TRAVELLING SALESMAN PROBLEM - PARALLEL ALGORITHMS

Ewa Dudek-Dyduch[1]
Tadeusz Dyduch[2]

[1] *Academy of Mining and Metallurgy, Institute of Automatics, al.Mickiewicza 30, PL 30-059 Krakow, edd@earth.ia.agh.edu.pl*
[2] *Cracow Univ. of Technology, Sect.of Descr.Geometry and Eng.Graphics ul. Warszawska 24, PL 31-155 Krakow, tdyduch@oeto.pk.edu.pl*

Abstract: The paper presents parallelization of Branch and Bound (B & B) algorithm for Travelling Salesman Problem. The algorithm generates and examines a binary decision tree according to back track strategy, with use of recursive procedure. Two different parallel computing methods based on another constructions of decision trees at the first stages of searching are presented. The results of computer experiments carried out with use of Parallel Virtual Machine software are given.

Key words: parallel computation, concurrent searches, branch and bound algorithms, travelling salesman problem, manufacturing systems, scheduling algorithms, backtracking, binary search trees, decomposition methods, optimal search techniques.

1 INTRODUCTION

The paper presents parallel algorithms for Travelling Salesman Problem (TSP). The algorithms are based on the described in (Syslo *et al.*, 1983) sequential branch and bound (B&B) algorithm, that is characterised shortly in the next section. The devised parallel algorithms have been implemented in language C and have been examined with use of package PVM (Parallel Virtual Machine) (Geist, *et al.*, 1993). The results of computer experiments are given and discussed in the paper.

The travelling salesman problem is as follows. A travelling salesman must visit every city in his territory exactly once and then return to his starting point. Given the cost of travel between all pairs of cities, how should he plan his itinerary so that he visits each city exactly once and so that the total cost of his entire tour is minimum? In network theory terms, the problem is to find a minimum-weight cycle of length n in a given weighted graph G of n nodes. The matrix A represents the costs of travel (or distances) between the pairs of cities; i.e. $a_{i,j}$ denotes the cost of travel between i-th city and j-th one (weight of (i, j) arc). The TSP is a classic problem in combinatorial optimisation. It arises in numerous applications. It arises also in manufacturing. Consider a machine shop that is to produce *n* different items, requiring retooling whenever the production item is changed. The cost of retooling between items is given. Finding an optimal production sequence is simply the TSP. The TSP belongs to the class of NP-hard problems (in the strong sense), i.e. the class of those problems for which no polynomial-time algorithm has been found. When the size of any NP-hard problem increases, the computing time increases exponentially and the exact solution cannot be found in reasonable time. Due to it, many of NP-hard problems that are solved in real life should be considered as large scale ones. The size of real life TSP causes that the problem should be also considered as large scale one. This is the reason for which parallel algorithms for it arouse interest of researchers. Application of package PVM enables one to distribute computing among many computers that work in parallel. Number of the computers depends on the size of TSP. The time of solving the problem is much shorter. The parallel algorithms may be also computed with use of multi-processor computer with distributed memory.

Different sequential B&B algorithms have been presented for TSP. Basis for the proposed in the paper parallelization is algorithm described in the next section. As it is known, B&B method lies in constructing a decision tree, the nodes of which correspond to the sets of admissible solutions. At each step the chosen subset of solutions is partitioned into some subsets. After this branching, the lower bounds are computed for new subsets. By way of eliminating the nodes, the successors of which do not contain better solution than the best known one, the number of calculations is reduced.

2. SEQUENTIAL B&B ALGORITHM FOR TRAVELLING SALESMAN PROBLEM

The B&B algorithm for TSP is given in (Syslo *et al.*, 1983). It belongs to the class of so-called constructing algorithms (Dudek-Dyduch, 1992a, 1992b). The partial solution is being augmented along the path of decision tree by means of adding new arcs that constitute a solution. Admissible solutions correspond to leafs. A subset of solutions associated with the decision tree node is determined by the subset of arcs that certainly will belong to generated solutions and the subset of arcs that certainly will not belong. The branching rule consists in partitioning the solution subset into two subsets: solutions that contain a specific arc (i, j) and those that do not. The specific arc is chosen from the minimal elements of the all active rows, in such a way so that the increase of lower bound is maximal, when the arc is rejected. The decision tree is generated and examined with use of backtrack strategy.

