数理解析研究所講究録
1120巻 1999年 98-109

PUBBによるPCクラスタ環境における並列分枝限定法

Parallel Branch-and-Bound Algorithms on a PC Cluster using PUBB

品野 勇治[1]　藤江 哲也[2]
Yuji SHINANO　Tetuya FUJIE

[1] 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
yshinano@cc.tuat.ac.jp

[2] 神戸商科大学管理科学科
〒652-2103 神戸市西区学園西町8-2-1
fujie@kobeuc.ac.jp

摘要: 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り，さまざまな問題に適用できる汎用的なフレームワークを提供する．また，分枝限定法のフレームワークには，並列処理適用可能な構造があり，素直な実装として汎用ツールの実現が考えられる．そのようなツールの1つにPUBB(Parallelization Utility for Branch-and-Bound algorithms)がある．本稿では，PUBBの概要とアプリケーション開発の概略を示す．また，最近行ったPCクラスタ環境での数値実験結果を示し，過去の結果と比較する．

キーワード: 組合せ最適化, 分枝限定法, 並列処理, PCクラスタ

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 [7]．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する分枝操作と，不要な列挙を停止する限定操作により構成される．このフレームワークへ，子問題の生成方法，下界値計算 (最大化問題の場合は上界値計算：本稿では，分枝限定法に関する一般的な記述は，常に最小化問題を仮定する) および探索規則などを，解くべき問題固有の性質や構造を利用して設計し，組み込むことで分枝限定法が完成する．厳密解が得られる問題の規模は，上下界値の良さなどに依存するため，主たる研究課題は，これらの計算方法にある．

一方，フレームワークとしての分枝限定法では，生成される子問題間には依存関係が少ないという傾向がある．よって，各子問題に対する下界値計算，その結果に基づく子問題の生成，あるいは限定操作（これら，1つの子問題に関する一連の処理を，本稿では**子問題の評価**と呼ぶことにする) は，他の子問題とは独立に行うことができる．したがって，複数の子問題を同時に評価するという並列処理が素直に適用でき，並列分枝限定法も汎用的なフレームワークを提供する．そこで，フレームワークを直接的に並列化する汎用ツールがいくつか開発された [2, 3, 13, 19]．**PUBB**(Parallelization Utility for Branch-and-Bound algorithms) は，そのような汎用ツールの1つである．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，このようなツールを利用することで，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 [3, 16]．

本稿では，PUBBの概要とアプリケーション開発の概略を紹介する．また，PUBBによってPCクラスタ環境上に並列分枝限定法を実現し，数値実験を行なった結果を，これまでの結果と比較しながら紹介する．

2 PUBB

2.1 PUBBの概要

PUBBは，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り，以下の特徴を

図 1: 汎用ツール PUBB

もつ．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．

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，並列・分散環境と同じインターフェースを持つ，逐次分枝限定法のための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を開発することができる．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)．

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，という操作を繰り返す．並列分枝限定法におけるハイブリッド探索の効果については，参考文献 [14] を参照されたい．

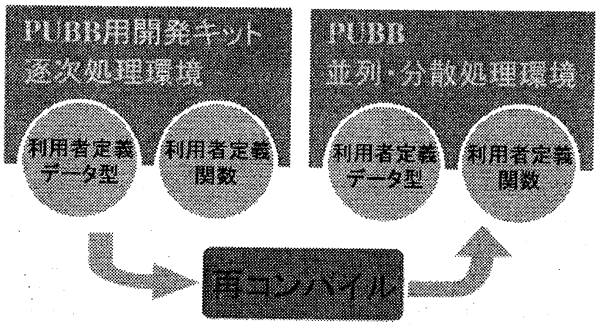

図 2: PUBB による並列化

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．

2.2 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 3)．

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** とデータ交換を行なって，負荷分散のための意思決定を行なう．

