統計数理（1998）
第46巻 第2号 411-431

特集「計算と最適化」
［原著論文］

汎用並列分枝限定法ツールPUBBによる組合せ最適化問題の厳密解法†

東京理科大学* 品野勇治
神戸商科大学** 藤江哲也

（受付 1998年3月20日；改訂 1998年8月8日）

要　旨

分枝限定法は，組合せ最適化問題に対する代表的な厳密解法である．近年では，並列計算機の発達にともなって並列分枝限定法の研究が盛んである．特に，分枝限定法が持つアルゴリズムの汎用性に着目した汎用ツールが開発されている．本稿では，品野らによって開発された汎用並列分枝限定法ツールPUBB（Parallelization Utility for Branch-and-Bound algorithms）の概要と適用例を示す．PUBBは，現在公開準備中であり，付録として，C言語によるユーザ・インターフェースの概略を与える．

キーワード：分枝限定法，並列処理，組合せ最適化，巡回セールスマン問題，最大クリーク問題．

1. はじめに

分枝限定法は，組合せ最適化問題に対する代表的な厳密解法である（茨木（1983））．この解法は列挙法であり，与えられた問題を幾つかの子問題に分割して解いていく（分枝操作）．ここで，不要な列挙を避けるため，解くべき問題の性質や構造を利用した限定操作や分枝方法，子問題の選択（探索）方法などが主たる研究課題である．一方，1980年代後半から，並列計算機の実用化に伴い，子問題を並列に解く並列分枝限定法の研究報告が増えている（Lüling and Monien (1989), Pardalos and Rodgers (1990), Quinn (1990), Taudes and Netousek (1991), Pekny and Miller (1992), 池上 他(1992), Eckstein (1994), Lüling et al. (1996)）．

分枝限定法に並列処理を適用することの利点は，演算処理能力の向上および演算時に利用できるメモリ容量の増大にある．さらに，並列計算機のアーキテクチャに依存するものの，演算処理能力とメモリ容量を必要に応じて拡張することができる．組合せ最適化問題の多くは本質的に難しく，分枝限定法の並列化がそれらの解決に決定的であるとはいえない．しかし，現実的な時間で解ける問題例を増やすための，重要なアプローチの1つである．実際，並列分枝限定法によって初めて最適解を得たという報告がいくつかなされている（Brüngger et al. (1997), Shinano et al. (1998)）．

* 工学部 経営工学科：〒162-0825 東京都新宿区神楽坂1-3.
** 商経学部 管理科学科：〒651-2103 兵庫県神戸市西区学園西町8-2-1.

† 本稿の一部は統計数理研究所共同研究（9-共研A－24）に基づくものである．

分枝限定法および並列分枝限定法は，個々の問題に依存しない汎用的な枠組みを持つ．近年では，この点に着目し，並列分枝限定法の枠組を汎用ツールとして提供するソフトウェアが開発されている（Benaïchoche et al. (1996), Brüngger (1997), Shinano et al. (1995), Shinano et al. (1997), Tschöke and Polzer (1996)）．良く知られているように，分枝限定法に並列処理を適用すると，異常加速（speedup anomaly）や異常損失（detrimental anomaly）などの現象が発生する（Lai and Sahni (1984)）．特に異常加速は，通常の並列処理では考えられない超線形加速を示す現象であり，この現象が多く発生するような汎用ツールを実装することが理想的である．

品野 他（1996）は，ワークステーション群を利用して並列分枝限定法の汎用ツール PUBB (Parallelization Utility for Branch-and-Bound algorithms) を実装した．PUBB には，異常加速が頻繁に発生しかつ異常損失が発生し難い，ハイブリッド探索と呼ばれる子問題の探索規則が実装された（品野 他(1996)）．しかし，PUBB には，列挙過程で生成される情報が全て集中管理されること（中央制御型）によるデメリットがあった．

PUBB の中央制御型構成を変更するために，完全分散型構成なる新たなシステムを再構築した（Shinano et al. (1997)）．再構築に際し，いくつかのメッセージ・パッシング・ライブラリを検討して，UNIX 環境でしか動作しなかった PUBB を，様々な並列計算機環境で動作するツールとすることも目標とした．1990 年代初頭から，いくつかのメッセージ・パッシング・ライブラリが開発され，1994 年には MPI(Message-Passing Interface)の標準仕様が提示された．計算機環境に依存しないツールとするためには，MPI を利用することが好ましいと思われたが，プロセスの生成・消去に関する仕様が開発当時にはなかったため，準標準の 1 つであった PVM（Parallel Virtual Machine）を利用した．さらに，C++による強力なクラス・ライブラリの利用を考慮して，開発言語を C 言語から C++へと変更し，C++によるユーザ・インターフェースを設計した．しかし，C++は言語仕様が巨大であり，しかも C 言語を利用するユーザが多いため，新たに C 言語によるユーザ・インターフェースも設計した（本稿付録参照）．

本稿では，中央制御型に加え，完全分散型，およびそれらを合わせた混合制御型の並列分枝限定法を実現した PUBB を紹介する．また，PUBB を利用した組合せ最適化問題の解法例として，巡回セールスマン問題と最大クリーク問題に対する数値実験結果を示す．前者では各子問題の処理に時間を要し，並列化の効果が示された．一方，後者では各子問題の処理時間がワークステーション間の通信時間よりも短く，この種の解法に対応するために PUBB そのものに改良を加えた．その結果，この種の解法に対しても並列化の効果が示された．

2. 分枝限定法と並列分枝限定法

本節では分枝限定法の概略を説明するために，目的関数を $f: R^n \to R$，実行可能解集合を $S \subseteq R^n$ とした最小化問題

$$(P) \quad \text{最小化 } f(\boldsymbol{x}) \qquad \text{条件 } \boldsymbol{x} \in S$$

を考える．分枝限定法は列挙解法であり，解集合 S を $S = \cup_{k=1}^{K} S_k$, $S_i \cap S_j = \emptyset\ (i \neq j)$ と分割していく．このとき各子問題

$$(P_k) \quad \text{最小化 } f(\boldsymbol{x}) \qquad \text{条件 } \boldsymbol{x} \in S_k$$

に対して，緩和問題に基づいて評価を行う．緩和問題とは，(P_k) の下界値を与える問題である．本節では，

$$(\overline{P_k})\quad \text{最小化 } f(\boldsymbol{x}) \qquad \text{条件 } \boldsymbol{x}\in\overline{S_k}$$

(ただし，$S_k\subseteq\overline{S_k}\subseteq R^n$ を満たす) を緩和問題としてとりあげる．

いま，ヒューリスティック解法等によって求められている (P) の実行可能解 (暫定解) を $\boldsymbol{x}^*\in S$ とするとき，子問題の評価は以下の手順を踏む:

(1) $(\overline{P_k})$ を解き，最適解を $\bar{\boldsymbol{x}}\in\overline{S_k}$ とする．
(2) $(\overline{P_k})$ が実行不可能 (すなわち $\overline{S_k}=\emptyset$) ならば，(P_k) も実行不可能なので (P_k) を見切る．
(3) $f(\bar{\boldsymbol{x}})\geq f(\boldsymbol{x}^*)$ ならば，(P_k) に $\boldsymbol{x}^*$ よりも良い解は存在しないので (P_k) を見切る．
(4) $\bar{\boldsymbol{x}}\in S_k$ ならば，$\bar{\boldsymbol{x}}$ は (P_k) の最適解である．また，(P) の実行可能解でもあるので，$f(\bar{\boldsymbol{x}})<f(\boldsymbol{x}^*)$ ならば $\bar{\boldsymbol{x}}$ を新しい暫定解として出力する．
(5) (2), (3), (4) を満たしていない，すなわち，「$(\overline{P_k})$ が実行可能で，その解 $\bar{\boldsymbol{x}}$ に対して $f(\bar{\boldsymbol{x}})<f(\boldsymbol{x}^*)$ かつ $\bar{\boldsymbol{x}}\notin S_k$」ならば，$S_k$ を分割し子問題を生成して出力する．

