情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 4　76–84 (Oct. 2011)

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[†1]　富永浩文[†1]　前川仁孝[†1]

本論文は，分枝限定法に有効な並列探索手法の 1 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て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てを行う．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,[†1] Hirobumi Tominaga[†1] and Yoshitaka Maekawa[†1]

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 1 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 1 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[5)] 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 1 つとして，階層的挟み撃ち探索が提案されている[6)]．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を行う．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，同じ階層を左右から 2 台のプロセッサが探索するた

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

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てする可能性がある．分枝限定法は 1 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，再割当て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[6]．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である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 1 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，求解する必要のな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 1 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いわれている[2]．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．

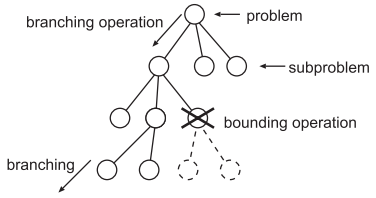

図 1　分枝限定法による問題の分割
Fig. 1　Subproblems by branch and bound method.

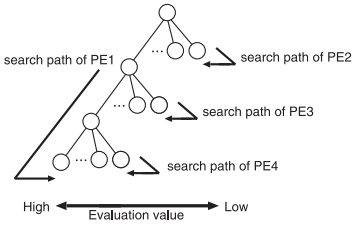

図 2　4PE における階層的挟み撃ち探索の振舞い
Fig. 2　Hierarchical pincers attack search by 4PEs.

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[6]．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[7] 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** 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いを示す．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[6]．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 3** 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，割当てノードを根とする部分探索木を割当て領域とし，割当て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てする．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，少ない再割当て回数で，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[6]．

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，効率良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て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て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て領域の探索を終了したか終了していないかという情報

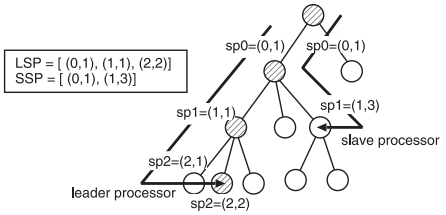

図 3　SP 値の例
Fig. 3　An example of a selection pointer (SP).

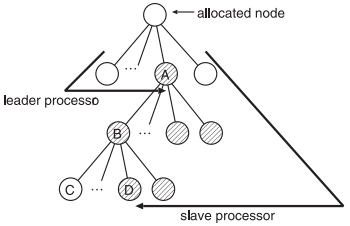

図 4　探索の重複領域の例
Fig. 4　An example of overlapping search area.

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，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ことになる．また，リーダプロセッサが再割当てするノードは，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** 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 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．し

たがって，図 4 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てされる．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，ある割当てノードによって設定される割当て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て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，割当て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，ある割当て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．

$$N_{overlap} = \frac{(\text{割当て領域のノード数})}{(\text{割当てノードの子ノード数})} \tag{1}$$

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，割当て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 1 つである DF/IHS 法[7] の並列探索手法として提案された手法である[6]．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 が良いといわれている[2]．この考え方に基づいて生成した探索木では，ある割当て領域に対する探索の重複領域の最大ノード数が，式 (1) により，割当て領域の 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．

3.2　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み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みノードの情報を参照するだけでよいため，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てする領域は，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てされるのを防ぐ．また，このスレーブプロセッサは，再割当て後に 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，再割当てされたノードよりも深さの浅いノードを割当て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

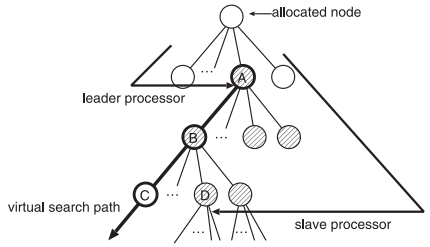

図 5　リーダプロセッサの仮想的な探索経路
Fig. 5　The virtual search path of the leader PE.

再探索しないように，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てされたノードより深さの浅いノードを再割当てしない．

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，再割当て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 5 に，図 4 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 5 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てされるノードは，ノード B である．また，ノード A により定められる割当て領域は，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 5 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て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，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て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みノードを 1 ノードずつ参照すると，リーダプロセッサの再割当て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 1 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 1 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，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 1 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てノードの深さを 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，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 5 中の太線のノードをたどるように，$d+1$ 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．

3.2.1　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

層にわた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て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て領域の探索を終了することができる．したがって，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て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，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，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，割当て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．

3.2.2　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

3.2 節で述べた操作を行うために，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，再割当てする必要のない階層および再割当てノードを検出する．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てノードの深さを 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，通常の再割当てを行う．

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て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，再割当てを行うために，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て領域のうち，$d'-1$ 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てしない．したがって，リーダプロセッサがスレーブプロセッサに再割当てするノードは，$d'-1$ 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，再割当て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てする．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，再割当てを受けたスレーブプロセッサは，再割当てされる割当て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て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てノード情報を書き換えても，スレーブプロセッサは，割当てノード情報の変化に関係なく割当て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て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，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 6** に，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 の擬似プログラムを示す．

```
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 6　探索の重複領域削減手続きの擬似プログラム
Fig. 6　Pseudo-program of reduction of overlapping search space.

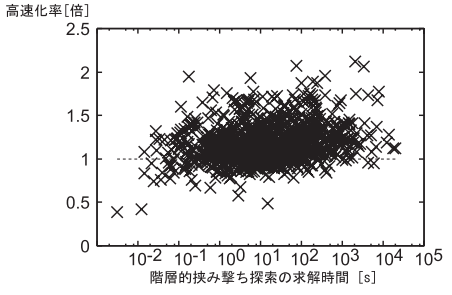

図 7　階層的挟み撃ち探索に対する提案手法の高速化率 [倍]
Fig. 7　Speedup ratio of the proposed method compared with hierarchical pincers attack search.

4. 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．求解対象とする問題は，組合せ最適化問題の 1 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，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．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[8] 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[9] 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で次数が最大のノードに接続する枝から 3 本を選択し，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

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 1,000 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 2.6 GHz）が 4CPU，メモリは 16 GB 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 の割当て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て結果を優先し，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 7 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 7 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 7 より，1,000 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 問であったことが分かる．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．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 7 の高速化率から，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できるといえる．

図 7 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，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 1** に，図 7 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 7 の高速化率を用い，階級幅を 0.2 とする．表 1 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，提案手法を用いても高速に求解可能な問題であることが分かる．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ことが分かる．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されることで，リーダプロセッサに

表 1　階層的挟み撃ち探索に対する提案手法高速化率の度数分布
Table 1　Frequency distribution of speedup ratio.

加速率 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	1,000

表 2　階層的挟み撃ち探索と提案手法の探索ノード数
Table 2　Number of nodes of hierarchical pincers attack search and the proposed method.

	階層的挟み撃ち探索	提案手法
P_{high}	1,995 (536)	1,807 (521)
P_{low}	91 (59)	91 (60)

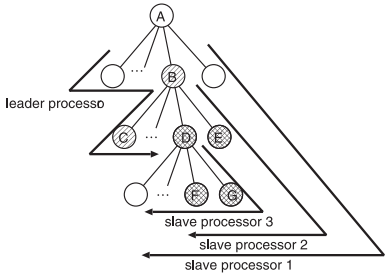

図 8　3 台以上で同じ領域を探索する例
Fig. 8　An example of overlapped search by 3PEs.

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．**表 2** に，図 7 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 2 より，問題

表 3　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 3　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

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 減少したことが分かる．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 2 より，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とが分かる．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，3.1 節より，1 つの割当て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 つの割当て領域の探索において，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．一方，図 7 より，階層的挟み撃ち探索に対する提案手法の高速化率が 1.5 倍を超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 8** 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 8 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．

最後に，**表 3** 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を示し，**表 4** 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 3，表 4 は，スーパリニアスピードアップが得られた問題と，逐

表 4　分枝限定法に対する提案手法の高速化率 [倍]
Table 4　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

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 3 より階層的挟み撃ち探索は 14 問，表 4 より提案手法は 18 問の問題でスーパリニアスピードアップが得られることが確認できた．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ることが確認できた．

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，分枝数が 2 または 3 の探索木において，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) 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).
6) 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).
7) 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).
8) Held, M. and Karp, R.M.: THE TRAVELING-SALESMAN PROBLEM AND MINIMUM SPANNING TREES: PART II, *Mathematical Programming*, Vol.1, pp.6–25 (1971).
9) 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).

(平成 23 年 1 月 28 日受付)
(平成 23 年 6 月 16 日採録)

中村あすか（学生会員）
1986 年生．2009 年千葉工業大学情報科学部情報工学科卒業．2011 年同大学大学院情報科学研究科情報科学専攻博士前期課程修了．同年同大学院情報科学研究科情報科学専攻博士後期課程入学．主として，並列探索に関する研究に従事．

富永　浩文（学生会員）
1984 年生．2007 年千葉工業大学情報科学部情報工学科卒業．2009 年同大学大学院情報科学研究科情報科学専攻博士前期課程修了．2010 年同大学院情報科学研究科情報科学専攻博士後期課程入学．主として，GPU 等のアクセラレータを用いた各種アプリケーション高速化の研究に従事．

前川　仁孝（正会員）
1967 年生．1990 年早稲田大学理工学部電気工学科卒業．1992 年同大学大学院理工学研究科電気工学専攻修士課程修了．1993 年日本学術振興会特別研究員．1994 年早稲田大学理工学部助手．1998 年千葉工業大学情報工学科講師．2002 年同大学助教授，2011 年同大学教授，現在に至る．博士（工学）．主として，各種アプリケーションの並列処理の研究に従事．