Solver： 主に，子問題の評価計算を行なう．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．

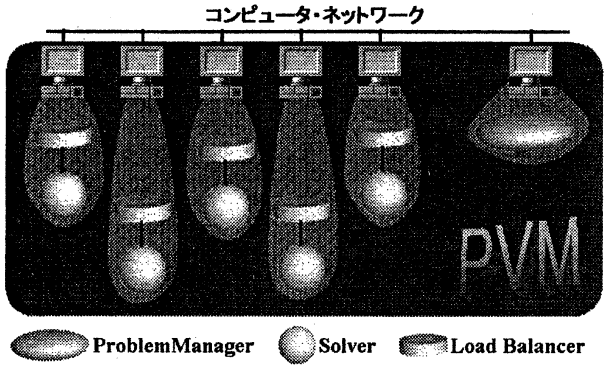

図 3: PUBB のタスク構成

通常，LB と Solver は同一ワークステーション・PC 上で動作させる．

2.3 各動作モードの概略

2.3.1 Master-Slave(MS) モード

初期化　まず PM が起動される．PM は問題データ (インスタンス) の読み込み，初期暫定解の計算，およびルート問題の生成を行なう．PM の初期化は，ルート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．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 (計算を行なっていない状態) 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行なう．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．

2.3.2 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ない，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なうために，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．ただし，Notification Interval Time パラメタが設定された場合，これらの情報の送信は，前回の送信から指定された時間 (Notification Interval Time) が経過した後にのみ行われる．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なう．負荷分散の詳細については本稿では割愛する．詳細は，参考文献[15, 17]を参照されたい．

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なっていない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．

2.3.3 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って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，[15]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が持つ子問題群は，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．

2.4 実現される並列分枝限定法の挙動

PUBBは，当初Ethernetで接続されたワークステーション群を利用し，子問題の評価計算に時間を要する解法の並列化を意図して開発された[17]．ここでは，ワークステーション群の環境で，評価計算に1秒程度を要した巡回セールスマン問題に対する数値実験結果[15]を示し，PUBBが実現する並列分枝限定法の挙動をみる．

2.4.1 ワークステーション群の環境

ワークステーションは，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は10M bpsのイーサネットにより接続されている．利用した台数は，最高101台(Solver数は100)である．

2.4.2 数値実験結果

図4,5,6は，Held-Karpの解法[5, 6]を用いて，TSPLIBベンチマーク問題の中から都市数70の問題を解いた結果である．図4, 図5, 図6は，それぞれ各Solver数に対しする計算時間，評価した子問題数，利用率を示す．ここで，

$$\text{利用率} = \frac{\text{Solverで評価計算を行った計算時間の和}}{\text{Solverの実行時間の和}}$$

と定義している．各制御方式に対して，下界値優先探索とハイブリッド探索を指定して実行している．また，各グラフにおいて，並列処理適用時に生じる非決定的な要素によるばらつきを排除するため，同じ動作環境，同じパラメタ設定により5回試行した平均値をプロットしている．

図4において，MSモードが線形あるいは超線形の結果を示しているのは，PUBBのMSモードが子問題を集中管理することにより，列挙木全体に対する探索規則が正確に制御されているためである．図5が示すように，いずれの探索規則においても，MSモードではSolver数が増加しても評価計算を行った子問題数は増加しない傾向がある．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

