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

中村　あすか[†]　富永　浩文[†]
前川　仁孝[†]

本論文は，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を，複数のプロセッサが探索する領域を削減することで高速化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．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．

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

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

This paper proposes a reduction algorithm of search space assigned some processors for the hierarchical pincers attack search which is one of the efficient parallel algorithm for the branch and bound method. In hierarchical pincers attack search, some processors search from right and left sides of each subtree in a whole tree. Hence, some of processors search the same search space. In a search tree consisting of a node connecting a few branches, overlapping search space occupies large search area in search area allocated of a slave processor. Therefore, the proposed method speeds up hierarchical pincers attack search by reducing overlapping search space. In this method, the leader processor detects overlapping search area so that it does not search a node which searched slave processors, and allocates a node so that some slave processors do not search the same node. As a results of evaluation, the speedup ratio of the proposal 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.

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[1)2)]．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につれ，求解時間が長くなる．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[3)4)]．

多くの並列分枝限定法は，早い段階に最適解を探索することで限定操作の効率を高めるために，評価の良いノードを早い段階で探索できるように設計されている[5)6)]．このように設計された手法を用いると，評価の良い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[7)]は，評価の悪い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，評価の悪いノードから優先的に探索でき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ノードに最適解が存在する問

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

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，評価の良いノードに最適解が存在する問題と，評価の悪い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．

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[8]．階層的挟み撃ち探索は，評価の良い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する．このため，評価の良いノードに最適解が存在する場合だけでなく，評価の悪い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．階層的挟み撃ち探索は，同じ階層を 2 プロセッサで探索するため，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．また，リーダプロセッサが，探索の重複領域を複数のプロセッサに再割当する可能性がある．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．そこで，本論文では，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．本手法は，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．

2. 階層的挟み撃ち探索による分枝限定法の並列化

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，最適解の探索木中の位置に関係なく高速に求解するために提案された手法である[8]．以下では，分枝限定法および階層的挟み撃ち探索について述べる．

2.1 分枝限定法

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．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い，生成した探索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，探索終了時の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．本操作では，緩和問題から求めた下界値を用いて各部分問題に最適解が存在するかどうか判定する．

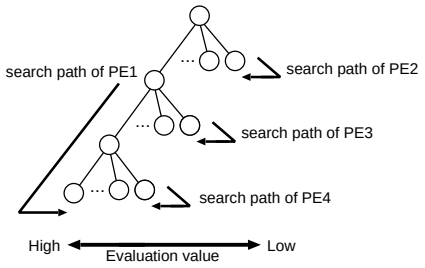

図 1 4PE における階層的挟み撃ち探索の振舞
Fig. 1 Hierarchical pincers attack search by 4 PEs

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[2]．また，限定操作の効率を上げることで，活性問題数が減少するため，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．

2.2 階層的挟み撃ち探索

階層的挟み撃ち探索は，複数のプロセッサが階層的に左右から挟み撃つ形で探索する並列探索手法である[8]．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[9] 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 1** 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[8]．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** 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

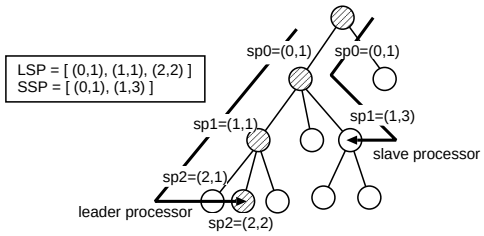

図 2 SP 値の例
Fig. 2 An example of a selection pointer(SP)

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．

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[8]．

3. 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．本論文では，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その大きさ，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．

3.1 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．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類できる．**図 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 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 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に探索の重複を検出しても，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．

3.2 無駄な探索によるオーバヘッドの大きさ

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，求解が終了する前に予測することができない．

ここで，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し，探索の重複領域が最も大きくなる場合を考える．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，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（1）で表すことができる．

