する精度の良い解を本章の提案手法が枝刈りしたためであると考えらえる．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，本章の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．

最後に，図 3–20 から図 3–24 に本章の提案手法を用いることで得られる探索ノード

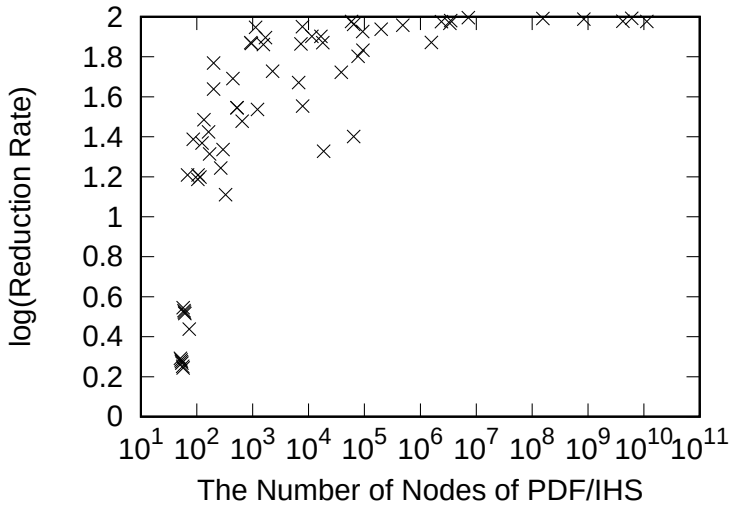

図 3–20 ： 1PE の削減率

数の削減率を示す．削減率は，式（3–12）で定義する．

$$\text{削減率}\,[\%] = (1 - \frac{\text{無駄な待ち状態の割当て削減手法の探索ノード数}}{\text{PDF/IHS の探索ノード数}}) \times 100 \tag{3–12}$$

また，グラフの縦軸は削減率の常用対数である．図 3–20 から図 3–24 より，探索ノード数の削減率にも図 3–15 から図 3–19 と同様の傾向が表れていることが分かる．

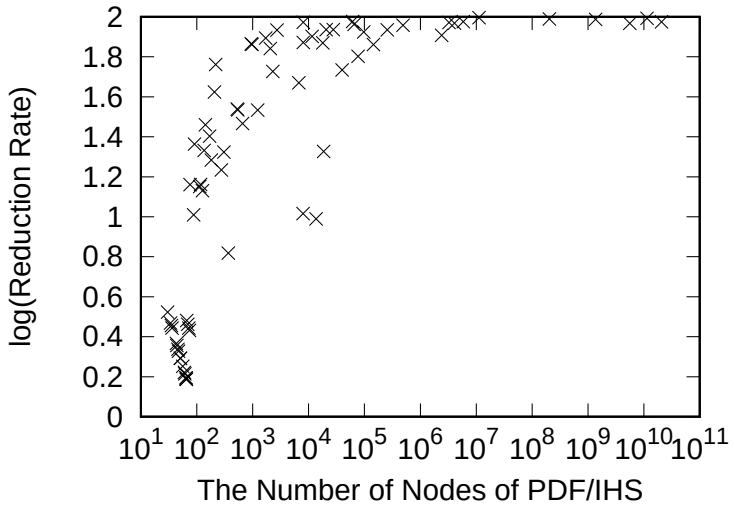

図 3–21 ： 2PE の削減率

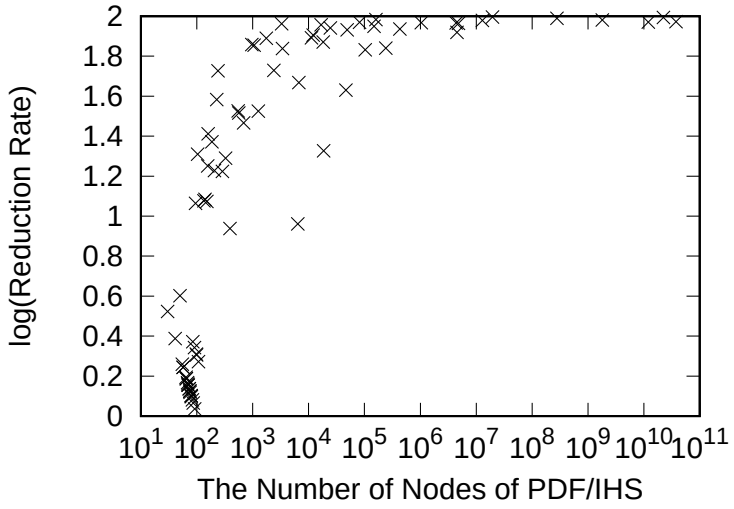

図 3–22 ： 4PE の削減率

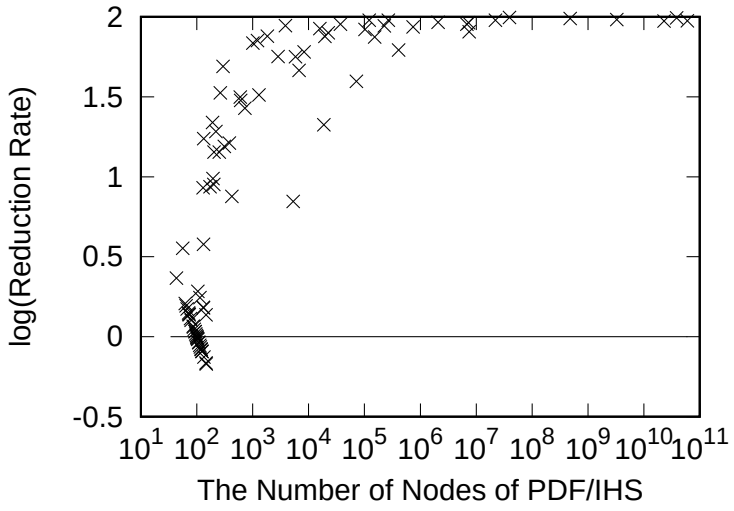

図 3–23 ： 8PE の削減率

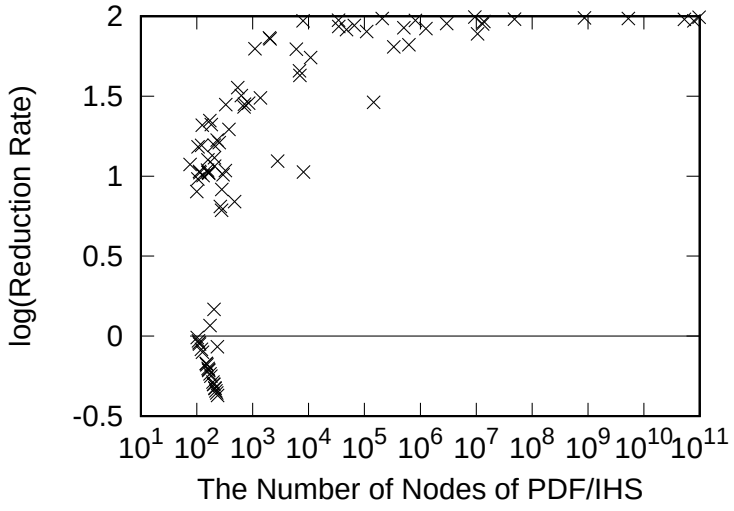

図 3–24 ： 16PE の削減率

3.8　本章のまとめ

本章では，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．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．評価の結果，本章の提案手法を用いることで，DF/IHS 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，本章の提案手法は，DF/IHS 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．

本章で提案した手法は，深さの浅い探索ノードを多く枝刈りすることができるため，問題規模が大きくなるほど高い効果が期待できる．一方で，本章の評価では，問題の特性が削減率にどう影響するかについては評価を行っていない．問題の特性が削減率に与える影響について評価することは，今後の課題のひとつである．また，本章で提案した手法は，本章で示した探索する必要の無い部分問題をすべて削減していない．このため，削減可能な部分問題を検出するための時間と探索ノード数の削減によって短縮される時間とのトレードオフについて検討することで，より高速なアルゴリズムを提案できる可能性がある．上記を踏まえ，問題の特性を基に，本章で示した探索する必要の無い部分問題を枝刈りすることで削減可能な探索ノード数が予測できれば，DF/IHS や PDF/IHS の探索効率をさらに向上することができると考えられる．

第4章

探索の重複領域削減

4.1 はじめに

本章では，代表的な組合せ最適化問題である巡回セールスマン問題 (TSP) において，階層的挟み撃ち探索を用いた並列分枝限定法の探索ノード数を削減する手法を提案し，その有効性を評価する．

TSP は，全都市を 1 回ずつ訪問する経路パターンのうち，経路に与えられたコストが最も小さい経路パターンを求める問題である [43]．本章では，TSP の中でも，都市をユークリッド平面上に配置た問題を扱う．本問題では，都市間の経路に与えられるコストがユークリッド距離となる．このため，2 都市を結ぶ経路は，どちらの都市から出発しても経路のコストが同じである．本問題の厳密解求解に有効な分枝限定法の探索アルゴリズムとして，Held-Karp アルゴリズム [44] が提案されている．

Held-Karp アルゴリズムを用いた分枝限定法は，最小 1-木を用いて分枝操作や限定操作を行う．このため，本手法が生成する探索木は，子ノード数が 2 または 3 の探索ノードから構成される．本手法は，限定操作の効率を高めるために，木探索を行う前に近似解法を用いて精度の高い暫定解を求める．このため，並列探索アルゴリズムに階層的挟み撃ち探索を用いることで，探索早期の最適解探索によって高い高速化率が得られることが期待できる．一方で，階層的挟み撃ち探索を用いた分枝限定法は，同じ階層を左右から 2 台のプロセッサが探索するため，各プロセッサの探索済領域が重

複し，同じノードを探索することがある．分枝限定法は一度探索したノードを再び探索する必要がないため，2 回目以降の探索が無駄な処理となる．Held-Karp アルゴリズムを用いた分枝限定法は，探索ノードの分枝数が少ないため，無駄な処理の割合が大きい．