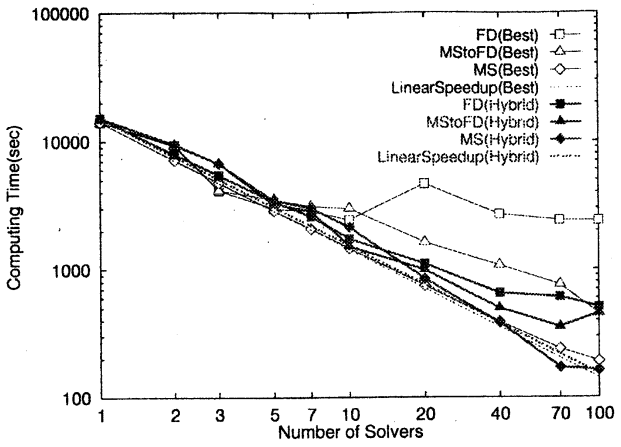

図 4: Solver 数と計算時間の関係

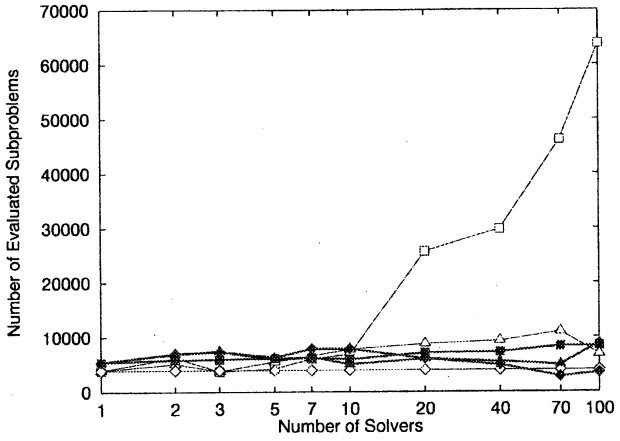

図 5: Solver 数と評価した子問題数の関係

数が 70 では超線形で計算時間を短縮している．図 6 が示すように，評価計算に時間を要する場合，MS モードで最も高い利用率が得られた．

図 4 が示すように，FD モードにおいては **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，図 5 に示されるように，**Solver** 数が増加すると，評価計算を行う子問題数が増加するためである．FD モードでは，各 **Solver** 内に保持されている子問題群に対して，起動時に指定された探索規則が適用されるため，列挙木全体としては無駄な子問題の評価計算を行うことがある．しかし，図 6 が示すように，**Solver** の利用率は特に悪いわけではない．さらに，MS-FD モードを適用することで大幅に改善がみられる (図 5)．

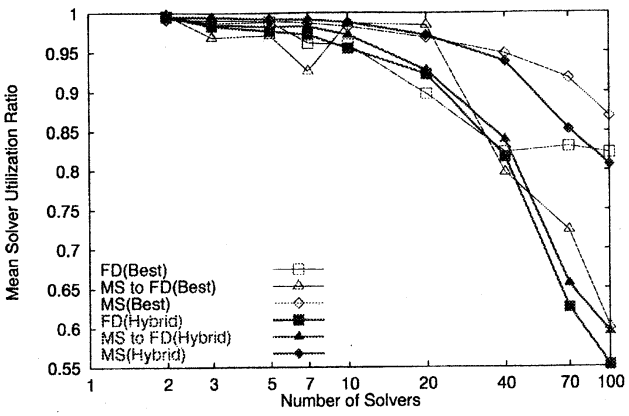

図 6: Solver 数と利用率の関係

2.5 評価計算の短い解法への対応

評価時間が短い解法を，PUBB を利用して並列化する場合，子問題の送受信に要する時間が評価計算時間に比べて相対的に長くなるため，MS モードでの実行は効果的でない．さらに，FD モードでさえ，**LB** と **Solver** 間の通信時間が，オーバーヘッドとして無視できなくなる．そこで，**LB** と **Solver** 間の通信量を減らすため用意したのが，前述の Notification Interval Time パラメタである．このパラメタを長くとると，**LB** と **Solver** 間のオーバーヘッドは減るが，**LB** が認識する **Solver** の子問題群の情報が不正確となるため，誤った負荷分散を行う可能性が高くなる．よって，このパラメタは適当に調整する必要がある．また，**Solver** 間で行われる子問題の送受信を減らすために，指定された探索規則によらず，**Solver** 内の子問題群の中から，下界値の最も良い子問題を転送する **Best Bound First Transfer 方式**を実装した．これらの改良を加えることで，評価計算の短い解法でも，PUBB による効果的な並列分枝限定法が実現できた [16].

2.6 アプリケーション開発

本節では，PUBB による分枝限定法の実装について概略を述べる．開発言語は C 言語および C++ 言語が用意されている．ユーザがプラグインとして開発する必要があるのは，利用者定義データ型と利用者定義関数群である．利用者定義データ型

は次の3種類である:

- **インスタンスデータ (Instance Data)**
  これは，解くべき問題のデータである．対象とする問題が最大化であるか最小化であるかを示すデータは，PUBBにより予約されている．

- **子問題データ (Subproblem Data)**
  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．下界値等，標準的に必要となデータは，PUBBにより予約されている．

- **解データ (Solution Data)**
  これは，実行可能解のデータである．目的関数値は，PUBBにより予約されている．

C言語では，各データに対する構造体を定義する．構造体名はPbbi (Parallel Branch-and-Bound Interface) から始まる名前が予約されている．ただし，PUBBが必要とする構造体のメンバ以外は，利用者が自由に記述して良い．

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作成する．

(b) 初期解の計算
インスタンスを読み込んだ直後に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のみから成る解を作成する．

