HAL
open science

Parallel branch and bound algorithms- an overview

C. Roucairol

To cite this version:

C. Roucairol. Parallel branch and bound algorithms- an overview. RR-0962, INRIA. 1989. inria-00075597

HAL Id: inria-00075597

https://inria.hal.science/inria-00075597v1

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Rapports de Recherche

N° 962

Programme 2

PARALLEL BRANCH AND BOUND ALGORITHMS - an overview

Catherine ROUCAIROL

Janvier 1989

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

RR_0962

PARALLEL BRANCH AND BOUND ALGORITHMS - an overview

PARALLELISATION DES PROCEDURES PAR SEPARATION ET EVALUATION - une revue

Catherine ROUCAIROL

Abstract :

In this paper, we present an overview of existing results on the parallelization of tree search algorithms like Branch and Bound procedure.
These methods are the most efficient ones solving exactly optimization problems of high complexity (NP hard problems). So, we give a characterization of the different ways B&B algorithms are designed both with a classification of different approaches based upon computer architecture.
We discuss also performances (particularly, anomalious behaviour) and report experiments on shared memory multiprocessors machine and distributed architecture.

Keywords : parallel algorithms, combinatorial optimization, parallel branch and bound.

Résumé :

Nous présentons ici l'état de l'art sur la parallélisation de méthodes de recherche arborescente comme les procédures Branch & Bound.
Ces méthodes sont les plus efficaces pour la résolution exacte de problèmes d'optimisation combinatoire difficile (NP hard en complexité). Nous donnons une caractérisation des différentes façons de paralléliser ces méthodes ainsi qu'une classification des différentes expériences réalisées suivant l'architecture utilisée. Nous discutons également des performances (en particulier, des anomalies) et reportons des tests effectués sur des machines multiprocesseurs à mémoire partagée et des réseaux distribués.

Mots clés : algorithmes parallèles, optimisation combinatoire, procédure par séparation et évaluation parallèle.

PAPIER RÉCUPÉRÉ ET RECYCLÉ

PARALLEL BRANCH AND BOUND ALGORITHMS - an overview

Catherine ROUCAIROL

1. INTRODUCTION

Branch and Bound (B&B) algorithms are the most efficient methods known solving many optimization problems of high complexity (NP-hard problems). They can also be viewed as the most general techniques for the search for solutions in a combinatorially large problem space which is a major problem in Operations Research and Artificial Intelligence. Backtracking dynamic programming, decision trees like AND-OR, and α-β search are variations of B&B algorithms. But as these algorithms realize an implicit enumeration of the space of solutions, their computational requirements (time and space) grow exponentially in the problem size and overflowing storage and excessive running time can stop the program before reaching the optimal solution. Therefore, the idea to parallelize them has naturally emerged. The goal of this parallelization is

to solve problems **faster**
to solve **larger** problems.

In the literature, up to know, different versions of parallel B&B have been proposed ; they have been implemented either on parallel machines which are emulated on sequential ones, or on commercial supercomputers and networks. Our aim is to give an overview of existing results. The second section presents a classification of different approaches based upon computer architecture ; the third presents a characterization of the different ways B&B algorithms are designed. Section four discusses performances and points out some anomalious behaviour under parallel computing proofs (theoretical and experimental) that parallel B&B can exhibit anomalous speed-up are presented. Section five reports on experiments on shared memory multiprocessors machine and other experiences from literature on distributed architecture.

2. PARALLEL ALGORITHM DESIGN AND COMPUTER ARCHITECTURE

MIMD		
SM (shared memory) processors share a common address space		MP (message passing)
TC (tightly coupled)	LC (loosely coupled)	all communication and synchronization of processor/memory nodes is achieved by message passing through an interconnection network.
physically realised as a single, global memory	formed by combining local memories associated with processing elements	
Cray x-MP Cray 2	Cm*(Univ. Carnegie Mellon) DPUP System (Univ. Colorado)	hypercube iPSC

Figure 1

The first classification of previous work we propose is based upon the characteristics of the parallel machine used. In fact, B&B algorithms have been implemented on various classes of computers such as array processors (AP), data flow machine (DT), MIMD machines. These last class is usually subdivided using as a criteria the way in which the various processing elements are connected and communicate each other (figure 1).

