ハイパフォーマンスコンピューティング 60－9
（1996．3．4）

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

吉松 敬仁郎† 安浦 寛人†

本稿では，最良優先探索を用いた分枝限定法の並列化手法として並列部分問題探索法（PSS:Parallel Subproblem Search）を新たに提案する．また，静的に負荷分散を行う手法を提案する．PSSにこの負荷分散手法を組み合わせることでさらに処理時間を短縮することが可能となる．実際にナップサック問題を複数のインスタンスを与えて解いた結果，1プロセッサに比べて負荷分散を行った64プロセッサで約10～80倍の速度向上が実測された．

Parallel Subproblem Search for Combinatorial Optimization Problems

KEIJIRO YOSHIMATSU† and HIROTO YASUURA†

We propose a parallel subproblem search (PSS) for branch-and-bound algorithm. We use static load balancing in the PSS to reduce execution time. To evaluate the PSS , we compare two cases : (i) host, and (ii) host + 64cells. Experimental results show that the execution time of (ii) is about 10～80times smaller than the execution time of (i).

1. はじめに

分枝限定法は組合せ最適化問題，とくにNP困難な問題のように非常に探索空間の広い問題の解法として良く用いられる．分枝限定法の計算時間は問題に対して指数的に増加するため大きな問題を実用的な時間で解く事は困難である．そこで問題を効率よく解くために分枝限定法の並列処理についてこれまで様々な手法が提案されてきた．

本稿では最良優先探索を用いた分枝限定法の並列化手法として並列部分問題探索法（PSS:Parallel Subproblem Search）を新たに提案する．この手法は1つのホストプロセッサと複数の子プロセッサから構成される並列計算機上へ実装することを前提としている．まずホストプロセッサが逐次処理で最良優先探索を用いて分枝限定法を行い部分問題を生成していく．ある部分問題があらかじめ設定しておいた探索の深さに到達すると，その部分問題を複数のプロセッサに割り当て並列処理を行う．発見された解はホストプロセッサへ送信される．この手法の新しい点は，各プロセッサへ部分問題を割り当てる際の探索の深さ，つまり逐次処理から並列処理へ処理を移行させる分岐点が変更可能な点である．プログラムが全てを決定するのではなく，利用者が与えられたプロセッサ数とインスタンスに応じてパラメータを変化させることで効率の良い処理を行う事が可能となる．逐次処理と並列処理を上手く組み合わせる事で速度向上を図っている．組合せ最適化問題の応用分野では，インスタンスを少しずつ変更しながら繰り返し問題を解くような場面が多い．本手法はこのような状況において効率よく並列処理を行う手法として利用できる．また，分枝限定法以外の近似解法などとの組合せも可能である．

また，負荷分散の手法についても新たに提案する．本稿で提案するのは静的に負荷分散を行う手法である．ホストプロセッサから各子プロセッサに割り当てられる部分問題の順序をローテートする手法であるが，PSSにこの負荷分散手法を組み合わせることで，さらに処理時間を短縮することが可能となる．実際にナップサック問題を複数のインスタンスを与えて解いた結果，1プロセッサに比べて64プロセッサで約10～80倍の速度向上が実測された．

本稿の内容としては，2章でナップサック問題と分枝限定法の説明を，またこれまでに提案されてきた並列化手法を述べる．3章では新たに提案するPSSについて述べ，4章ではPSSの性質と静的な負荷分散手法について述べる．5章ではPSSと負荷分散手法による処理の高速化について述べ，最後に6章で全体のまとめを述べる．

2. 分枝限定法

本章では本稿で提案する手法の対象の例として取り上げるナップサック問題[9)8)]の説明と，その基本的解

† 九州大学大学院総合理工学研究科
情報システム学専攻
Department of Information Systems,
Interdisciplinary Graduate School of Engineering
Sciences, Kyushu University