(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，解データの形で返す．

(d-2) 新しい子問題を作成する必要があれば，それらを子問題データの形で返す．

(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．

図7は分枝限定法の疑似コードであり，関数群の役割を示している．$\mathcal{L}$ は子問題プールであるが，その管理について，ユーザは全く考慮する必要のないことに注意する．

```
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) ルート問題を作成する;
 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 7: 分枝限定法の疑似コード

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，利用者定義データ型をpack/unpackする．

以下では，0-1ナップサック問題の実装例を示す．0-1ナップサック問題とは，次のようにして定義さ

れる問題である：

$$\text{(KP)} \left| \begin{array}{ll} \text{最大化} & z = \sum_{j=1}^{n} p_j x_j \\ \text{条件} & \sum_{j=1}^{n} w_j x_j \le c, \\ & x_j = 0 \text{ or } 1 \ (j = 1, \ldots, n). \end{array} \right.$$

ここで，各 j をアイテムと呼ぶ．また，p_j, w_j $(j = 1, \ldots, n)$ および c は正整数であり，$p_j/w_j \ge p_{j+1}/w_{j+1}$ $(j = 1, \ldots, n-1)$ と整列されていることを仮定する．

子問題を $\mathrm{KP}(S_1, S_0, F)$ と表現する．ただし，S_1, S_0 はそれぞれ 1,0 に固定されたアイテム集合であり，$F = \{1, \ldots, n\} \setminus (S_1 \cup S_0)$ である．よって 0-1 ナップサック問題自身は $\mathrm{KP}(\emptyset, \emptyset, \{1, \ldots, n\})$ となる．$\mathrm{KP}(S_1, S_0, F)$ は次のように定式化される．

$$\text{(KP}(S_1, S_0, F)\text{)} \left| \begin{array}{ll} \text{最大化} & z = \sum_{j \in F} p_j x_j + \sum_{j \in S_1} p_j \\ \text{条件} & \sum_{j \in F} w_j x_j \le c - \sum_{j \in S_1} w_j, \\ & x_j = 0 \text{ or } 1 \ (j \in F). \end{array} \right.$$

まず，利用者定義データ型は次のように設計すれば良い：

- **インスタンスデータ**
  $n, c, (p_j, w_j)$ $(j = 1, \ldots, n)$ を持つ．
- **子問題データ**
  S_1, F を持つ．
- **解データ**
  n, x_j $(j = 1, \ldots, n)$ を持つ．

また，関数の設計は次のように設計することができる．

(a) インスタンスの読み込み
$n, c, (p_j, w_j)$ $(j = 1, \ldots, n)$ をデータファイルから読み込み，インスタンスデータを作成する．

(b) 初期解の計算
貪欲解を初期暫定解としても良いし，自明な実行可能解 $(x_j = 0 \ (j = 1, \ldots, n))$ を初期暫定解としてもよい．

(c) ルート問題の作成
$S_1 = \emptyset, F = \{1, \ldots, n\}$ とする子問題データを作成する．

(d) 子問題の評価
簡単のため，ルート問題 (KP) に対する評価計算を考える．他の子問題の評価計算も同様である．

$s = \min\{j \mid \sum_{i=1}^{j} w_i > c\}$ と定義する（$\sum_{i=1}^{n} w_i > c$ ならば $s = n+1$ とする）．さらに，$\bar{c} = c - \sum_{j=1}^{s-1} w_j$ とする．このとき，$\mathrm{UB} = \sum_{j=1}^{s-1} p_j + \lfloor \bar{c} p_s / w_s \rfloor$ が (KP) に対する上界値であることが知られている（$\lfloor x \rfloor$ は x を超えない最大の整数）．評価計算は次のステップで実行する：

(d-0) $c < 0$ ならば問題は実行不可能であり，評価計算を終了する (分枝停止となる)．