Using these classification on computer architectures, three kinds of works can be distinguished (table1) :

- first, those proposing parallel B&B algorithms without any effective implementation or using a simulated parallelism on a sequential machine ; of course, they are the oldest ones because at this time, either no machine was available or access to them was difficult (reserved to military purposes or to physicists),

- the second category of works concerns the implementation on experimental machines (machines built in research laboratories with exotic architectures more or less),

- then, the latest experiments conducted on commercial supercomputers.

Machine	Vector computers Array processors Data Flow machine	Multiprocessors tightly coupled system	Network loosely coupled system
Theoretical (simulated parallelism)		75 Pruul + simulation Cornell Univ. (USA) 79 + simulation Imaî, Fukumura Nagoya Univ. (J) 80 El Dessouki, Huen (USA) 84 + simulation Lai, Sahni (USA) Wah, Yu + simulation Wah, Ma Purdue Univ.	83-84 Rayward - Smith Mac Keown et al. Univ. East Anglia (UK) 84 Lavallée, Roucairol Inria (F)
Experimental machine	85 Kindervater (NL) Manchester data Flow mach.	84 Mohan Cm* (H) Carnegie Mellon (USA)	75 Finkel, Manber (V) Crystal DIB Univ. Wisconsin Madison (USA) 86 Trienekens (H) DPU Boulder Univ. Colorado (USA)
Commercial machine	85 Kindervater Trienekens ICL Dap CDC Cyber 205 (NL)	85 Lavallée, Roucairol (F) Cray XMP 86 Roucairol (F) Cray 2 (H\V) 88 Boehring, Butler, Gillet (USA), Hep, Sequent, Encore (H), Rodgers Pardalos (USA), Cray XMP, IBM (H) 3090-600E, Plateau, Roucairol (F), Cray 2 (H)	87 Mraz, Seward (USA) Rodgers, Pardalos (USA) Intel iPSC hypercube (V) Vornberger (RFA) Transputer Inmos (V)

Table 1

Obviously, the architecture of the selected computer will influence the design of parallel algorithms which are implemented.
A decomposition of work into independant processes must be found ; necessary synchronization and communication mechanisms between concurent processes must be established in order to ensure a good trade-off between communication and computation.

3. TWO MAIN FRAMEWORKS FOR ASYNCHRONOUS PARALLEL B&B ALGORITHMS

We first briefly recall what is a B&B algorithm.

3.1. B&B algorithms on a single processor machine

The goal of the B&B algorithm is to solve a constrained optimization problem :

$$\min f(x)$$
$$x \in X ,$$

where X represents the domain of optimization,
x is a solution, x is feasible iff $x \in X$,
f(x) is the value of the solution.

The B&B method is based on the idea of intelligently enumerating all the feasible solutions of a combinatorial optimization problem.
The underlying idea of the B&B algorithm is to decompose a given problem into two or more subproblems of smaller size. Then, this partitionning process is repeatedly applied to the generated subproblems until each unexamined one is either partitioned, solved or shown not to yield an optimal solution of the original problem.

The process of excluding subproblem from further consideration is based upon the computation of a lower bound on the values of solutions within each subset : subproblems whose bounds exceeds the value of some known solution can be discarded.

The state of the partitionning process at any time can be represented as a partial tree in which subproblems are represented by nodes and the decomposition of problem into subproblems by edges from nodes to their successors.

More precisely, a B&B algorithm has three major components : a branching scheme, a bounding function, a search strategy.

Let $B=(S,\Gamma)$ be the tree built, S the set of nodes and Γ the application successor.

Branching principle Γ

The root of the tree SO is the set X (or a set of solutions U with X included in U). Successors $\Gamma(Sk)$ of every node Sk are defined by the branching scheme, which must satisfy three conditions :

(c1) the union of partitionned subsets is equal to the initial set
(c2) the number of node partitions necessary to reach an optimal solution is finite
(c3) a node -associated to a subset of solutions- cannot be partitionned (expanded) if :
- this subset is reduced to only one feasible solution
- this subset does not contain any feasible solution or feasible solutions whose cost is smaller then the cost value of the beat known feasible solution. Such a node is called a leaf of the B&B tree.