法であり並列化の対象である分枝限定法の説明を述べる．また，これまでに提案されてきた分枝限定法の並列化手法についても述べる．

2.1 ナップサック問題

ナップサック問題は，総重量を制約条件として与え，評価値の和を最大にするように品物を選択する整数計画問題である．ここで扱うナップサック問題は全ての変数が 0-1 変数なので，0-1 ナップサック問題ともいわれる．以下にその定義を述べる．

$$\text{目的関数}：\sum_{j=1}^{n} c_j x_j \text{を最大にする}$$

$$\text{制約条件}：\sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j = 0 \text{ または } 1 \quad (j=1,2,\ldots,n)$$

ただし，係数 c_j, a_j, b は全て正の整数とする．

係数 a_j を品物 j の重量，c_j をその価値とすると，総重量が b 以下という制約の下で価値の和を最大にするように品物を選択してナップサックに詰めるという問題と考えられる．ナップサック問題は広い応用分野を持つが，計算の複雑さの理論からは NP 困難であることが知られている．[8)]

2.2 分枝限定法

分枝限定法は，組合せ最適化問題，とくに NP 困難な問題を扱う要領の良い列挙法である．分枝限定法とは，分枝操作と限定操作の 2 つから成り立つ．まず分枝操作とは，直接解くことが困難な原問題をいくつかの部分問題に分解する操作である．ここで部分問題とは，1(0) に固定されている変数の添字集合 $J_i^1(J_i^0)$ と固定されていない自由変数の添字集合 $F_i = \{1,2,..,n\} - (J_i^1 \cup J_i^0)$ を用いて次のように定義される．

$$\text{目的関数}：\sum_{j\in J_i^1} c_j + \sum_{j\in F_i} c_j x_j \text{を最大にする}$$

$$\text{制約条件}：\sum_{j\in F_i} a_j x_j \leq b - \sum_{j\in J_i^1} a_j$$

$$x_j = 0 \text{ または } 1, \quad (j\in F_i)$$

これによって多数の部分問題を生成し，それらを解く事で最終的に原問題を解く．その分解の様子は 2 分木の分枝図と呼ばれる図によって表すことができる．

しかしこのまま分解を行うだけでは単なる列挙法になるため，限定操作が必要となる．限定操作とは，各部分問題が最適解を与える可能性があるかどうかをテストする操作である．もし最適解を与えないことがわかった場合，その部分問題をそれ以上分解せずに終端させる．そうでない場合はさらに分枝操作と限定操作を繰り返し続け，各部分問題で最適解が求まった場合もそこで終端させる．このようにして求まった各部分問題の解の内，最適なものが原問題の解となる．

このように分枝限定法ではテストを行う必要がある．ナップサック問題に分枝限定法を用いる場合，変数が実数値をとることを許すようにナップサック問題の制約条件を以下のように変えた問題を考える．

制約条件：$x_j = 0$ または $1 \Rightarrow 0 \leq x_j \leq 1$

これを連続ナップサック問題と言い，ナップサック問題と違って比較的簡単に最適解を得ることができる．ナップサック問題ではこれによって得られる解を分枝操作のテストにおける見積り値として使用する．連続ナップサック問題の解の方が良い解を得ることができるという性質を持っているため，分枝限定法の限定操作に用いる見積り値として使用することができる．

2.3 従来の分枝限定法の並列化手法

分枝限定法の並列化手法については様々な手法が提案されてきた．もっとも単純な手法としては，原問題を並列計算機のプロセッサ台数分の部分問題に分解して処理を行う手法が考えられる．しかし，これだけでは各プロセッサが独立して処理を行うために，暫定解が各プロセッサによって異なる．そのため各部分問題による探索回数のばらつきによって，プロセッサ間での負荷の不均衡が起こる．場合によっては逐次処理では探索しない余計なところまで探索してしまい，逐次処理の場合よりも処理時間がかかってしまう可能性もある．[5)]そこでこの問題を解決するために，次のような方法がとられている．