すなわち，更新された暫定解と新しい子問題群が出力される．ここで，新しい子問題群が生成される場合でも，(P_k) に対してヒューリスティック解法を適用するなどして暫定解が更新されることがある．したがって，暫定解と子問題群の両方が出力されることもある．

分枝限定法は，以上の操作を，元問題 (P) から始めて，評価されていない子問題がなくなるまで行う．ここで，評価されていない子問題群を子問題プールと呼ぶことにすると，分枝限定法は以下のように記述される．

分枝限定法

Step 1. 問題データ（インスタンス）を読み込む．
Step 2. 実行可能解（初期暫定解）を計算する．
Step 3. 初期暫定解を保存する．
Step 4. 元問題を生成する．
Step 5. 元問題を子問題プールに加える．
Step 6. 子問題プールが空なら Step 11 へ．さもなければ，子問題プールから子問題を選ぶ．
Step 7. 子問題の評価を行う．
Step 8. もし暫定解が更新されたならばそれを保存し，子問題プールから見切られる子問題を削除する．
Step 9. もし子問題が生成されたならば，それらを子問題プールに加える．
Step 10. Step 6 へ行く．
Step 11. 最適解を出力して終了．

Step 6 において，子問題プールの中から子問題の選択方法として，深さ優先探索，下界値優先探索，そしてハイブリッド探索などが知られている (茨木(1983)，品野 他(1996), Shinano et al. (1997))．

分枝限定法では，子問題プールに存在する子問題間の依存関係は一般に少ない．したがって，複数個の子問題の評価を同時に行うことができる．これは分枝限定法が持つ汎用的な並列性であり，ツールとしての実装が可能である．並列処理は，Step 2 の実行可能解（初期暫定解）の計算，Step 7 での 1 個の子問題の評価計算そのものに対しても導入可能であるが，これらは各問題別に対処する必要があるので本稿では扱わない．

ツールの実装に際しては，唯一の子問題プールを持ち子問題群が集中管理される中央制御型の実装と，複数の子問題プールが存在する分散制御型の実装に大別される．中央制御型は，子

問題の選択方法をほぼ完全に制御できる．その反面，並列処理を利用する利点の1つである，子問題管理用メモリ容量の増大は期待できない．また，子問題プールへアクセスが集中するため，並列に評価できる子問題数も制限されると思われる．一方，分散制御型では，効果的な子問題の選択方法を実現するには，子問題プール間の負荷分散に際して，子問題の適当な再配分が不可欠であり，このような操作は余分なオーバヘッドを生じる．子問題プール間の負荷分散が不適当であると，逐次処理と比較して極端に多くの子問題が評価されることとなり，並列化の効果は期待できない．

3. PUBBの概要

本節では，PUBBについて概観する．すでに述べたように，PUBBはPVM上で動作するシステムとして設計され，3つの動作モード（MS : Master-Slave, FD : Fully-Distributed, MS-FD : Master-Slave to Fully-Distributed）を持ち，中央制御型（MSモード），分散制御型（FDモード），混合制御型（MS-FDモード）の動作を1つのシステムでサポートしている．また，子問題の選択方法の指定を実行時に行える．

ユーザがPUBBを利用するには，個々の問題やその解法に依存する部分を用意すればよい．ここで，ユーザ記述部に対して，並列処理用と同じインターフェースを逐次処理用の枠組にも与えた．これによって，逐次処理のプログラムを開発すれば，再コンパイルだけで並列分枝限定法へ移行することができるようになった（Shinano et al.(1996)）．

3.1 タスク構成

PUBBは，PVM上の以下の3種類のタスクで構成される（図1）．

Problem Manager(PM)：主に，問題の情報や分枝限定法の実行中の情報を管理する．MSモードまたはMS-FDモードでは子問題プールを管理する．

Load Balancer(LB)：**Solver**を起動する．したがって**Solver**と対をなす．FDモードでは，対となる**Solver**の子問題プールの状態を受けとり，他の**LB**とデータ交換を行って，負荷分散

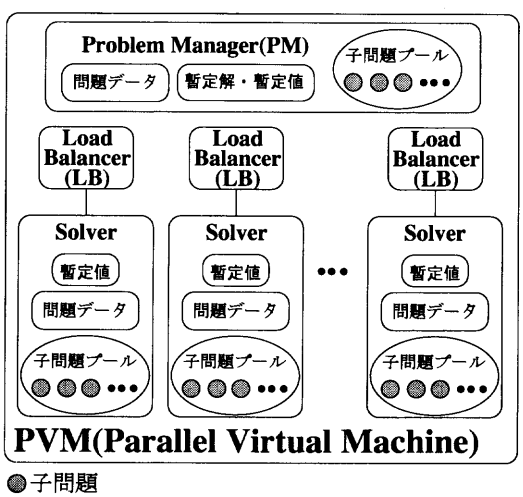

図1. PUBBのタスク構成．

のための意思決定を行う.

Solver：主に，子問題の評価計算を行う．FD モードまたは MS-FD モードでは，**Solver** ごとに子問題プールを管理する.

ワークステーション群上で PVM を利用する場合，通常，**LB** と **Solver** は同一ワークステーション上で動作させる.

3.2 各動作モードの概略

3.2.1 Master-Slave（MS）モード

初期化. まず **PM** が起動される．**PM** はユーザ・プログラム部を呼び出すことで，問題データの読み込み，初期暫定解の計算，および元問題の生成を行う．**PM** の初期化は，元問題を子問題プールへ入れることで完了する．**LB** は必要に応じて起動され，初めに対をなす **Solver** を起動する．そして起動された **Solver** は **PM** に対して初期化要求を出す．**PM** は，その要求に答えて，問題データや初期暫定値を **Solver** へ送る．**Solver** は，これらを受けとって初期化を完了する．そのとき，**Solver** の初期化完了は，対となる **LB** を経由して **PM** へ伝達される．**LB** および **PM** は，初期化の完了通知のあった **Solver** を計算に使用可能な **Solver** として認識する.

子問題の評価と送受信. **PM** は，使用可能であると認識した **Solver** 群中に IDLE 状態(計算を行っていない状態）の **Solver** が存在し，自タスクの子問題プールに子問題が存在する限り，次々に子問題を IDLE 状態の **Solver** へ割り当てる．**Solver** が子問題を受けとると，その問題の評価計算をなう．その結果にしたがって，次の動作が実行される：

(1) 新しく子問題群が生成された場合，それらは全て **PM** へ戻され，その **Solver** は次の子問題の受信待ち，すなわち IDLE 状態となる.
(2) 暫定解が更新された場合，その解を **PM** へ送信する．その後 **PM** は暫定値を各 **LB** へ送信する．各 **LB** は新たな暫定値を受けとると，その値を対をなす **Solver** へ伝達する.
(3) 評価された子問題が見切られた場合，計算終了の報告だけが **PM** へ伝達される.