そこで，本章では，探索の重複領域を削減することにより階層的挟み撃ち探索をさらに高速化する手法を提案する．提案するアルゴリズムは，スレーブプロセッサが探索済みの領域をリーダプロセッサが再探索しないように，スレーブプロセッサだけでなくリーダプロセッサも探索の重複を検出する．また，複数のスレーブプロセッサに探索の重複領域を割り当てないように，再割当にスレーブプロセッサの探索の進行状況を反映する．

以降の節では，まず，第 4.2 節で本章で扱う TSP について述べる．次に，第 4.3 節で Held-Karp アルゴリズムを用いた分枝限定法について述べる．第 4.4 節で探索の重複領域を削減した分枝限定法のアルゴリズムを提案し，第 4.5 節でその有効性を評価する．最後に，第 4.6 節で本章のまとめを行う．

4.2　巡回セールスマン問題

巡回セールスマン問題 (TSP) は，n 都市からなる都市集合において，すべての都市を 1 回ずつ訪問する経路パターンのうち，経路に与えられたコストが最も小さい経路パターンを求める問題である．ここで，都市数 n の TSP を，与えられた都市をノード集合 $V = \{1, 2, \cdots, n\}$ とし，移動可能な都市間をエッジ集合 $E = \{(i, j) \quad (i \in V, j \in V, i \neq j)\}$，都市 i と都市 j 間の距離をエッジ (i, j) のコストを $\mathbf{c}$= $\{c_{ij}\ ((i, j) \in E)\}$ とおくことで，無向重み付きグラフ $G = (V, E, \mathbf{c})$ で表すことができる．このとき，TSP は，グラフ $G(V, E, \mathbf{c})$ の最もコストの小さいハミルトン閉路を求める問題と言い換えることができる．

巡回セールスマン問題は，0-1 整数計画問題として以下のように表すことができる．

$$min. \quad \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \tag{4–1}$$

$$s.t. \quad \sum_{\substack{j=1\\ j\neq i}}^{n} \mathbf{x}_{ij} = 1 \qquad (i = 1, 2, \cdots, n) \tag{4–2}$$

$$\sum_{\substack{i=1\\ i\neq j}}^{n} \mathbf{x}_{ij} = 1 \qquad (j = 1, 2, \cdots, n) \tag{4–3}$$

$$\mathbf{x} \text{ の作るグラフは連結} \tag{4–4}$$

$$\mathbf{x}_{ij} = \{1, 0\} \qquad (i, j = 1, 2, \cdots, n, \quad i \neq j) \tag{4–5}$$

式 (4–2) と式 (4–3) は，$\mathbf{x}$ によって得られた経路が巡回路であることを保障するための制約である．本制約によって，実行可能解のグラフではすべてのノードの次数が 2 となる．

本章では，2 次元のユークリッド巡回セールスマン問題を扱う．本問題は，すべての都市が 2 次元平面上に存在し，都市 i と都市 j 間の距離であるコスト c_{ij} が都市間のユークリッド距離となる問題である．つまり，$c_{ij} = c_{ji}$ となる．このため，以降の節では，エッジ集合を $E = \{(i, j) \quad (i \in V, j \in V, i > j)\}$ とする．

4.3　分枝限定法による巡回セールスマン問題の求解

本章で巡回セールスマン問題の厳密解求解に用いる分枝限定法は，Held-Karp アルゴリズム [44][45] を用いる．Held-Karp のアルゴリズムは，最小 1-木と呼ばれるグラフを用いて分枝操作と限定操作を行う．このため，以下では，まず最小 1-木について述べた上で，分枝操作と限定操作について述べる．

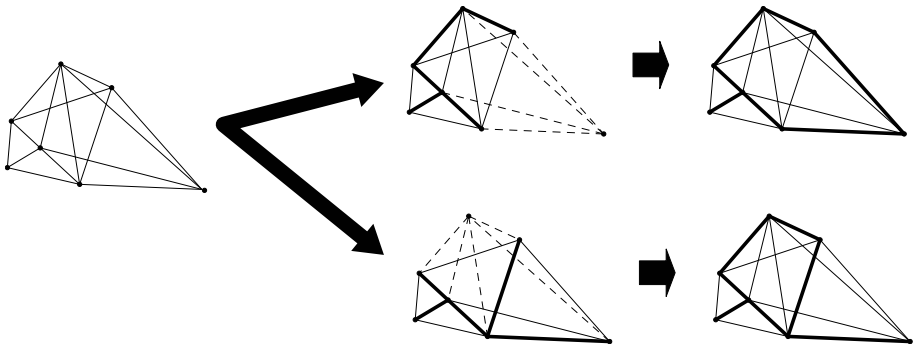

図 4–1 ： 最小 1-木の例

4.3.1　最小 1-木

最小 1-木は，最小コストの 1-木であり，Held と Karp によって定義されたグラフ形状である [46]．図 4–1 に，ノード数 7，エッジ数 15 のグラフに対する 1-木の例を示す．1-木は，図のように，元グラフから任意のノードを取り除いたグラフを作成し，そのグラフから作成した木に，取り除いたノードに接続するエッジを 2 本追加したグラフである．図中では，点線が取り除いたノードに接続するエッジを表しており，太線部分が木および最小 1-木を表す．本例のように，1-木は，閉路を 1 つだけ持つ連結グラフであると言い換えることができる．このため，巡回路も 1-木のひとつである．

最小 1-木は，任意のノードを取り除いたグラフから最小木を作成した上で，取り除かれたノードに接続するエッジのうち最もコストの小さい 2 本を選択し，最小木に加えることで作成できる．このため，最小 1-木の作成にはほとんど時間がかからない．また，最小 1-木を作成する問題は，巡回セールスマン問題の式 (4–2) と式 (4–3) をラグランジュ緩和した問題である．このため，最小 1-木には，元グラフから得られる最小コストの巡回路よりもコストが小さくなるという特性がある．この特性は，最初に除外するノードを任意に選択しても同様に成り立つ．上記のことから，Held-Karp アルゴリズムでは，最小 1-木を利用して下界値を算出する．

以下では，最小 1-木を $T(X,Y)$ とおく．ここで，X は最小 1-木に含まれるエッジの集合であり，Y は元グラフのエッジ集合 E のうち最小 1-木に含まれないエッジの集合である．

4.3.2　限定操作

Held-Karp アルゴリズムの限定操作は，緩和問題に用いた最小 1-木問題に対して反復計算を行うことで，精度の高い下界値を求める．本手法の反復計算では，反復ごとに変化する重み $\pi = (\pi_1, \pi_2, \cdots, \pi_n)$ を用いて式 (4–6) のようにエッジコスト $\mathbf{c}$ を更新する．

$$c'_{ij} = c_{ij} + \pi_i + \pi_j \tag{4–6}$$

式 (4–6) の重み π_i は，ノード i に接続するすべてのエッジに対して重み付を行う．式 (4–6) によって新たに反復ごとに生成される最小 1-木が変化する．このため，本手法は，元グラフから得られる巡回路のコストが変化しないように重み π を設定し，かつ，反復ごとに生成される最小 1-木が巡回路に近づくように重み π を更新することで，最終的に得られる下界値の精度を高める．

重み π は，各反復で生成された最小 1-木の次数から算出する．巡回路では，すべてのノードの次数が 2 になる．このため，最小 1-木を巡回路に近づけるためには，最小 1-木で次数が大きいノードに接続する枝に対して，エッジコストが大きくなるように π を設定すればよい．同様に，最小 1-木で次数が小さいノードに接続する枝に対しては，エッジコストが小さくなるように π を設定する．また，重み π によって巡回路のコストが変化しないようにするためには，巡回路に含まれるエッジに対する重み π の総和が 0 になるように設定する必要がある．これらを踏まえて，Held-Karp のアルゴ

リズムでは，反復重み π を式 (4–7) の漸化式から求める．

$$\pi^{m+1} = \pi^m + t_m \nu_k(\pi^m) \tag{4–7}$$

ここで，π^m は m 回目の反復で得られた π であり，ν_k は i 番目の要素が $d_{ik} - 2$ の n 次ベクトル，t_m はスカラ変数である．ただし，d_{ik} は k 回目の反復で生成した最小 1-木におけるノード i の次数である．

本手法は，k 回目の反復において，式 (4–8) が成り立つことを利用して，最適性を保障している．

$$C* + 2\sum_{i=1}^{n} \pi_i \geq min_k \left[c_k + \sum_{i=1}^{n} \pi_i d_{ik} \right] \tag{4–8}$$

ここで，$C*$ は最小コストの巡回路のコストであり，c_k は k 回目反復で生成した最小 1-木から得られるエッジコストである．式 (4–8) の左辺は，重み π を付加したグラフから得られる最小コストの巡回路のコストである．また，右辺は，重み π を付加したグラフから得られる最小 1-木のコストである．

4.3.3　分枝操作

Held-Karp アルゴリズムの分枝操作は，子ノードの部分問題で生成される最小 1-木が巡回路に近づくように，エッジに対して通る・通らないという情報を付加する．

図 4–2 に，本手法によって生成される探索木の例を示す．本例は，6 都市の巡回セールスマン問題を解く例であり，元問題のグラフは完全結合である．ただし，図中では，経路を見やすくするためにエッジコストを無視して都市を配置している．探索ノード内のグラフの太線は，限定操作で生成した最小 1-木を表す．また，点線は親ノードにおいて通らないように制約条件を加えられたエッジであり，最小 1-木のグラフの中で特に太い線は親ノードにおいて必ず通るように制約条件を加えられたエッジである．本

