Vol.2010-HPC-126 No.28
2010/8/4

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

中村　あすか[†1]　富永　浩文[†1]
前川　仁孝[†1]

本稿は，分枝限定法に有効な並列探索手法のひとつである階層的挟み撃ち探索において，探索の重複によって複数のプロセッサが探索する領域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では，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，複数のプロセッサで探索する領域が生じる．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は，各ノードの分枝数が少ない探索木ほど大きくなる．そこで，本稿では，すでに探索済みの領域をスレーブプロセッサに割当てないために，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．本手法の有効性を示すために評価を行い，各ノードの分枝数が少ない探索木において提案手法の有効性を確認した．

A Search Space Reduction Method of Overlapping Search Space for Hierarchical Pincers Attack Search

Asuka Nakamura,[†1] Hirobumi Tominaga[†1]
and Yoshitaka Maekawa[†1]

This paper proposes a reduction method of search space assigned some slave processors for the hierarchical pincers attack search which is one of the efficient parallel algorithm for branch and bound method. The hierarchical pincers attack search searches from right and left side of each subtree in a whole tree by a plurality of processors. Hence, some of processors search the same search space. A search tree consisting of a node connecting a few branches has higher ratio of overlapping search space of a whole tree. Therefore, the proposed method detects mutually overlapping search space by slave processors and a leader processor to prevent allocating already searched subtree to another slave processor. As a results of evaluation, we confirmed our proposed method has enough performance.

1. はじめに

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．多くの組合せ最適化問題の最適解を求める有効な手法のひとつとして，分枝限定法が知られている[1)]．分枝限定法は，求解対象の問題の規模が大きくなると求解に長い時間がかかる．このため，分枝限定法による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[2)3)]．

並列分枝限定法において，プロセッサ間の通信量を削減するために，探索領域を静的に分割する手法[4)] が提案されている．本手法では，各プロセッサごとに負荷が不均等になりやすいので，求解が難しいと考えられる評価の悪いノードから順に，プロセッサにノードを割り当てる．このため，逐次探索で高速に求解可能な評価の良い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．また，並列部分問題探索法[5)] は，ホストプロセッサが最良優先探索を用いて探索し，探索木において一定以上の深さを持つノードを子プロセッサに割り当てる．多くの並列分枝限定法は，限定操作の効率を上げるために，評価の良いノードから順にプロセッサを割り当てる[6)]．このため，評価の良いノードに最適解が存在する場合には高速に求解することができるが，評価の悪いノードに最適解が存在する場合には求解に時間がかかる．共有メモリ環境において，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[7)]．階層的挟み撃ち探索は，探索木を複数のプロセッサが左右から挟み込むように探索する並列探索手法である．分枝限定法における階層的挟み撃ち探索は，逐次探索に比べて高い高速化率を得ることができるが，問題規模の大きな問題では，探索ノード数が多くなり，求解に時間がかかる．そこで，本稿では，分枝限定法による階層的挟み撃ち探索の探索ノード数を削減し，高速化することを目的とする．

階層的挟み撃ち探索は，左右から挟み撃つ形で複数のプロセッサが並列に探索を行う．このため，左右から探索するプロセッサの探索経路が重複し，同じノードを複数のプロセッサで探索することがある．分枝限定法では探索済みのノードを再び探索する必要がないため，この探索は無駄になる．そこで，探索の重複領域を少なくすることで，探索の重複による

†1 千葉工業大学情報工学科
Department of Computer Science, Chiba Institute of Technology

Vol.2010-HPC-126 No.28
2010/8/4

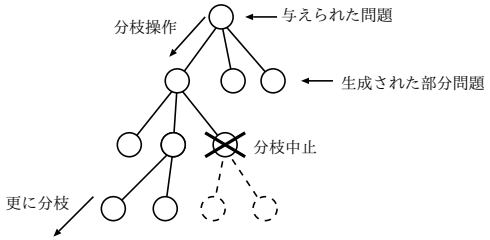

図 1 分枝限定法による問題の分割

オーバヘッドを抑える手法を提案する．本手法では，リーダプロセッサが探索領域の重複を検出することで，探索の重複が発生した時のスレーブプロセッサの探索位置情報を用いた再割当を行う．最後に，本手法の有効性を示すために，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を求解し，探索ノード数を従来の階層的挟み撃ち探索と比較することで，有効性を評価する．