The initial lower bound is computed on basis of matrix *A*. Then the matrix is being transformed (reduced) when generating the next nodes, so that with every node there is associated respectively reduced matrix. The lower bounds for the next nodes are computed with use of the reduced matrixes. Detailed description of matrix reduction and construction of lower bounds is not needed for presentation of parallelization, so it will be omitted here.

The heart of the branch & bound TSP algorithm according to (Syslo *et al.*, 1983). is recursive procedure EXPLORE. The procedure considers a given node of the decision tree and searches for better solution. It records the best found solution and its total cost. *A* is the matrix associated with the considered node. Let us denote:

REDUCE(*A*) - procedure that reduces matrix *A* and computes the increase of lower bound,

UNREDUCE(*A*) - procedure that restores the previous values of matrix *A*,

BESTEDGE(*A, r, c, most*) - procedure that computes the arc (r, c) that is the best for branching,

most - increase of lower bound that is caused by rejection of the arc (r, c),
cost - lower bound for the considered node,
edges - number of arcs included in the partial solution,
tweight - total cost of the best complete solution obtained so far

```
 procedure EXPLORE(edges, cost, A);
 begin
    cost ← cost + REDUCE(A);
    if cost < tweight then
       if edges = n - 2 then
       begin
        add the last two arcs;
        tweight ← cost;
        record the new solution;
*       broadcast the new cost value;
       end else
       begin
        BESTEDGE(A, r, c, most); {find the best
        edge for branching}
        lowerbound ← cost + most;
        prevent cycle;
        newA ← A - (row r) - (column c); {delete
        the row r and column c}
        EXPLORE(edges + 1, cost, newA); {left
        subtree}
        A ← newA + (row r) + (column c); {
        restore A by adding column c and row r}
        if lowerbound < tweight then
           begin {explore right subtree}
             $a_{r,c} \leftarrow \infty$;
             EXPLORE(edges,cost, A);
             $a_{r,c} \leftarrow 0$;
           end
       end;
    UNREDUCE(A);
 *  read message on new cost value;
 end   /*end of procedure EXPLORE*/
```

Backtrack strategy guarantees the ordered enumeration of solution subsets. It is realised with use of recursive procedure, so there is no need for storing partial solutions, what is very advantageous. On the other hand, however, one can not use another selection rule for node branching, what might improve effectiveness of the B&B algorithm. The conception of parallel algorithm that uses selection rules based on minimum lower bound is presented in the earlier paper of the authors (Dyduch and Dudek-Dyduch, 1994b).

3. PARALLEL B&B ALGORITHM FOR TSP

In this section, the conceptions of parallelization of the described algorithm will be presented. Some results of earlier research connected with the parallelization are presented in (Dyduch and Dudek-Dyduch, 1994a, 1994b).

We assume that a computer consists of N + 1 processors 0, 1,..N. The function of processors 1,2..N is to analyse in parallel suitable parts of decision tree, while the role of 0 processor (a main processor) consists in "managing". Some essential modification has been introduced to the previous algorithm in order to execute it in parallel. They result from the following requirements:

a) the loads of parallel processors should be balanced,

b) an enough good solution should be found possibly soon,

c) possibly many solutions' subsets should be eliminated.

It is easy to notice that the structure of decision tree described in the previous section is not suitable for efficient decomposition. (There is no possibility to assure the balanced loads of parallel processors.) Because of that, the main modifications refer to construction of the decision tree.

The paper presents two parallel algorithms. They differ mainly in:

- branching rules that are applied for the root and initial levels of the decision tree,

- manner of assigning the parts of decision tree to the particular processors.

The algorithms comply with the mentioned requirements a) - c) to different degrees.

Because every node of a decision tree may be considered as the subset of solution or the subproblem (to find the best solution that belongs to the solution subset) or the root of decision subtree, these notions will be used interchangeably. A subtree the root of which is a node of the first level (second level) of the decision tree will be called first stage (second stage) subtree.

3.1 Tasks distribution according to basic distribution row.

In order to perform this algorithm in parallel, some its essential modifications must be introduced.

The first one consists in assuming another branching rule applied for the root and the first level of the decision tree, so there is another construction of the decision tree. Now, the solution set associated with the decision tree root is partitioned into n-1 mutually disjoint subsets, instead of the two subsets, as it was previously. Then the subsets of the first level are partitioned each into n-2 mutually disjoint subsets, and only the subsets of the second and the further levels are partitioned into two sets according to the previous algorithm branching rule.

The second modification consists in the fact that only the second stage subtrees are generated and examined with use of recursive procedure EXPLORE. The construction of the modified decision tree and way of dynamic assignment of subproblems to free processors are described below.