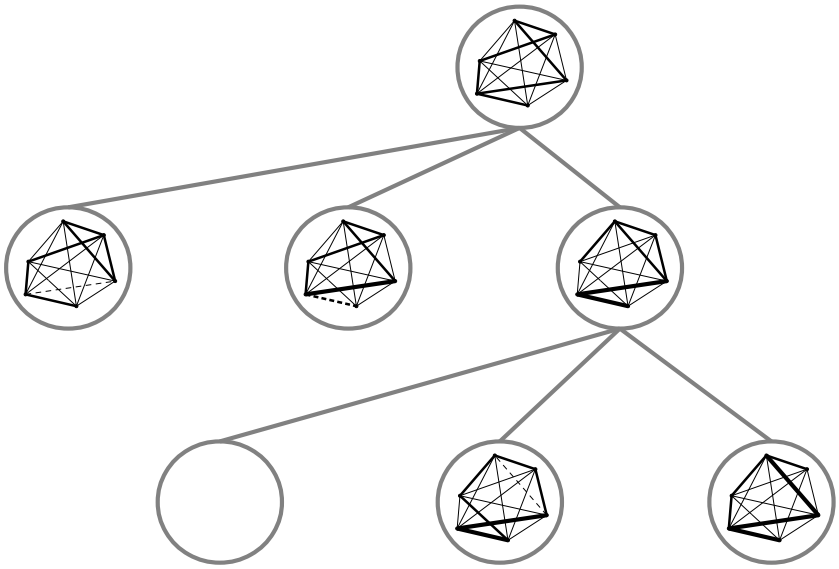

図 4–2 ： Held-Karp アルゴリズムの分枝操作

手法は，図 4–2 の深さ 1 の分枝例のように，最大分枝数が 3 となるように設定するのが良いと言われている．ただし，深さ 2 の分枝例のように，探索木上で浅い位置にある探索ノードによって付加された制約条件によって，3 つの子ノードを生成できない場合がある．このような場合は，本例のように分枝数を 2 とする．

Held-Karp アルゴリズムの分枝操作では，深さ d の探索ノードで生成された最小 1-木 $T(X,Y)$ にを利用してグラフから任意のエッジ h_1，h_2 を選択し，子ノードの制約条件を以下のように設定する．

- $sp_{d+1} = 1$ のノード$\cdots\cdots$ $X \cup h_1 \cup h_2$，　$Y \backslash (h_1 \cup h_2)$
- $sp_{d+1} = 1$ のノード$\cdots\cdots$ $X \cup h_1$，　$Y \cup h_2$
- $sp_{d+1} = 1$ のノード$\cdots\cdots$ X，　$Y \cup h1$

また，2 つの子ノードを生成する際に，各子ノードに追加される制約条件を以下に示す．

- $sp_{d+1} = 1$ のノード$\cdots\cdots$ $X \cup h_1$，　Y

表 4–1 ： 3 つの子ノードを生成する際の制約条件

		h_2	
		通る	通らない
h_1	通る	$sp_{d+1} = 1$	$sp_{d+1} = 2$
	通らない	$sp_{d+1} = 3$	$sp_{d+1} = 3$

表 4–2 ： 2 つの子ノードを生成する際の制約条件

h_1	通る	$sp_{d+1} = 1$
	通らない	$sp_{d+1} = 2$

- $sp_{d+1} = 2$ のノード$\cdots\cdots$ X，　$Y \cup h_1$

ここで，$A \backslash B$ は，集合 A から集合 B を取り除いた集合を表す．また，$sp_{d+1} = 1$ のノードは，第 2.6 節で示したように，深さ d の探索ノードから生成された兄弟ノードのうち探索木上で最も左側に位置する探索ノードを表す．つまり，sp_{d+1} の値が大きいほど探索木左側に位置する探索ノードであることを表す．

上記のように場合分けを行うことで，深さ d の探索ノードの制約条件を満たす全ての実行可能解を探索できる．表 4–1，表 4–2 に，深さ d の探索ノードに対して分枝操作を行った際に生成される子ノードに追加される制約条件を示す．表 4–1，表 4–2 に示すように，どちらの場合でもすべての実行可能解を網羅することができる．

Held-Karp アルゴリズムは，限定操作で生成された最小 1-木を巡回路に近づけるために，最小 1-木で最も次数の高いノードに対して通る・通らないという制約条件を加える．また，通る・通らないという情報を付加するエッジは貪欲法によって選択する．このため，h_1, h_2 には，最小 1-木で最も次数の高いノードに接続し，かつ，最小 1-木に接続する枝のうち，エッジコストが最も大きいエッジと 2 番目に大きいエッジが選

ばれる．

4.4　探索の重複領域を削減した階層的挟み撃ち探索

階層的挟み撃ち探索では，効率よく負荷分散を行うために，スレーブプロセッサが，リーダプロセッサの探索の進行状況を表す LSP を参照し，探索の重複検出操作を行うことで自身の割当領域を動的に決定する．このため，スレーブプロセッサがあとから探索するタイプの探索の重複が生じた場合，スレーブプロセッサは，無駄な探索を行わずに割当領域の探索を終了することができる．一方，リーダプロセッサがスレーブプロセッサから受け取る情報は，スレーブプロセッサが割当領域の探索を終了したか終了していないかという情報だけである．つまり，リーダプロセッサは，スレーブプロセッサがどのノードを探索済みか知ることができない．リーダプロセッサは，SSP を参照せずに探索を行うため，リーダプロセッサがあとから探索するタイプの探索の重複が生じた場合，すでにスレーブプロセッサが探索済みの領域を探索する可能性がある．このように，複数のプロセッサが同じノードを探索すると，2 回目以降の探索は無駄になる．本章では，探索木上において，探索の重複によって複数のプロセッサが探索した領域を，探索の重複領域と呼ぶ．

以下では，探索の重複領域の探索によるオーバヘッド，および，探索の重複検出操作による探索の重複領域の削減手法について述べる．

4.4.1　探索の重複領域の探索によるオーバヘッド

階層的挟み撃ち探索では，リーダプロセッサがあとから探索するタイプの探索の重複が生じると，探索の重複領域が生じる可能性がある．探索の重複が生じると，リーダプロセッサは，すでに探索済みのノードを含む階層を探索することになる．また，

リーダプロセッサが再割当するノードは，LSP 上に存在するため，すでに探索済みのノードを含む領域がスレーブプロセッサに割り当てられる場合がある．このように，探索の重複領域は，リーダプロセッサとスレーブプロセッサが探索する領域と，複数のスレーブプロセッサが探索する領域に分けることができる．図 4–3 に，リーダプロセッサがあとから探索するタイプの探索の重複が生じる例を示す．図 4–3 では，スレーブプロセッサがノード D を探索中に，リーダプロセッサがノード A の探索を開始する．本例では，ノード A を根とする部分探索木の探索が終了していないため，リーダプロセッサもノード A を根とする部分木を探索する．このとき，スレーブプロセッサがノード A 内における分枝条件などをすでに計算済みであるため，リーダプロセッサは，スレーブプロセッサと同じ計算を行うことになる．リーダプロセッサが次に探索するノード B においても同様に，2 台のプロセッサが同じ計算を行う．従って，図 4–3 の例では，リーダプロセッサとスレーブプロセッサの 2 台でノード A, B を探索するため，効率が悪くなる．また，本例において，図中のプロセッサとは別に待ち状態のスレーブプロセッサが存在するとき，リーダプロセッサは，すぐにノード A を根とする部分探索木を待ち状態のスレーブプロセッサに割り当てる．このとき，図中のスレーブプロセッサが早い段階でリーダプロセッサに探索が重複したことを通知しても，斜線のノードを 2 台のプロセッサが探索するため，効率が悪くなる．さらに，待ち状態のスレーブプロセッサが存在しない場合でも，図中のスレーブプロセッサが探索の重複を検出して待ち状態になると，ノード A が同じプロセッサに再割当される．この場合，同一プロセッサで斜線のノードを 2 回探索するため，効率が悪くなる．

探索の重複領域は，リーダプロセッサがあとから探索するタイプの探索の重複が生じたときに発生する．しかし，各階層の探索の重複が，スレーブプロセッサがあとから探索するタイプになるか，リーダプロセッサがあとから探索するタイプになるかは探索の重複が起こる前に知ることはできない．また，各プロセッサの探索の進行速度

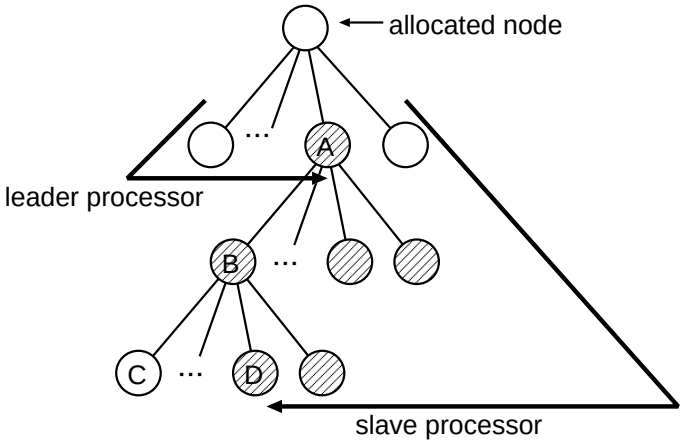

図 4–3 ： 探索の重複領域の例

はバックグラウンドジョブなどの影響を受けて変化するため，同じ問題を求解する場合でも，同じタイプの探索の重複が必ず起こるとは限らない．このため，リーダプロセッサがあとから探索するタイプの探索の重複の発生回数や，探索の重複領域の広さを，求解終了前に予測することは難しい．

ここで，ある割当ノードによって設定される割当領域に占める探索の重複領域の割合は，スレーブプロセッサが割当ノードの子ノードを根とする部分探索木の探索を終了する直前に探索の重複が生じた場合に，最も大きくなる．このため，割当ノードの子ノードを根とする各部分探索木のノード数は等しいと仮定すると，ある割当領域に対する探索の重複領域のノード数の最大値 $N_{overlap}$ は，式 (4–9) で表すことができる．

$$N_{overlap} = \frac{(\text{割当領域のノード数})}{(\text{割当ノードの子ノード数})} \tag{4–9}$$

式 (4–9) より，各ノードから分枝する子ノード数が多い探索木ほど，割当領域に占める探索の重複領域の割合が小さくなるといえる．

分枝限定法の階層的挟み撃ち探索は，分枝限定法のひとつである DF/IHS[26] の並列探索手法として提案された手法である [16]．DF/IHS で生成する探索木は，各ノー

