アルゴリズム 28−12
（1992. 7. 18）

並列分枝限定法に対するビジュアリゼーションシステム

大西克実　榎原博之　中野秀男

大阪大学工学部通信工学科

分枝限定法は、組合せ最適化問題の解法として適用される計算手法の一つである。分枝限定法の問題点として、問題の規模が大きくなると実行時間が指数関数的に増加する事が言われている。これを解決するために、分枝限定法の並列化と言う手法が考えられている。しかし、並列分枝限定法のような複雑な振る舞いをするアルゴリズムの有効性を的確に把握することは非常に難しい。そこで本報告では、並列分枝限定法のアルゴリズムの過程を逐次画面上に表示するシステム" Sapal-BB" を実現し、これを使用することで、アルゴリズムの有効性の判断を容易にすることを試みる。さらに、実際に"Sapal-BB" を実現する時の問題点について考察する。

A Visualization System for Parallel Branch-and-Bound Method

Katsumi Ohnishi　Hiroyuki Ebara　Hideo Nakano

Department of Communication Engineering, Faculty of Engineering, Osaka University

A bracnh-and-bound method is one of the algorithms that apply to combinatorial optimization problems. However, it has a drawback that it's running time grows exponentially in the problem size. Therefore, several parallel branch-and-bound methods are proposed. But, it is very difficult to understand precisely the behavior of parallel branch-and-bound algorithms. In this paper, we develop a system "Sapal-BB", which shows the running process of the parallel branch-and-bound algorithm in a bitmap display. To use Sapal-BB, it is made easy to recognize the efficiency of algorithms. Furthermore, we discuss problems to implement Sapal-BB.

1 はじめに

分枝限定法は、組合せ最適化問題の中でも特に難しい問題の解法に適用される計算手法の一つで、実用的にも有効な手法の一つである [1][2]。しかし、この分枝限定法を用いても解を得るために必要な計算時間は、問題の規模が大きくなるにつれて指数関数的に大きくなる傾向がある。このために、分枝限定法の並列化と言う手法が考え出され、いろいろな並列化手法が提案されている [3][4]。

分枝限定法では問題を次々と分解し、その過程で得られる与えられた問題についての知識を、次の段階での操作に利用する。このために分枝限定法のアルゴリズムが次の段階で行う操作を、正確に予測することは不可能である。分枝限定法のアルゴリズムの解析は今までにも行われているが、それらの解析においては単純化のために多くの仮定が行われている。さらに平均的な効率を予測するものではなく、最悪の場合を想定している場合が多い。

さらに、並列型のアルゴリズムと逐次型のアルゴリズムが同じ解を出しても、これら二つのアルゴリズムの実行過程はまったく異なることがあり得る。また、並列型にした場合、逐次型では考慮する必要がなかった条件も考える必要がある。これらのことから並列化したアルゴリズムの動作を予測することはますます難しくなる。

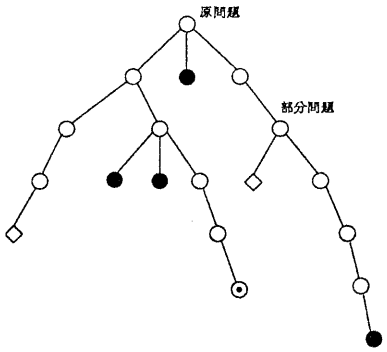
ところで、最近の新しい研究分野としてアルゴリズムアニメーション (*Algorithm Animation*) と呼ばれる分野がある [5][6]。これはアルゴリズムによって問題が解かれる過程を逐次ディスプレイ画面上に表示し、アルゴリズムが実際の問題を解いていく様子を動画 (*Animation*) として見せる技法であり、紙の上に書かれただけでは理解しにくいアルゴリズムの全体的な流れを容易に把握できるようにする。この考え方を振舞いの複雑な並列分枝限定法に応用すればアルゴリズムの開発に有効な手段を提供できると考えられる。

そこで本報告では、並列分枝限定法の振る舞いをワークステーション上の画面に逐次表示するシステム Sapal-BB (*Simulatar and Animator of Parallel Algorithm for Branch-and-Bound method*) を実際に作成し、これを実現するうえでの問題点について検討する。

2 並列分枝限定法

2.1 分枝限定法

分枝限定法 (*branch-and-bound method*) の基本的な考え方は、図1に示したように、直接解きがたい問題を部分問題 (*partial problems*) に分解し、それらすべてを解くことにより、与えられた問題を解くという考え方である。



⊙ : 可能解が求まった部分問題
• : 最適解の求まらない部分問題
◇ : 活性部分問題

図 1: 分枝限定法の探索木

部分問題に分解する操作のことを分枝操作 (*branching operation*) と呼ぶ。さらに不必要な部分問題の生成を抑えるために、可能解が求まる場合 (図の⊙)、あるいは最適解が求まらないことが結論できる場合 (図の•) には、新たな分枝操作を適用しないようにする。これを限定操作 (*bounding operation*) と呼ぶ。分枝操作も限定操作もなされていない問題は活性部分問題 (図の◇) と呼ばれ、次に分解すべき活性部分問題を探索木の中から選び出すことを探索 (*search*) と呼ぶ。このようにして探索、分枝操作、限定操作を繰り返していくと幾つかの可能解 (*feasible solution*) が得られる。実行中のある時点で最適と考えられる解を暫定解と言い、全ての部分問題を処理したときの暫定解が与えられた問題の最適解になる。

2.2 並列計算機のモデル

本報告で考える並列計算機のモデルは、図2に示すような星状結合ネットワーク型マルチプロセッサシステムである。ＲＰは親プロセッサを示し、ＮＰ$_i$(i = 1,2,...,m) は子プロセッサを示す。このシステムにおいて、各プロセッサは共有メモリを持たず、専用記憶装置だけを持っている。また、子プロセッサは全て同等の能力を持っており、プロセッサ間のデータの伝送はネットワークを通じて行われる。さらに、全てのプロセッサは複数の通信を同時に処理できないものとする。

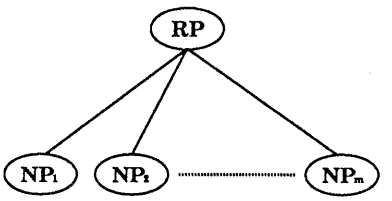


図 2: 星状結合ネットワーク型マルチプロセッサシステム

2.3 並列分枝限定法のアルゴリズム

非同期型並列分枝限定アルゴリズム (*asynchronous parallel branch-and-bound algorithm*) は、主に以下の二通りに大別される。

1. 垂直探索 (*vertical search*)
2. 水平探索 (*horizontal search*)

垂直探索の場合、親プロセッサは、データと解の管理、および子プロセッサへの問題の割当てを行う。子プロセッサは、親プロセッサとは独立に、探索と分枝操作、限定操作を繰り返し行う。このため、親プロセッサに関しては記憶領域及び通信時間が節約でき、負担は軽くなる。しかし、子プロセッサの探索終了時間に差が生じるため、空き状態のプロセッサができてしまい、子プロセッサを効率良く利用できない場合がある。

水平探索の場合、一台の親プロセッサが全ての活性部分問題及び、暫定解を保持し、子プロセッサの管理を行う。また、子プロセッサは親から与えられた一つの活性部分問題の処理のみを行う。この方法の場合、暫定解の更新が速やかにできるという点で有効であるが、プロセッサ数が増えると通信回数が増え、親プロセッサの仕事の負担が大きくなる。

Sapal-BB では、問題例として0－1ナップサック問題を用い、垂直探索を採用したシュミレータを構成し、その動作を視覚化することを試みる。

垂直探索においては、親プロセッサは図3に示したように、まず一台で、活性部分問題の個数が子プロセッサ数と等しくなるまで探索して分枝限定操作を行い、各部分問題を子プロセッサに割当てる。以後は子プロセッサからのメッセージに応じて適当な処理を行う。

子プロセッサは、図4に示したように、割当てられた問題を分解した部分問題の中に、活性部分問題が存在する限り独立して探索と分枝操作、限定操作を行い、新たに暫定解が出れば、親プロセッサに新しい暫定解を送り、活性部分問題がなくなれば親プロセッサに新たな活性部分問題を要求する。