Bounding function v

A lower bound function v assigns to each subset of solutions a real number representing a lower bound cost for all complete solutions in the set. It has the following properties :

(p1) $v : S \rightarrow \mathbb{R} \cup \{\infty\}$
(p2) for every node Sk of B associated with a set of solutions,
$v(Sk) \leq \min \{f(s) / s \in Sk\}$,
(p3) for every leaf node Sk of B, $v(Sk) = \min\{f(s) / s \in Sk\}$,

(p4) if no feasible solution belongs to Sk, $v(Sk)=+\infty$,

(p5) v is non decreasing on S : if Sl is a successor of Sk then $v(Sl) \geq v(Sk)$,

(p6) let us suppose that an upper bound UB of f^* is known (for example UB is the value of a feasible solution found by any heuristic method or the cost of the cheapest solution yet seen) ; whenever Sk is such that $v(Sk) \geq UB$ then Sk is not selected further for expansion.

Search strategy h

A strategy is a rule for choosing which of the currently active nodes the branching principle should be applied to. For conceptual simplicity and uniformity of notation, we assume that a heuristic function h, $h : S \rightarrow \mathbb{N}$, gives a priority to each node and thus the selected node is the one with the smallest h value.

The choice of a heuristic function depends on the application ; the most commonly used are :

depth first $h(Si) = l(Si)$ where $l(Si)$ is the level of node Si - length of the path from SO to Si -
best first $h(Si) = v(Si)$.

The first strategy leads quickly to a feasible solution and is space-saving : a node at a lower level will always be examined before a node at a higher level ; hence the set of active nodes can be maintened in a FIFO list.

The second strategy is more targed towards a good solution ; good solutions are determined first but all active nodes not yet examined must be stored. However, it can be shown that the total number of nodes expanded is minimised in the sense that any branching operation performed under this strategy must also be performed under other strategies, provided some condition on the bound's value hold.

Dominance relation may also be introduced in B&B algorithms ; it is a relation on subproblems such that if Si dominates Sj, then Sj cannot contain a better solution than Si and thus Sj can be eliminated.

Procedure B&B

```
liveset = set of active nodes
begin
        liveset = {root};
        if root is a solution node then UB = f(root) else UB = ∞ ;
        while liveset contains a node S with v(S) < UB do
            begin
            x = node in liveset with min h() ;
            delete x from liveset ;
            / branching scheme /
            create successors nodes of x ;
            / evaluation /
            for each successor y of x do
                    begin
                    if y is a solution and f(y) < UB then UB = f(y) ;
                    if y is not a leaf and v(y) < UB then add y to liveset ;
                    end
            end
end.
```

3.2. Asynchronous B&B algorithms

While dividing the work (i.e. expanding parts of the B&B tree) among a bounded number of processes, we have to avoid the following situations in order to reduce enumeration :

- several processes expand the same part of the tree,
- a process is working on a node of the subtree whose lower bound's value is greater than the value of a solution found by another process in another part of the tree.

Hence, each process must broadcast all information obtained and the upper bound UB must be constantly updated. This can influence the choice of the next expanded node and some branch which would occur in a sequential algorithm can be pruned.

Obviously, the synchronization of information exchange between processes (i.e. processes wait for others which have not completed their tasks) is a solution that can be handle more easily but it will increase the overall complexity of the algorithm. In B&B, the total work size (final size of the tree) is not complety known in advance and it is very difficult to divide it into equal parts. Synchronization will force some processes to wait for others and, hence, to lose time.

This hypothesis has been verified by experiments with parallel synchronous B&B algorithms conducted by Mohan on Cm* at Carnegie Mellon (MOH 84), and Kindervater on ICL DAP at Mancester (KIN 85).

There are two main methods to distribute the B&B algorithm's operations among a bounded number of processors.

Asynchronous parallel B&B algorithms

Vertical

Every process searches as a complete subtree independantly of the others. The information exchanged between various processes is reduced to one value per complete solution. As soon as this bound is communicated, it is the responsability of each process to eliminate nodes with lower bound greater than this value.
The principal difficulty comes from necessary synchronizations between processes which have terminated their enumeration in order to obtain some other work from their partners quickly and consistently.
This kind of algorithm is dedicated to computer networks or multiprocessors with limited memory space and slow interprocessor communication.
The principal criticism is that some processes may be working on nodes of the subtree that could be eliminated if a better communication (last upper bound) was designed.