Branching of the decision tree root is determined by the selected node of the graph G and all arcs outgoing from the node. Each of these arcs defines one node of the first level of the decision tree, as it is the element of each solution that belongs to the subset associated with the node. Costs of these arcs are given by elements of the same row of the matrix A . This row will be called a basic distribution row. The main processor determines the basic distribution row (i.e. determines the branching of the decision tree root) and sets its elements in order according to increasing their values. As there is one to one correspondence between the first level nodes and elements of the basic distribution row, the nodes are also ordered. The nodes (subproblems) of the first level are being assigned, according to this order, to the free processors 1, 2.. N . (We will say for short that the elements of the distribution row are assigned to particular processors.) Thus, each processor initially analyses a first stage subtree named here its basic subtree. Nodes of the subtree are examined with use of backtrack strategy.

Because N admissible solutions are computed in parallel, the total cost of the best found solution is, most likely, much better (lower) than in case of sequential algorithm executed by one processor. This best value, broadcasted to all the processors, is compared with the lower bounds of solution subsets, so that more subsets (nodes) are expected to be discarded than in the sequential version of algorithm. It is so, provided that the total costs for most of Hamilton cycles are essentially different.

The branching rule for the first level nodes has been devised so that loads of particular processors 1,2,..N may be balanced. All n-2 direct successors of a first level node are determined by arcs outgoing from the selected node of the graph G. This node is different from that assumed at 0 level. Thus, the arcs assigned to the path going from the decision tree root to a second level node, belong to each solution of subset associated with this node. Obviously, they cannot constitute a cycle. In order to determine the branching for a first level node, the main processor indicates for it the so-called local distribution row, analogously as for the root. Every local distribution row is ordered analogously as the basic distribution row (i.e. according to increasing elements). There is one to one correspondence between the elements of the local distribution row for the node and the direct successors of the node (similarly as for the basic distribution row and the first level nodes). Because of that, we will say for short, that the element of distribution row is examined, chosen or assigned to processor, having in mind that the corresponding node is examined, chosen or assigned to processor.

Consider situation when one of processors has finished examination of its basic subtree and there are no first stage subtree that may be assigned to it. The processor should start to examine a part of basic subtree assigned to another processor, and analysis of which is least advanced. In order to decide, a part of which subtree should be examined, states of local distribution rows are tested by the main processor. A state of the local distribution row of a node is defined as the number of examined elements of the row (number of direct successors that have been already generated). The state determines how far the examination of the basic subtree is advanced.

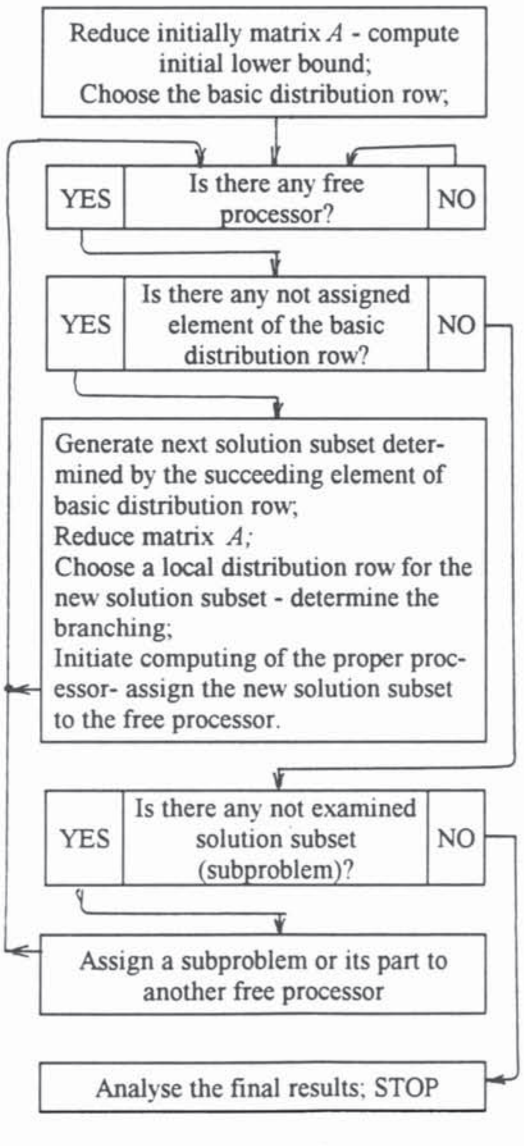
Reduce initially matrix A - compute initial lower bound;
Choose the basic distribution row;
YES
Is there any free processor?
NO
YES
Is there any not assigned element of the basic distribution row?
NO
Generate next solution subset determined by the succeeding element of basic distribution row;
Reduce matrix A;
Choose a local distribution row for the new solution subset - determine the branching;
Initiate computing of the proper processor- assign the new solution subset to the free processor.
YES
Is there any not examined solution subset (subproblem)?
NO
Assign a subproblem or its part to another free processor
Analyse the final results; STOP