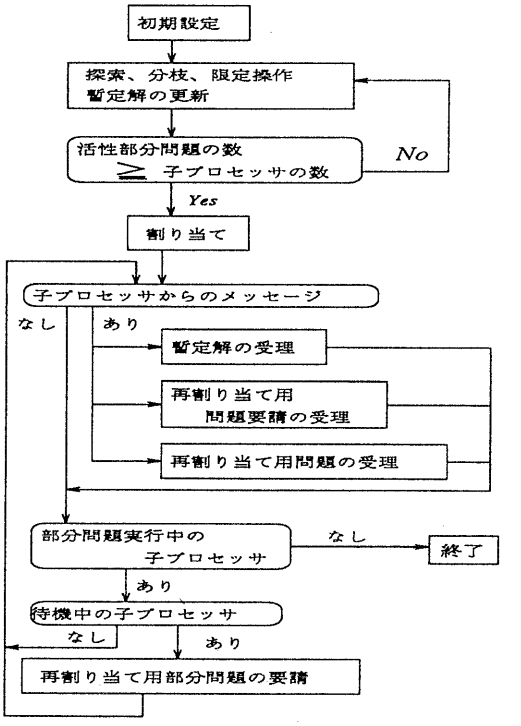


図 3: 親プロセッサのアルゴリズム

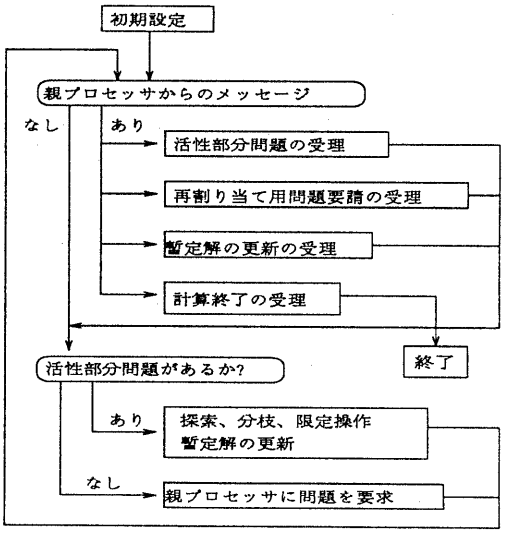


図 4: 子プロセッサのアルゴリズム

3 Sapal-BB の設計

Sapal-BB を設計する時、特に考慮した項目を以下に述べる。

1. 描画速度について
   視覚化の目的は、アルゴリズムの振舞いの全体的な流れをユーザに把握させることである。そのためには、描画をユーザが希望する(十分に把握できる)描画速度で見ることができ、かつ自由に停止、続行できるようにしなければならない。

2. 大規模な図形の描画について
   Sapal-BB では分枝図のような大規模な図形を描画しなければならない。しかし、描画画面の大きさと精細度には限度があり、全てのデータを一画面に描画しようとすると各データの表示が小さくなり詳しい情報を読み取ることができない。このような問題点を解決するために、大規模なデータに対する描画手法の工夫が必要である。

3. 複数のプロセスを用いる場合の制御について
   Sapal-BB の場合、複数のプロセスが実行され、全体として一つの仕事をする。この時、複数のプロセスからの部分的な情報を全体として時間的な矛盾を起こさせずに描画する必要がある。また、様々なアルゴリズムに対する視覚化を実現するためにはプロセスの生成、終了、通信等において共通して利用できる手続きを用意する必要がある。

4. 視覚化に必要な情報について
   Sapal-BB では、アニメータとシミュレータのプロセスの間では、描画に必要な情報が常に流れるようにしている。この量が必要以上に多いと、二つのプロセスの間で流れるデータの処理のための時間という本来は必要のない時間が生じて、シミュレーションが視覚化を行う場合と行わない場合とで大きな違いが生じるというようなことも充分考えられる。このようなことから、これら二つのプロセスの間で必要最小限のデータだけを送るための工夫が必要である。

4 Sapal-BB システムの構成

4.1 全体構成

Sapal-BB は以下の図5に示したようにアニメーションを行うプロセス(アニメータ)と、親プロセス及び子プロセスからなるシミュレータの二つの部分から構成されている。これらのプログラムは全て UNIX の上でC言語を用いて構成している。GUIを実現するためのツールとしては、X Window System と Motif Widget Set を用いている。通信を実現するための手段としては UNIX 上のソケットを使用している。