2. 分枝限定法

分枝限定法は，与えられた問題の解空間を分割し，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 1 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を活性問題という．新たに生成された活性問題に対して分枝操作を繰り返し行うと，図 1 のように，各部分問題をノードとした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を知ることができるため，図 1 のような木を探索することで与えられた問題を解くことができる．分枝操作を繰り返すことで解く必要のある問題数は増加するが，部分問題は元の問題よりも規模が小さくなるため，容易に解くことが可能である．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，それぞれの部分問題に対して緩和問題を解くことで下界値を求める．このとき，下界値の評価が暫定解よりも悪い部分問題は，最適解が存在しないと判断できるので，求解する必要が無い．このため，このような部分問題を活性問題から取り除く．図 1 中の破線部は，限定操作により取り除かれた問題を表す．破線のノードを探索する前に，×印のついたノードの下界値は，暫定解より評価が悪いと判断されたとする．このとき，こ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．×印のノードを根とする部分木内に最適解は存在しないため，×印のノードを根とする部分木を探索する必要が無い．

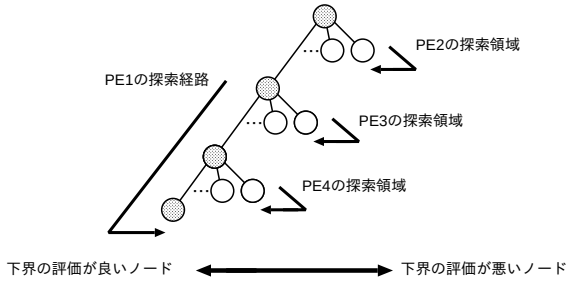

図 2 4PE における階層的挟み撃ち探索の振舞

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[8]．また，限定操作の効率を上げることで，活性問題の問題数を減らすことができ，探索ノード数が少なくなる．このため，並列分枝限定法で高速に求解するためには，なるべく早い段階で最適解を探索するようにアルゴリズムを設計する必要がある．

3. 階層的挟み撃ち探索による分枝限定法の並列化

階層的挟み撃ち探索による分枝限定法の並列化手法は，最適解の探索木上の位置に関係なく共有メモリ環境においてマルチプロセッサ実行時間最小化問題を高速に解くために提案された並列探索手法である[7]．図 2 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，アイドル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．スレーブプロセッサは，割り当てられたノードを根とする部分探索木を右側から探索する．左右から探索するプロセッサの探索領域が重複すると，スレーブプロセッサは，割り当てられた領