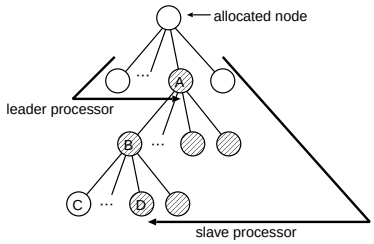

図 3 探索の重複領域の例
Fig. 3 An example of overlapping search area

$$N_{overlap} = \frac{\text{（割当領域のノード数）}}{\text{（割当ノードの子ノード数）}} \quad (1)$$

式（1）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 法[9] の並列探索手法として提案された手法である[8]．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 法では，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[2]．この考えに基づき生成した探索木は，式（1）より，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割当領域の 1/3 となる．また，0-1 計画問題として定式化される組合せ最適化問題の求解では，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全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，オーバヘッドが無視できなくなる．

3.3 探索の重複領域削減手法

3.1 節で示したように，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，本論文では，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．

提案手法では，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領域の探索は，スレーブプロセッサの探索済ノード情報の参照だけで良いため，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が探索中の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．また，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が他のスレーブプロセッサに再割当されないように，提案手法のリーダプロセッサは，現階層において，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，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の有無を知らせる通知を待つため，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，再割当オーバヘッドが大きくなる．よって，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 に，図 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 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．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 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 B は，リーダプロセッサの探索中ノード 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，提案手法では，SSP において，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．

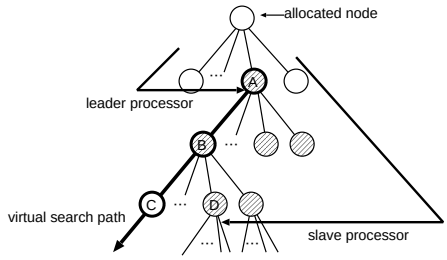

図 4 リーダプロセッサの仮想的な探索経路
Fig. 4 The virtual search path of the leader PE

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があるノード情報を一度に参照することで，無駄な探索が生じることを防ぐ．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，提案手法におけるリーダプロセッサは，リーダプロセッサがこれから探索する経路を仮想的な経路 $LSP_v = [\, lspv_1, \cdots, lspv_i, \ldots, lspv_\infty \,]$ を用いた探索の重複検出操作によってノードの位置情報を比較し，参照する必要があるノードを一度に検出する．つまり，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，LSP_v と探索の重複が生じる可能性が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，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が d であるとき，LSP_v は，$d+2$ 番目以降の SP が図 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．

3.3.1 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．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．

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，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，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，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，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．

3.3.2 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の手続き

第 3.3 節の操作を行うために，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，再割当不要な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，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，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．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．そして，$d'-1$ 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．このため，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．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．

4. 評　　価

提案手法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．求解対象とする問題は，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とする．以下の節では，巡回セールスマン問題および本論文で生成する探索木の特徴と，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について述べる．

4.1 巡回セールスマン問題

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されている Held-Karp の手法[10] を用いる．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[11] 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．分枝規則は，探索木の各ノードの分枝数が2 または3 となるように設定する．具体的には，下界値導出過程で生成されたグラフにおいて，最も次数の大きいノードに接続する枝を 3 本選択し，巡回路に含めるかどうかを設定する．

4.2 求解時間の測定

提案手法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 が 4CPU，メモリ 16GB である．スレーブプロセッサの探索の重複検出操作は，割当ノードの深さが d の割り当て領域で深さ $d+1$ 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，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 5 に，階層的挟み撃ち探索に対する提案手法の高速化率を示す．図 5 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 5 より，1000 問中，階層的挟み撃ち探索に対する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，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．階層的挟み撃ち探索に対する提案手法の高速化率の相乗平均をは約 1.2 倍となり，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可能であることが確認できた．図 5 の高速化率が 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，提案手法でも短い時間で求解することが可能である．また，階層的挟み撃ち探索の実行時間によらず，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した．階層的挟み撃ち探索の実行時間