ドが，それぞれ実行可能なレディタスクの数分の子ノードを持つので，各ノードから生成される子ノード数が多い．また，各ノードから子ノードを生成する際に，プライオリティリストを用いるため，各ノード内の計算時間が短い．このため，無駄な探索を検出する処理よりも，無駄な探索を行う処理の方が，処理時間が短くなる場合が多く存在すると考えられる．このように，DF/IHS では，探索の重複によるオーバヘッドは非常に小さく，階層的挟み撃ち探索が有効に働いたと考えられる．

一般的に，分枝限定法の分枝数は，ノードの増加を抑えるために，2 か 3 が良いと言われている [20]．この考え方に基づいて生成した探索木では，ある割当領域に対する探索の重複領域の最大ノード数が，式 (4–9) により，割当領域の 1/3 となる．また，組合せ最適化問題の中には，0-1 計画問題として定式化される問題がある．このような問題の求解では，分枝操作において二分木を生成する場合がしばしばある [1]．二分木探索中に探索の重複領域が生じた場合，スレーブプロセッサが探索したすべての領域が探索の重複領域となる．このため，各ノードから分枝する子ノード数が少なくなるように設計された分枝限定法において階層的挟み撃ち探索を行う場合，探索の重複領域が大きくなり，この領域の探索によるオーバヘッドが無視できなくなる．

4.4.2　探索の重複領域削減手法

第 4.4.1 項で示したように，階層的挟み撃ち探索は，リーダプロセッサが SSP を参照せずに探索し，探索の重複領域が生じるため，探索の効率が落ちる．そこで，本章では，複数のスレーブプロセッサが探索する領域および，リーダプロセッサとスレーブプロセッサが探索する領域を削減する．

本章で提案する手法では，まず，リーダプロセッサとスレーブプロセッサによる同一ノードの探索を削減するために，リーダプロセッサがスレーブプロセッサの探索済ノードの情報を利用して探索を行う．このとき，リーダプロセッサによる探索の重複

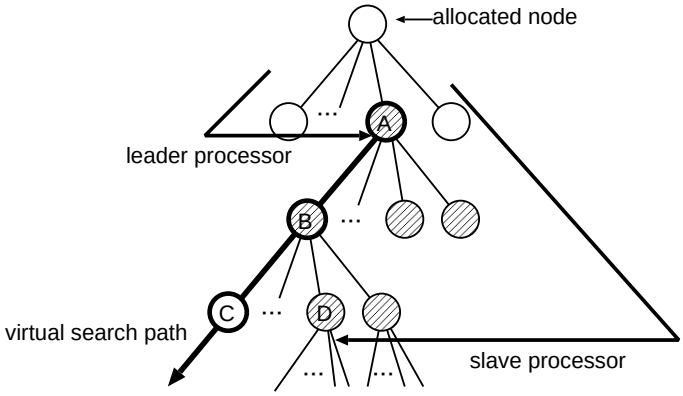

図 4–4 ： リーダプロセッサの仮想的な探索経路

領域の探索は，スレーブプロセッサの探索済ノードの情報を参照するだけで良いため，探索の重複領域を探索するオーバヘッドを削減できる．

次に，複数のスレーブプロセッサによる同一ノードの探索を削減するために，リーダプロセッサがあとから探索するタイプの探索の重複が生じたスレーブプロセッサに再割当する領域は，探索の重複が生じたときに，そのスレーブプロセッサが探索していたノードが含まれるように設定する．これにより，探索の重複が生じたスレーブプロセッサの探索中ノードが他のスレーブプロセッサに再割当されるのを防ぐ．また，このスレーブプロセッサは，再割当後に SP 値を再設定することなく，直前までの探索に使用していた SP 値を用いて探索を継続できるため，同一プロセッサによる再探索も防ぐことができる．ここで，探索の重複が生じたスレーブプロセッサは，探索の重複が生じた階層において，再割当されたノードよりも深さの浅いノードを割当ノードとする探索領域をすべて探索済みである．そこで，探索の重複が生じたスレーブプロセッサの探索済み領域を他のスレーブプロセッサが再探索しないように，本章の提案手法のリーダプロセッサは，現在探索中の階層において，探索の重複の生じたスレーブプロセッサに再割当されたノードより深さの浅いノードを再割当しない．

上記の操作を行うにあたり，リーダプロセッサは，新たなノードの探索を開始する前に，そのノードで探索の重複の有無を確認する必要がある．従来と同様にスレーブプロセッサのみで探索の重複を検出する場合，各スレーブプロセッサは，探索の重複が生じた場合だけでなく重複が生じていない場合も，探索の重複が生じたかどうかという情報をリーダプロセッサに対して通知する必要がある．また，リーダプロセッサは，新たにノードを探索するたびに，同じ階層を探索するスレーブプロセッサから探索の重複が生じたか生じていないかという通知を待つ必要がある．このとき，リーダプロセッサがスレーブプロセッサからの通知を待つオーバヘッドが生じるため，再割当オーバヘッドが大きくなる．よって，本章の提案手法では，リーダプロセッサがスレーブプロセッサの探索の進行状況を把握するために，スレーブプロセッサだけでなリーダプロセッサも探索の重複検出操作を行う．

図 4–4 に，図 4–3 の例における探索の重複領域の削減例を示す．リーダプロセッサとスレーブプロセッサによる同一ノードの探索を削減するために，図 4–4 中のリーダプロセッサは，自身の探索経路 LSP にノード A, B の情報を書き加えることで，自身が次に探索するノードを C に変更する．複数のスレーブプロセッサによる同一ノードの探索を削減するために，ノード D を探索中のスレーブプロセッサに再割当されるノードは，ノード B である．また，ノード A により定められる割当領域は，探索が終了した領域として登録され，今後どのスレーブプロセッサにも割り当てられない．図 4–4 より，リーダプロセッサがスレーブプロセッサから参照するノード A, B や，図中のスレーブプロセッサに再割当されるノードであるノード B は，リーダプロセッサがこれから探索するノードであり，リーダプロセッサが探索中のノード A を根とする部分木の最も左側のノードを通る太線で表された経路上に存在することが分かる．このため，本章の提案手法では，探索の重複が生じたスレーブプロセッサの SSP において，探索の重複が生じた時点の割当ノードを根とする部分木の最も左側のノードを検出するこ

とができれば，上記の操作を行うことができる．

リーダプロセッサが，新たにノードを探索するたびに，スレーブプロセッサの探索済ノードを 1 ノードずつ参照すると，リーダプロセッサの再割当中にスレーブプロセッサが探索の重複を検出してしまい，無駄な探索が生じる．このため，リーダプロセッサは，スレーブプロセッサから参照する必要のあるノード情報を一度に参照することで，スレーブプロセッサが待ち状態になることを防ぎ，無駄な探索を削減する．スレーブプロセッサのノード情報を一度に参照するために，本章の提案手法におけるリーダプロセッサは，探索の重複検出操作によって，ノードの位置情報を比較することで，参照する必要のあるノードを一度に検出する．ここで，探索の重複が生じる可能性のあるスレーブプロセッサの割当ノードの深さを d としたとき，LSP の要素数は $d+1$ となるため，$d+2$ 以上の深さのノードに対する比較操作を行うことができない．そこで，リーダプロセッサがこれから探索する経路を仮想的な LSP として $LSP_v = [\ lspv_1, \cdots, lspv_i, \ldots, lspv_\infty\ ]$ を設定することで，$d+2$ 以上の深さのノードに対する比較操作を行う．つまり，本章の提案手法におけるリーダプロセッサは，従来の階層的挟み撃ち探索の処理に加えて，LSP_v と探索の重複が生じる可能性のあるスレーブプロセッサの SSP に対して探索の重複検出操作を行う．ただし，LSP_v は，$d+2$ 番目以降の SP が図 4–4 中の太線のノードを辿るように，$d+1$ 番目までの SP に LSP を用い，$d+2$ 番目以降の SP に位置情報が 1 の SP を用いる．

4.4.3　探索の重複検出操作の頻度

探索の重複は，なるべく早い段階で検出することで，探索の重複領域を少なくすることができる．本章の提案手法では，各プロセッサが新たなノードを探索する際に探索の重複検出操作を行うと，自身があとから探索するタイプの探索の重複を早い段階に発見することができる．つまり，各プロセッサが，自身があとから探索するタイプ

の探索の重複さえ検出すれば，すべてのタイプの探索の重複を検出することができる．

リーダプロセッサがあとから探索するタイプの探索の重複が生じる可能性があるのは，リーダプロセッサが，前に探索したノードと同じ深さのノードか，前に探索したノードより浅いノードを新たに探索するときである．このとき，探索の重複が生じたノードを探索済みのスレーブプロセッサは，リーダプロセッサが新たに探索するノードの親ノードが割り当てられたスレーブプロセッサだけである．また，リーダプロセッサは，限定操作によって複数の階層に渡ってバックトラックを行う場合がある．バックトラックにより通過したノードを割当ノードとする階層の探索が終了するので，リーダプロセッサは，すべての階層で探索の重複検出操作を行う必要がない．このため，リーダプロセッサは，すべてのバックトラックが終了してから，探索の重複検出操作を行う．このとき，リーダプロセッサから探索が重複したという情報を通知されなかったスレーブプロセッサは，階層的挟み撃ち探索と同様の操作によって割当領域の探索を終了することができる．従って，本章の提案手法においてリーダプロセッサは，新たにノードを探索する際に，1 台のスレーブプロセッサのみに対して探索の重複検出操作を行うだけでよい．

一方，スレーブプロセッサがあとから探索するタイプの探索の重複が生じる可能性があるのは，スレーブプロセッサが割当ノードの子ノードを新たに探索するときである．このため，本章の提案手法においてスレーブプロセッサは，階層的挟み撃ち探索のように，新たにノードを探索するたびに，探索の検出操作を行う必要がない．

これらより，本章の提案手法において探索の重複検出操作を行う回数は，従来の階層的挟み撃ち探索よりも少ないため，探索の重複を検出するオーバヘッドを少なく抑えることができる．また，探索の重複検出操作は，左右から探索するプロセッサの SP を比較する処理を行うだけなので，本操作によるオーバヘッドは小さい．このため，本章の提案手法は，割当ノードから分枝する子ノード数が少ない探索木でも高速に求解