Horizontal

A specialized process - the scheduler-keeps track of live nodes, maintains the current value of the best known feasible solution, and schedules the processes which expand either a live node (one or two levels) or a subtree with a bounded number of nodes (before returning for re-scheduling).
The advantage is that information about the current bound is kept reasonably up-to-date.
But as the number of processors grows, the scheduler process and message transport mechanism will become bootlenecks. Hence, this algorithm is dedicated to a system with a modest number of processors and an efficient message passing scheme, like a multiprocessors with a shared memory.

These algorithms are respectively denoted by H(horizontal) and V(vertical) in table 1.

4. PERFORMANCES OF PARALLEL B&B

Classical measures for the efficiency of algorithms implemented on multiprocessors are speed-up and efficiency.
The speed-up S achieved by a parallel algorithm A executing an instance of a problem π, is defined as the ratio between T1 and Tp : $S(A,\pi) = \frac{T1}{Tp}$, where Tp is the time taken by the parallel computer executing the parallel algorithm using p processors, T1 is the time taken by the execution of the best known sequential algorithm with only one processor.

Remarks.

(i) This time is measured

- either theoretically : average number of operations, or number of iterations between two synchronizations in case of parallel synchronous algorithm, (in case of asynchronous machine, a parallel algorithm is linked with several execution times, which is due to non-determinism ; thus, it would be preferable to compute the average time) ;

- or experimentally (dedicated CPU time).

(ii) Some authors use as T1, the time of the parallel algorithm with one processor. But, synchronization and communication statements are included in the parallel program and penalize the serial execution. It is a simple way if comparing experimental parallel executions with an increasing number of processors.

(iii) Average speed up on all the same instances size of a combinatorial optimization problem can be usefully considered.

Efficiency

The efficiency of a parallel algorithm running on n processors is the speed-up divided by p. In general, the speed-up less than p but close to p is expected. So the following cases are called anomalies :

detrimental anomaly	$Sp < 1$
deceleration anomaly	$Sp << p$
acceleration anomaly	$Sp > p$

So superlinear speed-up are possible ; because it is not always possible to choose for a given problem instance the best serial algorithm and because in parallel, extra processors may enable the parallel algorithm to find the optimal solution very quickly. Of course, one can say it is not possible because a sequential computer can always emulate a parallel computer.

These anomalies are due to the fact that, for a problem's instance, the search strategy which leads to the best serial B&B algorithm is not known "a priori". Anomalious behaviours of B&B algorithms under parallel processing have been studied by many authors : Lai and Sahni (LAS 84), Lai and Sprague (LASP 85), Li and Wah (LIW 84), Roucairol (ROU 87).

The computational efficiency of these algorithms depends on the bounding function, the data structure and the search strategy.

Let **B&B tree** denote the tree generated by the B&B algorithm, **critical tree** denote the subtree whose nodes Si are such taht $v(Si) < f^*$, **minimum tree** denote the subtree whose nodes Si are such that $v(Si) \leq f^*$ (contains minimal number of nodes to expand in order to find an optimal solution).

A B&B algorithm A = (Γ branching principle, v bounding function, h strategy) is h-optimal if the B&B tree is minimum.

Theorem : if T^*1 is the time with a h-optimal B&B (called A) to expand all the nodes of the minimum tree, Tp the time with a parallelization of A on p processors, speed-up

$$T^*1 / Tp \leq p$$

This theorem shows that anomaly can occur when the serial B&B algorithm is not h-optimal.

Let us define some properties of the bounding function :

v is discriminating	iff $\forall (Si,Sj)$ $v(Si) \neq v(Sj)$
h is consistent with v	iff $h(Si) \leq (Sj) \Rightarrow v(Si) \leq v(Sj)$
v is weak	iff $\forall$ Si which is not an optimal solution $v(Si) \neq f^*$.

Theoretical results have been obtained under some assumptions :

- horizontal parallelization,
- all processors are synchronized.

Time is measured as a number of iterations required ; at each iteration, p processes expand in parallel the p first priority nodes of the list (provided this list contains at least p nodes) and add their children to the share list. We summarize the contents of several theorems whose proofs can be found in Lai and Sahni (Lais83), Lai and Sprague (LAI 85), Li and Wah (LIW 84), Wah and Yu (WAY 82,85), Roucairol (ROU87) :

- for a sequential B&B algorithm, if v is discriminating, a best first strategy h builds the critical tree,
- for a parallel B&B algorithm, if h is a best first strategy and v is discriminating, the speed-up is $Sp \leq p$,

- moreover, under these conditions, this strategy is robust - i.e. $Sp \cong p$ $(p-t(p-1)/Tp) \leq T1 / Tp \leq p$, is the t height of the B&B tree with 1 processor),
- necessary condition for good anomalies is that h is inconsistent with v,
- sufficient conditions for bad anomalies are that v is discriminating and h is consistent with v.