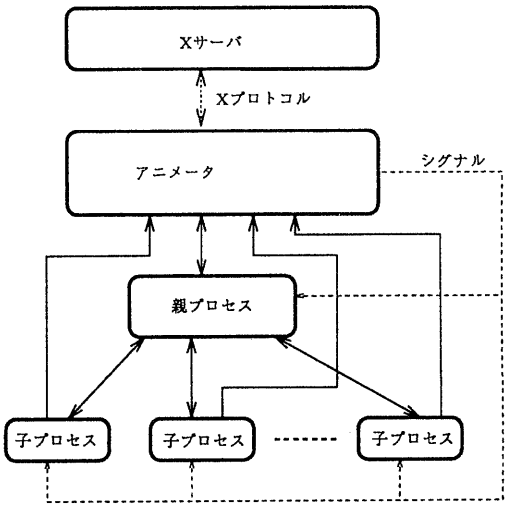


図 5: プログラムの全体構成

X Window System では、ワークステーション上の入出力装置はXサーバと呼ばれるプログラムが管理している。Xサーバは各ハードウェアメーカーから提供される。ユーザがプログラム中から画面に表示を行いたい場合、Xサーバに対してXプロトコルを用いて、描画要求を行う。

Sapal-BB では図5に示したように、Xサーバと分枝限定法を実行するシミュレータの部分とは完全に分離されていて、これら二つのプロセスの間にアニメータを置いている。アニメータとXサーバはXプロトコルを用いてイベントの通知や描画要求などの通信を行う。シミュレータからアニメータへの通信はソケットで構成した通信路を用いている。アニメータとシミュレータの間には、このほかに使用者からシミュレータを停止させたいという要求があったとき、それを実現するためにシグナルという UNIX に用意されているソフトウェア割り込みの仕組みも利用している。

次に、実際にプログラムが起動されてシミュレーションが始まるまでを述べる。UNIX のコマンドラインから次の様にシミュレータ用のプロセス、アニメータを各々起動する。ここで、host1、host2 はそれぞれアニメータ、親プロセスが実行されるホスト

を表す。"pbbviewd" はアニメータ、"ropro" は親プロセス、"copro" は子プロセスの実行を指示している。"MakeProb" は問題を作成するプログラムで、変数の数、問題のデータの最大値、最小値を与えている。

```
% pbbviewd &
% MakeProb 20 100 80 | ropro host1 host2 &
% copro host1 host2 &
% copro host1 host2 &
% copro host1 host2 &
```

ナップサック問題のデータは、親プロセスを起動する際にパイプを用いて与える。起動されたアニメータは、シミュレータからの接続要求を受け付けるソケットを用意し、接続要求を待つ。親プロセスは与えられた問題のデータを読み込み、アニメータに接続要求を行う。さらに、子プロセスからの接続要求を受け付けるためのソケットを用意し、接続要求を待つ。起動された子プロセスはアニメータ、親プロセスに接続要求を行い、接続が確立されれば親プロセスから問題が送られてくるのを待つ。

シミュレータの各プロセスの起動時には実行を停止するためのシグナルと停止状態を解除して実行を再開するためのシグナルを受け付けられるようにソフトウェア割り込みを登録する。

親プロセスは通信路の設定が終了すると、すべての子プロセスに対して与えられた問題を送り、その後は 2.3 で述べたアルゴリズムに従って、シミュレーションの実行を開始する。子プロセスも通信路の設定が終了すると、問題を受け付ける状態に入り、問題を受け付けると直ちにシミュレーションの実行を開始する。

4.2 シミュレータの構成

Sapal-BB における分枝限定法のシミュレータの構成を図6に示す。

Sapal-BB で構成したシミュレータは親プロセスが一つ、子プロセスが三つという構成である。子プロセスの数についてはプログラムのコンパイル時に指定できるがここでは視覚化する問題の規模や視覚化したときの見やすさなどを考えてこのような構成としている。

各プロセス中では、生成された部分問題の情報は木構造をなして保持されている。これによって、探索方法を容易に変更できるという利点がある。このデータ構造に対して用意したルーチンの内で主なものは次のものである。

- 活性問題を追加する。

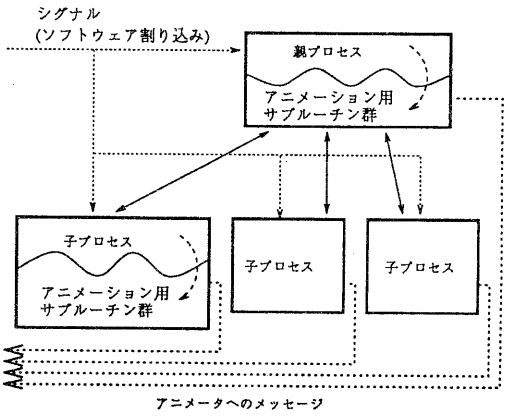


図 6: シミュレータプログラムの構造

- 次に処理すべき問題を得る。

分枝限定法のアルゴリズムからはこのレベルのルーチンしか呼ばないようにしている。このためシミュレータの各プロセス中で、部分問題を保持しているデータ構造をアルゴリズムや探索方法の特徴を生かしたものに変える時にも分枝限定法のアルゴリズムの部分には変更を加えずにすむようにしている。