- 先に処理が終了したプロセッサが，まだ処理をしているプロセッサの部分問題をを分けてもらう[1)6)3)]
- あるプロセッサで暫定解が更新された場合に，各プロセッサ間で通信を行い全プロセッサの暫定解を更新し，枝刈りの効率を上げる[1)10)3)]
- 部分問題を展開し，新たに部分問題を生成する度に隣接プロセッサ間で一部の部分問題の交換を行う[1)4)7)]
- 統括するプロセッサがある深さまで探索し生成した部分問題を各プロセッサへ一つずつ割り当てて処理を行う[1)10)]

このような手法によって負荷の分散を図り，分枝限定法の並列処理を効率よく行う事が考えられてきた．しかし，解が見つかる可能性のない部分問題を解いていては必ず無駄な探索を行ってしまう．また，プロセッサ間で頻繁に部分問題のやり取りをしていては通信コストを増大させ，そのための処理も複雑になる．そこで，解の見つかる可能性の高い部分問題について並列処理を行い，通信のコストを押えた，なおかつプロセッサのアイドル時間を少なくするような手法が必要であると考えられる．

3. 並列部分問題探索法

本稿で提案する並列部分問題探索法 (PSS:Parallel Subproblem Search) は，単純にプロセッサの台数分の部分問題を生成してそれぞれを並列に解くのではなく，本当に必要な部分問題だけに並列処理を用いる事によって効率よく探索を行うという考え方に沿っている．

本稿で提案する手法は分散メモリ型のメッセージ通信を基本とした並列計算機上への実装を前提として

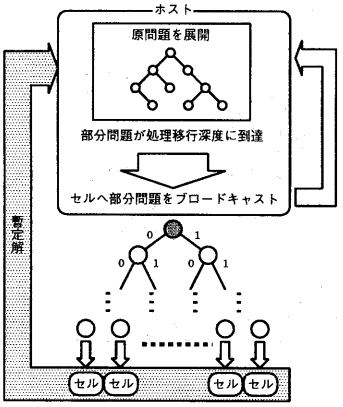

図1 並列部分問題探索法

いる．この並列計算機には全体を統括するホストプロセッサと子プロセッサであるセルプロセッサが存在し，各々ホスト，セルと略称する．使用するメッセージ通信はホスト・セル間のブロードキャストと1対1通信のみである．

PSS は，逐次処理により生成した部分問題の中で探索木のある深さに達したものだけを選択し，それについて並列処理を行う手法である．探索法は最良優先探索を用いる．以下にホスト，セルでの処理について示す．

- ホスト
  - (1) 暫定解の初期値として近似最適解を求め，分枝限定法を実行し原問題を展開
  - (2) 生成された部分問題がある深さに到達したら全セルへ送信
  - (3) セルから解が送信されてきたら受信，ホストの暫定解より良い値であれば更新し全セルへ送信
  - (4) 部分問題がなくなるまで (2)～(4) を繰り返し
  - (5) 全セルへ終了メッセージを送信
  - (6) 全セルでの処理が終了するまでにセルから解が送信されてきたら受信，暫定解より良い値であれば更新し，全セルへ送信
  - (7) 全セルでの処理が終了した時点での暫定解が原問題の解
- セル
  - (1) ホストから送信された部分問題を受信，この時暫定解がホストから送信されていたら受信し，セルでの暫定解より良い値であったら更新
  - (2) 部分問題をセルの台数分に分割し，各セルが担当する部分問題について分枝限定法を実行
  - (3) 処理終了後，解が更新された時のみホストへ解を送信
  - (4) ホストからの部分問題を受信する度に (1)～(3) を繰り返し
  - (5) ホストからの終了メッセージを受信したら処理を終了

ホストからセルへ部分問題を割り当てる際にはブロードキャストで並列処理移行深度に到達した部分問題を送信し，各セルが自分が割り当てられるべき部分問題を生成して処理を行う．そのため1対1通信で部分問題を送信する場合に比べると通信コストは軽減される．

4. PSS の性質と負荷分散手法

前章で提案した PSS の特徴は，並列処理移行深度が可変な点である．この章では並列処理移行深度を変化させた時の処理時間の変化の特徴について示す．また，負荷分散手法についても示す．