(d-1) $s = n+1$ または $\bar{c} = 0$ ならば，$x_1 = \cdots = x_{s-1} = 1, x_s = \cdots = x_n = 0$ が (子) 問題の最適解である．よって，この新しい解が暫定解を更新するならば，新しい解を解データの形にして返す．そして評価計算を終了する (分枝停止となる)．

(d-2) 上界値 UB を計算する． UB が暫定値よりも大きければ，$k \in F$ を選んで分枝する．評価されている子問題が $\mathrm{KP}(S_1, S_0, F)$ の場合，$\mathrm{KP}(S_1 \cup \{k\}, S_0, F \setminus \{k\})$, $\mathrm{KP}(S_1, S_0 \cup \{k\}, F \setminus \{k\})$ が新しい子問題となるので，各々を子問題データの形にして返す．

紙面の都合上，他の関数群の設計については省略する．

3 PC クラスタ環境における並列分枝限定法

本節では，まず最大クリーク問題をワークステーション群で解いた結果と PC クラスタで解いた結果を比較する．次に，二次割当問題を PC クラスタ環境で解いた結果を示す．

3.1 PC クラスタ環境

PC クラスタを構成する PC は，エプソンダイレクト社の Endeavor AT-700C(CPU:Pentium II 400MHz，メモリ:256M) である．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．また，OS には Linux (Kernel 2.2.5-15,package RedHat6.0) を使用した．

3.2 最大クリーク問題

無向グラフ $G=(V,E)$ のクリーク C $(\subseteq V)$ とは，C の任意の 2 頂点が枝で結ばれているものをいう．最大クリーク問題とは，要素数最大のクリークを求める問題である．[16] では，ワークステーション 51 台を用いて，その時点で最適性が保証されていなかった DIMACS ベンチマーク問題を 5 問解いた．3.2.3 節では，この結果と PC クラスタ環境における計算結果を比較する．

3.2.1 解法の概略

解法の概略は次の通りである．詳細は [16] に与えられている．

初期解

Ikebe and Tamura [8] による STABJOIN を使用した．STABJOIN は，ほとんどのインスタンスに対して最適解もしくはそれに近い解を出力した．

上界値の計算

G の安定集合 S $(\subseteq V)$ とは，S の任意の 2 頂点が枝で結ばれていないものをいう．$\mathcal{S}=\{S_1,\ldots,S_K\}$ を安定集合による頂点集合 V の分割とする．$\mathcal{S}$ は G の点彩色と呼ばれる．そして，よく知られているように，クリークの最大要素数は高々 K であり，点彩色の彩色数が上界値を与える．一般のグラフに対して彩色数を求めることは $\mathcal{NP}$-困難であるため，ヒューリスティック解法が多く提案されている．[16] では，逐次分枝限定法において，総計算時間の意味で最良の結果を示した，greedy な解法を採用した．

分枝方法

[12] に基づいている．1 回の分枝操作で生成される子問題数は暫定解の大きさに依存するため，STABJOIN は総子問題数を減らすためにも有効であった．

子問題の探索

一般に，1 回の分枝操作によって多くの子問題を生成するため，深さ優先探索を適用している．

3.2.2 ワークステーション群と PC クラスタの比較

ワークステーション群は，2.4.1 節の環境を用いている．この環境で最大クリーク問題を解いた結果は，[16, 17] に示されている．数値実験には，ランダムに生成した 200 頂点，枝密度 90%の問題 10 問を用いた．PC クラスタ環境での数値実験でも，[16, 17] と同じプログラム，同じデータを用いた．ただし，初期暫定解を得るときに利用している乱数が動作環境に依存したため，初期暫定値が異なる場合がある．両環境とも 21 台 (**Solver** 数は 20) のワークステーション・PC を使用している．

まず，今回利用したワークステーションおよび PC 単体の計算時間を比較するため，10 問の中で暫定解の更新がなかった (初期暫定解が最適であった)6 問について計算時間の和をとった．すると，

$$\frac{\text{ワークステーション単体の計算時間の和}}{\text{PC 単体の計算時間の和}}=5.0$$

という結果を得た．すなわち，計算機単体の計算速度は PC の方が 5 倍程度速いと考えられる．また，この 6 問について，**Solver** 内の子問題群の中から深さの最も大きい子問題を転送する **Depth First Transfer 方式**を実行し，負荷分散を頻繁に生じさせた結果を図 8,9 に示す．この場合も，並列動作に伴う非決定的な挙動の影響を排除するため，5 回の試行による平均値でプロットしている．これらの図より，ワークステーション群の環境では，Notification Interval Time を 0.5 秒に設定すると必ず計算時間が短縮するのに対して，PC クラスタ環境では，必ずしも計算時間は減少しない．また，負荷分散を誤り，**Solver** の利用率が低下することも多くなる．