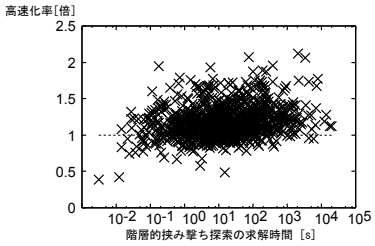

図 5 階層的挟み撃ち探索に対する提案手法の高速化率
Fig. 5 Speedup ratio of the proposed method compared with hierarchical pincers attack search

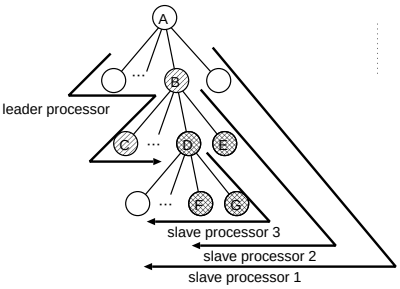

図 6 3 台以上で同じ領域を探索する例
Fig. 6 An example of overlapped search by 3PEs

t の範囲ごとの相乗平均は，$t \leq 10$ のとき約 1.1 倍，$10 < t \leq 10^2$ のとき約 1.2 倍，$10^2 < t \leq 10^3$ のとき約 1.2 倍，$10^3 < t$ のとき約 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．

多くの問題で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提案手法の探索ノード数が減少したことが考えられる．図 5 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提案手法の各プロセッサの探索ノード数の和は，階層的挟み撃ち探索で P_{high} が 1995，P_{low} が 91，提案手法で P_{high} が 1807，P_{low} が 91 であった．問題 P_{high} では，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．また，P_{low} では，各手法の総探索ノード数が等しくなったが，リーダプロセッサが探索したノード数は P_{high} が 59，P_{low} が 60 であった．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階層的挟み撃ち探索より多いにも関わらず，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，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．

本評価で用いた分枝規則では多くのノードで分枝数

表 1 分枝限定法に対する階層的挟み撃ち探索の高速化率 [倍]
Table 1 Speedup ratio of hierarchical pincers attack search compared with branch and bound method

	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 2 分枝限定法に対する提案手法の高速化率 [倍]
Table 2 Speedup ratio of the proposed method compared with branch and bound method

	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

が 3 となるため，第 3.1 節より，1 つの割当領域において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ける階層的挟み撃ち探索に対する提案手法の高速化率は，最大 1.5 倍になると考えられる．しかし図 5 において高速化率が 1.5 倍を越える問題が多数存在した．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 6** 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 6 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 を新たに探索すると，ノード D, E, F, G は，2 台のスレーブプロセッサが探索することになる．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．

最後に，**表 1** 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，**表 2** に逐次分枝限定法に対する 4 スレッドによる提案手法の高速化率の相乗平均を示す．表 1，表 2 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 1 より階層的挟み撃ち探索は 14 問，表 2 より提案手法は 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，提案手法で約 19 倍であり，提案手法の方が高い高速化率が得られた．

5. おわりに

本論文では，探索の重複領域の削減手法を用いた階層的挟み撃ち探索を提案し，有効性を評価した．評価の結果，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．これにより，提案手法は，各ノードから分枝する子ノード数が 2 または 3 のとき，従来の階層的挟み撃ち探索よりも高速に求解できることを示した．

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) 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).
3) 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).
4) 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).
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, Vol. 1995-HPC-060, pp. 49–54 (1996).
6) Bernard, G. and Teodor, Gabriel, C.: Parallel Branch-And-Bound Algorithms: Survey and Synthesis, *Operations Resarch*, Vol. 42, No. 6, pp. 1042–1066 (1994).
7) 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, Vol. FTS-95, No. 23, pp. 15–22 (1995).
8) 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).
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: PART II, *Mathematical Programming*, Vol. 1, pp. 6–25 (1971).
11) 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).