A Scalable Fully Distributed Parallel Branch & Bound Algorithm on PVM cluster

Chuan Yi Tang, Solomon K.C. Wu, Chee Kane Chang

Department of Computer Science

National Tsing Hua University

Hsinchu, Taiwan, 30043, Republic of China

Correspondence Address:

Prof. Chuan-Yi Tang

Department of Computer Science

National Tsing Hua University

Hsinchu, Taiwan, 30043, Republic of China

Phone: 886-35-731077

Fax: 886-35-723694

E-mail: cytang@cs.nthu.edu.tw

A Scalable Fully Distributed Parallel Branch & Bound Algorithm on PVM cluster

Chuan Yi Tang, Solomon K.C. Wu, Chee Kane Chang

Department of Computer Science
National Tsing Hua University
Hsinchu, 30043
Taiwan, R.O.C.

Correspondence e-mail address: cytang@cs.nthu.edu.tw

Abstract

The branch & bound is an important design strategy of algorithm to solve NP-complete combinatorial optimization problems. In this paper, we present a best-first branch & bound scheme for parallel computers to solve the traveling salesperson problem.

Load-imbalancement is usually the main reason that degrades the performance of a parallel program. In branch & bound algorithms, because it is impossible to predict the size of each branch beforehand, the workload has to be balanced again and again at run-time. In this paper, we propose a dynamic load-balancing algorithm, which is much simpler and easier to implement, in comparison with [17], but still equally efficient.

To show the performance of our scheme, we solve the traveling salesperson problems on a PVM/cluster. Using 16 workstations, we are able to achieve a speedup 16.811 for sixty-city problem sizes. These results verify that our scheme is efficient.

Keywords: Parallel computing, dynamic load-balancing, branch & bound, traveling salesman problem, PVM.

1 Introduction

The branch & bound strategy is widely used to design algorithms for solving NP-complete problems. A survey for this strategy can be found in [9]. The branch & bound strategy, due to its characteristics, is naturally suitable for parallelization. Since a branch & bound strategy is a kind of tree searching technique, we can let several processors simultaneously search through the different parts of the search tree. Much research about parallel branch & bound has been done [1, 4, 7, 8, 10, 12, 15].

There are two implementation models for parallel branch & bound algorithms: the centralized model and the distributed model. In this paper, we propose a fully distributed balancing scheme for efficient parallelization of the best-first branch & bound algorithm. In this scheme, we try to let all the processors expand nodes which have the same or nearly the same lower bounds. During the expansion, some of the processors will generate nodes which are best among all the processors. The main job of our balancing algorithm is to let these nodes be fairly distributed among the processors and expanded. In [25] we propose a fully distributed nearest-neighbor to achieve this goal. The idea is very simple. Before select a best node for expansion, each processor, say p_i, sends the lower bound of its best node to all its neighbors. On receiving this lower bound, a processor will send its best node to p_i if this lower bound is greater than the one of its best node. Note that this scheme will let the best nodes among the processor be fairly distributed among the processors, but it does not involve much communication, because the message size is very small. Hence search overhead can be expected to decrease with low cost of communication. On PVM, we also adopt this strategy as we adopted it on nCUBE in [25]. Although there is no *neighborhood* relationship on PVM, we assign a "logical" one to them artificially. We will discuss this issue in detail in Section 3.

The remaining of this paper is organized as follows: Section 2 describes the branch & bound algorithm, and how to reduce search overhead is given in Section 3. The performance results of the scheme on the PVM/cluster are reported in Section 4. Section 5 gives the conclusion and future researching direction.

2 Parallel Branch & Bound Algorithms

Our program works in an SPMD (Single Program Multiple Data) stye. It has an initialization step to let every processor derive its branch to work on. Then during processing, for a parallel branch & bound algorithm, it is necessary to broadcast the newly computed UB to other processors such that each processor can prune those nodes with lower bound greater or equal to this UB. This work can be done after step 10. Recall in Section 1 that we need a load balancing algorithm to balance the best node each processor. This work can be done after step 19, because the best lower bound of a processor alters only when it has expanded the best node.

In [25] each node only communicate with its *neighbors* so that there is only $\log n$ communications. Here on PVM environment, we also numbering the workstations in a sequence so that they can also be regarded as connected like a hypercube. Thus we can simulate the algorithm as we did in [25].

When a processor run out of node (after step 20), it sends a ∞, which means it is in the idle state, to its neighbors and enters the idle state. If all the processors enter idle state, the latest idle processor will generate a termination signal and broadcast it to other processors. We adapt the *snapshot termination algorithm* of Huang [6] to do this work. Before receiving the termination signal and output the optimal solution (if any), a processor might receiving a node from its neighbor. If this occurs, the processor should put this node into its priority queue and go to step 5. We'll combine such circumstance with the balance procedure and explain how to implement it in Section 3. These two works (termination detection and node receiving) should be placed after step 20.

Therefore, our parallel branch & bound algorithm should be:

```
1   Create the initial subproblems
2   Insert these subproblems into priority queue PQ
3   Set the upper bound UB to ∞
4   Set the best solution BS to Ø
    Do
5       While PQ not empty do
6           Delete the best node SP from PQ
7           If SP is feasible then
8               If cost of SP < UB then
9                   UB = cost of SP
10                  BS = SP
                    Broadcast UB to all processors
11              EndIf
12          Else
```

Expand P to obtain $SP_1, SP_2, \ldots, SP_r$
For $i = 0$ to r **do**
If $LowerBound(SP_i) < UB$ **then**
Insert SP_i into PQ
EndIf
EndFor
EndIf
Invoke the Balancing Procedure /∗ balance with respect to lower bound ∗/
EndWhile
Send (Inform, ∞) to all neighbors
Invoke Balance Procedure /∗ receive nodes ∗/
Invoke the Snapshot Termination Algorithm
Until detected TERMINATE signal
Output BS,

3 Balancing Algorithm

From the above discussions, one can realize that the balancing of the number of unexamined nodes in each processor is not suitable for a parallel best-first branch & bound algorithm and that using the lower bound as the balancing criterion is better.

Although on our PVM environment, all workstations are connected on the Ethernet bus, and thus they don't have the *neighborhood* relationship as in [25], we still wanna give them a "logical" neighborhood relationship so that the quantity of communication can be reduced.

The following is our balancing algorithm. In this algorithm, the message comes in the form (A, B), where A represents the message type, and B is the message. There are three message types, *NewUpperBound*, *SubProblem* and *Inform*. They represent the newly computed bound, the subproblem (node), and the information which contains the lower bound of the second best node in a processor, respectively. The function *LowerBound*() returns the lower bound of a node, and the symbol *MySecondBest* represents the node having the second smallest lower bound in the processor's priority queue.

Procedure BALANCE
While receiving (NewUpperBound, NU) **do If** $NU < UB$ **then** $UB = NU$

While receiving (SubProblem, SP) **do**

```
3         If LowerBound(SP) < UB then insert SP to PQ
4     EndWhile

5     For i = 0 to dims - 1 do HasInform[i] = False

6     While receiving (Inform, LB) from neighbor i do
7         BestNode[i].LB = LB
8         HasInform[i] = True
9     EndWhile

10    Sort the array BestNode.LB in descending order
11    For i = 0 to dims - 1 do
12        If HasInform[i] then
13            If BestNode[i].LB > LowerBound(MySecondBestNode) then
14                Delete MySecondBestNode SP from PQ
15                Send (SubProblem, SP) to neighbor i
16                BestNode[i].LB = LowerBound(SP)
17            EndIf
18        EndIf
19    EndFor

20    Send (Inform, LowerBound(MySecondBestNode)) to all neighbors
EndProcedure
```

On receiving a new upper bound, the processor compares it with the current upper bound. If this new upper bound is less than the old one, it replaces the old one (step 1). When a node coming, the processor inserts it into its priority queue PQ if the lower bound of this node does not excess the current upper bound (steps 2–4). Steps 5–9 update the latest information which represent the second best lower bounds of the processor's neighbors. Then a loop is performed to send a node to each neighbor i if the neighbor has a lower bound greater than the lower bound of the second best node in the processor's priority queue (steps 10–19). Finally, the processor sends its lower bound of the second best node to all its neighbors.

The idea behind this algorithm is simple. Every time when a processor performed the branching rule, it will apply this algorithm. The key point is in steps 10–20. After receiving the lower bound information or additional tasks from each neighbor, the processor sorts and checks these lower bounds and sends a node to those neighbors which have lower bounds greater than the one of the second best node in the processor's priority queue. After checking all the neighbors' lower bounds, the processor multi-casts the current lower bound of its second best node to all its neighbors.

Here, we sort the lower bounds because a neighbor which has the larger lower bound is more likely to produce the redundant nodes than those of smaller lower bounds. Besides, the reason why we use the second best node instead of the first best node for checking the lower bound and sending it to other processors is that a processor would be preferable to expand the best node instead of sending it to another processor for expansion, which will cause additional communication.

4 Experimental Results

Here we report the experimental results of our algorithm. All the experiments were done on IBM SP2 workstations running the traveling salesperson problem. This PVM/cluster is composed of 16 workstations.

In our following description, the *runtime* is the execution time excluding the I/O time for data loading and solution outputing. The notation T_1 is used to denote the computation time when we solve the problem with only a single processor. P represents the number of processors and T_P represents the runtime when we parallelize the program onto P processors. We now present our experimental results for the traveling salesman problems below.

4.1 Results

We performed our experiments on different sizes of *asymmetric* traveling salesperson problem, in which the edge costs of in-coming and out-going may be different. All the edges costs are generated randomly with integers which distribute uniformly in the range $[0, 5t]$, where t is the problem size. The problem size (the number of cities) ranges from 30 to 60. For each problem size, we generate 16 data instances for our experiments. The graph for the traveling problems are all fully connected, which means that there is an edge between any two cities. Table 1 shows the average runtime for different sizes of traveling salesperson problems.

problem size	number of processors				
	1	2	4	8	16
30	10.86	6.08	3.23	2.27	1.81
40	97.68	48.57	25.93	13.66	9.01
50	348.43	165.82	82.23	42.58	22.41
60	2335.52	1478.457	728.171	292.10	138.92

Figure 1: The runtime for the asymmetric traveling salesman problems.

As one can see, the runtime increases as the problem size increases, and decrease as the number processor increases. Note also that, for a specific problem size, the runtime decreases nearly by half as we double the number of processors. This gives us the first impression that our results can achieve good efficiency.

Table 2 shows the average speedups for different sizes of traveling salesperson problems. We can see that, using 16 workstations to run the program in parallel, it achieves a speedup of 15.543 for 50-city problems, and a speedup of 16.811 for 60-city problems.

problem size	number of processors				
	1	2	4	8	16
30	1.000	1.786	3.365	4.7686	5.984
40	1.000	2.011	3.767	7.1504	10.836
50	1.000	2.101	4.237	8.1826	15.543
60	1.000	1.580	3.207	7.9956	16.811

Figure 2: The speedups $S = \frac{T_1}{T_p}$ for the asymmetric TSP.

Figure 1 shows that our algorithm approaches to nearly linear speedups for large problems.

5 Conclusions and Future Work

We have presented a best-first branch & bound scheme for parallel computers and applied this scheme to solve the traveling salesperson problem on PVM/Cluster. We demonstrate that this parallel algorithm based upon the lower bound balancing scheme is efficient on a cluster of workstations. When we run it on 16 workstations, it achieves the speedup of 16.811. This phenomenon of superlinear speedup shows the value for parallel processing.

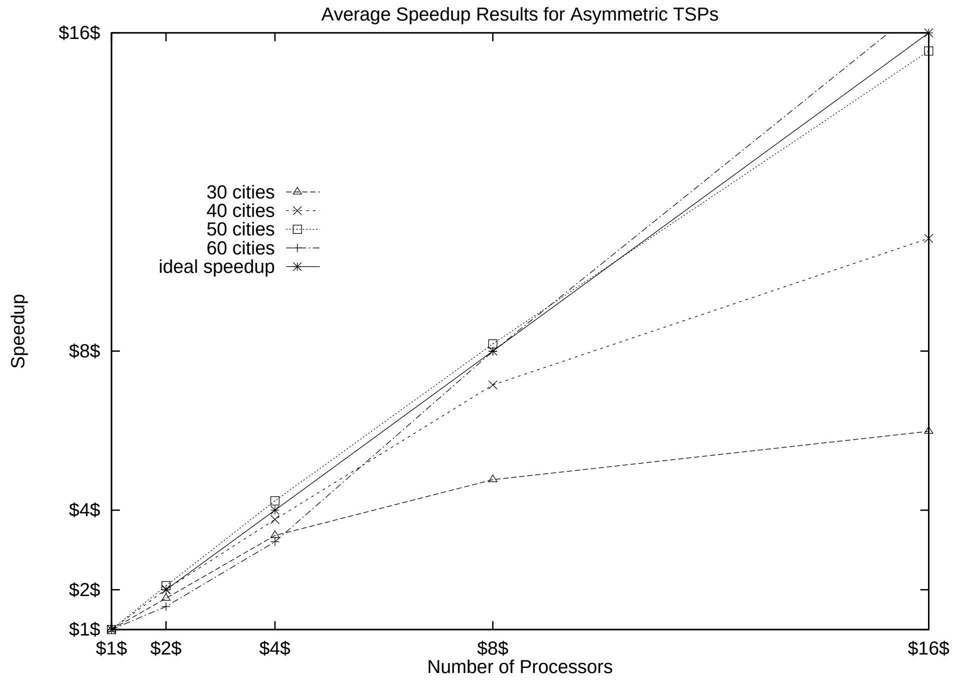

Figure 1: The speedups of different problem sizes.

The superlinear phenomenon looks odd at a first glance. Theorectically it's impossible to be 16.811 times faster if we only use 16 workstations. However, the running time does not count CPU time only. TSP is a large problem which will consume much of memory space. When we run it on a single machine, the large demand of memory will cause the OS to adopt virtual memory management system to handle it. Thus lots of swapping between memeory and disks will take place. When we divide TSP to 16 workstations, each of them only gets a small portion so that their memory can accomodate the program. Therefore, the disk swapping time can be saved. Because disk I/O is always slow, this can save lots of time.

Our lower bound balancing algorithm only gather the information for balancing purpose from a portion of workstations. To reduce the communication overhead, we do not instruct them to exchange information with all other workstations.

The platform we are using now is a homogeneous environment. How to apply our strategy to a heterogeneous environment where machines have different computing power is an interesting direction for research.

References

[1] T. S. Abdelrahman and T. N. Mudge. Parallel Branch and Bound Algorithms on Hypercube Multiprocessors. In *Proc. of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 1492–1499, 1988.

[2] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a Large-scale Traveling Salesman problem. *Operation Research*, 2:393–410, 1954.

[3] O. Eriksen. A Termination Detection Protocol and Its Formal Verification. *Journal of the Parallel Distributed Computing*, 5:82–91, 1988.

[4] E. W. Felten. Best-first Branch-and-Bound on Hypercube. In *Proc. of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 1500–1504, 1988.

[5] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Inc., 1978.

[6] S.-T. Huang. Termination Detection by Using Distributed Snapshots. *Information Processing Letter*, 32:113–119, 1989.

[7] V. K. Janakiram, D. P. Agrawal., and R. Mehrota. A Randomized Parallel Branch and Bound Algorithm. In *Proc. of the International Conference on Parallel processing*, pages 69–75, 1988.

[8] R. M. Karp and Y. Zhang. A Randomized Parallel Branch and Bound Procedure. In *Proc. of the ACM Symposium on Teory of Computing*, pages 290–300, 1988.

[9] E. L. Lawler and D. E. Wood. Branch and Bound Methods: A survey. *Operation Research*, 14:699–719, 1966.

[10] Y. L. Liou and C. Y. Tang. Distributed Branch-and-Bound Algorithms on the Single-Bus Brocast Communication Model. In *Proc. of International Computer Symposium*, pages 1391–1396, 1988.

[11] Kurt Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness.* Spring-Verlag, Berlin, 1984.

[12] R. P. Pargas and E. D. Wooster. Branch-and-bound algorithms on a hypercube. In *Proc. of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 1514–1519, 1988.

[13] M. J. Quinn. Analysis and Implementation of Branch-and-Bound Algorithms on a Hypercube Multicomputer. *IEEE Trans.on Computers*, 39(3):384–387, March 1990.

[14] S. P. Rana. A Distributed Solution to the Distributed Termination Probelm. *Information letter*, 17:43–46, 1983.

[15] J. M. Troya and M. Ortega. A Study of Parallel Branch-and-Bound Algorithms with Best Bound First Search. *Parallel Computing*, pages 121–126, November 1989.

[16] B. W. Wah, G. J. Li, and C. F. Yu. Multiprocessing of Combinatorial Search Problems. *Computer*, 18(9):93–108, June 1985.

[17] R. Lüling and B. Monien. Load Balancing for Distributed Branch & Bound Algorithms. *Proc of Int. Parallel Processing Symposium (IPPS)*, 1992, pages 543–549.

[18] R. Lüling, B. Monien, M. Räcke, and S. Tschöke. Efficient Parallelization of a Branch & Bound Algorithms for the Symmetric Traveling Salesman Problem. *European Workshop on Parallel Computing (EWPC), 1992, Barcelona.*

[19] R. Lüling, B. Monien, and F. Ramme. Load Balancing in Large Networks: A Comparative Study. *Proc. of IEEE Symposium on Parallel and Distributed Processing*, 1991, pages 686–689.

[20] Olivier Vornberger. Implementing Branch-and-Bound in A Ring of Processors. *Technical Report 29, Univ. of Paderborn, FRG*, 1986.

[21] Benjamin W. Wah and Y. W. Eva Ma. Manip - A Multicomputer architecture for solving combinatorial extremum-search problems. *IEEE Transactions on Computers*, c-33, May 1984.

[22] Vipin Kumar and Ananth Y. Grama and V.Nageshwara Rao. Scalable Load Balancing Techniques for Parallel Computers. *Technical Report, Tech Report 91-55, Computer Science Department, University of Minnesota*, 1991.

[23] Vipin Kumar, K. Ramesh, and V. Nageshwara Rao. Parallel Best-First Search of State-Space Graph: Asummary of Results. *Proceeedings of the 1988 National Conference on Artificial Intelligence*, August 1988, pages 122-126.

[24] Vipin Kumar, V.N. Rao. Parallel depth-first search, part II : Analysis. *International Journal of Paralle Programming*, 16(6):501-519, 1987

[25] C. Y. Tang, Solomon K.C. Wu, Y. T. Tsai, C. K. Chang. A Scalable Fully Distributed Parallel Branch and Bound Algorithm on nCUBE *High Performance Computing - Asia*, September 1995