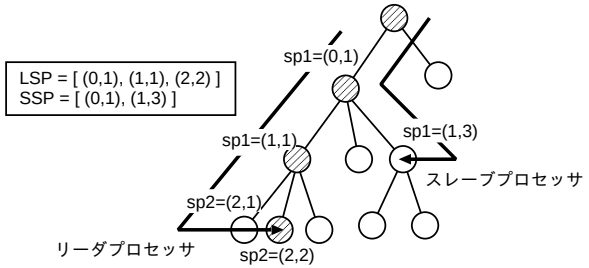

図 3 SP 値の例

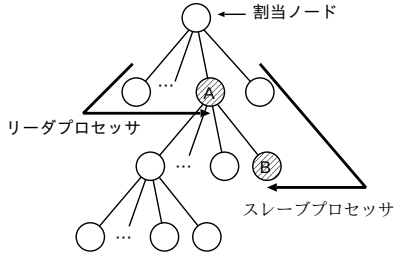

図 4 探索の重複領域が狭い場合

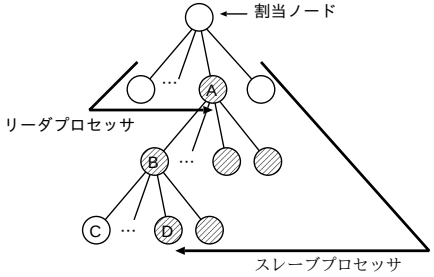

図 5 探索の重複領域が広い場合

域の探索を終了し，リーダプロセッサに新しい領域の割当を要求する．リーダプロセッサは，探索領域の広い処理を割り当てるため，処理の割り当て回数を抑えて，動的に負荷分散を行うことができる．本手法を用いることにより，評価値の良いノードを多くのプロセッサで探索することができる．また，スレーブプロセッサは評価値の悪いノードから探索するため，探索木上における最適解の位置に関係なく，高速に求解することができる．

3.1 探索領域の重複と再割当

探索の重複は，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 種類ある．どちらの場合の探索の重複が生じた場合も，スレーブプロセッサが探索の重複を検出する．本稿では，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には，各ノードの探索木上における位置情報を表すポインタとしてセレクションポインタ (SP) を用いる．SP 値は，根ノードから探索中ノードまでの経路上の各ノードが左から何番目のノードであるかを，深さの浅い順に列挙することで表す．図 3 に SP 値の例を示す．図 3 のように，深さ i の SP 値は $sp_i = (\text{深さ}, \text{枝番号})$ で表す．SP 値は，探索中ノードの深さ分の要素によって構成され，探索木上の経路を記憶する．本稿では，リーダプロセッサの探索経路を表す SP 値列を LSP，スレーブプロセッサの探索経路を表す SP 値列を SSP とおき，スレーブプロセッサ自身に割り当てられたノードの深さを d とおく．

探索の重複検出操作は，スレーブプロセッサが行い，スレーブプロセッサ自身の探索経路 $SSP = [ssp_1, ssp_2, \cdots]$ とリーダの探索経路（$LSP = [lsp_1, lsp_2, \cdots]$）を根ノードから比較する．探索の重複が生じると，深さ $(d+1)$ までに $ssp_i \leq lsp_i$ となる SP 値が存在する．このため，スレーブプロセッサは，SP 値の各要素を，深さの浅い方から順に $(d+1)$ 要素分比較することで，探索の重複を検出することができる．スレーブプロセッサが探索の重複を検出すると，スレーブプロセッサは検出したことをリーダプロセッサに知らせ，アイドル状態となる．また，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起きたことを知らされると，重複を検出し重複を検出したプロセッサに新たな探索領域を再割り当てする．スレーブプロセッサになるべく広い探索領域を割り当てるために，リーダプロセッサは未割当のノードのうち，最も浅いノードを新たな割当ノードに選択する．

3.2 探索の重複によるオーバヘッド

階層的挟み撃ち探索の求解過程では，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起こると，複数のスレーブプロセッサが同じ領域を探索することがある．このとき，あとから探索したプロセッサの探索結果が無駄になる．本稿では，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．図 4 および図 5 に，探索の重複により，同一の領域を複数のスレーブプロセッサが探索する例を示す．

図 4 は，スレーブプロセッサがノード B を探索中であり，リーダプロセッサがスレーブプロセッサのあとからノード A を探索する例である．階層的挟み撃ち探索では，リーダプロセッサが，すぐに，ノード A を根とする部分探索木をアイドル状態のスレーブプロセッサに割り当てる．このため，スレーブプロセッサが探索の重複を早い段階でリーダプロセッサに通知しても，斜線部分のノードを複数のスレーブプロセッサが探索することになる．

図 5 は，スレーブプロセッサがノード D を探索中であり，リーダプロセッサがスレーブプロセッサのあとからノード A を探索する例である．図 5 中の探索の重複領域は図 4 よりも広いため，無駄な探索によるオーバヘッドが大きい．このように，探索の重複領域が最も大