4.1 並列処理移行深度が処理時間に与える影響

並列処理移行深度の変化が処理時間にどのような影響を与えるのかを調べるために，並列処理移行深度を変化させた時の処理時間を測定する．ここで処理時間とは分枝限定法を用いて最適解を見つけるまでの時間とする．実験方法は，4,16,64 の 3 種類のセル構成の並列計算機 AP1000[2] で PSS を用いてナップサック問題を解く．与えるインスタンスは表1に示す2つでその要素は乱数によって各々範囲内に重量と価値を決定したものを使用した．インスタンス I-a を与えた時のセル構成を変えた際の並列処理移行深度と処理時間の関係を図2に示す．いずれのセル構成の場合にも，ある並列処理移行深度で最も処理時間が短くなり，そこをピークとして同じようなカーブを描いている．また，セルの台数が増加すると処理時間が最短となる並列処理移行深度は浅くなっているが，これはセルの台数の増加がそのまま並列処理の部分の処理時間の短縮につながり，全体での処理時間の短縮につながっている事を示している．

このように並列処理移行深度を適切に選択することにより効率の良い処理が可能となる．

4.2 静的負荷分散手法

4.2.1 負荷の偏りと負荷分散

図3に16セル構成で，インスタンス I-a を用いた場合の各セルの探索回数を示す．セル ID とは各セルの番号である．図3からある1つの並列処理移行深度に着目すると負荷の偏りには周期性がある事が見てとれる．この性質を利用して静的に負荷の均衡を図ることが可能だと考えられる．

表1 与えるインスタンス

	要素数	重量制限	重量範囲	価値範囲
I-a	50	重量総和の$\frac{1}{2}$	30～35	5～10
I-b	50	重量総和の$\frac{1}{4}$	30～35	5～10

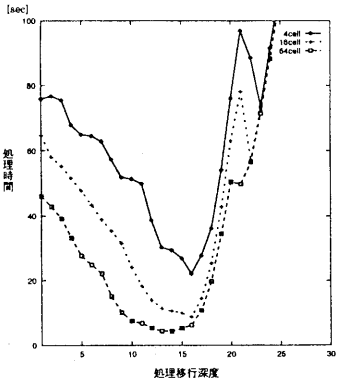

図2　並列処理移行深度と処理時間の関係

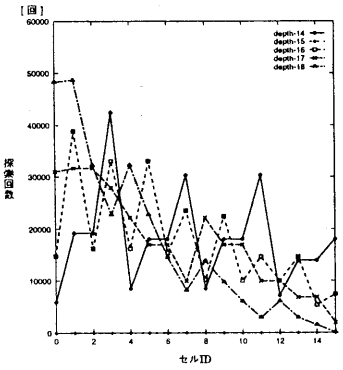

図3　各セルの探索回数

本稿で提案するのは探索を行う前に静的に負荷分散を図る手法である．静的に負荷分散を図る手法として次の2つを考える．

rotational load balancing(RLB)

図4に示すようにホストから割り当てられる部分問題をその度にローテートを行いセルで処理をする方法．ホストから送信される部分問題の数が多ければ全てのセルへ偏りなく全ての0,1の組合せの部分問題が行き渡る．

complementary rotational load balancing (CRLB)

図4に示すようにホストから割り当てられる部分問題をさらにもう1段階分解し，0,1の相補性を考えて2種類の部分問題を組合せる．これをRLBの様にホストから割り当てられる度にローテートを行いセルで処理をする．負荷分散を時間的，空間的に図る．

4.3　負荷分散手法の有効性

静的な負荷分散手法であるRLBとCRLBの有効性を調べるために，1の2つのインスタンスを与えた場合の処理時間と探索回数を測定する．探索回数とは探索の際に生成した部分問題の数とする．実験方法は負荷分散を図らない場合(nobalanced:NB)と,RLB,CRLBの2つの静的な負荷分散手法の3つの場合について

