分枝限定法並列化ツールPUBB

品野　勇治

1. はじめに

分枝限定法は，組合せ最適化問題に対する列挙解法に基づく古典的なアルゴリズムである．1970年代初頭に，M.HeldとR.Karpが分枝限定法により巡回セールスマン問題を解くことに成功して以来，様々な組合せ最適化問題に適用されるようになった．その後，解法の一般化も行われ，今では分枝限定法は組合せ最適化問題に対する厳密解法を設計するための基本的な道具として定着している．しかし，よく知られている事実ではあるが，たとえ分枝限定法を用いても，問題の規模が大きくなると現実的な時間内では解が得られなくなる[1]．

1980年代後半になり，並列計算機が実用化されるにつれ，分枝限定法に並列処理を取り入れる試みがなされるようになる．当初は，対象とする問題個別に，利用する計算機アーキテクチャの特徴を生かして実装された．並列分枝限定法の実装は，並列計算機の知識と対象とする解法に対する知識の両方を持ち合わせる必要があり，困難であった．1990年代になると，特定の問題を特定の並列計算機環境へ実装するのではなく，分枝限定法の汎用的なフレームワークをツール化することで，様々な問題に対する並列分枝限定法の開発を容易にする汎用のシステムがいくつか登場した．本稿では，それらのツールの1つである分枝限定法並列化ツールPUBB(Parallelization Utility for Branch-and-Bound algorithms)について，特に利用者を対象として紹介する．PUBBを利用した並列分枝限定法の開発には，C言語およびC++言語が可能である．しかし，本稿ではC言語による開発を前提として解説する．また，説明は最小化問題を仮定して行う．

しなの ゆうじ 東京農工大学工学部情報コミュニケーション工学科
〒184-8588 東京都小金井市中町2-24-16
E-mail:yshinano@cc.tuat.ac.jp

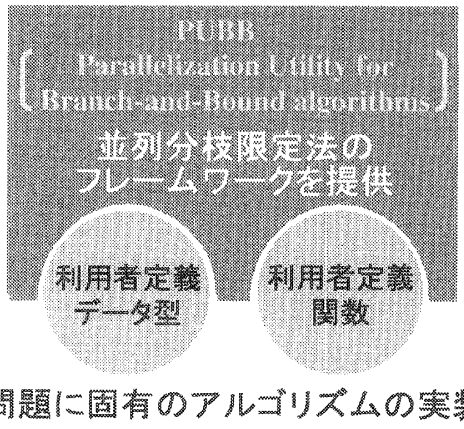

図1: 汎用ツールPUBB

2. PUBBの概要

PUBBは，以下の特徴をもつ並列分枝限定法の開発・実行環境を構築するための汎用ツールである．並列・分散環境の構築，データ転送には標準メッセージパッシング・ライブラリの1つであるPVM(Parallel Virtual Machine)を利用している．よって，PVMが動作する並列計算機，ワークステーション・クラスタ，PCクラスタ上で動作可能である．

汎用ツール 並列分枝限定法のフレームワークを提供する汎用ツールである．ユーザは問題固有の利用者定義データ型と，それを用いた利用者定義関数をプラグインとして開発することで，各問題に対する並列分枝限定法が実現できる(図1)．

逐次処理として開発可能 PUBBでは，プラグインするデータ型・関数の開発環境として，同じインターフェースを持つ逐次分枝限定法の実行ファイルを生成する開発キットを提供する．このため，ユーザは並列処理を意識することなく，並列分枝限定法のプログラムが開発できる．ユーザが開発したプラグインは，再コンパイル程度の手間で並列化される(図2)．