できる．

4.4.4　リーダプロセッサの探索の重複検出操作

第 4.4.2 項で述べた操作を行うために，本章の提案手法のリーダプロセッサは，仮想的な探索経路を用いて探索の重複検出操作を行い，再割当する必要のない階層および再割当ノードを検出する．以下に，リーダプロセッサの探索の重複検出操作による探索の重複領域の削減手法の具体的手順について述べる．ただし，探索の重複が生じる可能性のあるスレーブプロセッサの割当ノードの深さを d とする．

まず，リーダプロセッサは，d 番目の要素までの LSP_v と SSP に対して探索の重複検出操作を行う．d 番目の要素までの LSP_v と SSP の比較において探索の重複を検出した場合，スレーブプロセッサは待ち状態である．この場合，リーダプロセッサは，探索の重複検出操作を終了し，通常の再割当を行う．

次に，リーダプロセッサは，$d+1$ 番目の要素である $lspv_{d+1}$ と ssp_{d+1} の位置情報を比較する．$d+1$ 番目の要素の比較において探索の重複を検出しなかった場合，リーダプロセッサは再割当を行わずに探索の重複検出操作を終了する．一方，$d+1$ 番目の要素までの比較において探索の重複を検出した場合，再割当を行うために，LSP_v とスレーブプロセッサの SP 値である SSP のすべての要素に対して深さの浅い順に探索の重複検出操作を行う．$d+2$ 番目以降の要素に対する比較において，はじめて位置情報が異なる要素が d' 番目であるとする．ただし，SSP のすべての要素が LSP の対応する要素と等しい場合は，d' を SSP の要素数とする．ここで，リーダプロセッサは左側から探索を行うため，LSP_v の上に存在する SP は，リーダプロセッサがこれから探索するノードである．このため，リーダプロセッサは，LSP の位置情報を d' 番目までの SSP と同一の値に書き換える．また，探索の重複が生じたスレーブプロセッサの割当領域のうち，$d'-1$ よりも深さの浅い階層は，探索が終了している．そのため，リーダ

プロセッサは，$d'-1$ よりも深さの浅い階層を探索が終了した階層として登録し，スレーブプロセッサに再割当しない．従って，リーダプロセッサがスレーブプロセッサに再割当するノードは，$d'-1$ 以上の深さを持つノードとなる．

最後に，リーダプロセッサは，再割当可能なノードのうち最も浅いノードである $d'-1$ の深さのノードを探索の重複が生じたスレーブプロセッサに再割当する．探索の重複が生じたスレーブプロセッサが待ち状態でない場合，再割当を受けたスレーブプロセッサは，再割当される割当ノードの位置情報をすでに SSP 上に保持しているため，リーダプロセッサの SP 値をたどることなく割当ノードの探索経路を再生することができる．つまり，本操作において，リーダプロセッサがスレーブプロセッサの割当ノード情報を書き換えても，スレーブプロセッサは，割当ノード情報の変化に関係なく割当ノードの情報を再生できる．このため，リーダプロセッサがスレーブプロセッサに割当ノードの情報が変化したことを知らせなくても，スレーブプロセッサは探索を続けることができる．

リーダプロセッサによる探索の重複検出操作を実装するにあたり，LSP_v は，$d+2$ 番目以降の要素 SP の位置情報が 1 なので，$d+2$ 番目以降の SP をメモリに確保する必要がない．さらに，LSP_v の要素は無限に存在するが，SSP の要素数は有限であるため，リーダプロセッサによる探索の重複検出操作は，有限回の比較で実行することができる．また，スレーブプロセッサの処理内容は従来の階層的挟み撃ち探索から変更する必要がない．このため，本章の提案手法は，従来の階層的挟み撃ち探索にリーダプロセッサによる探索の重複検出操作を加えるだけで実装することができる．図 4–5 に，本章の提案手法が従来の階層的挟み撃ち探索に追加する手続きであるリーダプロセッサの探索の重複検出操作による探索の重複領域を削減する手続き leader_check の擬似コードを示す．

```
leader_check(){
  SSP ← 同じ階層のスレーブ PE の SSP
  for(i=0; i<d+1; i++)
    if(lspv_i ≠ ssp_i)
      return; /* 従来と同様の再割当 */
  if(lspv_{d+1} ≠ ssp_{d+1})
    return; /* 探索が重複していない */
  for(i=d+2; i≤SSP の要素数; i++){
    if(ssp_i ≠ (i,1)) /*lspv_i ≠ ssp_i*/
       break;
    lsp_i ← ssp_i;
  }
  スレーブ PE に深さ i − 1 のノードを再割当
}
```

図 4–5 ： 探索の重複領域削減手続きの擬似コード

4.5　評価

探索の重複領域を削減した階層的挟み撃ち探索の有効性を確認するために，本手法と階層的挟み撃ち探索による求解を行い，求解時間を評価する．

分枝限定法の探索を行う前に，ランダム挿入法 [43] による近似解を求め，暫定解とする．ただし，初期解の精度が異なると限定操作に影響するため，同一問題に対する初期解が等しくなるように乱数を生成する．本手法で新たに内挿する都市は，擬似乱数 as183[47] を用いて選択する．

4.5.1　探索の重複検出操作を排他的に行う手法の評価

本項の評価では，本章の提案手法において，スレーブプロセッサとリーダプロセッサの探索の重複領域検出操作を同時に実行することを禁止して評価を行う．探索の重

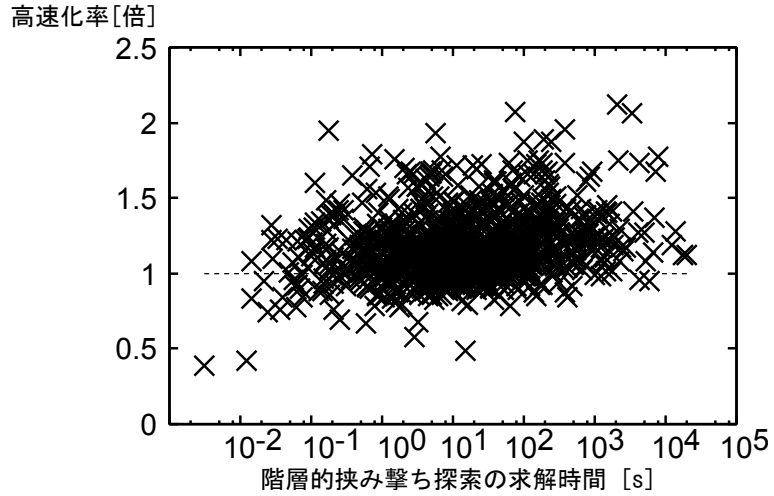

図 4–6 ： 階層的挟み撃ち探索に対する本章の提案手法の高速化率 [倍]

複検出操作が同時に起こった場合はスレーブプロセッサの操作を優先する．また，探索の重複検出操作のオーバヘッドはアーキテクチャによって異なるため，本項の評価では，すべてのノードに対する下界値導出時間を一定とし，階層的挟み撃ち探索と本章の提案手法の探索ノード数を比較する．ただし，階層的挟み撃ち探索における探索の重複検出操作や，再割当に必要な実行時間は短いため，下界値導出以外の処理時間は 0 として扱う．

スレーブプロセッサの探索の重複検出操作は，深さ d の割当ノードを根とする部分探索木において深さ $(d+1)$ のときのみ行う．また，本評価では，各プロセッサが探索したノード数を，各プロセッサが下界値を計算したノード数とし，同一ノードにおいて複数回下界値を導出する場合も計数して測定する．表 4–3 に，4PE による，探索の重複領域を削減した階層的挟み撃ち探索と層的挟み撃ち探索において，各 PE が探索したノード数の和を示す．表 4–3 より, リーダプロセッサが探索の重複検出操作を行うことで，すべての問題で探索ノード数が減少したことが分かる．提案する手法の探索ノード数は，従来の階層的挟み撃ち探索に対して，最大 85% 削減することが確認でき

表 4–3 ： 4PE における総探索ノード数 [個]

問題番号	階層的挟み撃ち探索		探索の重複を削減した階層的挟み撃ち探索
	総探索ノード	無駄な探索	
1	5678339	3668649	2028554
2	4669307	2048253	2038106
3	29033	12551	4498
4	4676206	1650754	1672063
5	1212667	901401	409027

た．最も無駄な探索領域の割合が大きい問題は，問題 5 であり，無駄な探索を行ったノード数が総探索ノードの 74% を占めることが確認できた．

表 4–3 の各問題において，それぞれのノード数の関係は，無駄な探索領域のノード数 n_{opt}，探索の重複領域を削減した階層的挟み撃ち探索の探索ノード数 n_p，階層的挟み撃ち探索の探索ノード数 n_h とすると，$n_{opt} + n_p = n_h$ とならなかった．これは，ノードを探索する順序の変化により，暫定解が更新されるタイミングが変化し，限定操作の効率が変化したためであると考えられる．

従来の階層的挟み撃ち探索の探索ノード数に対して，リーダプロセッサが探索の重複を検出することで最も探索ノード数の割合が減少したのは，問題 3 の場合であった．問題 3 で探索ノード数が削減した理由として，最適解を早い段階に探索することで限定操作の効率が上がり，探索する必要のあるノードの数を減らすことができたためであると考えられる．そこで，問題 3 の最適解が存在するノードの探索木上の位置情報である SP_{opt} を調べたところ，$SP_{opt} = [(0,3),(1,1),(2,3),(3,3),(4,1)\cdots]$ であり，最適解が存在するノードが探索木の右側に存在することが分かった．これより，提案する手法が問題 3 を高速に求解したのは，探索木右側の探索が効率よく行われたからであ

ると考えられる．従来の階層的挟み撃ち探索では，リーダプロセッサがあとから探索するタイプの探索の重複が生じると，すでに探索済みのノードが他のプロセッサの割当ノードになる可能性がある．このため，探索木右側の探索の進行が遅れ，最適解を探索するタイミングが遅くなったと考えられる．一方，リーダプロセッサが探索の重複検出操作を行う階層的挟み撃ち探索では，スレーブプロセッサがすでに探索したノードが他のプロセッサに割り当てられることが無い．このため，右側から探索する領域の探索効率が上がり，早い段階で最適解を探索することができたと考えられる．このように，本章の提案手法は，最適解が探索木の右側に存在する問題において探索ノード数を大きく削減することが可能である．また，最適解が探索木右側に存在する問題は，並列分枝限定法において求解に時間がかかる問題であると考えられるため，本手法は高い高速化率を得ることができると考えられる．

4.5.2　探索の重複検出操作を非排他的に行う手法の評価

本項の評価では，本章の提案手法においてリーダプロセッサとスレーブプロセッサによる探索の重複検出操作が同時に行われることを許容して評価を行う．本評価は，第 4.5.1 項と同様に Opteron885（Dual Core 2.6GHz）が 4CPU，メモリ 16GB を用いる．

表 4–4，表 4–5 に，逐次分枝限定法に対する 4 スレッドによる階層的挟み撃ち探索の高速化率の相乗平均と，逐次分枝限定法に対する 4 スレッドによる本章の提案手法の高速化率の相乗平均を示す．表 4–4，表 4–5 は，スーパーリニアスピードアップが得られた問題と，逐次分枝限定法よりも遅くなった問題，および，それ以外の問題に場合分けして求めた．表 4–5 より，階層的挟み撃ち探索は 14 問，本章の提案手法は 18 問の問題でスーパーリニアスピードアップが得られた．また，高速化率の最大値は，階層的挟み撃ち探索で約 17 倍，本章の提案手法で約 19 倍であり，本章の提案手法の方が高い高速化率が得られた．

表 4–5 より，階層的挟み撃ち探索や，本章の提案手法を用いることで，逐次分枝限定法に対する並列探索手法の高速化率が 1 以下となる問題が確認できた．これは，並列処理によるオーバヘッドが大きかったためであると考えられる．探索木左側に最適解が存在する場合，逐次分枝限定法は，早い段階に最適解を探索できる．このような問題では，探索木右側の評価が悪いノードの多くは，枝刈され，探索されないことが多い．本評価で用いた並列探索手法は，逐次分枝限定法で枝刈されるノードをスレーブプロセッサが探索したため，求解時間が逐次分枝限定法よりも長くなったと考えられる．

また，逐次分枝限定法に対する並列探索手法の高速化率がスレッド数以上となる問題が確認できた．これは，並列処理により，限定操作の効率が向上したためであると考えられる．並列探索手法では，複数のスレッドが同時に異なるノードを探索することで，評価の良い暫定解が逐次探索よりも早い段階に発見することができる．特に本

表 4–4 ： 分枝限定法に対する階層的挟み撃ち探索の高速化率 [倍]

	count	ave	min	max
$0 < R \leq 1$	31	0.84	0.14	1.00
$1 < R \leq 4$	955	1.42	1.00	3.97
$4 < R$	14	6.17	4.43	16.77

表 4–5 ： 分枝限定法に対する本章の提案手法の高速化率 [倍]

	count	ave	min	max
$0 < R \leq 1$	12	0.77	0.13	1.00
$1 < R \leq 4$	970	1.63	1.01	3.98
$4 < R$	18	6.05	4.02	19.11

評価で用いた並列探索手法では，逐次分枝限定法で探索するまでに時間のかかるノードを早い段階に探索することができる．これにより，逐次分枝限定法よりも求解過程で探索する必要のあるノード数が減少し，高速に求解できたと考えられる．

評価に用いた 1000 問のうち，階層的挟み撃ち探索に対する本章の提案手法の高速化率が 1 未満となった問題は 186 問であったが，逐次分枝限定法に対する本章の提案手法の高速化率が 1 未満となった問題は 12 問であった．このため，本章の提案手法を用いると階層的挟み撃ち探索よりも求解に時間がかかる問題においても，多くの問題は，逐次分枝限定法よりも高速に求解できたと言える．

4.5.3　階層的挟み撃ち探索に対する探索の重複領域削減手法の評価

探索の重複領域を削減した階層的挟み撃ち探索の有効性を示すために，30 都市の巡回セールスマン問題 1000 問を一様乱数を用いて生成し，求解する．評価環境は Opteron885（Dual Core 2.6GHz）が 4CPU，メモリは 16GB である．スレーブプロセッサの探索の重複検出操作は，深さ d の割当ノードを根とする部分探索木において深さ $d+1$ のときのみ行う．また，スレーブプロセッサとリーダプロセッサの探索の重複検出操作が同時に起こった場合は，リーダプロセッサの検出結果および再割当結果を優先し，スレーブプロセッサを待ち状態にしないようにする．本評価では，第 3.6.3 項と同様に，求解手法 a, b の各求解時間 t_a, t_b において，a に対する b の高速化率 R を $R\,(a, b) = t_a/t_b$ と定義する．

図 4–6 に，階層的挟み撃ち探索に対する本章の提案手法の高速化率を示す．図 4–6 の横軸は階層的挟み撃ち探索の求解時間，縦軸は階層的挟み撃ち探索に対する本章の提案手法の高速化率である．各手法とも，4 スレッドで実行した．図 4–6 より，1000 問中，階層的挟み撃ち探索に対する本章の提案手法の高速化率が 1 以上となった問題は 814 問，1 未満となった問題は 186 問であった．また，本章の提案手法は，階層的挟み撃ち探索に対して最大約 2.1 倍高速に求解することが確認できた．ここで，高速化率のように倍率を表す値の平均には，一般的に相乗平均が用いられる．図 4–6 の高速化率から，階層的挟み撃ち探索に対する本章の提案手法の高速化率の相乗平均は約 1.2 倍となり，本章の提案手法は多くの問題で階層的挟み撃ち探索よりも高速に求解できると言える．

図 4–6 において高速化率が 1 未満の問題も存在するが，このような問題の多くは，階層的挟み撃ち探索の求解時間が短いため，本章の提案手法でも短い時間で求解することが可能である．ここで，表 4–6 に，図 4–6 の高速化率の度数分布を階層的挟み撃ち探索の実行時間 t[秒] ごとに問題を分類して示す．ただし，度数分布の階級には図 4–6

表 4–6 ： 階層的挟み撃ち探索に対する本章の提案手法高速化率の度数分布

高速化率 r の範囲 [倍]	$t \leq 10$	$10 < t \leq 10^2$	$10^2 < t \leq 10^3$	$10^3 < t$	合計
$r \leq 0.2$	0	0	0	0	0
$0.2 < r \leq 0.4$	1	0	0	0	1
$0.4 < r \leq 0.6$	2	1	0	0	3
$0.6 < r \leq 0.8$	10	2	0	0	12
$0.8 < r \leq 1.0$	98	60	15	4	177
$1.0 < r \leq 1.2$	189	151	57	16	413
$1.2 < r \leq 1.4$	103	82	55	13	253
$1.4 < r \leq 1.6$	30	30	20	6	86
$1.6 < r \leq 1.8$	17	11	14	4	46
$1.8 < r \leq 2.0$	2	0	4	0	6
$2.0 < r \leq 2.2$	0	1	0	2	3
$2.2 < r$	0	0	0	0	0
合計	452	338	165	45	1000

の高速化率を用い，階級幅を 0.2 とする．表 4–6 より，高速化率 r が 0.8 以下の問題は，すべて階層的挟み撃ち探索で 100 秒以内に求解可能であり，本章の提案手法を用いても高速に求解可能な問題である．また，階層的挟み撃ち探索の実行時間 t によらず，高速化率 r が $1.0 < r \leq 1.2$ の範囲をとる問題が多く存在する．しかし，t の範囲ごとの相乗平均は，$t \leq 10$ で約 1.1 倍，$10 < t \leq 10^2$ で約 1.2 倍，$10^2 < t \leq 10^3$ で約 1.2 倍，$10^3 < t$ で約 1.3 倍となり，階層的挟み撃ち探索による実行時間が長い問題ほど高い高速化率が得られやすいことが確認できた．これは，実行時間の長い問題は，求解過程に

表 4–7 ： 階層的挟み撃ち探索と本章の提案手法の探索ノード数

	階層的挟み撃ち探索	本章の提案手法
P_{high}	1995(536)	1807(521)
P_{low}	91(59)	91(60)

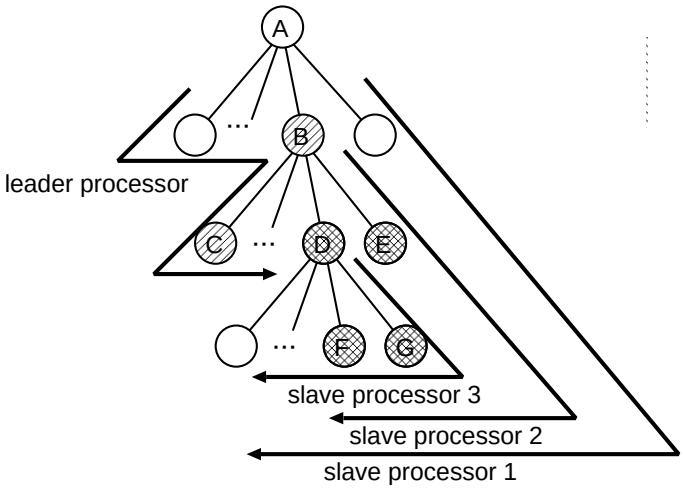

図 4–7 ： 3 台以上で同じ領域を探索する例

おいて探索する必要がある領域が広い問題であり，広い領域が無駄な探索領域になりやすいためであると考えられる．これにより，実行時間の長い問題では，無駄な探索領域が大きく削減することで，リーダプロセッサによる探索の重複検査のオーバヘッドよりも無駄な探索領域の削減による求解時間短縮の方が効果が大きくなったと考えられる．

多くの問題で本章の提案手法が高速に求解できた理由として，階層的挟み撃ち探索よりも本章の提案手法の探索ノード数が減少したことが考えられる．表 4–7 に，図 4–6 の高速化率が最も高い問題 P_{high} および最も低い問題 P_{low} において，階層的挟み撃ち探索と本章の提案手法の各プロセッサが探索したノード数を示す．表中の総探索ノー