終了時. **PM** のみが管理している子問題プールが枯渇し，計算中の **Solver** が存在しないことを **PM** が認識したとき，**PM** は保持している暫定解を最適解として出力し終了する.

3.2.2 Fully-Distributed(FD)モード

初期化. 初期化の手順は次の 2 点を除き，MS モードと同じである．すなわち，(1) **PM** は子問題プールを持たず，**PM** で生成された元問題は最初に起動された **Solver** に渡される；(2) **PM** は **Solver** 群を管理しない.

子問題の評価と送受信(1). このモードでは，各 **Solver** は自タスクの子問題プールを用いた逐次処理の分枝限定法を **Solver** 内だけで実行する．最初に起動された **Solver** は **PM** から元問題を受け取るが，その後は各 **Solver** と対をなす **LB** 群が負荷分散の意思決定を行い，**LB** の指示に従って，ある **Solver** から別の **Solver** へと子問題が渡される.

負荷分散. **LB** 群だけで負荷分散の意思決定を行うために，各 **LB** は対をなす **Solver** の子問題プールの情報を知る必要がある．そのため，**Solver** は 1 個の子問題の評価計算が終了すると，対をなす **LB** に対して，(1)保持している子問題数，(2)最良下界値，および(3)最悪下界値を送信する．**LB** はこれらの情報と他の **LB** とやり取りして得た負荷分散の意思決定をもとに，対をなす

Solver が他の **Solver** へ子問題を送信するかどうか判断し，送信する場合には送信先の **Solver** を指示する．**Solver** は，この指示にしたがって子問題を送信するか，もしくは他の **Solver** から子問題を受信し，子問題プールを更新してから次の子問題評価計算を行う．

LB は，対をなす **Solver** の持つ子問題数がパラメータで指定された閾値を下回ったとき，あるいは **Solver** から子問題の溢れが通知されると，子問題の送受信先を探しだす処理を開始する．本稿では，子問題数が閾値を下回ったときの処理方法を説明する．この場合，**LB** は，子問題の送信元となる **Solver** に対をなす **LB** を探す処理となる．ここで，**LB** と **Solver** は別タスクなので，探索の間も **Solver** での子問題評価計算が中断されることはない．探索を開始した **LB** をイニシエータと呼ぶ．そして，イニシエータと対をなす **Solver** と，子問題を送受信する **Solver** が見つかったとき，その **Solver** に対をなす **LB** をパートナーと呼ぶ．また，負荷分散の意思決定を協調して行う **LB** 群を **Load Balancing Group** と呼ぶ．**Load Balancing Group** は，以下のプロトコルによりイニシエータごとに生成・拡張・消去される：

1. イニシエータは全ての **LB** へロック要求を出す．この要求にはシーケンス番号が含まれる．この番号は要求を出したイニシエータ内に保存される．
2. ロック要求を受け取った **LB** は，これに答えて，対をなす **Solver** の持つ子問題数や，**LB** 自身のステータス(「負荷分散の要求を発行している」，「対をなす **Solver** が他の **Solver** と子問題の送受信を行っている」といった情報)，および，受けとったシーケンス番号をロック要求の応答としてイニシエータへ返す．
3. イニシエータは，ロック要求の応答を受信するとシーケンス番号がタスク内に保存しているシーケンス番号と一致するロック要求の応答は受理し，応答した **LB** を，イニシエータが認識する **Load Balancing Group** へ追加する．シーケンス番号の異なった応答は，既に消去された **Load Balancing Group** に対するものなので捨てる．
4. パートナーの探索が終了すると，全 **LB** に対してロック解放要求が送信される．

LB と **Solver** の対は，少なくとも1つのイニシエータからのロック要求に応答すると，対応する解放要求を受けとるまで，タスクを終了させないことを保証する．現在の PUBB では，並列分枝限定法の実行中にタスクの追加・削除が可能であるため，負荷分散の処理中にタスクが削除されないことを保証する機能がこのプロトコルに含まれている．

パートナーの探索は，以下の手順により行なわれる．

Step 1. イニシエータは，全ての **LB** に対してロック要求を発行する．要素数 0 の **Load Balancing Group** を生成する．

Step 2. イニシエータは，対をなす **Solver** から1回の子問題評価計算の終了が通知された場合，ロック要求の応答を待つ．既に子問題が枯渇している場合には1秒間ロック要求の応答を待つ．

Step 3. もし，既に全ての **LB** からの応答が **Load Balancing Group** に含まれていたなら，パートナーの探索は失敗して終了する．さもなければ，応答のあった **LB** 群を，**Load Balancing Group** に加えて拡張する．

Step 4. ロック要求の応答に含まれている情報から，**Load Balancing Group** に属する **Solver** が保持している平均子問題数を計算する．

Step 5. 以下の条件を満足する **LB** をパートナーの候補とし，子問題数の多い **Solver** と対をなす **LB** から順に優先順位をつける：

・イニシエータでなく，しかも対をなす **Solver** は負荷分散のための子問題の送受信をしていない．
・対をなす **Solver** の子問題数が，**Load Balancing Group** に属する **Solver** が保持している平均子問題数よりも多い．

Step 6. イニシエータは，優先順位の高いパートナーの候補から順に子問題の送受信が可能であるかどうかを調べる．子問題の送受信が可能であれば，パートナーの選択は成功して終了する．子問題の送受信が不可能であれば（この選択処理中にも状態は変化するため，不可能なこともあり得る），次の優先順位のパートナーの候補を次々と調べる．**Load Balancing Group** 内にパートナーの候補がいなくなった場合には **Step 2** へ．

子問題の溢れが通知されたときの処理方法もほぼ同様の手順である．このようなパートナーの選択処理は，各 **LB** で独立して行われる．ここで，パートナーの選択は，基本的に，レスポンスタイムで定義される距離の近い **LB** が優先される．

このようにして成立した **Solver** 間の子問題の送受信は，子問題評価計算が 1 回終了するごとに 1 個の子問題の受け渡しをする．1 個の子問題さえ受信すれば，受信側の **Solver** は子問題評価計算が実行でき，その後受信側のみで子問題数を増やすことができる．そのため，並列分枝限定法全体で転送される総子問題数がおさえられる．このような送受信は，送信側あるいは受信側のどちらかの子問題数が，**Load Balancing Group** に属する **Solver** が保持する平均子問題数に達するまで続けられる．イニシエータ側が先に，その平均に達した場合には，この負荷分散処理は終了する．しかし，パートナー側が先に平均に達した場合，イニシエータは再度，パートナーの探索を開始し，新たなパートナーを見つけて負荷分散処理を行う．

子問題の評価と送受信(2). ある **Solver** で暫定解が更新された場合，その解は **PM** へ通知され，その後，暫定値が全ての **LB** へ通知される．各 **LB** は，対をなす **Solver** の暫定値を更新する．

終了時. **Solver** は，子問題の枯渇が生じて計算を行っていない IDLE 状態か，子問題の溢れが生じて子問題を処理できない FULL 状態になったときにのみ，**PM** へ **Solver** の情報を通知する．全ての **Solver** が IDLE であることを **PM** が検知したとき，**PM** が持つ暫定解を最適解として出力し終了する．あるいは，全ての **Solver** が FULL であることを **PM** が検知したとき，**PM** が持つ暫定解を最良解として出力し終了する．