5. SOME EXPERIMENTS

5.1. Experiments on asynchronous shared memory multiprocessor machines

The main characteristics of an horizontal parallel B&B algorithm for shared memory multiprocessor machines (as Cray X-MP or Cray 2) are the following :

The *distribution of work* among the different processes (one process assigns to one of the four processors) is done by giving access to a shared list which contains information about every node to be expanded.

This list is implemented as a heap (binary tree) in order to use fast existing algorithms to insert, sort and remove items.
A priority (adapted to the problem to be solved) is associated to each node. Nodes are ranked by decreasing priority : every active process finds at the top of the list the node it has in charge.

Insertion of items is done whenever the expansion of a node generates several successors whose evaluation is less than the best known upper bound. The best known upper bound (BKUB) is a shared variable which is updated whenever a local upper bound (lub) less than BKUB is found at a node to be expanded.

Items are suppressed either at the beginning or at the end of the list. The former case occurs whenever an inactive process looks for a new job, the latter case occurs whenever a lub is less than BKUB : every node whose evaluation is greater than lub is eliminated because it cannot lead to an optimal solution.

The algorithm terminates when the list is empty and all processes are inactive.

We report first on our experiments with the horizontal parallelization of a serial algorithm for the asymetric traveling salesman problem (TSP) (other results for TSP can be found in (ROU86), for Quadratic assignment in (ROU84), for Multiknapsack in (PRG88)).

Given a complete oriented and weighted graph in which the weight of an arc c(i,j) represents the cost of traveling from i to j, the traveling salesman problem is to find a circuit of minimum cost that goes through every node exactly once.

First results have been obtained on an emulator called CREM running on a Multics machine BULL DPS68 at INRIA, France and later results on the Cray X-MP 48 at CRAY RESEARCH, Minneapolis, USA.

Speed-up :

The coefficients c(i,j) of the matrix C are generated from a uniform distribution in a range [a,b[. The best first strategy is used. Speed-up is computed here as : T1/Ti where T1 is the solution time for one processor and Ti the solution time for i processors.

Number of processors (from 1 to 3 on CREM)	Average simulated time in seconds (CREM)	Average number of nodes in the B&B tree N
1	373	10
3	150	11
Average speed-up = 2.5	Average increase of nodes = N1/N3 = 1.1	

Table 2. 40 TSPs of size 200, $c(i,j) \in [0,100[$.

Moreover, with one processor it was impossible to find optimal solution for 6 problems of size 200 and 21 of size 150 in a time limited to 900 seconds.

Number of processors	Average simulated time in seconds (CREM)	Average number of nodes in the B&B tree
1	> 687	> 8
3	424	11,5
Average speed-up > 1.6	Average increase of nodes = N1/N3 < 1.4	

Table 3. 40 TSPs of size 150, $c(i,j) \in [0,1000[$.

But, these results hide a part of reality : worst speed-up is 0.5, best 12.5. The next table shows the differences between problem instances.

Anomaly	Class	Number of TSPs	Average speed-up S	N3/N1
	$S < 0.8$	3	0.7	2.8
detrimental	$S \in [0.8,1.2]$	11	1	2.8
deceleration	$S \in [1.2,2.8]$	11	1.8	1.5
	$S \in [2.8,4]$	7	3.1	0.9
acceleration	$S > 4$	8	6.9	0.5

Table 4. Anomalies

This table indicates that the time is directly related to the number of nodes expanded in the search tree.

For this combinatorial optimization problem, the size of the B&B tree to be built is small (about twenty nodes), but the time taken to evaluate a node is rather important. Hence, a larger number of nodes will increase the total time.

But, whenever the bounding procedure does not take enough time, some granularity problems may occur because the size of the work performed in parallel is to small.