各プロセスからアニメータに通信するときには、直接ソケットにそれらの情報を書き込むのではなく、図6にも示したように通信用のサブルーチン群を用意し、それらを分枝限定法のアルゴリズム中に埋め込むことで実現している。このためアニメーションを行わないシミュレータが必要な場合、通信用のサブルーチンを呼び出している部分を削除し、さらに初期化時に行っているアニメータとの接続用の手続きを変更すれば良い。

アニメーション用のサブルーチン群の中の主なものとしては、問題の状態（活性であるか、枝刈りされたか、終端されたか）をアニメータに報告するためのサブルーチンと、各プロセスで保持している暫定解が変更した事を報告するためのサブルーチンが用意されている。

4.3 アニメータの構成

Sapal-BB におけるアニメータの構成を図7に示す。

シミュレータ用の全てのプロセスとの接続を確立すると、アニメータ用のルーチンが実行を開始する。X toolkit を用いたプログラムで必要になる初期化を行った後、Xサーバからのイベントを受け付ける状

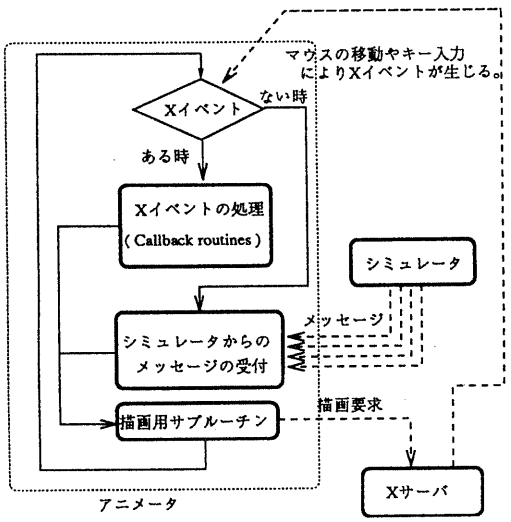


図 7: アニメータプログラムの構造

態になる。このままではシミュレータからの要求を受け付けられないため、X toolkit で用意されている WorkProc と呼ばれる機能を利用する。この機能はXサーバからのイベントの通知がないときには、あらかじめ登録しておいたルーチンを処理するようにする機能の事である。これを用いる事でシミュレータからのメッセージを受け付けられるようになる。

シミュレータからアニメータには、図5にも示したように親、子プロセスから一つずつ通信路を用意している。ところが、アニメータとそれからの要求を受け付けるXサーバにも処理能力の限界があり、これが原因になって通信路上にメッセージがたまる事がある。

アニメータは、これら複数の通信路にたまっているメッセージの中で、時間的に一番古いメッセージから処理してなければ、画面上に描いている分枝図に矛盾を生じさせる可能性がある。通信路ではキューの形でメッセージがたまるので、一本の通信路だけを見た場合は、時間的な問題は発生していない。そこでまず、各通信路から最新のメッセージを一つずつ読み込んで、それをアニメータ中のバッファにしまい込む。

次に、そのバッファの中から最新のものを一つ選ぶために、前に述べたシミュレータ側で呼び出されるアニメータとの通信用のサブルーチン群が実行される時、各メッセージに時間情報を付加する。UNIX では時間を知るために gettimeofday() と言うシステムコールが用意されており、これを利用する。

すべてのメッセージに付加されるこの情報を用いて一番古いメッセージを見つけ、それに対して処理を行い、必要な処理を終了すると、そのメッセージを読み込んだ通信路から次のメッセージを読み込み、次に処理すべきメッセージを見つけると言う仕組みを用意している。この仕組みがあるために時間的な矛盾が生じないように描画を行う事が出来る。