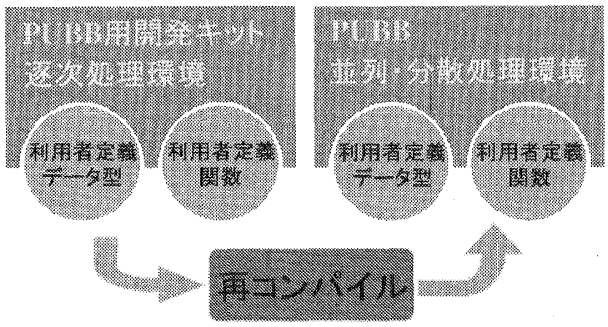

図 2: PUBB による並列化

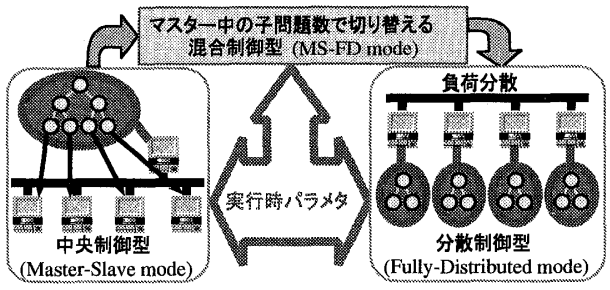

図 3: PUBB が提供する制御方式

多様な探索規則 PUBB では，標準として深さ優先，広がり優先，下界値優先，ユーザが定義した子問題の優先順位の昇順・降順の各探索規則をプログラム実行時に指定できる．また，PUBB の開発過程において提案・実装された探索規則で，深さ優先探索と下界値優先探索を組み合わせた**ハイブリッド探索**も標準で指定できる．ハイブリッド探索では，まず，深さ優先探索が適用される．実行可能解が見つかるか，暫定解よりも良い実行可能解が見つからないことが保証されることで分枝が停止すると，生成された子問題群の中から最良の下界値の子問題を選択する．その後，分枝が停止するまで，再度深さ優先探索が適用される．分枝が停止すると再び最良の下界値の子問題を選択する．以上の操作を繰り返す．

多様な制御方式 PUBB では，並列分枝限定法の実装上での代表的な制御方式である中央制御型（Master-Slave モード），分散制御型（Fully-Distributed モード），混合制御型（Master-Slave to Fully Distributed モード）を 1 つのシステムで実現している．また，制御方式は実行時に指定できる (図 3)．

3. PUBB が提供するフレームワーク

PUBB が提供する（並列）分枝限定法のフレームワークの概略を述べる．まず，図 4 に，分枝限定法の疑似コードを PUBB のフレームワークにおける利用者定義関数群により示す．アルゴリズムの記述としては，メモリの解放等冗長な部分もあるが，並列を意識しない自然な表現であることを確認されたい．

```
algorithm Branch_and_Bound
begin
 (a) インスタンスを読み込む;
 (b) 初期解を計算する;
 (c) ルート問題を生成する;
 ℒ = {(ルート問題)};

 while ℒ ≠ ∅ do
  begin
   子問題 P ∈ ℒ を選び，ℒ = ℒ \ {P} とする;
   (d) 子問題の評価計算を行なう;
   (d-1) if 暫定解が更新された then
    begin
     ℒ から不要な子問題を除去する;
     (f) 古い暫定解のメモリを解放する
    end;
   (d-2) if 新たに子問題が生成された then
    ℒ に新しい子問題を追加する;
   (f) 子問題 P のメモリを解放する
  end;

 (e) 解を出力する;
 (f) インスタンスのメモリを解放する
end.
```

図 4: 分枝限定法の疑似コード

$\mathcal{L}$ は子問題プールと呼ばれるものであるが，その管理は PUBB によって行われ，ユーザが考慮する必要はない．実行時のパラメタ指定にしたがって，PUBB が子問題プールから子問題を取り出し，利用者により定義された評価計算を行う関数へ渡す．つまり，分枝木の標準的な探索順序（探索規則）は，利用者定義関数開発時に意識しなくても実行時に変更できる．並列実行環境においては，“(d) 子問題の評価計算を行う” の部分が，複数の計算機上で実行される．また，子問題プールの個数も PUBB の動作モードにより異なる．

Master-Slave モード時は，実行環境全体に 1 つ子問題プールが生成される．子問題プールから子問題が取り出されると，動作環境上で空きのある計算機へ対して送られ，評価後結果として子問題群，あるいは解を

受信する．したがって，このモードでは子問題の転送のコストが大きい．しかし，探索順序の制御はプールが1つであるため正確に行え，子問題の評価計算に時間を要する解法に対しては有効である．

Fully-Distributedモード時は，動作環境上の各計算機に子問題プールが生成され，図4の疑似コードにおけるwhileループの部分は，それぞれの計算機上で，それぞれの子問題プール内の子問題に対して実行される．子問題プールにおいて子問題の枯渇やあふれが起こらない限り，他の計算機との子問題の転送は行わない．しかし，計算機間の負荷分散処理のためオーバーヘッドは生じる．また，分枝木における探索順序の正確な制御は困難となるため，一般に最適解が得られたときに構成される分枝木は大きくなる．評価計算がデータ転送時間と比較して相対的に短時間の場合や，大規模な問題で子問題プールの大きさが1台の計算機では足りない場合に有効である．