For example, let us consider the following 0.1 multiconstraint problem :

$$\begin{array}{ll} \text{maximize} & cx \\ \text{subject to} & Ax \leq b \; ; \; x \in V \; , \end{array}$$

whose data are such that

$c \in \mathbb{N}^n$, $b \in \mathbb{N}^m$, A is a mxn dense non-negative integer matrix,
and where $V = \{x \in \mathbb{R}^n \mid x_j = 0 \text{ or } 1,\ j=1,2,\ldots,n\}$.

An horizontal parallel B&B algorithm has been implemented on the Cray 2 (Plateau and Roucairol, (PRG 88)).

In order to balance the loads of each processor, we decide to define a priority rule for accessing the shared list of node. Namely, a process is allowed to treat a node if its current load does not exceed t% of the mean load.

	t	time(s) Cray 2	Processors 1	2	3	4
N p	0	3.29	79 25%	123 34%	89 22%	67 19%
N p q	10%	2.51	92 26% 2	86 24% 2	91 25% 24	89 25% 3
N p q	30%	2.13	84 23% 0	95 27% 2	87 24% 0	92 26% 10
N p q	50%	1.93	95 26,5% 0	95 26,5% 3	80 22% 0	88 25% 0

Table 5. Granularity's problem on Fleisher's example (m=10, n=20)
(N number of nodes treated by a processor, p percentage of work done by a processor, q number of times a processor waits)

5.2. Experiments on others architectures

We choose to present the results obtained by Rodgers and Pardalos (RP 88) for quadratic 0.1 programming on various parallel computer architectures :

$$\min c^T x + 1/2\, x^T A x$$
$$x \in \{0,1\}^n \text{ where } A \in \mathbb{R}^{n \times n},\ c \in \mathbb{R}^n \text{ are given}$$

For the shared memory multiprocessor machine, they propose a synchronous / asynchronous vertical parallelization ; a variable M controles the number of nodes that processes are allowed to expand in parallel at each cycle of the synchronous algorithm.

For distributed memory processors, they design a vertical parallelization : a process expand a subtree of M nodes then send subproblem to neighbor.

From their results (table below) it is clear that the algorithm for the hypercube were nearly as efficient as the shared memory algorithms.

Maximum Achieved Speedups					
MACHINE	T	T	Speedup	Efficiency	Number of processors
CRAY XMP/48	17.52	4.61	3.80	.95	4
IBM 3090 600E	15.48	2.97	5.22	.87	6
Intel iPSC/1	1903.35	95.32	19.97	.62	32
Intel iPSC/2	428.26	35.10	12.20	.76	16

Table 6. Speed-ups for an example from (RP 88)
with A = (100 × 100)

8. CONCLUSION

It is clear that parallelism or distribution of work can improve significantly the performance of B&B algorithms. Vertical parallelization seems more adapted to distributed architecture,

horizontal one is well suited to share memory multiprocessors. Furthermore, it is interesting to stress the two following points :

- even with a single processor machine and a simulated parallelism, optimal results have been obtained for problems (experiments with asymetric traveling salesman problem (ROU 86)) which were not solved in sequential ; it means that asynchronous or merely non determinism of choice can improve the performance of a pure sequential and deterministic algorithm ; of course, performances can be improved with a real parallel machine.
- speed-up is almost linear with respect to the number of processors.

More precisely, there exist speed-ups which are more than linear and these cases are more frequent than sub-linear speed-ups. Up to now, only the exploration of the B&B tree has been parallelized ; but in many combinatorial optimization problems like the TSP or the Quadratic assignment problem, the evaluation procedure called in each node of the B&B tree takes a long time (about 80%) with respect to the time of buiding the tree. So the parallelization of this procedure should be studied. From our point of view, it may be enough to vectorize the evaluation procedure (in most cases a linear program). This has to be tested on a machine having many processors, each one owning a vector facility.

We must also recognize that there is still a lack of experiments for "real problems" : problem of larger size must also be included into experiments.

REFERENCES

(BBG88) Boehning R., Butler R., Gilett B., A parallel integer linear programming algorithm, EJOR 34, 393-398, 1988.

(EDH80) El-Dessouki O., and Huen W.H., Distributed enumeration on network computers, IEEE Trans. on Comp. 29, 818-825, 1980.

(FM85) Finkel R. and U. Manber, "DIB- a ditributed implementation of backtracking", IEEE, 446-452, 1975.