Fig. 1 Block schema of the main processor computing

The basic distribution row as well as the local distribution rows is computed by the main processor.

Because every basic subtree is examined according to back track strategy but only the second stage subtrees are generated and tested with use of recursive procedure, there is possible to assign dynamically parts of tha basic subtrees to the free processors. Notice that it is impossible if recursive procedure is applied for a first stage subtree, as a recursive procedure uses a stack.

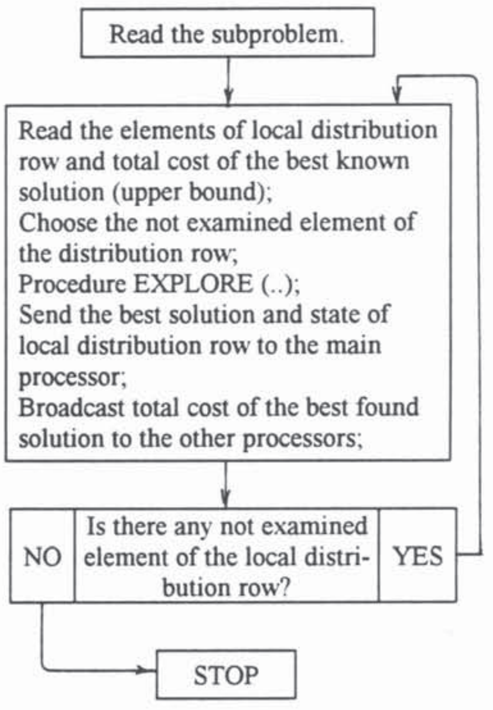
Read the subproblem.
Read the elements of local distribution row and total cost of the best known solution (upper bound);
Choose the not examined element of the distribution row;
Procedure EXPLORE (..);
Send the best solution and state of local distribution row to the main processor;
Broadcast total cost of the best found solution to the other processors;
NO
Is there any not examined element of the local distribution row?
YES
STOP

Fig. 2. Block schema for each of parallel processors computing.

3.2. Parallelization with fixed initial subsets of arcs

A lot of experiments have been carried out with the described above parallel algorithm. In case of large TSP problems, it appeared that when the matrix A was not reduced yet or was reduced to very little extent, the chosen with use of the BESTEDGE strategy (procedure) arcs constituted parts of optimal solution with probability close to 1. This result had impact on working out another parallel algorithm for TSP problem.

In this new algorithm some number of the best arcs are selected at the beginning with use of BESTEDGE procedure. Then all combinations of these arcs are created, i.e. the subsets containing some fixed and some rejected arcs. Each subset defines an initial part of any solution. The generated, in this way, subsets are assigned to the particular processors that work in parallel. Each processor computes solutions the initial part of which is the same and which do not contain the rejected initially arcs. The processors realise identical programs, driven by data red from files. The programs have ability to communicate and exchange the best values of the cost variable, as in the previous parallel algorithm.

The novelty of these programs is BABTSP2 procedure. It prepares the initial data and starts EXPLORE recursive procedure for the first time. The role of BABTSP2 procedure consists in:

- replacing in the matrix A the weights of the rejected arcs by the infinity symbol,
- removing the rows and columns of matrix A, that correspond to the fixed elements,
- preventing subcycles,
- simulating the proper level of recurrence.

Then it starts EXPLORE.