Master-Slave to Fully-Distributedモード時は，最初はMaster-Slaveモードとして動作し，パラメタで指定された個数の子問題が生成されるとFully-Distributedモードへ切り替わる．切り替え時には，Master-Slaveモードにおける子問題プールから，各子問題を下界値順に選び，各計算機へラウンド・ロビン方式で分配する．このため，Fully-Distributedモードへ切り替わった時点では，子問題の個数，および，各計算機が保持している子問題の下界値の合計という観点でもほぼ均一になる．この状態からの負荷分散は，過去の数値実験結果では，比較的少なく済み，特に大規模な問題を解く場合に有効であった．

4. アプリケーション開発

PUBBによるC言語での分枝限定法の実装概略を述べる．ユーザが実装する必要があるのは，データ型と問題固有の関数群である．データ型は次の3種類である：

- **インスタンスデータ**
  これは，解くべき問題のデータである．
- **子問題データ**
  各子問題はインスタンスデータと付加的な情報によって記述できる．子問題データは付加的な情報に関するものである．
- **解データ**
  これは，実行可能解のデータである．

図4の擬似コードに対応する問題固有の関数群は，次のように分類される：

(a) **インスタンスデータの読み込み**
インスタンスを読み込み，インスタンスデータを生成する．

(b) **初期解の計算**
一番初めの評価計算に先だって，初期暫定解を計算する．初期解を求めない場合でも，自明なboundのみを持つ解を生成する．

(c) **ルート問題の作成**
ルート問題に関する子問題データを作成する．

(d) **子問題の評価**
子問題の評価計算をする．このとき，

(d-1) 暫定解を更新する解を見つけたら，解データ型で返す．

(d-2) 新しい子問題を生成する必要があれば，それらを子問題データ型で返す．

(e) **解の出力**
初期解とPUBB終了後の解(一般に最適解)を出力する．

(f) **メモリの解放**
データ型のメモリを解放する．

(g) **Dump関数**
デバッグのために，データ型の情報を出力する．

逐次分枝限定法の実装は以上で充分である．そして，並列化のみに必要な関数は次の1つである：

(h) **データのPack/Unpack**
PVMの提供する基本pack/unpack関数を用いて，データ型をpack/unpackする．

5. 並列化の効果について

並列分枝限定法と呼ばれる解法の中には，各子問題の評価計算にも並列処理を適用している場合がある．しかし，ここではPUBBのみによって実現できる，複数の子問題を同時に処理するような並列化に限定する．複数の子問題を同時に評価する並列動作を行った場合，異常加速(speed-up anomaly)，異常損失(detrimental anomaly)という現象が生じることが良く知ら

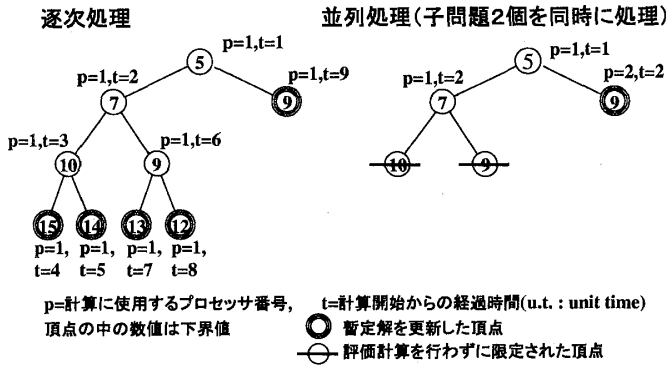

図 5: 異常加速の例

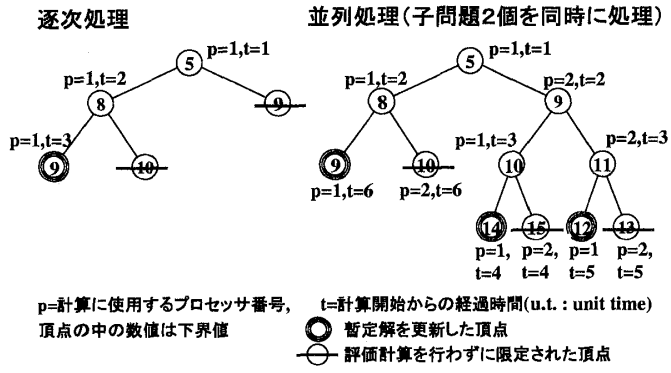

図 6: 異常損失の例

れている[2]. まず, この現象について具体的に説明しよう.