(IF79) Imaï M. and T. Fukuruma, "A parallelized Branch and Bound algorithms implementation and efficiency", Systems computers controls, Vol. 10, n°3, 62-70, 1979.

(KIN85) Kindervater G., Trinekens H., Experiments with parallel algorithms for combinatorial problems, Report OS-R85 12, center for Mathematics and Computer Science, Amsterdam, 1985.

(LAS84) Lai H., Sahni S., Anomalies in Branch and Bound, Com. of ACM, Vol 27, n°6, 594-602, 1984.

(LASP85) Lai T., Sprague, Performance of Branch and Bound algorithms, IEEE Trans. on Comp., C-34, n°10, 194-201, 1985.

(LAROU85) Lavallée I., Roucairol C., Parallel Branch and Bound algorithms, presented at Euro VIII Congress, Bologna, Italy, RR MASI n°112, Univ. Paris VI, 1985.

(LIW84) Li G., Wah B., Computational efficiency of parallel approximate branch and bound algorithms, Proc. of the Int. Conf. on Parallel Processing, 473-480, 1984.

(LMR88) Lichnewsky A., Marchand D. and Roucairol C., A fast vectorized version of a linear assignment algorithm, RR INRIA, à paraître 88.

(MS87) Mraz R. and W. Seward, "Performance evaluation of parallel Branch and Bound search with the intel iPSC hypercube computer", Proceedings of Supercomputing 88, Boston, 82-91, 1988.

(MOH84) Mohan J., A study in parallel computation : the traveling salesman problem, Carnegie Mellon University, CMU-CS-82-136, 1982.

(PRG88) Plateau G., Roucairol C., Gachet S., Algorithm PR2 88 for the parallel resolution of the 0.1 multiknapsack problem, RR INRIA to appear 88.

(PRU75) Pruul E.A., "Parallel processing and a B&B algorithm", Thesis Cornell University, Ithaca, N.Y., 1975.

(QUI87) Quinn M.J., Designing efficient algorithms for parallel computers, MC Graw Hill Series in Supercomputing and Artificial Intelligence.

(RK83) Burton F.W., M.M. Huntback, G.M. McKeown and U.J. Rayward-Smith, "Parallelism in branch and bound algorithms", Mathematical Algorithms group-2, Internal report CSA/3, University of East Anglia, Norwick, UK, 1983.

(RK84) McKeown G. and Rayward-Smith U., "Chaotic computing", RR CSA 15/1984, University of East Anglia, Norwick, UK, 1984.

(RP88) Rodgers G. and Pardalos P., Parallel Branch and Bound algorithms for Quadratic 0-1 programming, RR CS 88-17, Dept of Computer Science, Pennsylvania State University, 1988.

(ROU84) Roucairol C., An efficient branching scheme in branch and bound procedure, presented at TIMS XXV, Copenhagen, RR Université Paris VI 79-4, 1983.

(ROU84) Roucairol C., A parallel branch and bound algorithm for the quadratic assignment problem, Disc. Appl. Math., 18, 211-225, 1987.

(ROU86) Roucairol C., Experiments with parallel algorithms for the asymetric salesman problem, preesented at EURO VIII, Lisboa, Portugal.

(ROU87) Roucairol C., Du séquentiel au parallèle : la recherche arborescente et son application à la programmation quadratique en variables 0-1, Thèse d'état, Université Paris VI, Juin 1987.

(ROU88) Roucairol C., Parallel computing in Combinatorial optimization, Proceedings of Numerical methods for parallel vector computers, North Holland, to appear 1989.

(TRI86) Trinekens H., Parallel Branch and Bound on an MIMD system, RR 8640/A Econometric Institute Rotterdam, 1986.

(VOR87) Vornberger O., "Load balancing in a network of transputers", Lectures notes in Computer Science 312, Distributed Algorithms, Springer-Verlag, 116-126, 1988.

(WAM84) Wah B., Ma Y., Manip : a multicomputer architecture for solving combinatorial extremum search problems, IEEE Trans. on Comp. C3, 5, 1984.

(WAY82) Wah B. Yu C., Probabilistic modeling of branch and bound algorithms, Proc. COMPSAC, 647-653, 1982.

(WAY85) Wah B., Yu C., Stochastic modeling of branch and bound algorithms with best first search, IEEE Trans. on Software Engineering, Vol SE-11, n°9, 922-934, 1985.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique