A Parallel Branch and Bound Algorithm for Vehicle Routing Problem

G.H. Dastghaibifard, E. Ansari, S.M. Sheykhalishahi, A. Bavandpouri, E. Ashoor

Abstract— **In Operation Research, Branch and Bound is one of the basic methods to solve Integer programming (IP) problems. According to dividing property of branch and bound, parallel algorithms for solving Integer Programming are common.**

In this paper, a new parallel branch and bound algorithm is proposed for muli-computers. This algorithm instead using of shared memory multi-processor environment, uses multi computers and dynamic load balancing. As well as reduce drastically the intercommunication between processes. This algorithm is implemented for well known Capacitated Vehicle Routing Problem (CVRP). In addition, our results are quite good comparing to other algorithms.

Index Terms— **CVRP, Integer Programming, Parallel Branch and Bound**

I. Introduction

The linear-programming models all have been continuous, in the sense that decision variables are allowed to be fractional. Often this is a realistic assumption. At other times, however, fractional solutions are not realistic, and we must consider the Integer-programming (IP) model of some optimization problems [1].

Integer-programming models arise in practically every area of application of mathematical programming. Popular NP-Hard problems like: Warehouse location problem, TSP, Scheduling and Knapsack problem develop a preliminary appreciation for the importance of IP models and have showed how integer variables can be used to provide broad modeling capabilities beyond those available in linear programming.[3],[4]

Capacitated Vehicle Routing Problem (CVRP) is a famous integer programming problem which has lots of application in [14] and our proposed algorithm is implemented for CVRP.

There is no single technique to solving integer programs; however the Simplex method is effective for solving linear programs. Although a number of procedures have been developed, but the performance of any particular technique appears to be highly problem-dependent. Currently, the algorithms use one of the following classical approaches:

1) Enumeration techniques, including the branch-and-bound procedure.
2) Cutting-plane techniques
3) Group-theoretic techniques.

In addition, several composite procedures have been proposed [4]. In this paper, the first classical approaches will be considered in detail. Various sequential algorithms and heuristics for solving integer programming problems with branch and bound method can be found in [6]- [9].

Large and/or computationally expensive optimization problems sometimes require parallel or high-performance computing systems to achieve reasonable running times.

Even though many parallel algorithms [10]- [13] have been developed for branch and bound problems, one of the big issues in these algorithms is how to tackle huge amount of communication between sub-problems. In most algorithms for tackling this problem they have used parallel shared memory computers. But these computers are not affordable these days. To overcome this problem, in this paper we will consider multi-computer environment instead of shared-memory which are more affordable.

One of the main obstacles in using multi computers is how to tackle the centralized communication used in parallel shared memory computers. In order to tackle this problem we can use a central process for performing the communication among sub-problems, but this will reduce the efficiency drastically in problems with high communication. In this paper, a new method is proposed such that all sub-problems communicate to each other in the way that there is no need to have a central processor, so increase the efficiency. In addition also to reduce the idle times of processors and keep them as busy as possible, the following heuristic mechanism is proposed. First of all sub problems are distributed between different processes and each process works on problems in its queue for a specific period of time. Then processes send best value of solution and some extra information to other processes. If a process becomes idle, an unsolved problem from other processes will be assigned to it. These operations will be continued till the queue of all processes become empty.

In some phases of solving linear programming problems with

Manuscript received November 4, 2007.

G.H. Dastghaibifard is assistant professor of Computer Science and Engineering Department, Shiraz University, Shiraz, iran. (email: dstghaib@shirazu.ac.ir)

Ebrahim Ansari Chelche is Msc Student of Computer Science and Engineering Department, Shiraz University, Shiraz, iran. (email: ansari@cse.shirazu.ac.ir)

S.M. Sheykhalishahi is Msc Student of Computer Science and Engineering Department, Shiraz University, Shiraz, iran. (email: alishahi@cse.shirazu.ac.ir)

Amir Bavandpouri Chelche is Msc Student of Computer Science and Engineering Department, Shiraz University, Shiraz, iran. (email: bavandpouri@cse.shirazu.ac.ir)

Elmira Ashoor Mahani is Msc Student of Computer Science and Engineering Department, Shiraz University, Shiraz, iran. (email: ashoor@cse.shirazu.ac.ir)

branch and bound, sub-problems must being solved with classical linear programming methods such as Dual Simplex [1], [2]. For this purpose a program has been written to performing dual simplex methods on linear programming model.