T_p を p 個のプロセッサによる計算時間とする. p 個のプロセッサを使用して計算を行ったとき, 加速率 S_p は, $S_p = T_1/T_p$ と定義される. 通常, 並列処理を適用した場合, 並列化に伴うオーバーヘッドにより, $S_p < p$ となる. しかし, 分枝限定法に対して並列処理を適用した場合, 図5のように, 最終的に生成される分枝木の大きさが, 逐次処理と比較して小さくなることがある. 計算開始後のある同じ経過時点において逐次処理と並列処理を比べたとき, 逐次処理では見つからなかった実行可能解が, 並列処理では見つかることがある. すると, このみつかった実行可能解により, いくつかの子問題が限定され, 分枝木が小さくなる. この場合, $S_p > p$ という極めて有効な並列処理の効果が期待できる. このような現象が異常加速である.

一方, 多くのプロセッサを利用して実行したにも関わらず, 逆に遅くなるという現象がある. つまり, $p_1 < p_2$ の場合に, $T_{p_1} < T_{p_2}$ が生じる. このような現象は, 異常損失と呼ばれる. 具体的には, 図6に示すように, 並列実行時に, 逐次処理では生成されなかった子問題が生成され, 余分な子問題の処理を行うことで生じる. 図6では, $t = 3$ の時点で, プロセッサへ分枝木の右側に位置する子問題が割当られている. 逐次処理と比較して子問題の処理順が変わっているように思われるかもしれないが, $t = 2$ で行った評価計算後, 同時に4つの子問題が生成される場合, 生成された4つの子問題中での処理順は非決定的である. 現実的にも, わずかに右側に位置する子問題が先に生成された場合, このような現象が生じる.

PUBB により並列分枝限定法を実現して行った数値実験結果においても, これらの現象は観測されている. 実験結果から, 探索規則を比較的制御できる中央制御型を用いて, ハイブリッド探索を行った場合, 少ない台数 (5台程度まで) による並列処理では異常加速を生じやすい. このような現象が生じるのは, 分枝限定法を実行した場合に, 解が更新されることが前提である. 初期暫定解の精度が高い場合には, 生じ難いのは当然であり, 特に初期暫定解が最適解であった場合には上記の観点からの異常加速は起こり得ない. また, 経験上, 利用する台数が 100 台程度になると, 特殊なデータを用意することがなければ, 極端な異常加速は生じ難い. 一方, 異常損失についても, 全体としての探索順序が正確に制御できる場合には, 特殊なデータを用いない限り, 生じることは少ない. 比較的, 異常損失を生じやすいのは深さ優先探索を行った場合であった. ただし, 用意されていたプロセッサの台数が, 本来解くべき問題に対して多すぎた場合や, 子問題プールを複数持つような実装に対して, 負荷分散を誤ると生じやすくなる.

図7は, PUBB を利用して並列分枝限定法を実行した場合に, 計算時間が短縮する様子を示す典型的な例である. ワークステーションとして, IBM RS/6000 Model 25T (CPU : PowerPC601 (66MHz), メモリ : 64M) を用い, 各ワークステーションが 10M bps のイーサネットにより接続されてた環境を用いた. 利用したワークステーションの台数は, 最高101台(全体の管理用にワークステーションを1台用いるので, 評価計算を行う部分である **Solver** は最大100) である. 古典的な Held-Karp の解法を用いて, TSPLIB のデータから都市数が 70(st70) の巡回セールスマン問題を解いた. 探索規則として, 下界値優先探索 (Best), ハイブリッド探索 (Hybrid) を適用し, PUBB が提供する3