きくなる場合は，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，その部分探索木において探索の重複が生じる場合である．ここで，割当ノードの子ノードを根とするそれぞれの部分探索木のノード数が等しいと仮定する．このとき，最悪の場合における探索の重複領域のノード数は，式（1）で表すことができる．式 (1) より，分枝数の多い探索木ほど，割当領域に占める探索の重複領域の割合が小さくなる．

$$(\text{探索の重複領域のノード数}) = (\text{割当領域}) \times \frac{1}{(\text{割当ノードの子ノード数})} \qquad (1)$$

分枝限定法の階層的挟み撃ち探索は，DF/IHS 法[9] の並列探索手法として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おいて提案された手法である[7]．DF/IHS 法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，探索の重複によるオーバヘッドは非常に小さい．

一般的に，分枝限定法の分枝数は，部分問題の増加を抑えるために，2 か 3 が良いと言われている[1]．この考え方に基づいて生成した探索木では，式（1）により，最悪の場合の探索の重複領域が，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]．二分木の探索において，探索の重複領域をあとから探索したプロセッサがリーダプロセッサである場合，スレーブプロセッサに割り当てられたすべての領域が探索の重複領域となる．このため，各ノードの分枝数の少ない探索木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. 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

本章では，探索の重複領域を削減することで，各ノードの分枝数が少ない探索木を生成する問題に対して有効な階層的挟み撃ち探索手法について述べる．図 4 において，リーダプロセッサがノード B を探索中のスレーブプロセッサに対してノード A を再割り当てすることができれば，ノード B を探索するスレーブプロセッサが 1 台になるため，探索の重複領域を削減することができる．しかし，3.2 節で示したように，スレーブプロセッサだけが探索の重複チェックを行う場合，探索の重複領域が発生する．そこで，本稿では，リーダプロセッサが，探索が重複したスレーブプロセッサの探索中であるノードの位置情報を用いて再割当を行う．これにより，求解全体を通してすべてのプロセッサが探索するノード数が減少するので，求解時間を短縮できる．また，このとき，探索が重複したスレーブプロセッサは前の割当領域の探索を中断することなく次の割当領域の探索をすることができるため，再割当のオーバヘッドが小さくなる．さらに，図 5 のような場合，リーダプロセッサがノード A をスレーブプロセッサに再割当してもノード C を複数のプロセッサが探索することになる．このような場合は，リーダプロセッサは，ノード C を探索が重複したスレーブプロセッサに再割当することで，探索の重複領域を削減する．

4.1 リーダプロセッサによる重複の重複検出操作

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は，探索の重複領域が狭い．このため，リーダプロセッサは，自身があとから探索するタイプの探索の重複を検出し，探索の重複による無駄な探索を削減する．このタイプの探索の重複が生じるタイミングは，リーダプロセッサがバックトラックを行った時である．また，このとき探索の重複が起こる可能性のあるスレーブプロセッサは，親ノードを割当ノードとして通知したスレーブプロセッサである．このため，リーダプロセッサは，そのノードの親ノードを割り当てたスレーブプロセッサに対して探索の重複検出操作を行う．

リーダプロセッサがスレーブプロセッサと同様の方法で探索の重複確認操作を行うと，スレーブの探索位置を再割当に用いることができない．そこで，リーダプロセッサは，探索の重複を検出すると，探索の重複が生じた最も深いノードを調べる．ただし，リーダプロセッサはバックトラック時に探索の重複確認操作を行うため，リーダプロセッサが探索中であるノードの深さは $(d+1)$ である．リーダプロセッサの探索順序は，左側から順に深さ優先探索を行うため，LSP を式 (2) のように仮定して重複検出操作を行う．

$$LSP = [\cdots, (d,i), (d+1,1), (d+2,1), (d+3,1), (d+4,1), \cdots] \qquad (2)$$

図 6 に式 (2) の経路を示す．LSP の要素数を仮想的にスレーブプロセッサと同数にすることで，スレーブプロセッサの SP 値が続く限り，探索の重複検出操作を行うことができる．図 6 中のリーダプロセッサは，ノード A を探索中であるが，ノード B,C を仮想的に探索経路として持つことで，ノード A を根とする部分木内も SP 値を比較することができる．リーダプロセッサは，$ssp_i > lsp_i$ となる深さより 1 つ浅いノードを次の割当ノードしてスレーブプロセッサに通知する．図 6 中のスレーブプロセッサはノード D を探索中であるため，ノード B を新たな割当ノードとする．新たな割当ノードは，SSP と LSP の比較により決定するため，スレーブプロセッサは新たな割当ノードをすでに ssp_d として SSP に保持している．このため，本手法により再割当を受けたスレーブプロセッサは，リーダプロセッサ

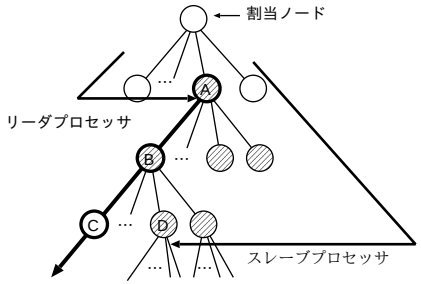

図 6 リーダプロセッサの仮想的な探索経路

の SP 値をたどることなく探索経路を再生することができ，探索が重複したノード内の計算を二重に行う手間を省くことができる．リーダプロセッサは，探索の重複が起きたプロセッサに再割当が終了してから，他のプロセッサの再割当を行う．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．このとき，バックトラックにより通過した階層を割り当てられたスレーブプロセッサは割当領域を探索する必要がなくなるので，リーダプロセッサが各階層ごとに探索の重複検出操作を行うと効率が悪い．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．探索の重複が通知されなかったスレーブプロセッサは，限定操作において，割当領域の探索が終了したことを知ることができる．

階層的挟み撃ち探索は，各プロセッサに広い探索領域を割り当てることで，再割当のオーバヘッドを小さくしている．しかし，割当ノードが葉ノード付近に存在する場合，探索領域が狭くなるため，再割当が頻繁に起こる．このオーバヘッドは，スレーブプロセッサを割当可能な深さに制限を加えるとで削減することができる．このとき，割当制限に設定した深さより深いノードにおいて探索経過の情報が切り捨てられるが，従来の階層的挟み撃ち探索でも，このような現象は生じるため，問題がない．また，ノード内の計算に時間がかかるような探索木に対して階層的挟み撃ち探索を行う場合は，スレーブプロセッサの SP 値だけでなく，スレーブプロセッサが計算したノード情報も受け取ることで，再計算のオーバヘッドを削減することができる．

4.2 スレーブプロセッサによる重複検出操作

スレーブプロセッサは，従来の階層的挟み撃ち探索と同様に探索の重複を検出する操作を行う．ただし，割当ノードから生成される子ノードのうち，最も左側のノードは，必ずリーダプロセッサに割り当てられる．このため，スレーブプロセッサがこのノードを探索する場合，リーダプロセッサの SP 値を確認せずに，探索の重複を検出することができる．

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，重複した領域内の探索はすべて無駄になる．このため，探索の重複によるオーバヘッドを減らすためには，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をなるべく早い段階に検出する必要がある．