5 Sapal-BB の実現例

5.1 親画面の機能

親画面には分枝図の全体が描かれ、その例を図8に示す。

- 実際に画面上に描画されるのは、分枝図全体の一部であるが右横および下に用意したスクロールバーを用いることで画面上に描画される領域を変更することが可能である。
- 図中の点線は、枝刈りされた問題を表している。四角の枠は子画面がどの範囲を表示しているかを表している。また、プロセッサ１つ１つに異なる色を設定することが可能である。
- 図中の数字は子プロセッサから新たに親プロセッサに報告された暫定解を表しており、その位置によってその暫定解が得られる深さなどを知ることが出来る。
- 分枝図が描画されている領域の任意の点をマウスの左ボタンでクリックすると孫画面が表示され、分枝図中には孫画面がどの領域を表しているかを示す三角形が描かれる。この孫画面は、プログラムのコンパイル時に設定した数だけ同時に表示可能である。また、この時シミュレータは強制的に停止させられる。
- “Processes” メニューを選ぶと実行されているプロセッサ数分のサブメニューが表れる。この中のひとつを選択することにより、表示させなくした子画面を再び表示させることが可能である。
- “Control” メニューには “シミュレータ実行停止” と “シミュレータ実行継続” の二つのサブメニューがある。ここから適当なメニューを選ぶことにより、シミュレータの実行を制御することが可能である。また、停止中の分枝図上でマウスの中央ボタンを押すことで実行の継続を指示することも出来る。

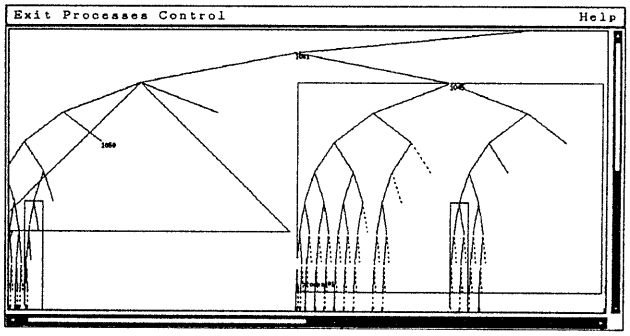


図 8: 親画面の描画例

5.2 子画面の機能

子画面には各々のプロセッサが処理中の問題を葉とする分枝図の一部が描かれる。この画面は新たな活性問題が出来た場合や、枝刈りが生じた場合に描き換えなければならない。しかし、パラメータの設定により、常に書き換えをさせるのではなく、数回に一度書き換えをさせる様にできる。描画例を図9に示す。

- 画面中には、この画面がどのプロセッサに対応しているかを示すラベルがある。その下に各プロセッサごとの情報が示される。一つ目はそのプロセッサが現在持っている暫定解であり、二つ目は現在表示されている葉の実際の分枝図中における深さである。最後の数字はそのプロセッサが現在持っている活性問題の数である。

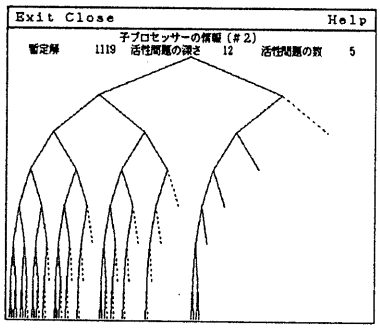


図 9: 子画面の描画例

5.3 孫画面の機能

孫画面にも分枝図の一部が描かれる。この画面は親画面の分枝図上をマウスの左ボタンでクリックすることで表示され、実際に孫画面で観察可能な領域は親画面の分枝図上に描かれた三角形で表される。描画例を図10に示す。

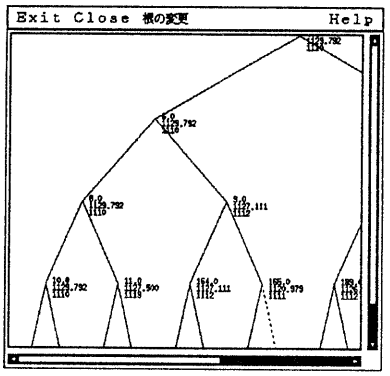


図 10: 孫画面の描画例

- 孫画面は親と同様に、観察可能な領域すべてを一度には表示しない。見えていない領域はスクロールバーを用いたり、孫画面全体の大きさをウィンドウマネージャのリサイズ機能を使って変更することで見えるようにできる。
- 各節点の脇に振られた数字の意味を以下に示す。

一段目 “,” で区切られた二つの数字がある。第一の数字をたどることによりどのように変数の固定が進んできたかについて知ることが出来る。第二の数字はその問題が枝刈りされた順序を表していて、“0” である節点の問題は枝刈りがなされていないということを表す。

二段目 その部分問題の上界値である。

三段目 その部分問題の下界値である。

- “根の変更” メニューから適当な移動先を選ぶことにより孫画面に表示されている部分木の根を変更することが可能である。“上へ” を選ぶと根がひとつ上の部分木を再描画し、親画面中の観察可能領域を示す三角形も上にずれる。“左下へ”、“右下へ” も同様に動作する。

5.4 今後の課題

今回作成したSapal-BB を用いることで、並列分枝限定法のアルゴリズムの振舞いを使用者に対して動的に視覚化するという初期の目的は充分達成できたと思われる。しかし、このプログラムにはまだ改良すべき点や、追加されるべき機能があると思われる。これらを以下に列挙する。

1. 問題の順序の表示について

   孫画面に表示される問題の順序を表す数字であるが、深さ優先探索のように探索が直線的に進む探索法に対しては、充分その動きを追いかけられ非常に有効である。しかし、最良上界値探索のように探索が直線的には進まず、次に処理する問題の分枝図上での位置が大きく変わるような探索方法に対しては、その動きを手作業で追いかけるのは難しい。そこで、アニメータ側で受け付けた情報に対して時間的なリンクを付加し、この情報を用いて使用者によりわかりやすく時間的な関係を示す必要があると考える。

2. プロセッサの状態表示について

   現在のアニメーションは分枝図についてのみの情報であり、実際に問題を処理しているプロセッサ自身についての情報はあまり表示されない。並列分枝限定法のアルゴリズムの設計を行う際には各プロセッサについての情報が必要になることも充分考えられる。この様な要求にも答えられるべきであろう。

6 結論

Sapal-BB では、並列分枝限定法の視覚化を 0—1 ナップサック問題を例として取り上げ実現した。

視覚化する時に、分枝限定法のシミュレータ本体から、アニメーションの要求を直接表示装置に行うのではなく、それらに関する機能はシミュレータから分離するようにした。具体的には、アニメータとシミュレータを UNIX のプロセスのレベルで分離した。Sapal-BB 本体のプログラミング中でも、この設計方針はプログラミングの負担の軽減という面で有効であった。また、この考え方を進めることで、たとえば水平探索のような、様々な種類の分枝限定法のシミュレータに対してもアニメーションを行うことが可能になると考えられる。

シミュレータに視覚化の機能を付け加えるための変更を出来るだけ少なくするためにアニメーション用のサブルーチン群を用意した。このため、シミュレータ用のプログラム中に関数呼出しを入れるだけで視覚化が可能である。

視覚化された図は、全体を大きく見るものから、具体的な数値を乗せた図まで親、子、孫と 3 レベルの画面を用意した。親画面中に子、孫の表示領域も示したので相互間の関係も把握できる。

一度実行が始まると終了まで使用者は、なにも指示が出来ないというものではなくシミュレータの停止と再実行を指示することは可能である。しかし、アニメーションの表示の制御には他にも速度を任意の時点で変更したり、前の状態に戻らせるなどの制御もある。このようなより高度な要求に対しても考慮する必要がある。

さらに今後は、アニメータの機能を充実させるとともに、このツールの機能を使用して実際に並列分枝限定法のアルゴリズムを評価、開発することが考えられる。実際にアルゴリズムの開発に用いれば、その過程でこのツールの問題点や有効性などがより明らかになると考える。

参考文献

[1] 茨木俊秀 : “組合せ最適化問題”, 産業図書, 1983.

[2] 西山、三宮、茨木 : “最適化”, 岩波書店, 1982.

[3] Catherine Roucarirol : “Parallel branch and bound algorithms - an overview”, Parallel and Distributed Alogorithm, M.Cosnard et al.(Eds), North-Holland, 153-163, 1989.

[4] R.Kan and H.Trienekens : “A simulation tool for the performance evaluation of parallel branch and bound algorithm”, Mathematical Programming ,Vol.42, No.2, 245-271, 1988.

[5] M.H.Brown : “Exploring Algorithms Using Balsa-2”, IEEE Computer, Vol.21, No.5, pp.14-36, May 1988.

[6] M.H.Brown : “Algorithm Animation”, The MIT Press, 1988.