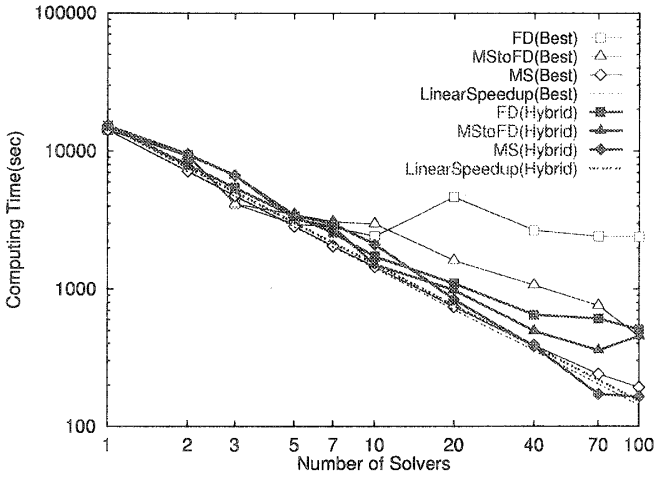

図 7: 計算時間の短縮効果を示す例

種類の制御方式, 中央制御型 (MS), 分散制御型 (FD), 混合制御型 (MStoFD) により実行した. 図7中の各プロットは, 並列実行にともなう非決定的な振る舞いを排除するため, 5回の実行の平均値を用いている.

まず, ここに示した結果は, 中央制御型で動作することから, 問題の規模という観点からは比較的小規模であることに注意されたい. この場合, 中央制御型 (MS) がもっとも効果的な並列分枝限定法を実現している. これは, 解法が評価計算に時間を要したので, 子問題の転送コストよりも探索順の制御が有効に機能しているためである. ハイブリッド探索は下界値優先探索よりも, さらに計算時間の短縮効果が大きい場合がある. 特に, Solver 数が 70 では超線形で計算時間を短縮している. 分散制御型 (FD) においては Solver 数が増加しても計算時間が短縮しない場合が生じる. これは, PUBB が, レスポンスタイムで定義される距離の近い部分に位置する計算機とのローカルなやり取りにより負荷分散を行うため, 全体としての負荷分散には失敗していると考えられる. このような状態が, 混合制御型 (MStoFD) を適用することで大幅に改善されることがわかる.

最後に, 子問題の評価計算が短時間である解法を用い, 生成される子問題数が極端に多かった場合についても言及する (詳細は, [3] 参照). この場合, 複数の子問題プールを保持する, 分散制御型, あるいは, 混合制御型での動作が前提となる. 通常, 動作環境に存在する全ての計算機に, 子問題を3個程度割り当てられるだけの子問題が生成された後, 分散制御型に切り替えている. また, 大規模な問題を扱う場合には, 現実的には深さ優先探索でないと実行不可能となる. このようなパラメタ設定において, 最大クリーク問題に対する DIMACS のベンチマーク問題, および, 二次割当問題に対するベンチマーク問題 (QAPLIB) を, PC クラスタ環境で解いた結果について紹介する (表 1,2 参照). PC には, エプソンダイレクト社の Endeavor AT-700C(CPU:Pentium II 400MHz, メモリ: 256M) を利用した. この PC 21 台を 3Com 社製 Super Stack II Baseline 10/100 Switch 24 port で接続して PC クラスタを構成した.

表 1: DIMACS の問題に対する結果

Prob.	Time(sec)	子問題数	更新
p_hat1500-2	34810	623538050	無
p_hat1000-3	1256613	23901006812	無

表 2: QAPLIB の問題に対する結果

Prob.	Time(sec)	子問題数	更新
nug21	13287	593656913	無
nug22	147378	6712276783	無
nug24	1269218	44317904109	無

表 1,2 中の "更新" は, 分枝限定法実行における, 解の更新の有無を示す. つまり, 表 1,2 中のいずれの結果においても, 初期暫定解を求める近似解法により, 最適解が得られていたことになり, 全ての計算は最適性を保証するために費やされている. この場合, 異常加速や異常損失は生じない. おおよそではあるが, この環境での加速率 S_{20} は, 16 程度であった.

6. PUBB 開発キット

PUBB のアプリケーションを開発するための開発キット, および, マニュアルを以下の URL にて配布している.