The following denotation is assumed.

cost - lower bound for the considered node,
edges - number of arcs included in the partial solution,
rowA, colA - lists of actual rows and columns of *A* weights matrix.
inf - large number, infinity symbol;

```
procedure BABTSP2();
begin
    rowA ← all rows;
    colA ← all columns;
    cost ← 0;
    edges ← 0;
    read loff, lon, xoff, yoff, xon, yon;
        for i ← 1 to loff
            A[xoff[i]][yoff[i]] ← inf;
        for i ← 1 to lon
        begin
            cost ← cost + A[xon[i]][yon[i]];
            edges ← edges +1;
            rowA ← rowA - (row xon[i]);
            colA ← colA - (col yon[i]);
            A[yon[i]][xon[i]] ← inf;
            prevent subcycle;
        end;
        EXPLORE(edges, cost, rowA, colA);
    write down the best solution's path;
end
```

4. RESULTS OF EXPERIMENTS AND CONCLUSIONS

The two parallel algorithms have been implemented in language C and have been examined with use of package PVM (Parallel Virtual Machine) (Geist, *et al.*, 1993). Table 1 presents results of experiments carried out for 10 instances of TSP and comparison of the two parallel algorithms.

Table 1. Results of experiments with 10 TSP instances of random data

VALUE/NUMBER	min	max.	mean
Total cost of optimal solution	108	162	130
Initial lower bound	72	140	103
Optimal solution total cost minus initial lower bound	18	36	27
Experiments with sequential program.			
Difference between total cost for the first and optimal solution	0	22	8
Number of EXPLORE calls needed for optimal solution	39	8 789	1 978
Total number of EXPLORE calls	507	1436 2	4 214
Experiments with first parallel algorithm "basic distribution row"			
Difference between total cost for the first and optimal solution	0	19	4
Number of EXPLORE calls needed for optimal solution	39	4 372	827
Maximal number of EXPLORE calls for one processor	258	9 213	2 219
Experiments with second parallel algorithm "combinations of the best arcs"			
Difference between total cost for the first and optimal solution	0	21	7
Number of EXPLORE calls needed for optimal solution	39	7 232	1 828
Maximal number of EXPLORE calls for one processor	258	6 173	1 696

The network *G* of each instance had 40 nodes and randomly generated weights of arcs. The weights were integers from the range [0,99]. The networks were presented in form of matrixes. For each matrix, the instance was first solved by means of sequential algorithm that used recursive procedure EXPLORE and was implemented in C language. Then it was solved by the two parallel algorithms, using 5 processors with shared memory. Because different computers were used, the number of procedure EXPLORE calls was chosen as the more informative factor.

The experiments have confirmed the expectation referring to effectiveness of the proposed parallel algorithms. For most experiments, the number of iterations performed by each processor 1,2,..N was considerably fewer than the number of iterations in sequential algorithm.

Comparison between the two devised methods for parallel computing of TSP problem showed that the second method is more effective. The total number of EXPLORE procedure calls for the whole problem was much less in this case as well as the maximal number of EXPLORE calls during solving one subproblem was significantly less.

Application of recursive procedure at the lower stage of the proposed parallel computing methods enables one to save computer memory.

Acknowledgement

The paper has been supported by the grant of KBN No.: 8 T11A 013 08.

REFERENCES

Ben-Ari, M. (1990). *Principles of concurrent and distributed programming.* Prentice-Hall, Englewood Cliffs, NY.

Dudek-Dyduch, E. (1992a). Control of discrete event processes - branch and bound method. *Prepr. of IFAC/ Ifors/Imacs Symposium Large Scale Systems: Theory and Applications, Chinese Association of Automation*, **vol. 2**, pp. 573 - 578.

Dudek-Dyduch, E. (1992b). Optimization of discrete manufacturing processes - branch and bound method. In *Operations Research '92* P. Gritzman, R. Hettich, R. Horst, E. Sachs (Ed.). pp. 19 - 22. Springer-Verlag, Heidelberg.

Dyduch, T., and E. Dudek-Dyduch (1994a). Parallel branch and bound algorithm for travelling salesman problem. In: *Operations Research Proceedings 1994; Selected Papers* Derigs U. Bachem A., Drexl A. (Ed.). pp. 127-132. Springer- Verlag, Heidelberg.

.Dyduch, T., and E. Dudek-Dyduch (1994b). Parallel Branch & Bound algorithm for solving TSP. (in Polish). *Scientific Bulletins of Silesian University, Automatics*, **No. 114**, pp. 61-70.

Geist, Al *et al.* (1993). *Oak Ridge National Laboratory Report.* TM - 12187.

Hernandez, V., and A.M. Vidal (1989). Parallel orthogonal triangularization of a rectangular matrix on a distributed memory multiprocessor. *Working Conf. on Decentralized Systems, Lyon, December 1989.*

Syslo, M., N. Deo and J. Kowalik (1983). Discrete Optimization Algorithms. Prentice-Hall, Englewood Cliffs, NY.