次に，PC クラスタ環境において，Depth First Transfer 方式で転送し，負荷分散を頻繁に生じさせた場合と，Best Bound First Transfer 方式で転送した場合を比較した結果を図 10, 11 に示す．この場合も，並列動作に伴う非決定的な挙動の影響

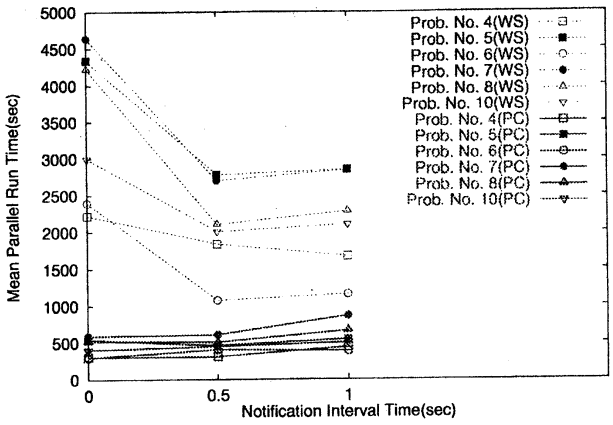

図 8: 動作環境の違いによる比較 (計算時間)

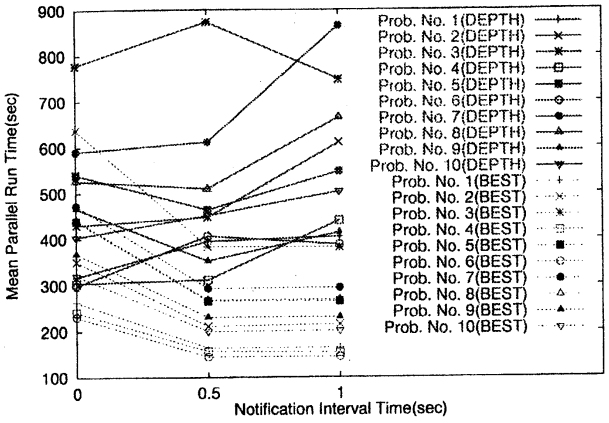

図 10: 転送方式の違いによる比較 (計算時間)

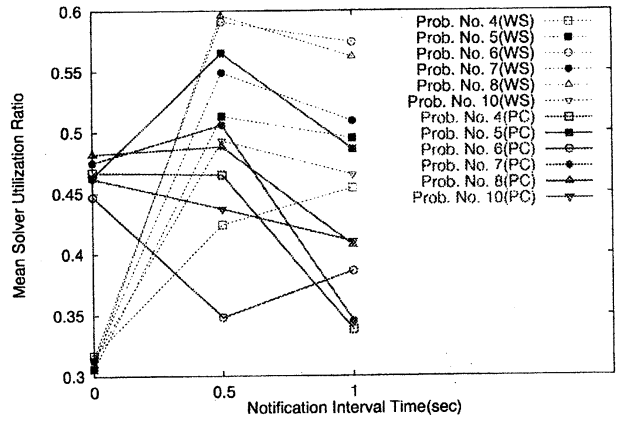

図 9: 動作環境の違いによる比較 (利用率)

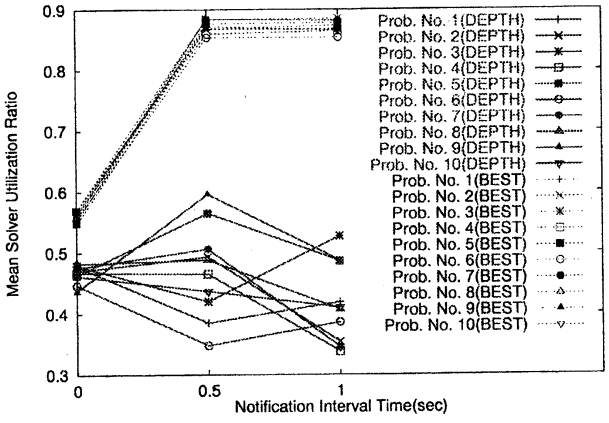

図 11: 転送方式の違いによる比較 (利用率)

を排除するため，5 回の試行による平均値でプロットしている．

図 10, 11 から，負荷分散が頻繁に生じる Depth First Transfer 方式では，パラメタ値 (Notification Interval Time) を大きくすると，誤った負荷分散によるオーバーヘッドが大きくなることが多いことがわかる．よって，負荷分散が多い場合は，パラメタ値を 0 に設定することが適当である．しかし，Best Bound First Transfer 方式を採用する場合には，パラメタ値を 0.5 秒に設定することで，0.3 程度利用率が向上している．ただし，0.5 秒より長くしても計算時間に顕著な短縮効果は現れていない．したがって，PC クラスタ環境では Notification Interval Time を 0.5 秒に設定した．

最後に，ワークステーション群環境では 1 秒，PC クラスタ環境では 0.5 秒に Notification Interval Time を設定し，どちらも Best Bound First Transfer 方式で転送した場合の数値実験結果を表 1 に示す．表 1 において，“加速率” は次のように定義されている：

$$\text{加速率} = \frac{\text{逐次処理での計算時間}}{\text{20 個の Solver による並列計算時間}}$$

また，“更新” は，暫定解の更新の有無を示している．表 1 が示すように，Notification Interval Time を適切に設定することによって，PC クラスタ環境においても十分な加速率が得られ，並列の場合にもワークステーション群環境よりも 5 倍程度速く解くことができた．

表 1: ワークステーション群とPCクラスタの比較

Prob. No.	ワークステーション群			PCクラスタ		
	Time(sec)	加速率	更新	Time(sec)	加速率	更新
1	1006.08	24.1	有	164.18	16.6	無
2	1082.22	20.2	有	211.12	17.4	有
3	1950.58	16.1	無	383.22	16.8	有
4	821.63	15.6	無	156.46	16.2	無
5	1468.25	15.9	無	265.33	16.9	無
6	769.58	12.9	無	144.41	16.3	無
7	1570.68	16.1	無	293.01	17.0	無
8	1462.07	16.1	無	267.71	17.0	無
9	1214.67	15.7	有	230.47	15.8	有
10	1073.00	16.0	無	199.96	16.7	無

3.2.3 数値実験結果

[17]では，DIMACSチャレンジ問題集から，1997年の時点において最適解が得られていなかった問題を5問解いた．その中で最も計算時間を要したp_hat1500-2(頂点数:1500，枝密度:50.6%)をPCクラスタ環境で解いた．その結果を表2に示す．

ワークステーション群の環境では，51台(**Solver**数は50)のワークステーションを使用した(PCクラスタ環境では21台のPCを使用)．この実験の試行回数は1回である．表中の“子問題数”は，評価計算を行った子問題の数である．

表 2: p_hat1500-2に対する計算結果

ワークステーション群			PCクラスタ		
Time(sec)	子問題数	更新	Time(sec)	子問題数	更新
61694	628386468	有	34810	623538050	無

また，新たにp_hat1000-3(頂点数:1000，枝密度:74.3%)もPCクラスタ環境で解くことができた．その結果を表3に示す．

表 3: p_hat1000-3に対する計算結果

Time(sec)	子問題数	更新
1256613	23901006812	無