`http://al.ei.tuat.ac.jp/~yshinano/pubb`

PUBB 開発キットでは, プラグインとなる関数群から1つの実行モジュールが生成されるため, 並列処理は全く意識せずに開発が行える. また, PUBB のフレームワークが提供する機能が利用できることにより, 分枝限定法の開発環境としても有効である. 例えば, 探索規則はパラメタを変更することで実行時に切り替えられる. 複数の探索法を適用し, 得られた解の目的関数値が一致することを確認することは, デバッ

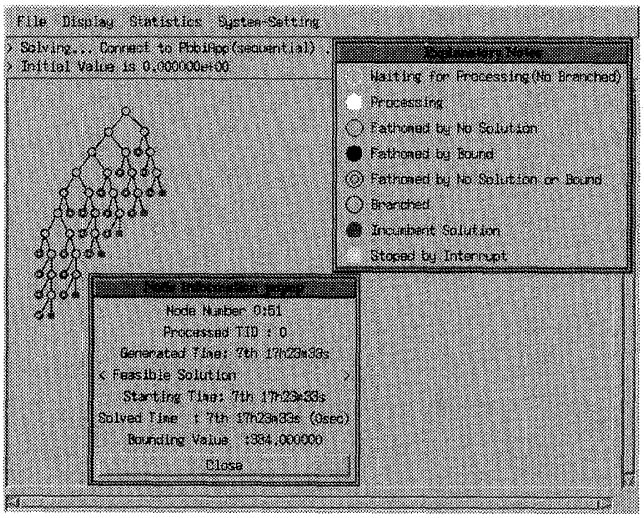

図 8: VABB による分枝木の表示例

グに有効である．また，桧垣正浩氏(現Sarion Systems Research) による VABB(Visualizer and Analyzer for Branch-and-Bound algorithms) を動作させることで，分枝限定法実行時に生成される分枝木を可視化できる (図 8 参照)．アプリケーション開発時には，可視化のためのコードは 1 行も書く必要はない．こちらは，デバッグに有効なのはもちろんのこと，分枝限定法の動作説明にも有効である．

7. おわりに

ベンチマーク問題に対する数値実験結果は，近似解法で得られた結果がたとえ最適解であっても，その最適性を保証することが如何に困難であるかを示している．近似解法の研究と比較して，厳密解法を実装した研究が少ないのは，実装の困難さにも関わらず，近似解法が扱う問題規模と比較して，厳密解法の扱う問題規模が極端に小さいことにも起因すると考えられる．しかし，厳密解が必要な局面や，少なくとも精度がある程度保証された解が必要な場合には，分枝限定法が唯一の解法となることさえある．そのような場面で，実装の手間を省くことに PUBB が貢献できれば幸いである．

本稿では，PUBB の内部構造についての解説は割愛し，利用する場合に必要となる手間と，実際に PUBB により並列分枝限定法を実現した場合に期待できる効果を示すため，過去の数値実験結果を紹介した．PUBB のタスク構成等の詳細については，上記 URL より PUBB に関連する論文がダウンロードできるので，興味のある方はそちらを参照されたい．

また，本稿では，C 言語による開発を紹介したが，C++言語による開発も可能である．PUBB は，C++言語によって記述されている．PUBB のように，利用者定義関数をプラグインとして開発するシステムにおいて，C 言語で開発されたシステムに対して，プラグインが C++言語で記述された場合，問題を生じることがある (開発当時は，リンクそのものが困難であった)．現在，STL や LEDA など強力なクラスライブラリが利用できるようになっているので，C++言語によるアプリケーション開発のメリットは大きい．C++言語によるインターフェースについては，参考文献[4]を参照されたい．

PUBB そのものも，STL 等を利用したコードへと書き換えたいと考えている．PUBB は，他の汎用の並列分枝限定法に対するツールと比較して，利用者定義関数の定義は，もっとも単純であり，その数も少ない．単純なユーザインターフェースにより，十分な汎用性が維持できるのであれば，インターフェースとしては好ましい．書き換えに際しても，C 言語によるユーザインターフェースは，現状を継承するように心がけるつもりである．しかし，ユーザインターフェースに不自然に感じる部分等あれば，PUBB と並列分枝限定法の汎用ツールの向上のため，幅広いご意見を頂ければ幸いである．

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.

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

[3] Y. Shinano, T. Fujie, Y. Ikebe and R. Hirabayashi, "Solving the Maximum Clique Problem using PUBB," *Proc. of the 12th International Parallel Processing Symposium* (1998) 326–332.

[4] Y.Shinano, M.Higaki and R.Hirabayashi, "An Interface Design for General Parallel Branch-and-Bound Algorithms," A.Ferreira, J.Rolim, Y.Saad and T.Yang(eds.), *Parallel Algorithms for Irregularly Structured Problems, LNCS*, Springer-Verlag, **1117** (1996) 277-284.