ド数は，各プロセッサの探索ノード数の和であり，同一ノードを複数回探索してた場合も総探索ノード数に計数した．また，括弧内の数値は，総探索ノードのうちリーダプロセッサが探索したノード数を表す．表 4–7 より，問題 P_{high} では，本章の提案手法を用いることで，求解に必要な探索ノード数が 188 減少したことが分かる．本章の提案手法は，リーダプロセッサがスレーブプロセッサの探索の進捗状況を監視するため，1 ノードあたりのリーダプロセッサの処理が多い．しかし，表 4–7 より，本章の提案手法を用いることで，リーダプロセッサの探索ノード数が増加する場合がある．これは，本章の提案手法においてリーダプロセッサが分枝条件などをスレーブプロセッサから参照したことにより，探索の重複領域中の分枝を高速化できたためであると考えられる．

本評価で用いた分枝規則では多くのノードで分枝数が 3 となるため，第 4.4.1 項より，1 つの割当領域において本章の提案手法が探索するノード数は，探索の重複領域が最大のとき階層的挟み撃ち探索の 2/3 となる．このため，1 つの割当領域の探索において，階層的挟み撃ち探索に対する本章の提案手法の高速化率は，最大 1.5 倍になると考えられる．一方，図 4–6 より，階層的挟み撃ち探索に対する本章の提案手法の高速化率が 1.5 倍を越える問題が多数存在する．この理由として，階層的挟み撃ち探索において，同一領域を 3 台以上のプロセッサで探索する場合が存在したことが考えられる．図 4–7 に，同一領域を 3 台以上のプロセッサが探索する例を示す．図 4–7 では，ノード A を割り当てられたスレーブプロセッサ 1 がノード F を探索中に，ノード B で探索の重複が生じている．階層的挟み撃ち探索では，スレーブプロセッサ 1 がどんなに早い段階に探索の重複をリーダプロセッサに通知しても，ノード B がスレーブプロセッサ 2 に割り当てられる．スレーブプロセッサ 2 がノード F まで探索したときに，リーダプロセッサがノード D を新たに探索したとする．このとき，ノード D, E, F, G は，2 台のスレーブプロセッサが探索したことになる．また，リーダプロセッサ

は，ノード D をスレーブプロセッサ 3 に割り当てる．スレーブプロセッサ 3 の探索がノード F まで進むと，ノード F, G を 3 台のプロセッサが探索することになる．

4.6　本章のまとめ

本章では，巡回セールスマン問題において，階層的挟み撃ち探索を用いた並列分枝限定法の探索ノード数を削減する手法を提案し，有効性を評価した．本手法は，リーダプロセッサの探索手順に探索の重複を検出する操作を加えることで，階層的挟み撃ち探索によって生じる探索の重複領域を削減する．評価の結果，本章の提案手法は，階層的挟み撃ち探索に対して最大約 2.1 倍，逐次分枝限定法に対して最大約 19.1 倍高速に求解することが確認できた．

本章で提案した手法は，探索の効率が探索木形状のみに依存する手法である．このため，本手法は，子ノード数の少ない探索木を生成する多くの探索アルゴリズムに対して有効に働くと考えられる．また，木探索は，組合せ最適化問題の求解だけでなく広い分野で用いられている．以上を踏まえて，巡回セールスマン問題や分枝限定法に留まらず，多くの木探索アルゴリズムに対する提案手法の有効性の評価が今後の課題である．

他にも，本章の評価では，ランダムに生成した巡回セールスマン問題を求解した．本章の評価に用いた問題は都市数が 30 であるため，比較的小規模な問題である．このため，大規模な問題やベンチマーク問題を用いた評価を行い，他の並列分枝限定法アルゴリズムとの比較や検討を行うことも今後の課題のひとつである．

第5章

結論

5.1 本研究により得られた成果

本章では，これまでに述べてきた提案手法とその評価結果をまとめ，本研究全体の総括を行う．

本研究では，階層的挟み撃ち探索を用いた並列分枝限定法を高速化するために，代表的な組合せ最適化問題のひとつである巡回セールスマン問題および，分枝限定法における階層的挟み撃ち探索が最初に提案されたタスクスケジューリング問題において，探索ノード数を削減する手法を提案し，その有効性を評価した．階層的挟み撃ち探索を用いた分枝限定法は，探索の早い段階に最適解を発見することで，高速に枝刈りすることができる並列探索アルゴリズムであるが，大規模な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．

3 章「無駄な待ち状態の割当て削減」では，タスクスケジューリング問題を解くために提案された分枝限定法のひとつである DF/IHS およびその並列探索アルゴリズム PDF/IHS の探索ノード数を削減する手法を提案した．本手法は，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．このため，下界の精度に関係なく多くの探索ノードを削減することができる．評価の結果，提案手法は，提案手法を用いない探索アルゴリズムと比較して，DF/IHS において最大約 79 倍，PDF/IHS において最大約 96 倍高速に求解できることを確認した．

4 章「探索の重複領域削減」では，巡回セールスマン問題の求解において，階層的挟み撃ち探索を用いた分枝限定法の探索ノード数を削減する手法を提案した．本手法は，スレーブプロセッサの探索済領域をリーダプロセッサが探索しないように，スレーブプロセッサだけでなくリーダプロセッサも探索の重複を検出する．また，複数のスレーブプロセッサが同じノードを探索しないように，スレーブプロセッサの探索経路を反映した再割当を行う．評価の結果，提案手法は階層的挟み撃ち探索に対して相乗平均で約 1.2 倍，最大約 2.1 倍高速に求解できることを確認した．

以上の結果より，巡回セールスマン問題および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を確認できた．

5.2　今後の課題

本研究に関する今後の課題を以下に述べる．

3 章で提案した PDF/IHS において無駄な待ち状態の割当てを削減する手法は，本章で示した探索する必要の無い部分問題をすべて削減していない．本手法は，深さの浅い探索ノードを多く枝刈りすることができるため，問題規模が大きくなるほど高い効果が期待できる．このため，削減可能な部分問題を検出するための時間と探索ノード数の削減によって短縮される時間とのトレードオフについて検討することが今後の課題である．

4 章で提案した階層的挟み撃ち探索の探索の重複領域を削減する手法は，探索の効率が探索木形状のみに依存する手法である．このため，本手法は，巡回セールスマン問題や分枝限定法に留まらず，子ノード数の少ない探索木を生成する多くの探索アルゴリズムに対して有効に働くと考えられる．巡回セールスマン問題と異なる問題や，分枝限定法以外の木探索アルゴリズムに対する提案手法の有効性の評価が今後の課題である．また，4 章の評価では，ランダムに生成した巡回セールスマン問題を求解し

た．このため，大規模な問題やベンチマーク問題に対する評価も今後の課題のひとつである．

謝辞

本研究は，千葉工業大学大学院 情報科学研究科 情報科学専攻の在学期間中になされたものです．非常に多くの方々の協力によって本研究を博士論文としてまとめることができました．ここに謝意を表したいと思います．

初めに，御指導ご鞭撻を賜りました指導教員の富井規雄 教授に深く感謝をいたします．また，研究活動をはじめてから現在まで，長い期間にわたって御指導いただいた前川仁孝 教授には，多くの助言をいただきましたことを，ここに深謝いたします．本博士論文を完成させるにあたり，有益な御指摘・御助言をいただきました国立情報学研究所 合田憲人 教授に深謝いたします．

宮崎収兄 教授，藤田茂 教授をはじめとする本学 情報工学科の先生方には，研究面だけでなく多くの面で支えていただきました．本当にありがとうございました．

研究室内外を問わず，多くの方に研究面や生活面でお世話になりました．先輩方や同輩，後輩に感謝いたします．特に，論文の証明部分に関しては，ゼミやディスカッションによって精錬したことでこのように成果としてまとめることができたのだと思います．

そして最後に，両親，友人，および本論文を執筆できる環境へおいてくださった全ての皆様に感謝いたします．

2019 年 2 月 13 日

参考文献

(1) 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).

(2) B. コルテ，J. フィーゲン著，浅野孝夫，平田富夫，浅野泰仁訳 : 組合せ最適化—理論とアルゴリズム—，シュプリンガー・フェアラーク東京株式会社 (2005).

(3) Sait, S. M.，Habib Youssef 著，白石洋一訳：組合せ最適化アルゴリズムの最新手法—基礎から工学応用まで—，丸善株式会社 (2002).

(4) 人工知能学会編：人工知能学会辞典，共立出版 (2005).

(5) Hiroyuki Mori, K. T.: Parallel Simulated Annealing for Power System Decomposition, *IEEE Transactions on Power Systems*, Vol. 9, No. 2, pp. 785–789 (1994).

(6) 久保典弘，村本勝洋，下薗真一：平面巡回セールスマン問題の高速な近似アルゴリズム，情報処理学会研究報告アルゴリズム，Vol. 74, No. 7, pp. 49–56 (2002).

(7) 中山慶一郎：分枝限定法について (1987).

(8) 今井正治，吉田雄二，福村晃夫：分枝限定アルゴリズムの並列化とその評価，電子情報通信学会論文誌 D，Vol. J62-D, No. 6, pp. 403–410 (1979).

(9) Li, G. J. and Wah, B. W.: Coping with anomalies in parallel branch-and-bound algorithms, *IEEE TRANS.Comput.*, Vol. C-35, No. 6, pp. 568–573 (1986).

(10) Bernard, G. and Teodor, Gabriel, C.: Parallel Branch-and-Branch Algorithms: Survey and Synthesis, *OPERATIONS RESEARCH*, Vol. 42, No. 6, pp. 1042–1066 (1994).

(11) 宮本隆宏，岩崎一彦，萩原兼一：分枝限定法の並列化と並列計算機での実行，電子情報通信学会技術研究報告．FTS, フォールトトレラントシステム， Vol. 95, No. 23, pp. 15–22 (1995).

(12) 岡本秀輔，渡辺一衛，飯塚肇 : 分散記憶型マルチプロセッサにおけるフローショップスケジューリング問題の並列解法，電子情報通信学会論文誌 D-I，Vol. J77-D-I, No. 6, pp. 415–423 (1994).

(13) 川口剛，真栄田保，増山博 : 非同期式並列分枝限定アルゴリズム，電子情報通信学会論文誌 D-I，Vol. J74-D-I, No. 1, pp. 23–32 (1992).

(14) 池上敦子，青柳雄大，飯塚肇 : 分散メモリ型マシンにおける並列整数計画法，電子情報通信学会論文誌 D，Vol. J74-D-I, No. 9, pp. 801–808 (1992).

(15) 吉松敬仁郎，安浦寛人 : 並列部分問題探索法による組合せ最適化問題の並列処理，情報処理学会研究報告，ハイパフォーマンスコンピューティング，Vol. 96, No. 22, pp. 49–54 (1996).

(16) 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).

(17) 島内剛一，野下浩平，伏見正則，有沢誠，浜田穂積 : アルゴリズム辞典，共立出版 (1994).

(18) 監役者 : 森村英典，刀根薫，伊理正夫 : 経営科学 OR 用語大辞典，朝倉書店 (1999).

(19) 茨木俊秀：組合せ最適化の理論，電子通信学会 (1979).

(20) 今野浩史，鈴木久敏：ORライブラリー 7 整数計画法と組合せ最適化，日科技連出版社 (1982).

(21) 大西克実，榎原博之，中野秀男：並列分枝限定法における分枝変数の選択に関する考察，電子情報通信学会論文誌 D，Vol. J84-D-I, No. 9, pp. 24–32 (2001).

(22) Valenzuela, C. L. and Jones, A. J.: Estimating the Held-Karp lower bound for the geometric TSP, *European Journal of Operational Research*, Vol. 102, pp. 157–175 (1997).

(23) Padberg, M. and Rinaldi, G.: A Branch-and-Cut Algorithm for Resolution of Large-Scale Symmetric Traveling Salesman Problems, *SIAM Review*, Vol. 33, No. 1, pp. 60–100 (1991).

(24) Fischetti, M., Gonzalez, J. J. S. and Toth, P.: A Branch-And-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem, *Operations Research*, Vol. 45, No. 3, pp. 378–394 (1997).

(25) 藤江哲也：整数計画問題に対する分枝カット法とカットの理論，オペレーションズ・リサーチ: 経営の科学， Vol. 48, No. 12, pp. 935–940 (2003).

(26) Hironori, K. and Seinosuke, N.: Practical Multiprocessor Scheduliing Algorithms for Efficient Parallel Processing, *IEEE TRANS.Comput.*, Vol. C-33, No. 11, pp. 1023–1029 (1984).

(27) 合田憲人，二方克昌，原辰次：並列分散計算システム上でのBMI固有値問題解法，情報処理学会論文誌ハイパフォーマンスコンピューティングシステム，Vol. 42, No. SIG 12(HPS 4), pp. 132–141 (2001).

(28) 甲斐宗徳，小林和男，笠原博徳：階層的挟み打ち探索による PROLOG OR 並列処理手法，情報処理学会論文誌， Vol. 29, No. 7, pp. 647–655 (1988).

(29) 笠原博徳，田中久充，伊藤敬介：並列化マルチプロセッサ・スケジューリング・アルゴリズム，情報処理学会研究報告，Vol. 90, No. 90(ARC-83), pp. 91–96 (1990).

(30) 澁谷智則，栗田浩一，甲斐宗徳：B-017 通信を考慮したタスクスケジューリング問題のための並列分枝限定法とその評価 (B 分野:ソフトウェア, 一般論文)，情報科学技術フォーラム講演論文集，Vol. 13, No. 1, pp. 149–154 (2014).

(31) Land, A. and Doig, A.: An Automatic Method of Solving Discrete Programming Problems, *Econometrica*, Vol. 28, No. 3, pp. 497–520 (1960).

(32) El-Rewini, H., Ali, H. and Lewis, T.: Task scheduling in multiprocessing systems, *IEEE Computer*, Vol. 28, No. 12, pp. 27–37 (1995).

(33) Rashtbar, S., Isazadeh, A. and Khanly, L.: A new hybrid approach for multiprocessor system scheduling with genetic algorithm and tabu search (HGTS) (2010).

(34) 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).

(35) 笠原博徳：並列処理技術，コロナ社 (1991).

(36) Coffman, E.: *Computer and Job-Shop Scheduling Tehory*, John Wiley & Sons (1976).

(37) 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).

(38) 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).

(39) 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).

(40) 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)，Vol. 2002, No. 8, pp. 289–296 (2002).

(41) 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).

(42) 笠原研究室 Standard Task Graph Set(STG),
available from <http://www.kasahara.elec.waseda.ac.jp/schedule/>
(accessed 2018-02-13).

(43) 山本芳嗣，久保幹夫：巡回セールスマン問題への招待，朝倉書店 (1997).

(44) Held, M. and Karp, R. M.: The Traveling-salesman problem and minimum spanning trees II, *Mathematical Programming*, Vol. 1, pp. 6–25 (1971).

(45) Hansen, K. H. and Krarup, J.: Improvements of the Held-Karp algorithm for the symmetric traveling salesman problem, *Mathematical Programming North-Holland Publishing Company*, Vol. 7, pp. 87–96 (1974). x.

(46) Held, M. and Karp, R. M.: The Traveling-salesman problem and minimum spanning trees, *Operations Resarch*, Vol. 18, pp. 1138–1162 (1970).

(47) Wichmann, B. A. and Hill, D.: A Pseudo-random Number Generator (1982).

研究業績

論文

[1] 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/HISの高速化”，情報処理学会論文誌，Vol.58, No.3, pp.654-662，(2017年3月15日).

[2] 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年1月22日).

[3] 中村あすか，富永浩文，前川仁孝，”探索の重複領域削減による階層的挟み撃ち探索の高速化”，情報処理学会論文誌コンピューティングシステム(ACS)，Vol.4, No.4, pp.76-84，(2011年10月5日).

講演(査読有シンポジウム)

[1] 中村あすか，富永浩文，前川仁孝，”探索の重複領域削減による階層的挟み撃ち探索の高速化”，先進的計算基盤システムシンポジウム論文集，Vol. 2011, pp.348–355, (2011年5月27日).

講演 (研究会)

[1] 富永浩文，中村あすか，前川仁孝，”レジスタ最適化を用いた CUDA による格子ボルツマン法の高速化手法”，第 95 回プログラミング研究発表会，2017-4-(5), pp. 1–7, (2018 年 1 月 15 日).

[2] 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 法の探索ノード数削減”，第 110 回プログラミング研究発表会，2016-2-(4), pp.1–6, (2016 年 8 月 10 日).

[3] 鈴木宏明，富永浩文，中村あすか，前川仁孝，”GPU を用いた格子ボルツマン法のループ展開を利用した同期オーバヘッド削減による高速化”，第 105 回プログラミング研究発表会，2015-2-(1), pp.1–7, (2015 年 8 月 5 日).

[4] 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”，第 95 回プログラミング研究発表会，2013-2-(6), pp. 1–8, (2013 年 8 月 2 日).

[5] 篠塚研太，富永浩文，中村あすか，前川仁孝，”時間依存性のない線形素子による定数伝播を用いた電子回路シミュレータ SPICE3 の高速化”，情報処理学会研究報告ハイパフォーマンスコンピューティング (HPC)，Vol. 2011-HPC-132, No. 3, pp. 1–6, (2011 年 11 月 28 日).

[6] 中村あすか，前川仁孝，”階層的挟み撃ち探索における探索の重複領域の削減手法”，情報処理学会研究報告ハイパフォーマンスコンピューティング (HPC)，Vol. 2010-HPC-126, No. 28, pp. 1–7, (2010 年 7 月 27 日).

講演 (全国大会)

[1] 野崎真也，中村あすか，前川仁孝，”展開済みノードを用いたLGRF-2による囲碁プログラムFuegoの効率化”，情報処理学会第79回全国大会講演論文集第2分冊，6P-02, pp. 461–462, (2017年3月18日).

[2] 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回全国大会講演論文集第1分冊，4H-06, pp. 197–198, (2016年3月10日).

[3] 鈴木宏明，富永浩文，佐藤一馬，中村あすか，前川仁孝，”GPUを用いた格子ボルツマン法のループ展開を利用したメモリアクセスの局所性向上による高速化”，情報処理学会第77回全国大会第1分冊，3J-06, pp. 71–72, (2015年3月17日).

[4] 佐藤一馬，富永浩文，中村あすか，前川仁孝，松林祐，若松真治，”有限要素法による電磁場解析の透磁率を用いた演算回数削減手法”，情報処理学会第75回全国大会第1分冊，1K-5, pp. 93–94, (2013年3月6日).

[5] 松井南実，富永浩文，中村あすか，前川仁孝，”GPUのキャッシュヒット率向上によるDEMの高速化”，情報処理学会第74回全国大会第1分冊，4K-8, pp. 215–216, (2012年3月6日).

[6] 富永浩文，中村あすか，篠塚研太，前川仁孝，”GPUのための回路方程式求解における命令レベル並列度の評価”，情報科学技術フォーラム講演論文集，Vol. 10, No. 1, pp. 343-344, (2011年9月9日).

[7] 中村あすか，富永浩文，前川仁孝，”探索の重複領域を削減した階層的挟み撃ち探索による実行時間最小マルチプロセッサスケジューリング問題の求解”，情報科学技術フォーラム講演論文集，Vol. 10, No. 1, pp. 379-380, (2011年9月7日).

[8] 篠塚研太，中村あすか，富永浩文，前川仁孝，”ストリーミング SIMD 拡張命令を用いた電子回路シミュレータ SPICE3 の高速化”，情報科学技術フォーラム講演論文集， Vol. 9, No. 1, pp. 339–340, (2010 年 8 月 20 日).

[9] 中村あすか，前川仁孝，”評価関数の精度による階層的挟み撃ち探索の性能評価”，情報科学技術フォーラム講演論文集， Vol. 9, No. 1, pp. 333–334, (2010 年 8 月 20 日).