3.3 二次割当問題

二次割当問題は，2つの$n \times n$行列$\boldsymbol{F} = (F_{ij})$，$\boldsymbol{D} = (D_{ij})$が与えられたとき，次のように定式化される問題である：

$$\min_{\pi \in \mathcal{S}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} F_{\pi(i)\pi(j)} D_{ij}.$$

ここで，$\mathcal{S}_n$は$\{1,\ldots,n\}$上の置換集合である．3.3.2節では，QAPLIBベンチマーク問題をPCクラスタ環境で解いた結果を示す．

3.3.1 解法の概略

初期解

Taillard[18]によって提案されたrobust taboo searchを用いた．また，QAPLIBのホームページで公開されているプログラムコードを参考にした．

下界値の計算

Gilmore-Lawler bound [4, 10]を用いた．すなわち，$i,j = 1,\ldots,n$に対して，L_{ij}を

$$L_{ij} \equiv \min_{\substack{\pi \in \mathcal{S}_n \\ \pi(j)=i}} \sum_{k=1}^{n} F_{i\pi(k)} D_{jk}$$

と計算して求め，行列$\boldsymbol{L} = (L_{ij})$に関する(線形)割当問題の最適値を求める．良く知られているように，L_{ij}の計算において割当問題を解く必要はなく，Gilmore-Lawler boundの計算は高速に行なうことができる．

分枝方法

固定されていない添字iを選び，「$\pi(i) = 1, \pi(i) = 2,\ldots,\pi(i) = n$」または「$\pi(1) = i, \pi(2) = i,\ldots,\pi(n) = i$」と$n$個の子問題を生成する．このとき，行列$\boldsymbol{L}$に関する割当問題の最適解の情報や，行列$\boldsymbol{F}, \boldsymbol{D}$の情報を利用して無駄な分枝を省くことができる．

子問題の探索

深さ優先探索を適用している．

3.3.2 数値実験結果

QAPLIBベンチマーク問題，nug22 ($n = 22$)およびnug24 ($n = 24$)を解くことができた．結果を表4に示す．この実験の試行回数も1回である．1997年の論文[3]では，nug22を，NEC Cenju-3のプロセッサを48 ~ 96台使用して(restart機能があるため，利用プロセッサ数を変更することができる)，766800(sec)で解いた．現在，nug-typeで解かれている最大規模の問題はnug25 ($n = 25$)である[11]．

表 4: 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

4 おわりに

本稿では，PUBB を紹介し，PC クラスタ環境における数値実験結果を示した．その結果，PUBB を利用することで，比較的低価格の PC クラスタが並列分枝限定法の動作環境として十分機能することがわかった．パソコンの性能は，今でも急速に向上しているので，PC クラスタは並列分枝限定法の実行環境として期待できる．

本稿で紹介した PUBB は，逐次処理の開発環境で，すなわちワークステーション・PC が 1 台あれば，並列分枝限定法を実現するプログラムの開発が可能である．つまり，並列分枝限定法に対する一つの開発手法も提供している．PUBB による開発が適当であるかどうかは PUBB のプラグインとなるユーザ定義データ型，ユーザ定義関数群のインターフェースが適当であるかによる．逐次処理の開発キットは以下の URL よりダウンロードでき，分枝限定法のプログラムの開発に利用することができる：

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

PUBB と並列分枝限定法に対する汎用ツールの向上のため，幅広いご意見が頂ければ幸いである．

[謝辞]

東京理科大学の仁木直人教授には，PC クラスタ環境の入手にご尽力頂きました．ここに謝意を表します．

参考文献

[1] E. Balas and J. Xue, "Weighted and Unweighted Maximum Clique Algorithms with Upper Bounds from Fractional Coloring," *Algorithmica* **15** (1996) 397–412.

[2] M. Benaïchoche, V. D. Cung, S. Dowaji, B. L. Cun, T. Mautor and C. Roucairol, "Building a Parallel Branch and Bound Library," *Lecture Notes in Computer Science* **1054** (1996) 201–231.