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．リーダプロセッサがあとから探索するパターンの探索の重複は，リーダプロセッサが早い段階で検出できる．このため，少ないコストでスレーブプロセッサとリーダプロセッサが同時に探索の重複を検出しないことが保証できる場合，スレーブプロセッサは，リーダが探索中のノードをあとから探索する場合のみ探索の重複を検出する操作を行えばよい．この場合，スレーブプロセッサは，深さ d の割当ノードで指定される割当領域の探索において，深さ $(d+1)$ のノードを探索する際に探索の重複検出操作を行う．ただし，実装環境によっては，スレーブプロセッサが従来の階層的挟み撃ち探索と同様に探索の重複を検出する操作を行うことで，リーダプロセッサにおける探索の重複を検出する操作の負荷を小さくすることができる．一方，スレーブプロセッサとリーダプロセッサが同時に探索の重複を検出しないことを保証するコストが大きい場合は，スレーブプロセッサが，従来手法と同様の頻度で探索の重複検出操作を行う必要がある．

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，探索ノード数を評価する．求解対象の問題には，組み合わせ最適化問題のひとつである巡回セールスマン問題を用いる．以下の節では，巡回セールスマン問題および本稿で生成する探索木の特徴と，スレーブプロセッサとリーダプロセッサによる階層的挟み撃ち探索の有効性について述べる．

5.1 巡回セールスマン問題

評価には，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を用いる．巡回セールスマン問題は，指定されたすべての都市を 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．各ノードの下界値の導出や，分枝規則には，分枝限定法における巡回セールスマン問題の求解に有効であるととされてい

Vol.2010-HPC-126 No.28
2010/8/4

る Held-Karp の手法を用いる[10)11)]．本分枝規則により，本稿で生成する探索木の各ノードの分枝数は，2 または 3 となる．また，根ノードにおいて初期解を求める近似解法には，ランダム挿入法を[12)] を用いる．ただし，初期解の精度が異なると限定操作に影響を与えるため，ランダム挿入法により求まる近似解が各問題で等しくなるように乱数を生成した．

5.2 探索ノード数の測定

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を一様乱数を用いて生成し，求解することで評価する．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，従来の階層的挟み撃ち探索と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を比較する．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 $(d+1)$ のときのみ行い，スレーブプロセッサとリーダプロセッサの探索領域の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，階層的挟み撃ち探索では，各割当領域において最低でも 1 ノード分の無駄な探索が生じる．このため，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て測定する．表 1 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．ただし，表 1 において，総探索ノード数は，無駄な探索ノード数を含むものとする．表 1 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 1 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．分枝限定法の並列探索では，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，$SP_{opt} = [(0,3),(1,1),(2,3),(3,3),(4,1)\cdots]$

表 1 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

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，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．

6. おわりに

本稿では，分枝限定法における階層的挟み撃ち探索において，探索の重複領域を削減するために，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並列探索手法を提案し，有効性を評価した．評価の結果，従来の階層的挟み撃ち探索に比べて探索ノード数が最大 85% 削減できることを確認した．これにより，提案した手法を用いることで，従来の並列分枝限定法において求解に時間がかかる問題ほど，高い高速化率を得られると考えられる．

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).

Vol.2010-HPC-126 No.28
2010/8/4

2) 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).
3) 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).
4) 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).
5) 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).
6) Bernard, G. and Teodor, Gabriel, C.: Parallel Branch-and-Branch Algorithms: Survey and Synthesis, *OPERATIONS RESEARCH*, Vol. 42, No. 6, pp. 1042–1066 (1994).
7) 笠原博徳，伊藤敦，田中久充，伊藤敬介：実行時間最小マルチプロセッサスケジューリング問題に対する並列最適化アルゴリズム，電子情報通信学会論文誌 D，Vol.J74-D1, No.11, pp.755–764 (1991).
8) 今野浩史，鈴木久敏：整数計画法と組合せ最適化，日科技連出版社 (1982).
9) 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).
10) Held, M. and Karp, R.M.: The Traveling-salesman problem and minimum spanning trees, *Operations Resarch*, Vol.18, pp.1138–1162 (1970).
11) Held, M. and Karp, R.M.: The Traveling-salesman problem and minimum spanning trees II, *Mathematical Programming*, Vol.1, pp.6–25 (1971).
12) 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).