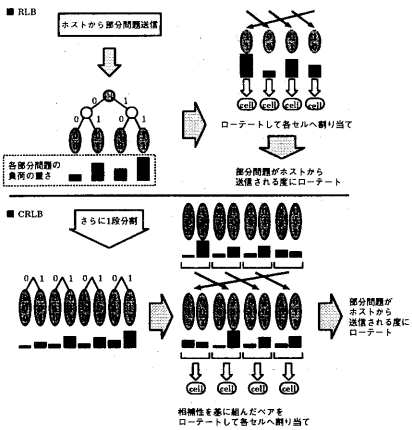

図4　負荷分散手法

表2　静的負荷分散手法による速度向上

	I-a	I-b
NB(16cell)	1	1
RLB(16cell)	1.7	1.6
CRLB(16cell)	3.2	3.7

16台のセル構成のAP1000で測定する．

以下にその結果を示す．表2には各インスタンスを与えた際のNB,RLB,CRLBの3つの場合に並列処理移行深度を変化させた時の最短の処理時間を元にした速度向上率を示している．値はNBの場合の処理時間を各場合における処理時間で割ったものである．

どちらのインスタンスの場合にもRLB,CRLBの負荷分散手法が効果的に機能している事がわかる．NBのプログラムにわずかな工夫を加えることで約2倍から3倍の高速化を実現している．

図5に，各セルでの探索回数を示す．いずれも処理移行深度が15の時の結果である．NBの場合では各セルでの探索回数に大きな偏りがあることが一目でわかる．しかしRLB,CRLBを用いた場合ではローテートを行うというシンプルな手法であるにもかかわらずかなりの負荷分散が図れていることがわかる．また，RLBよりもCRLBを用いた場合の方が探索回数が少なくなっている事がわかる．つまりCRLBにおいては負荷分散だけではなく探索領域も縮小する効果があることがわかる．

5.　PSSとRLB,CRLBによる処理の高速化

以上に述べたPSSとRLB,CRLBを用いた負荷分散によってどの程度処理の高速化が可能なのかを示す．表1の2つのインスタンスの重量，価値の範囲内で乱数を用いて作成した各3種類の計6種類のインスタンスを与えた時の処理時間を比較する．比較対象は逐次処理の場合と，4,16,64セル構成でRLB,CRLBを用

表3　PSSとRLB,CRLBによる処理速度の向上率

	I-a(1)		I-a(2)		I-a(3)		I-b(1)		I-b(2)		I-b(3)	
	SU	ED	SU	ED	SU	ED	SU	ED	SU	ED	SU	ED
逐次処理	1	-	1	-	1	-	1	-	1	-	1	-
RLB(4cell)	3.0	15	3.1	15	3.0	14	3.0	9	3.3	10	3.2	11
RLB(16cell)	10.1	12	10.3	15	10.4	12	11.1	9	12.8	9	12.4	9
RLB(64cell)	26.7	10	24.0	12	25.5	10	33.2	8	43.9	9	43.3	9
CRLB(4cell)	8.9	12	7.9	12	9.2	13	10.5	8	9.8	9	9.4	9
CRLB(16cell)	20.4	12	19.4	12	16.5	11	24.8	8	31.2	9	30.4	8
CRLB(64cell)	34.8	11	33.3	11	28.8	9	44.4	6	80.9	7	79.7	7

SU:速度向上率　　ED:並列処理移行深度

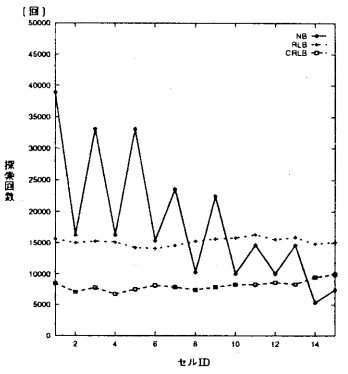

図5　NB,RLB,CRLB, 各場合における各セルの探索回数:I-a

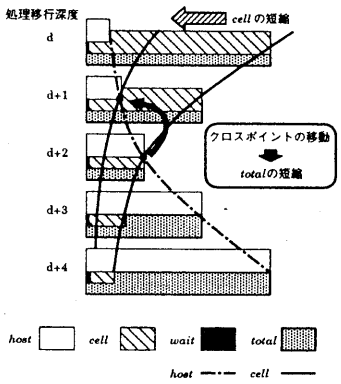

図6　EDを変化させた時のT_{host}, T_{cell},T_{wait},T_{total}の変化

いた場合である.

表3に結果を示す. 値は各インスタンスにおけるNBの処理時間を割った速度向上率(speedup:SU)と処理時間が最短となった時の並列処理移行深度(Entry Depth of Parallel Processing:ED)を表している. ここで各々のセル構成における処理時間はEDを変化させた際の最短の値である.

RLB,CRLB, いずれの場合にも台数の増加と共に速度も向上している. セルの台数が4倍に増加するにともなって, 速度向上率は約2~3倍に増加している. インスタンスによっては64台のセルを使用することで約80倍の高速化になっている. 台数効果を上回った結果となっているが, これはCRLBを用いた場合に負荷分散と同時に探索領域も縮小されているからである. この結果からPSSでは台数効果と共に, CRLBを用いることでそれ以上の効果も期待できると結論される.

5.1 考　　察

表3のEDに着目すると, 同じインスタンスでは負荷分散によって処理時間が短くなるほどEDは浅くなっている. これはセルの台数を増やした際にも同じ結果が見られる. ホストでの処理時間はEDが浅いほど短く, セルでの処理時間はEDが浅いほど長いと考えられるので, 全体の処理が終了するまでの時間T_{total}は, ホストでの処理時間をT_{host}, セルでの処理時間をT_{cell}, セルでの処理が始まるまでの時間をT_{wait}とすると, 次のように表される.

$$\begin{cases} T_{total} = T_{wait} + T_{cell} & T_{host} < T_{wait} + T_{cell} \\ T_{total} = T_{host} & T_{host} > T_{wait} + T_{cell} \end{cases}$$

図6にEDが変化する時のT_{host}, T_{cell}, T_{wait}, T_{total}の変化を摸式化したものを示す. この図から処理時間が最短となるのはT_{host}=T_{wait}+T_{cell}となる時であるのがわかる. 台数効果や負荷分散を図ることによってT_{cell}が短縮されると図6に示すようにT_{cell}を表す直線が上へ並行移動し, T_{host}=T_{wait}+T_{cell}となるクロスポイントがEDの浅い所へ移動する. この時T_{host}に大きな変化はないためT_{total}は短縮される.

PSSによって処理の高速化を図るには, インスタンスとプロセッサ数に適したEDを前もって見積もることが必要となる. 実験の結果から, インスタンスの各要素の重量, 価値の値, 重量制限, 要素数を考慮に入れた見積り方法が必要だと考えられる. そこで各要素の単位重量あたりの価値を計算しその値の大きい順に要素を並び換えた時に, 1番目の要素から重量制限を満たす範囲で取る事ができる数を調べる. この値をD_{cal}とする. しかしこれにはプロセッサの数を考慮に入れていない. そこで表3のEDに着目すると, プロセッサ数が4倍になる度に約1ずつEDが浅くなっている. 以上の事から見積り値をD_{est}, プロセッサ数をPEとすると, 次のように表される.

$$D_{est} = D_{cal} - \lceil \log_4 PE \rceil$$

この式を用いて表5のインスタンスの見積りを行い,

表4　見積り値と実測値の速度向上率の比較

	I-c				I-d				I-e			
	見積り		実測		見積り		実測		見積り		実測	
	SU	ED	SU	ED	SU	ED	SU	ED	SU	ED	SU	ED
逐次処理	1	-	1	-	1	-	1	-	1	-	1	-
CRLB(4cell)	9.8	32	9.8	32	3.2	38	3.2	38	3.9	34	4.8	31
CRLB(16cell)	25.0	31	29.3	32	3.8	37	8.0	39	8.1	33	11.6	34
CRLB(64cell)	50.5	30	53.8	31	6.2	36	9.1	39	14.0	32	19.1	30

表5　与えるインスタンス

	要素数	重量制限	重量範囲	価値範囲
I-c	100	総重量の$\frac{1}{3}$	30～35	5～10
I-d	80	総重量の$\frac{1}{3}$	10～50	40～50
I-e	80	総重量の$\frac{1}{3}$	$1.0 \leq \frac{c_j}{a_j} \leq 3.5$	1～100

CRLB を用いた場合の 4,16,64 セル構成での実測値との比較を行った．表 5の$\frac{c_j}{a_j}$は各要素の単位重量あたりの価値である．結果を表 4に示す．

いずれのインスタンスにおいても D_{eet}は最短の処理時間となる ED に近い値となっている．インスタンスによるばらつきはあるが，実測値の SU に比べて約 7～8 割の速度向上は確保している．この見積りによって，与えられたインスタンスの情報とプロセッサ数に適した ED を処理を行う前に知る事ができ，PSS の効果を活かす事が可能となる．

6. おわりに

本稿では，最良優先探索を用いた分枝限定法の並列化手法として並列部分問題探索法 (PSS:Parallel Subproblem Search) を新たに提案した．逐次処理で有望な部分問題を選択し，それらについてのみ並列処理を行う事で効率よく処理を行う手法である．この手法では，並列処理移行深度を使用可能なプロセッサ数と与えられたインスタンスに応じて変化させることで効率の良い処理を行う事が可能となる．事前に適した並列処理移行深度を知るための見積り手法を用いることによって，これが可能となる．また，負荷分散の手法として静的に負荷分散を行う 2 つの手法 rotational load balancing(RLB) と complementaryrotational load-balancing(CRLB) を新たに提案した．PSS にこの負荷分散手法を組み合わせることでさらに処理時間を短縮することができた．実際にナップサック問題を複数のインスタンスを与えて解いた結果，1 プロセッサに比べて 64 プロセッサで約 10～80 倍の速度向上が実測された．

並列化手法，負荷分散手法のどちらもシンプルであるがそれだけに実用性があり，応用性が高いと考えられる．今後の課題としては，アプリケーションへの適用，他の手法との比較などを考えている．

謝辞　日頃から御討論頂く，九州大学大学院総合理工学研究科の村上和彰助教授，岩井原瑞穂助手に深く感謝致します．また，有益な助言を頂く安浦研究室の皆様に感謝致します．

参考文献

1) *ADVANCES IN PARALLEL ALGORITHMS.* Blackwell Scientific Publications, 1992.
2) *AP1000* 使用手引書. 富士通研究所, 1992.
3) T. S. Abdelrahman and T. N. Mudge. Parallel branch and bound algorithms on hypercube multiprocessors. *THE 3rd Conf. ON Hypercube Concurrent Computers and Applications*, II-Applications:1492–1499, January 1988.
4) E. W. Felten. Best-first branch-and-bound on a hypercube. *THE 3rd Conf. ON Hypercube Concurrent Computers and Applications*, II-Applications:1500–1504, January 1988.
5) T. Lai and S. Sahni. Anomalies in parallel branch-and-bound algorithms. *Communications of the ACM*, 27(6):594–602, June 1984.
6) Tsung F. Ma, R. P. and M. Ma. A dynamic load balancer for a parallel branch and bound algorithm. *THE 3rd Conf. ON Hypercube Concurrent Computers and Applications*, II-Applications:1505–1513, January 1988.
7) M. J. Quinn. Analysis and implementation of branch-and-bound algorithms on a hypercube multicomputer. *IEEE Tra. on Computers*, 39(3):384–387, March 1990.
8) 福島 雅夫 茨木 俊秀. 最適化プログラミング. 岩波書店, 1991.
9) 石畑 清. アルゴリズムとデータ構造. 岩波書店, 1989.
10) 川岸 太郎. 並列分枝限定法による混合整数計画問題の解法. *JSPP '92*, pages 79–84, June 1992.