[3] A. Brüngger, A. Marzetta, J. Clausen and M. Perregaard, "Joining Forces in Solving Large-Scale Quadratic Assignment Problems in Parallel," *Proc. of the 11th International Parallel Processing Symposium* (1997) 418–427.

[4] P. C. Gilmore, "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *Computational Optimization and Applications* **3** (1994) 243–258.

[5] M. Held and R. M. Karp, "The Traveling-Salesman Problem and Minimum Spanning Trees," *Operations Research* **18** (1970) 1138–1162.

[6] M. Held and R. M. Karp, "The Traveling-Salesman Problem and Minimum Spanning Trees: PartII," *Mathematical Programming* **1** (1971) 6–25.

[7] 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.

[8] Y. Ikebe and A. Tamura, "Ideal Polytopes and Face Structures of Some Combinatorial Optimization Problems," *Mathematical Programming* **71** (1995) 1–15.

[9] T. H. Lai and S. Sahni, "Anomalies in Parallel Branch-and-Bound Algorithms," *Communications of the ACM* **27** (1984) 594–602.

[10] E. L. Lawler, "The Quadratic Assignment Problem," *Management Science* **9** (1963) 586–599.

[11] A.Marzetta and A.Brüngger, "A Dynamic-Programming Bound for the Quadratic Assignment Problem," *Lecture Notes in Computer Science* **1627** (1999) 339–348.

[12] C. Manino and A. Sassano, "An Exact Algorithm for the Maximum Stable Set Problems," *Computational Optimization and Applications* **3** (1994) 243–258.

[13] Y. Shinano, M. Higaki and R. Hirabayashi, "A Generalized Utility for Parallel Branch and Bound Algorithms," *Proc. of the 7th IEEE Symposium on Parallel and Distributed Processing* (1995) 392–401.

[14] 品野, 桧垣, 平林, "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** (1996) 1379–1387.

[15] Y. Shinano, K. Harada and R. Hirabayashi, "Control Schemes in a Generalized Utility for Parallel Branch-and-Bound Algorithms," *Proc. of the 11th International Parallel Processing Symposium* (1997) 621–627.

[16] 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.

[17] 品野, 藤江, "汎用分枝限定法ツール PUBB による組合せ最適化問題の厳密解法," 統計数理 **46** (1998) 411–431.

[18] E. Taillard, "Robust Taboo Search for the Quadratic Assignment Problem," *Parallel Computing* **17** (1991) 443–455.

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