In the remainder of the paper, we first describe branch and bound algorithm, and then describe CVRP in detail. Section 4 describes proposed algorithm in detail. Experimental results and analysis are given in section 5. Finally section concludes the paper.

II. Branch and Bound

Branch and bound is a technique for solving optimization problems that uses divide and conquer strategy to partition the solution space into sub-problems and then solves each sub-problem recursively.

Linear programming methods, such as simplex can be used to solve every sub-problem. If any of variables is fractional, we select one of fractional variables, and divide our B&B tree, into two branches. For example if our fractional variable A is 3.45, first branch is constraint is $A \leq 3$ and second branch is $A \geq 4$. Then we put these branches into B&B list of candidate sub-problems, and continue our algorithm.

In each step, one of the candidate sub-problems is selected, removed from the list, and simplex method will be applied. There are four possible cases.

1) Feasible solution better than the current best value, is found: In this case the current best value will be replaced by the new solution and continue.

2) We may also find that the sub-problem is infeasible so prune it. Otherwise, we compare solution of it to the upper bound yielded by the current best solution.

3) If it is greater than or equal to our current upper bound, then we may again prune the sub-problem.

4) Finally if we cannot prune the sub-problem, we are forced to branch and add children of this sub-problem to the list of candidates.

This process will be continued until the list of sub problems being empty. Finally the best answer we achieve so far is the answer of problem.

III. CVRP

We consider the Vehicle Routing Problem (VRP), introduced by Dantzig and Ramser [5], in which a quantity d_i of a single commodity is to be delivered to each customer $i \in N = \{1,.., n\}$ from a central depot $\{0\}$ using k independent delivery vehicles of identical capacity C. Delivery is to be accomplished at minimum total cost, with $C_{ij} \geq 0$ denoting the transit cost from i to j, for $0 \leq i, j \leq n$. The cost structure is assumed symmetric, i.e.,

$C_{ji} = C_{ij}$ and $C_{ii} = 0$.

A solution for this problem consists of a partition $\{R_1, R_2, \ldots, R_k\}$ of N into k routes, each satisfying $\sum_{j \in R_i} d_i \leq C$, and a corresponding permutation σ_i of each route specifying the service ordering. This problem is naturally associated with the complete undirected graph consisting of nodes $N \cup \{0\}$, edges E, and edge-traversal costs C_{ij}, $\{i,j\} \in E$. In this graph, a solution is the union of k cycles whose only intersection is depot node. Each cycle corresponds to the route serviced by one of the k vehicles. By associating a binary variable with each edge in the graph, we obtain the following integer programming formulation:

$$\min \sum_{e \in E} c_e x_e$$

$$\sum_{e \in \{0,j\} \in E} x_e = 2k \quad (1)$$

$$\sum_{e \in \{i,j\} \in E} x_e = 2 \qquad \forall_{i \in N} \quad (2)$$

$$\sum_{e \in \{i,j\} \in E, i \in S, j \notin S} x_e \geq 2b(s) \quad \forall_{S \subset N,\ |S|>1} \quad (3)$$

$$0 \leq x_e \leq 1 \quad \forall_{e=\{i,j\} \in E,\ i,j \neq 0} \quad (4)$$

$$0 \leq x_e \leq 2 \quad \forall_{e=\{0,j\} \in E,\ i,j \neq 0} \quad (5)$$

$$x_e \ \ is\ Integer \qquad \forall_{e \in E} \quad (6)$$

For ease of computation, we define:

$$b(s) = \left\lceil \sum_{j \in S} \frac{d_j}{C} \right\rceil$$

, an obvious lower bound on the number of trucks needed to service the customers in set S.

Constraint (1) ensures that there are exactly k vehicles, while constraints (2) ensure that each customer is serviced by exactly one vehicle, as well as ensuring that the solution is the union of edge sets of routes.

Constraints (3) can be viewed as a generalization of the sub-tour elimination constraints from the TSP and serve to enforce the connectivity of the solution, as well as to ensure that no route has total demand exceeding the capacity C.

It is clear from our description that the VRP is closely related to two difficult combinatorial problems. By setting $C = \infty$, we get an instance of the multiple traveling salesman problem and by setting $C_e = 0$, we get a feasibility version of the bin packing problem with a fixed number of bins.

IV. New Algorithm

In the first step, a new program has been written for solving linear programming problems. This program uses Dual Simplex method and its input is a simple form of a linear programming problem including an n*m array for constraints, goal function and constraints properties. In this program some functions have been written for inserting a new constraint to problem model and or assigning a fixed value to one of variables. These functions perform their computation and problem model changes in an efficient time. The proposed program, have good performance and reasonable speed in comparison with other linear programming packages such as Lingo [17].

In next step, a sequential program has been proposed to solve branch and bound problems. In this program we exploited some heuristic methods to select best branch in available branches such as Strongly Branch in [6] and some

of proposed methods in [7].

In linear programming branch and bound methods, one problem is the large amount of data in each sub-problem. So after adding a constraint or any other changes in sub-problems, saving the whole new sub-problem needs high memory space. As we know after several iterations the memory will become full.

To conquer this problem, a data structure has been used for saving only the changes not the whole changed sub-problems. This data structure is a linked list that each elements is an indicator to a new branch (constraint) for branch and bound tree and has a pointer to its parent (previous constraint).

In branch and bound process, in each fields of problem queue, there is just a pointer to its last constraint. For solving each sub-problem by using this pointer and tracking linked list, all of sub-problem constraints being added to original problem and new sub-problem being constructed.

When each process wants to send one sub-problem to another process, first reconstructs the whole sub-problem then sends it to corresponding process. Each process after receiving a new sub-problem, assigns it as like as a new original problem hereafter. And for building posterior branches (sub-problems), uses this sub-problem as a beginning problem (root of tree).

After solving constraints size problem, there is another problem yet. If a central process manages all communication between other processes, this process becomes a bottleneck for our algorithm. To overcome this problem we use Decentralized Load Balancing. For this purpose sending and receiving sub-problems being performed by processes personally. Although Master process just finding the idle processes and demand sender processes by doing a heuristic algorithm by considering the length of queues and the number of idle processes to find the best senders and receivers in each iteration. In addition, it sends label of sender and receiver to corresponding processes. Main idea of Assignment algorithm is sending sub-problems from high length queue processes to idle processes. Now processes doing communication instead of master process partnership. This method helps us to reduce idle time of slave processes. Ipso facto coordinator process named Assignment Process.

For synchronizing processes to do communication in same time, time variable T_p has been defined. Namely each processes doing its job for T_p and then sends best answer and other information to assignment process.

Small T_p causes more sending and receiving between processes than large T_p. And if T_p value was large processes relationship and so parallelization would become low. As we know, in initial iterations best-values being changed very quickly. So in preliminary iterations we set T_p a low value, and in posterior iterations increase value of it because in final iterations best-values changing will be rare. In our implementation and result testing, some strategies for T_p assigning are performed. The results of this examination are in the final part of this paper.

To decrease processes idle times when assignment process is doing its duty, below time parameters has been defined:

T_w: time for assignment process calculation plus sending and receiving time between assignment process and other processes. Namely a process is idle when it can do its job. Value of T_w gets updated at the end of per iteration for use in the next iteration. This job is performed by using statistical results from previous iterations. So in per iteration every process after sending its information to assignment process, do its job for T_w.

T_r: time required to receive a sub-problem by each empty process from sender process. In a normal situation value of T_r is equal to zero and when a process receives a sub-problem, this time variable being valued by receiving time. Receiver process consider this time to next calculation, namely minus it from T_p.

For parallelization implementation the MPI [15] library function has been used.

Part 4.1 and 4.2 are explanation of our branch and bound algorithm, and in fig.1 (at the end of paper) there is flowchart to algorithm illustration.

A. Assignment Process algorithm

A- Doing sequential branch and bound algorithm until there is exactly (*numproc*-1) sub-problem in branch and bound queue.

B- Build and sending sub-problems to other processes

C- Receiving best answer and queues information from other processes by *MPI_Gather*

D- By considering received information from other processes, building their array. If queues of all processes are empty, insert *Terminate_tag* in arrays. Otherwise if required, for each process, insert receiver(s) or sender processes label in corresponding array. Finally insert the best value so far in these arrays.

E- Sending built arrays to their process by *MPI_Scatter*.

T- If in Step D *Terminate_tag* had been sent to all process, Terminate algorithm.

B. Other Processes Algorithm

B- Receive sub-problem from assignment process.

B-1- Doing algorithm for T_p - T_r. finally set T_r=0.

C- Send best-value and queue information to assignment process. (*MPI_Gatther*)

C-1- Checking *MYQueueLength* (process queue length).

If (*MYQueueLength*==1)

- C-1-1- Doing job for T_w
- E- Receive best-value so far from assignment process by *MPI_Scatter*.

If (*MYQueueLength*>1)

- C-1-2- Doing job for T_w. if in this step *MyQueueLength*< numproc-2 process must pauses. Because may need to send its (numproc-2) sub-problem to other processes (in a rare situation).
- E- Receive *NumberofSend*, receiver processes labels and general best-value.
- F- If *NumberofSend* is greater than zero; send *NumberofSend* sub-problem(s), to receiver processes that have been determined by assignment process.
- G- Advance forward start pointer of queue NumberofSend room(s).

If (MYQueueLength==0)

- E- Receive best-value so far and *Sender* process label.
- E-1 Check *Sender*
  - If receive Terminate_tag, Final.
  - If Sender==0 means there is no available sub-problem, so goto C.
  - If Sender > 0

F- Receive a sub-problem from *Sender* and insert it in queue.

F-1- By considering receiving time, set T_r.

H- Goto B-1

In per iteration the new value of T_w being calculated for next iteration.

V. Implementation Result

Our algorithm has been coded in the C programming language using the Microsoft Visual C++.NET 2005 compiler and for parallelization MPI library has been used. The source code is available upon any request.

All experiments have been done on 11 Computer with 3.2 GHz Intel Pentium 4 processor and 512 MB of RAM running Microsoft Windows XP Professional Edition SP2. We have done our experiments on the so-called A, B, and P benchmark CVRP instances, which are available in [16].

In all experiments, program has been executed for maximum 30 minutes, and after finishing this time, best-value so far assigned as our answer.

In table-1 results of running program on some famous examples in both sequential and parallel case have been showed. For performing these tests we assign 30 non-central processes and set time variable T_p = 350ms. First row shows the name of example. Second row shows sequential running time for each example. By the way, Row 3 shows parallel running time. Forth row indicates the number of branches have been used for solving corresponding example. In 5th row there are best-values for each example. Every time unit in this table is second. As the previous discussion when a time is equal 1800, means that the 30 minutes deadline for execution has been finished. The results show the performance of the proposed algorithm.

In final step of our examination there is a comparison for value of T_p effects. For these purpose some example has been evaluated by several value of T_p. In the first phase we set T_p equal to small fixed value 100ms. In the next phase we set the value of T_p to a fixed large value 1s. Furthermore in 3rd phase we set T_p a changing value. Namely in primary iterations, value of T_p is equal to 100ms and after each iterations the value of T_p being decreased uniformly. The results of the experiment have been illustrated in table.2. After doing this triple examination, these conclusions have been achieved.

Whereas in primary iterations best-values change quickly, so in these iterations a small value of T_p is a good choice. Because a process with better best-value, can prunes its branch and bound tree faster.

In the secondary steps (especially in big problems), best-values changes becomes less than previous steps. So if T_p has a small value, communication overhead will being very large. Because just repeated best-values have been exchanged between processes and have no profile for parallelization.

In very big problems, assigning a large T_p is an optimized choice, because in these problems solving the sub-problems have much importance than sharing the best-values.

Table 1- A comparison between sequential and parallel execution with 33 processes

Example name	Seq. time	Parallel time	Number of branches	Best-Value
A-n37–k6	50	22	10000	945
A-n39–k5	20	10	6000	829
A-n53-k7	100	30	25000	1013
A-n54-k7	1800	1200	400000	1180
B-n50-k8	1800	1800	500000	1314
B-n66-k9	1800	1800	500000	1321
B-n67-k10	450	150	65000	375
B-n78-k10	1800	1800	450000	1226
P-n50-k8	1800	1800	500000	630
P-n55-k7	1800	1052	35000	569.5
P-n76-k4	50	18	9000	599.2
P-n76-k5	1800	1750	450000	3124
P-n101-k4	69	25	11000	691.2
P-n50-k10	1800	1800	48000	697

Table 2- Effect of T_p values on execution time

Example name	Phase 1	Phase 2	Phase 3
A-n37–k6	22	24	21
A-n39–k5	11	12	11
A-n53-k7	35	31	30
A-n54-k7	1250	1220	1190
B-n67-k10	154	154	149
B-n78-k10	1800	1800	1800
P-n55-k7	1100	1080	1050
P-n76-k4	20	22	19
P-n76-k5	1800	1800	1740
P-n101-k4	27	28	25
P-n50-k10	1800	1800	1800

VI. Conclusion

In this paper a new parallel branch and bound algorithm has been proposed. This algorithm instead of using shared-memory uses a multi-computer environment. A decentralized load balancing method has been used for this algorithm. Also shows that by revising existed algorithms can archive a good performance and lowers communication between processes. This algorithm is implemented for famous Capacitated Vehicle Routing Problem (CVRP). And the experimental results show the efficiency of this algorithm.

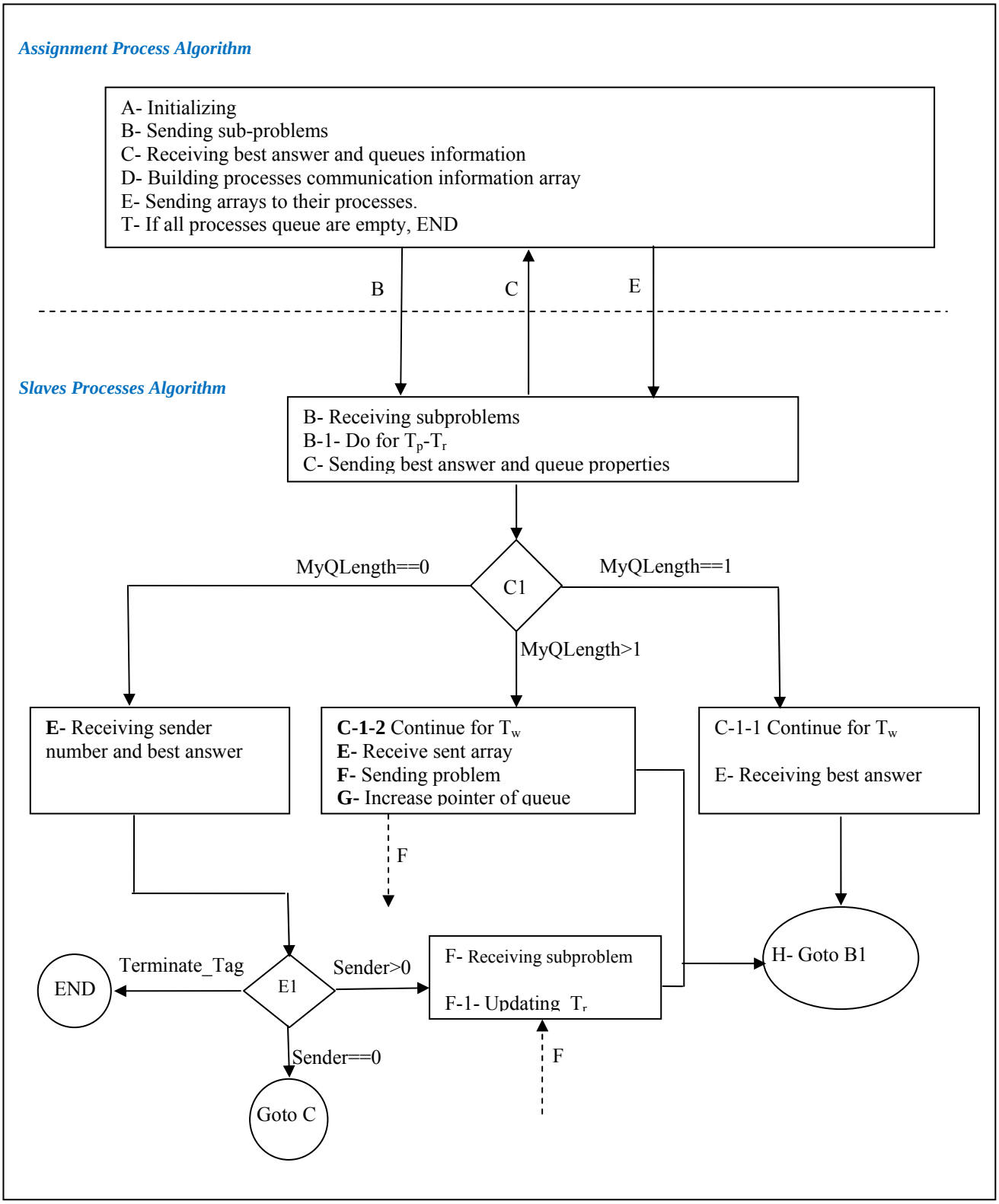

Fig 1- algorithm flowchart

ACKNOWLEDGMENT

The Authors thanks ITRC (Iranian Telecommunication Research Center) for their financial support. And thanks Dr. K. Ziarati for him guidance.

REFERENCES

[1] Hamdy A. Taha, "Operations Research: An Introduction", 6th Edition, Prentice Hall, 2003.
[2] Frederick S. Hillier, and Gerald J. Lieberman, Introduction to Operation Research, 7th Edition, McGraw-Hill, 2002.
[3] Mokhtar S. Bazaraa, J. John Jarvis, and Hanif D. Sherali, Linear Programming and Network Flows, 2nd Edition, Wiley, 1990.
[4] Laurence A. Wolsey, and George L. Nemhauser, Integer and Combinatorial Optimization, 1st Edition, Wiley-Interscience, 1999.
[5] G.B. Dantzig, and R.H. Ramser, The truck dispatching problem, Management Science 6 (1959) 80.
[6] D. Applegate, R.E. Bixby, V. Chv´atal, and W. Cook, Finding cuts in the TSP (A preliminary report), Tech. Rep. 95-05, DIMACS, Rutgers University, New Brunswick, NJ 08903, 1995.
[7] R. Baldick, A Randomized Heuristic for Inequality-Constrained Mixed-Integer Programming, Tech. rep., Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, 1992.
[8] E.M.L. Beale, Branch and Bound Methods for Mathematical Programming System, Annals of Discrete Mathematics 5 (1979), 201–219.
[9] J.A. Tomlin, An Improved Branch and Bound Method for Integer programming, Operations Research 19 (1971), 1070–1075.
[10] R. Bixby, W. Cook, A. Cox, and E.K. Lee, Parallel mixed Integer Programming, Rice University Center for Research on Parallel Computation Research Monograph CRPC-TR95554, 1995.
[11] R. Correa, and A. Ferreira, Parallel best-first branch and bound in discrete optimization: a framework, Center for Discrete Mathematics and Theoretical Computer Science Technical Report 95-03.
[12] A. Grama, and V. Kumar, Parallel search algorithms for discrete optimization problems, ORSA Journal on Computing 7 (1995) 365.
[13] G. Mitra, I. Hai, and M.T. Hajian, A distributed processing algorithm for solving integer programs using a cluster of workstations, Parallel Computing 23 (1997) p-733.
[14] T.K. Ralphs, Parallel branch and cut for capacitated vehicle routing, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, USA, 2002.
[15] Mark Snir, Stive Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra, "MPI: The Complete Reference", The MIT Press, 1996.
[16] http://neo.lcc.uma.es/radi-aeb/WebVRP; (accessed 1st May 2006), Examples and Benchmark for CVRP.
[17] LINGO Optimization Modeling Language, College of Engineering, North Carolina State University, Raleigh, NC 27695.