3.2.3 Master-Slave to Fully-Distributed（MS-FD）モード

まず，MS モードとして初期化し，実行する．その後，**PM** が保持する子問題数がパラメータで指定された値を上回ったとき FD モードへ切替える．切替え時には，モードの切替え要求を **LB** 経由で全ての **Solver** へ通知する．この通知を受けた **Solver** は，通知に対する応答を **PM** へ返し，その後，受けとった子問題の評価計算により生成される子問題群を **PM** へ返却せず，自タスク内の子問題プールとのやりとりによるローカルな逐次処理の分枝限定法の実行に切り替える．**PM** は，全ての **Solver** からのモードの切替え要求に対する応答を受信すると，**PM** 内に残っている子問題群を（最小化問題の場合）下界値の良い子問題から順にラウンド・ロビン方式によって，1 個ずつ全ての **Solver** へ送信する（子問題プールは，子問題選択規則順，最良下界値順，最悪下界値順に操作できるデータ構造を持つ．詳しくは，Shinano et al. (1997) を参照）．このような分配により，FD モードへの切替えが終了した時点での各 **Solver** の持つ子問題群は，子問題数のみならず，各子問題群の下界値の合計という点でもほぼ均等に分散した状態

となる．その後の動作はFDモードと同じである．

4. 組合せ最適化問題への適用例

4.1 巡回セールスマン問題

本節では，対称型巡回セールスマン問題に対する数値実験を通じて，並列化の効果を示す．特に，前節で紹介した3種類の動作モードおよび2種類の子問題探索規則（下界値優先探索，ハイブリッド探索）の比較・検討を行う．子問題の評価計算には，1-木による緩和問題を利用したHeld and Karp (1970, 1971) の解法を用いた．なお，詳細はShinano et al. (1997) に

表1. 逐次処理での計算時間（秒）．

問題名	下界値優先探索				ハイブリッド探索			
	実行時間	1子問題評価時間			実行時間	1子問題評価時間		
		平均 (標準偏差)	最大	最小		平均 (標準偏差)	最大	最小
berlin52	343.190	2.373(1.442)	5.640	0.320	32.700	2.047(1.893)	5.880	0.360
dantzig42	16605.200	1.351(0.629)	3.760	0.010	6539.660	0.931(0.845)	3.780	0.020
eil101	4859.470	6.201(3.553)	21.390	0.570	5850.590	5.541(4.199)	21.750	0.170
eil51	717.640	1.324(0.784)	4.820	0.070	1120.830	1.321(0.838)	5.300	0.030
eil76	447.130	4.425(0.859)	8.680	1.890	618.430	3.373(2.580)	11.720	0.110
gr48	26152.500	1.221(0.950)	4.730	0.060	23605.860	1.155(0.980)	4.910	0.020
rat99	687.980	6.733(1.069)	8.720	1.820	2027.100	6.008(2.953)	15.430	0.150
rd100	50463.030	7.980(5.619)	22.160	0.180	36573.590	6.895(5.676)	22.170	0.160
st70	14209.280	3.598(0.848)	9.550	0.390	15245.440	2.798(2.372)	10.280	0.060
swiss42	23.070	1.132(0.747)	2.870	0.290	23.130	0.964(0.919)	3.480	0.060

表2. 逐次処理での計算結果．

問題名	下界値優先探索				ハイブリッド探索			
	最大プール使用量	生成子問題数	評価子問題数	暫定解の更新回数	最大プール使用量	生成子問題数	評価子問題数	暫定解の更新回数
berlin52	92	221	141	2	3	15	15	1
dantzig42	10661	24429	12217	2	891	7537	6966	6
eil101	672	1362	778	1	170	1322	1049	3
eil51	490	999	534	2	527	1601	843	5
eil76	192	291	100	1	56	229	180	3
gr48	8274	26612	21281	2	3732	22315	20292	8
rat99	174	273	101	2	164	654	335	10
rd100	1686	8122	6280	3	585	5703	5269	4
st70	5697	10502	3923	3	564	5568	5418	4
swiss42	19	30	15	1	6	25	23	1

述べられている．

4.1.1 予備実験

まず，逐次処理による数値実験を行った．ここで，適用した子問題の評価計算部は，並列処理で使用するものと同一である．データは，世界中の研究者が利用するベンチマーク問題集 TSPLIB の中から 10 問使用した．動作環境は，IBM RS/6000 Model 25 T (CPU：PowerPC 601，66 MHz，メモリ：64 M) である．各問題の計算時間が表 1 に示され，また，主な特徴が表 2 に示されている．

表 1 および表 2 より，問題のサイズ（問題名の数値が都市数を示す）が同程度でも，計算時間および評価した子問題数は大きく異なることがわかる．また，同じデータでも子問題によって，評価時間が大きくばらつく．これは，暫定解を更新することができないと判断できた時点で，評価計算が終了するためである．

表 2 によると，暫定解の更新回数が多いのはハイブリッド探索の方である．ハイブリッド探索は，初めて子問題が見切られるまで深さ優先探索を行い，次に，下界値の最も良い子問題を選択して再び深さ優先探索を行う，という操作を繰り返す．ここで，深さ優先探索を行うため，実行可能解が生成される頻度が高く，暫定解の更新回数も多くなる．

さらに，深さ優先探索開始の子問題は，最良下界値を基準に選択されるため，単純な深さ優先探索より，比較的良い実行可能解が早期に得られ，生成子問題数が抑えられると考えられる．実際，一般に，単純な深さ優先探索は下界値優先探索よりも生成子問題数が多いが，表 2 の多くのケースで見られるように，ハイブリッド探索は下界値優先探索よりも生成子問題数は少なく，また最大プール使用量も少ない．したがって，ハイブリッド探索は限定操作を効率よく機能させる探索であるといえる．

4.1.2 動作モードによる効果比較

3 種類の動作モードの効果比較のため，逐次処理で計算に時間を要した，dantzig 42，gr 48，rd 100，st 70 の 4 問を並列処理で解くことにした．動作環境は，イーサネットにより接続された，逐次処理で使用したものと同じワークステーション群である．**Solver** は，ワークステーション 1 台につき 1 個起動し，2，3，5，7，10，20，40，70，100 個使用する場合について実験を行った．ただし，**PM** は **Solver** とは別のワークステーションで動作させたため，最高 101 台のワークステーションを使用した．FD モードおよび MS-FD モードでは，閾値を 3 とした．MS-FD モードでは，**PM** の管理する子問題プール中の子問題数が，(閾値＋1)×(**Solver** 数) を超えたとき，FD モードへ切り替えた．このパラメータ設定では，計算開始から短時間で FD モードに切り替わると考えられる．並列分枝限定法では，制御不可能な要因によって結果が異なる場合がある．そのため，以下の結果は 5 回の実行の平均を示している．紙数の制限により，結果の一部を図 2，3，4，5 に示す．図 3，5 の凡例は，図 2，4 の凡例と同じである．凡例中の Best は下界値優先探索を示し，Hybrid はハイブリッド探索を示す．dantzig 42 と rd 100 は似た傾向を示し，gr 48 と st 70 も似た傾向を示した．

図 2，4 において，同じ **Solver** 数，同じ探索規則で 3 種類のモードの比較をすると，多くの場合 MS モードの計算時間が最も短い．このように MS モードが良好な結果を示す背景は，本実験で用いた問題では，1 個の子問題の評価計算に十分な時間を費やしている（1 秒以上）ため，MS モードで発生しうるボトルネックが生じにくいことである．

図 5 では，FD モードにおいて評価された子問題数が，**Solver** 数の増加に伴って大幅に増加している．これは，FD モードでは，幾つかの **Solver** が，グローバルな視点で下界値の良くない子問題ばかりを処理したためであると考えられる．しかし，この現象は MS-FD モードの場合に

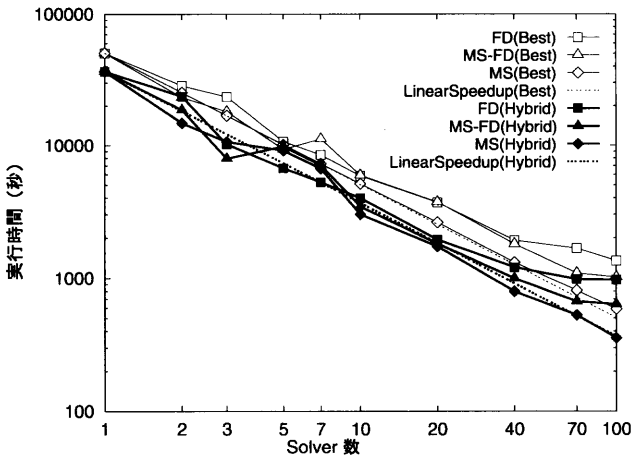

図2. Solver 数の増加に対する実行時間（問題名：rd 100）.

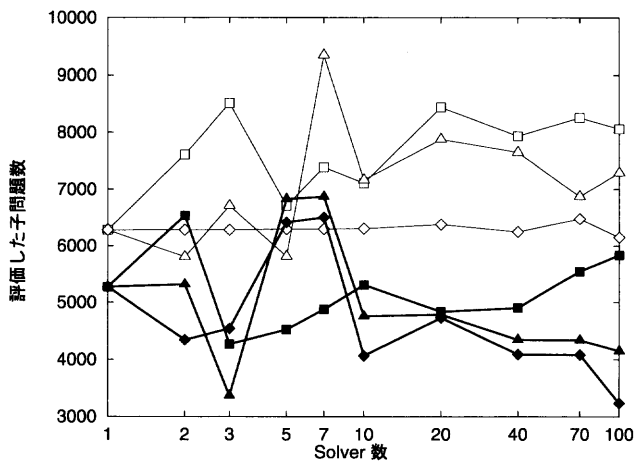

図3. Solver 数の増加に対する評価した子問題数（問題名：rd 100）.

は大幅に改善されている．また，図3，5より，ハイブリッド探索では，下界値優先探索と比較すると，**Solver** 数が増加しても評価される子問題数をおさえる傾向があることがわかる．

4.2 最大クリーク問題

本節では，最大クリーク問題に対する数値実験結果を通じて，並列化の効果を示す．本節の特徴は，子問題の評価計算時間がワークステーション間の伝達時間よりもはるかに短いことにある．その結果として，PUBB の子問題管理に改良を要した．以下では，改良した点とその効果について報告する．子問題の評価計算には，頂点彩色上界を求める貪欲算法を用いた（例えば Balas and Xue(1996) を参照のこと）．なお，詳細は Shinano et al. (1998) に述べられている．

4.2.1 PUBB の改良

動作環境は，IBM RS/6000 Model 25 T（CPU : PowerPC 601，66 MHz，メモリ：64 M）であり，イーサネットで結ばれている．また以下では，頂点数 200，枝密度 90%のランダムグ

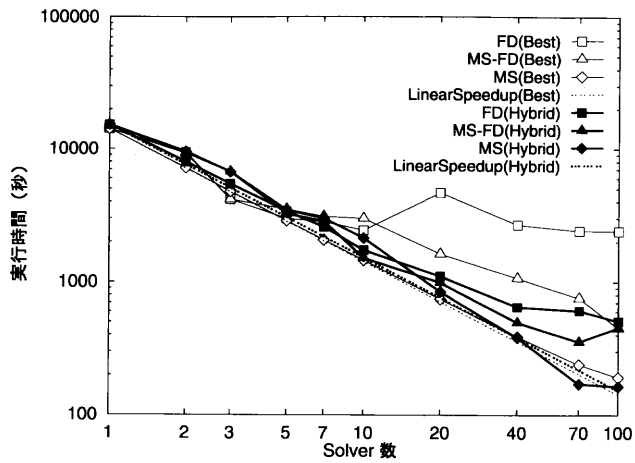

図 4.　Solver 数の増加に対する実行時間（問題名：st 70）.

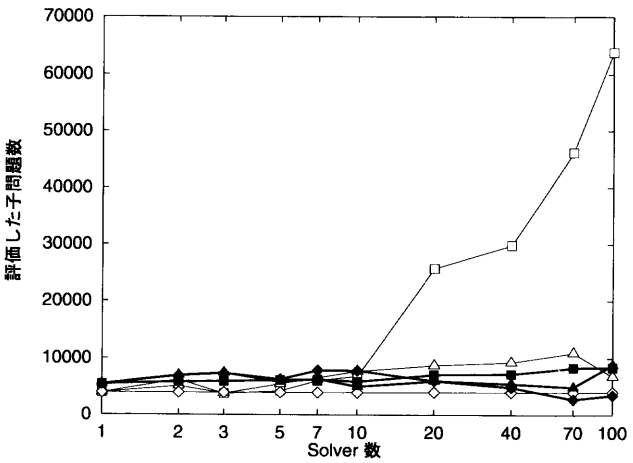

図 5.　Solver 数の増加に対する評価した子問題数（問題名：st 70）.

ラフ 10 問を対象にする．これらの問題を逐次処理で解いた結果を表 3 に示す．

PUBB の改良 1. MS モードで実行する場合，1 個の子問題評価計算が終るたびに **PM** と **Solver** の間で子問題の送受信を行っている．したがって，本節のように，送受信時間が子問題評価時間よりも長いときには並列化の効果は期待できない．

一方，FD モードの場合，**Solver** が管理する子問題プールがほとんど空，もしくは溢れが発生したときにのみ，子問題が **Solver** 間で送受信される．さらに，対をなす **LB** がパートナーを探索している間，**Solver** は子問題評価計算を中断されることはない．したがって，このモードでは改良が見込まれる．

FD モードの場合，各 **Solver** は 1 個の子問題評価計算を終えるたび，対をなす **LB** に子問題プールの情報を送る．各 **LB** は，その情報を基にパートナーを探索する．ここで，**Solver** と **LB** の間の通信はワークステーション内で行われるが，その通信時間は，子問題評価時間が短い場合には無視できなくなる．そこで，**Solver** が子問題プールの情報をある一定時間（α 秒）送ら

表3. 逐次処理による計算結果（頂点数200，枝密度90%）.

問題番号	計算時間(秒)	子問題評価の平均時間(標準偏差)(秒)	生成された子問題数	評価された子問題数	暫定解の更新回数
1	24274.93	0.0012 (0.0033)	17076112	17076112	1
2	21910.94	0.0013 (0.0034)	14635886	14635886	2
3	31418.37	0.0012 (0.0033)	22242646	22242646	0
4	12842.94	0.0013 (0.0034)	8360920	8360920	0
5	23307.84	0.0012 (0.0033)	16495272	16495272	0
6	9959.94	0.0011 (0.0031)	7862052	7862052	0
7	25392.77	0.0013 (0.0034)	17019196	17019196	0
8	23601.35	0.0013 (0.0033)	16161238	16161238	0
9	19017.74	0.0012 (0.0033)	13572984	13572982	1
10	17147.96	0.0013 (0.0033)	11570975	11570975	0

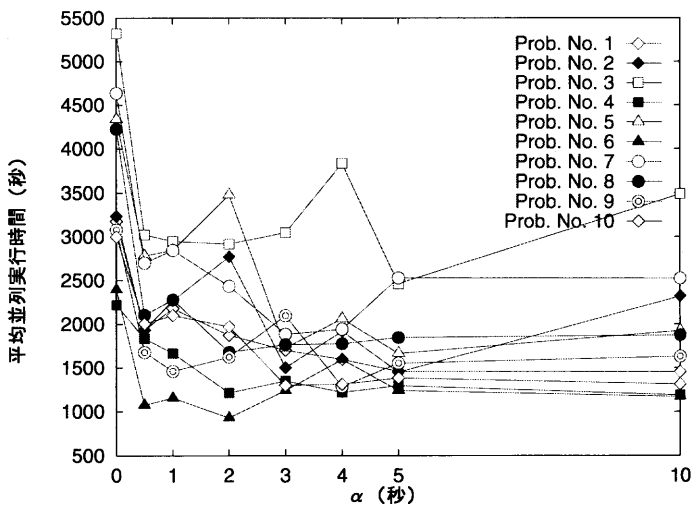

図6. αと並列実行時間（Solver数20）.

ないようにした．ただし，(1)子問題プールがほとんど空，すなわち，残りの子問題数がパラメータよりも少ない，もしくは，(2)**Solver**が子問題の送受信中である場合は例外とする．

αを正の値に設定することによって，**LB**と**Solver**の間の通信コストが削減される．しかし，αがあまりにも大きく設定されると**LB**の認識する情報が正確でなくなり，誤った負荷分散を行い，かえって全体の並列実行時間が遅くなる可能性がある．そのために，**Solver**数20でαの設定を行った．図6は，平均並列実行時間を示している．この図が示すように，αを1まで増加させると，並列実行時間は大幅に減少することがわかる．しかし，αを1より大きくしていくとそれ以上の効果は見られない．したがって，α=1とした．

表 4. 子問題の転送戦略の違いによる効果.

問題番号	転送戦略	並列実行時間 (秒)			転送子問題数		
		平均	最小	最大	平均	最小	最大
1	深さ優先	2179.28	1501.61	2884.47	259991	170106	327651
	上界値優先	1006.08	906.00	1372.75	13631	10511	15794
2	深さ優先	2269.64	2017.45	2562.68	252986	198468	321987
	上界値優先	1082.22	1077.04	1092.12	14443	12409	16685
3	深さ優先	2947.66	2283.82	4071.38	328107	170455	488439
	上界値優先	1950.58	1933.30	1974.37	20101	17504	22769
4	深さ優先	1668.40	1015.86	1968.98	172703	90678	237112
	上界値優先	821.63	809.89	831.06	14049	10683	16617
5	深さ優先	2845.97	2295.11	3652.39	380516	233904	480927
	上界値優先	1468.25	1452.99	1492.37	18750	15331	23181
6	深さ優先	1156.16	938.65	1575.85	120232	79115	200219
	上界値優先	769.58	751.97	796.58	11766	9511	14708
7	深さ優先	2845.39	2357.35	3436.10	331042	262169	419683
	上界値優先	1570.68	1548.15	1599.40	17451	13979	25608
8	深さ優先	2279.22	1709.44	2502.05	231953	96859	369319
	上界値優先	1462.07	1445.58	1492.34	15378	12237	17268
9	深さ優先	1462.18	1222.38	1611.45	96955	59047	145142
	上界値優先	1214.67	956.86	1359.50	10862	7358	16427
10	深さ優先	2100.77	1534.42	2725.48	223286	147135	287155
	上界値優先	1073.00	1055.16	1099.78	11772	8500	17259

表 5. 計算結果.

問題名	頂点数	枝密度 (%)	計算時間 (sec)	生成された子問題数	評価された子問題数	暫定解の更新回数
C250.9	250	89.9	25768.79	687551803	687551803	0
p_hat500-3	500	75.2	856.65	12398047	12398047	0
p_hat700-3	700	74.8	10637.92	131620573	131620573	0
p_hat1000-2	1000	49.0	1111.23	16098000	16098000	0
p_hat1500-2	1500	50.6	61694.40	628386474	628386468	1

PUBB の改良 2. 次に子問題の探索方法に注目する. 本節では深さ優先探索を適用したが, FD モードの場合, 各 **Solver** は深さ優先探索の逐次分枝限定法を実行し, さらに **Solver** 間の子問題の送受信においても, 送信側の **Solver** は深さ優先で子問題を選択し送信する (深さ優先転送). そして子問題を受け取った **Solver** は, その子問題を優先して解いていた.

しかし, この方法では, 送信される子問題は, 将来さらに多くの子問題を生成しないと思わ

れる．つまり，子問題評価計算時間に対して通信時間が長い場合には，**Solver** 間で行われる子問題の送受信の効果が期待できない．そのため，送信側の **Solver** は上界値優先で子問題を選択し送信することにした(上界値優先転送)．表4に，転送戦略の違いによる，並列実行時間および転送された子問題数を示す．この表からわかるように，送受信される子問題数が劇的に減少し，並列実行時間も大幅に減少した．

4.2.2 DIMACS ベンチマーク問題の結果

最後に，世界中の研究者が利用する DIMACS のベンチマーク問題集の中から5問を選び，それらを解いた結果を示す．我々の知る限りでは，これらの問題は今まで最適性の保証がなされていなかったものである．

表5は **Solver** 数50で並列分枝限定法を実行した結果である．この表が示すように，暫定解の更新回数がほとんどないことより，今回使用したヒューリスティック解法 STABJOIN (Ikebe and Tamura(1995)) は強力であった．それにもかかわらず，生成された子問題数は膨大であったが，前節のような改良を施すことによって並列化の効果が示された．この結果から，並列分枝限定法が現実的な時間内で解ける問題例を増やすためには，重要なアプローチであることが示される．

5. まとめと今後の課題

本稿では，並列分枝限定法の汎用ツール PUBB の設計および有効性について説明した．今後はさらに多くの組合せ最適化問題へ適用することにより，汎用ツールとしての有効性を高める必要があると思われる．また，近年では，分枝限定法に切除平面法を組み入れた分枝カット法が提案され注目されている．分枝カット法は，分枝限定法と同様に汎用的な構造を有しており，ツール化を行うことが PUBB の今後の課題である．

謝　　辞

本研究における開発・デバッグ環境の入手および，常時その計算機環境の改善に配慮して下さった，統計数理研究所・水野眞治助教授に深く感謝致します．また，本稿の不適切な表現を指摘し，改稿にあたって非常に参考となるコメントをして下さった査読者のかたに感謝致します．

付録：C 言語による PUBB のユーザ・インターフェース

現在，PUBB の公開準備中である(詳細は，shinano@ms.kagu.sut.ac.jp，fujie@kobeuc.ac.jp まで)．本付録では，PUBB を利用して分枝限定法を実行するためのユーザ・インターフェースについて説明する．ユーザが設計するのは以下の通りである：

ユーザ構造体……問題データ，子問題，実行可能解をそれぞれ表現する構造体．

ユーザ関数……実行可能解(初期暫定解)を計算する（**分枝限定法**の Step 2）関数，子問題を評価する関数（同じく Step 7）など．

また，ユーザ関数設計の際，

サービス関数……暫定値を返す関数，子問題の深さを返す関数など．

が PUBB に用意されている．

A.1 節から A.3 節において，これらの詳細な解説を行う．これ以降，ユーザ構造体とそのメンバ名は先頭に 'Pbbi' を付け，ユーザ関数とサービス関数名は先頭に 'pbbi' を付ける．'Pbbi' および 'pbbi' は，'Parallel Branch-and-Bound Interface' の省略形である．

A.1 ユーザ構造体

□ **PbbiInstance**　問題データを表現する構造体である．構造体のメンバはユーザが設計するが，次の 2 つは必ずメンバにする：

PbbiProblemType：最大化問題（PbbiMaximize），最小化問題（PbbiMinimize）を示す．
PbbiDominanceTest：子問題の優越関係をユーザがテストする(PbbiUse)，しない(PbbiNoUse) を示す．

すなわち，書式は次のように与える：

```
typedef struct PbbiInstance_t{
  enum PbbiProblemType_e{PbbiMaximize=0, PbbiMinimize=1}
                        PbbiProblemType ;
  enum PbbiDominanceTest_e{PbbiNoUse=0,    PbbiNoUse=1}
                          PbbiDominanceTest :
  /*上は PUBB に必要なメンバである．                    */
  /*続いて，ユーザは自由にメンバを宣言することができる．  */
          :
}PbbiInstance ;
```

他のユーザ構造体も同様である．

□ **PbbiSubproblem**　子問題を表現する構造体であり，次の 3 つは必ずメンバにする：

PbbiBound :　子問題の下界値（最小化問題の場合）を示す，double 型変数．
PbbiPriority :子問題プール中における優先順位を示す，double 型変数．値が小さいほど優先順位が高い．
PbbiInternalVariables : pbbiEvaluation () 関数内で利用された内部変数（PbbiEvaluation()で確保されたメモリ領域にある変数）が，親問題の評価計算終了時の状態であるか（CanUse）否か（CannotUse）を示す．
親問題の評価計算終了時の状態が保証される時には，余分な計算を省略し内部変数の内容を評価計算に利用できる．

□ **PbbiSolution**　実行可能解を表現する構造体であり，次の 1 つは必ずメンバにする：

PbbiObjectiveValue：目的関数値を示す，double 型変数．

A.2 ユーザ関数

ユーザ関数は 6 つのグループに分類される．あるグループに属する関数は，他のグループに属する関数との共有変数をもってはいけない．グループの異なる関数は，プログラム・ファイ

ルを別にすることが推奨される.

A.2.1 入出力関数

□ **pbbiReadProblem** 問題データを読み込み, PbbiInstance 型にセットアップする関数である.

書式

```
PbbiInstance *instance=ppbiReadProblem (int argc, char **argv)
```

□ **pbbiPrintSolution** 最適解を出力する関数である.

書式

```
void pbbiPrintSolution (FILE *fp, PbbiInstance *instance,
                        PbbiSolution *solution)
```

最適解は, ユーザ構造体 PbbiSolution 型へのポインタ solution として与えられる. また, FILE の open/close は PUBB が行う.

A.2.2 初期実行関数

□ **pbbiGetInitialSolution** ヒューリスティック解法などにより実行可能解(初期暫定解)を計算する関数である.

書式

```
PbbiSolution *solution=pbbiGetInitialSolution (PbbiInstance *instance)
```

□ **pbbiGetRootProblem** PbbiInstance 型で保存されている問題データから PbbiSubproblem 型の元問題を作る関数である.

書式

```
PbbiSubproblem *subproblem=pbbiGetRootProblem (PbbiInstance *instance)
```

A.2.3 評価関数

□ **pbbiInitializeEvaluation** 問題の評価を始める前に(必要ならば)初期化を行う関数である. すなわち, pbbiEvaluation() を含むファイル内で静的変数あるいは動的変数を用い, その初期化が必要な場合等に利用する.

書式

```
void pbbiInitializeEvaluation (PbbiInstance *instance)
```

□ **pbbiEvaluation** PUBB から送られる子問題の評価を行う関数である.

書式

```
void pbbiEvaluation (PbbiInstance *inInstance,
                     PbbiSubproblem *inSubproblem,
                     PbbiSolution **outSolution,
                     PbbiSubproblem ***outSubproblems)
```

入力は, 問題データ *inInstance と子問題 *inSubproblem である. 更新した暫定解があれば *outSolution へ解へのポインタを返し, なければ, *outSolution=NULL として返す. 子問題がK個生成された場合,*outSubproblems[0] から *outSubproblems[K-1] に子問題へのポイ

ンタを格納し，さらに *outSubproblems[K]=NULL を返す．子問題が生成されなかった場合は，*outSubproblems=NULL として返す．

□ **pbbiFinalizeEvaluation**　すべての子問題の評価が終わった後の処理を行う関数である．すなわち，pbbiInitializeEvaluation() でメモリを確保した動的変数の領域等，後処理が必要な場合に利用する．

書式

```
void pbbiFinalizeEvaluation()
```

A.2.4　メモリ解放関数

□ **pbbiFreeInstance, pbbiFreeSubproblem, pbbiFreeSolution**　すでに確保されたユーザ構造体のメモリ領域を解放する関数である．

書式

```
void pbbiFreeInstance (PbbiInstance *instance)
void pbbiFreeSubproblem (PbbiSubproblem *subproblem)
void pbbiFreeSolution (PbbiSolution *solution)
```

A.2.5　Pack/Unpack 関数

□ **pbbiPackInstance, pbbiPackSubproblem, pbbiPackSolution**　PVM の提供する基本 pack 関数を利用して，ユーザ構造体を pack する．戻り値は，PVM の基本 pack 関数に準じて，result が 0 未満はエラーを示す．

書式

```
int result=pbbiPackInstance (PbbiInstance *instance)
int result=pbbiPackSubproblem (PbbiSubproblem *subproblem)
int result=pbbiPackSolution (PbbiSolution *solution)
```

□ **pbbiUnpackInstance, pbbiUnpackSubproblem, pbbiUnpackSolution**　PVM の提供する基本 unpack 関数を利用して，ユーザ構造体を unpack する．戻り値は，PVM の基本 unpack 関数に準じて，result が 0 未満はエラーを示す．

書式

```
int result=pbbiUnpackInstance (PbbiInstance *instance)
int result=pbbiUnpackSubproblem (PbbiSubproblem *subproblem)
int result=pbbiUnpackSolution (PbbiSolution *solution)
```

A.2.6　Dump 関数

□ **pbbiDumpInstance, pbbiDumpSubproblem, pbbiDumpSolution**　デバッグのために，ユーザ構造体の情報を出力する関数である．出力のときはサービス関数 pbbiPrintf() を用いる必要がある．

書式

```
void pbbiDumpInstance (PbbiInstance *instance)
void pbbiDumpSubproblem (PbbiSubproblem *subproblem)
void pbbiDumpSolution (PbbiSolution *solution)
```

A.3 サービス関数

□ **pbbiGetIncumbentValue** 暫定値を得る関数である.

書式

```
double pbbiGetIncumbentValue()
```

□ **pbbiGetDepth** pbbiEvaluation() の中でのみ利用可能な関数であり, 入力された子問題の深さを示す.

書式

```
int pbbiGetDepth()
```

□ **pbbiPrintf** ユーザが編集して出力したい情報を PUBB へ渡す. 仕様は, printf() 関数と等しい. 出力先は, PUBB の動作モードに応じて変更される.

書式

```
void pbbiPrintf (char* fmt, ...)
```

A.4 再び分枝限定法について

今まで紹介したユーザ構造体およびユーザ関数の役割を明確にするため, 分枝限定法の C 言語疑似コードを与える.

分枝限定法

```
int main (int argc, char **argv)
{
  int i ;
  PbbiInstance *instance ;
  PbbiSubproblem *subproblem ;
  PbbiSolution *solution ;
  PbbiSubproblem **outSubproblems ;
  PbbiSolution **outSolution ;

  /* 問題を読み込む */
  instance=ppbiReadProblem (argc,argv) ;
  /* 初期暫定解の計算 */
  solution=pbbiGetInitialSolution (instance) ;
  暫定解 solution を記憶する ;
  /* 元問題の生成 */
  subproblem=pbbiGetRootProblem (instance) ;
  子問題プールに元問題 subproblem を追加する ;
  /* pbbiEvaluation の前処理 */
  pbbiInitializeEvaluation (instance) ;

  /* 分枝限定法のメインループ */
  while (subproblem=PUBB から子問題を得る ){
    /* 子問題 subproblem の計算 */
    pbbiEvaluation (instance, subproblem, &outSolution, &outSubproblems) ;
    /* 暫定解の更新 */
```

```
    if (outSolution)
      暫定解 outSolution を更新・記憶する ;
    /* 子問題の生成 */
    if (outSubproblems)
      for (i=0 ; outSubproblems[i] ; i++)
        子問題プールに outSubproblem[i] を追加する  ;
  }

  /* pbbiEvaluation の後処理 */
  pbbiFinalizeEvaluation () ;
  /* 最適解の出力 */
  pbbiPrintSolution() ;
  instance のメモリを解放する ;

  return 0 ;
}
```

参 考 文 献

Balas, E. and Xue, J. (1996). Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring, *Algorithmica*, **15**, 397-412.

Benaïchoche, M., Cung, V. D., Dowaji, S., Cun, B. L., Mautor, T. and Roucairol, C. (1996). Building a parallel branch and bound library, *Lecture Notes in Comput. Sci.* (eds. A. Ferreira and P. Pardalos), No. 1054, 201-231, Springer, Berlin.

Brüngger, A., Marzetta, A., Clausen, J. and Perregaard, M. (1997). Joining forces in solving large-scale quadratic assignment problems in parallel, *Proceedings of the 11th International Parallel Processing Symposium*, 418-427, IEEE Computer Society Press, Los Alamitos, California.

Eckstein, J. (1994). Control strategies for parallel mixed integer branch and bound, *Proceedings of Supercomputing '94*, 41-48, IEEE Computer Society Press, Los Alamitos, California.

Held, M. and Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees, *Oper. Res.*, **18**, 1138-1162.

Held, M. and Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part II, *Math. Programming*, **1**, 6-25.

茨木俊秀 (1983).『組合せ最適化 分枝限定法を中心として』, 産業図書, 東京.

Ikebe, Y. and Tamura, A. (1995). Ideal polytopes and face structures of some combinatorial optimization problems, *Math. Programming*, **71**, 1-15.

池上淳子, 青柳雄大, 飯塚肇(1992). 分散記憶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-I, **J75-D-I-9**, 801-808.

Lai, T. H. and Sahni, S. (1984). Anomalies in parallel branch-and-bound algorithms, *Communications of the ACM*, **27**, 594-602.

Lüling, R. and Monien, B. (1989). Two strategies for solving the vertex cover problem on transputer network, *Lecture Notes in Comput. Sci.* (eds. J-C. Bermond and M. Raynal), No. 392, 160-170, Springer, Berlin.

Lüling, R., Monien, B., Reinefeld, A. and Tschöke, S. (1996). Mapping tree-structured combinatorial optimization problem onto parallel computers, *Lecture Notes in Comput. Sci.* (eds. A. Ferreira and P. Pardalos), No. 1054, 115-144, Springer, Berlin.

Pardalos, P. M. and Rodgers, G. P. (1990). Parallel branch and bound algorithms for quadratic zero-one programs on the hypercube architecture, *Ann. Oper. Res.*, **22**, 271-292.

Pekny, J. F. and Miller, D.L. (1992). A parallel branch and bound algorithm for solving large symmetric

traveling salesman problems, *Math. Programming*, **55**, 17-33.

Quinn, M. (1990). Analysis and implementation of branch and bound algorithms on a hypercube multicomputer, *IEEE Trans. Comput.*, **39**, 384-387.

Shinano, Y., Higaki, M. and Hirabayashi, R. (1995). A generalized utility for parallel branch and bound algorithms, *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, 392-401, IEEE Computer Society Press, Los Alamitos, California.

品野勇治，桧垣正浩，平林隆一(1996)．並列分枝限定法における解の探索規則，計測自動制御学会論文集，**32-9**，1379-1387.

Shinano, Y., Higaki, M. and Hirabayashi, R. (1996). An interface design for general parallel branch-and-bound algorithms, *Lecture Notes in Comput. Sci.* (eds. A. Ferreira, J. Rolim, Y. Saad and T. Yang), No. 1117, 227-284, Springer, Berlin.

Shinano, Y., Harada, K. and Hirabayashi, R. (1997). Control schemes in a generalized utility for parallel branch-and-bound algorithms, *Proceedigs of the 11th International Parallel Processing Symposium*, 621-627, IEEE Computer Society Press, Los Alamitos, California.

Shinano, Y., Fujie, T., Ikebe, Y. and Hirabayashi, R. (1998). Solving the maximum clique problem using PUBB, *Proceedings of the 12th International Parallel Processing Symposium*, 326-332, IEEE Computer Society Press, Los Alamitos, California.

Taudes, A. and Netousek, T. (1991). Implementing branch-and-bound algorithms on a cluster of workstations —A survey, some new results and open problems, *Lecture Notes in Econom. and Math. Systems*, No. 367, 79-102, Springer, Berlin.

Tschöke, S. and Polzer, T. (1996). *Portable Parallel Branch-and-Bound Library (PPBB-Lib): User Manual Version 1.1*, University of Paderborn, Germany.

Proceedings of the Institute of Statistical Mathematics Vol. 46, No. 2, 411-431 (1998)

Exact Algorithms for Combinatorial Optimization Problems Using PUBB —— A Generalized Utility for Parallel Branch-and-bound Algorithms ——

Yuji Shinano

(Department of Management Science, Science University of Tokyo)

Tetsuya Fujie

(Department of Management Science, Kobe University of Commerce)

The branch-and-bound algorithm is mostly applied for solving combinatorial optimization problems exactly. In practice, however, there is a limit of a problem size which can be solved in a reasonable amount of time. Parallelization of the branch-and-bound algorithm is an important approach to solve more problems beyond the limit, and according to the recent advance of parallel machines, much effort has been devoted to the parallelization. Especially, based on the general framework of the branch-and-bound algorithm, generalized tool development is a trend of the implementations.

PUBB (Parallelization Utility for Branch-and-Bound algorithms), developed by Shinano et al., is one of the generalized tools. Since the first development of PUBB, several improvements have been made in order to apply to a wide variety of problems, and the current PUBB realizes a central, a distributed and a mixed control schemes of a search tree management.

In this paper, we provide an outline of PUBB, in which the three control schemes are introduced. We also provide applications of PUBB to classical combinatorial optimization problems, the Traveling Salesman Problem and the Maximum Clique Problem. We are now preparing PUBB available via the Internet, and as an appendix, we give the user interface for PUBB designed with the C language.

Key words: Branch-and-bound algorithm, parallel processing, combinatorial optimization, traveling-salesman problem, maximum clique problem.