Solving Asymmetric Traveling Salesman Problems Using Dynamic Scheduling on a Heterogeneous Computing System

Janez Brest, Viljem Žumer
University of Maribor
Faculty of Electrical Engineering
and Computer Science
Smetanova 17, 2000 Maribor, Slovenia
janez.brest@uni-mb.si

Janez Žerovnik*
University of Maribor
Faculty of Mechanical Engineering
Smetanova 17, 2000 Maribor, Slovenia

Abstract *A method for dynamic scheduling on a network computing system and an approximation algorithm for solving the asymmetric traveling salesman problem (ATSP) are presented in this paper. Dynamic scheduling was implemented to minimize the application program execution time. Our method decomposes the program workload into computationally homogeneous subtasks, which may be of different size, depending on the current load of each machine in a heterogeneous computing system.*

We present the experimental results of a practical application, the asymmetric traveling salesman problem. All test problems are taken from the literature.

Keywords: heterogeneous computing, dynamic scheduling, optimization method, asymmetric traveling salesman problem

1 Introduction

From the scientific community to the federal government, heterogeneous computing [5] has become an important area of research and interest. Heterogeneous computing includes both parallel [6] and distributed processing.

In general, the goal of heterogeneous computing is to assign each subtask to one of the machines in the system with the objective that the total execution time (computation time and inter-machine communication time) of the application program is minimized [16]. Mapping can be specified statically or determined at runtime by load balancing algorithms.

In heterogeneous computing the structure of a problem may be known, but the structure of the system can change dynamically.

The traveling salesman problem (TSP) is one of the most studied problems in combinatorial optimization [19, 15]. The TSP is simply stated, has practical applications and is representative of a large class of important scientific and engineering problems.

The rest of the paper is organized as follows. In Section 2 a brief overview of the network heterogeneous computing is given. The algorithm for the asymmetric traveling salesman problem is presented in Section 3. In Section 4 task scheduling and load balancing are described. The simple task scheduling scheme and the algorithm for dynamic scheduling on a heterogeneous computing system are presented. The results of the implemented algorithm obtained on optimization problems are presented in Section 5. In Section 6 the concluding remarks are given.

*Also at IMFM/DTCS, Jadranska 19, 1111 Ljubljana, Slovenia.

2 Network Heterogeneous Computing

In network heterogeneous computing (NHC) a number of connected autonomous computers can execute one or more computational tasks concurrently. There are two types of NHC: multi-machine, where the computers used are all identical, and mixed-mode, where the computers used do not need to be identical.

The network layer can provide interconnectivity between computing sites. Communication tools suitable for NHC are the message passing interface (MPI) [2], the parallel virtual machine (PVM), portable programs for parallel processors (P4), etc. [5]. We used MPI.

A network heterogeneous computing system consists of a number of autonomous and independently scheduled computers. A primary objective in many researches dealing with heterogeneous computing is the minimization of the job completion time. In order to focus on the aspects of heterogeneity related to processor time and speed availability we adopt a simple paradigm. We have modeled the computing capacity of each computer with a single parameter: its response time needed for the execution of a small piece of program or task.

Figure 1 shows a part of a heterogeneous computing system that has been used. There are many different operating systems (Solaris, Linux, etc.) on computers connected in a local-area network.

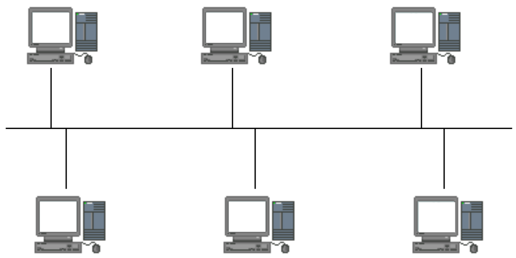

Figure 1: An example of a heterogeneous computing system

Local Area Multicomputer (LAM) [2, 10] is a parallel environment development system of independent computers. It features the MPI programming standard, supported by extensive monitoring and debugging tools.

3 An Application: The Traveling Salesman Problem

3.1 Introduction

The TSP can be viewed as a graph–theory problem if the cities are identified with the nodes of a graph, and the links between the cities are associated with arcs. A weight corresponding to the inter–city distance is assigned to each arc.

An instance of the TSP is given by the distance matrix $D = (d_{ij})$ of dimension $n \times n$; where d_{ij} represents the weight of the edge between city i and city j in $N = \{1, ..., n\}$. If $d_{ij} = d_{ji}$ for every pair i and j in N then the TSP is *symmetric*, otherwise it is *asymmetric*. The TSP satisfies the *triangle inequality* if $d_{ij} \le d_{ik} + d_{kj}$ for all i, j and k in N.

TSP is an example of a NP-Hard problem [12]. There is a simple $O(log\ n)$ approximation algorithm [7], but the existence of a constant factor algorithm is open [13]. Therefore it is reasonable to design approximate algorithms which are able to give near–optimum solutions for NP-Hard problems. Many approximate algorithms were tried on TSP with various success including simulated annealing (SA) [20], threshold accepting [9], genetic algorithms [14], etc. However, it has to be noted that very large instances of TSP were exactly solved by a branch and cut approach [3].

3.2 The Algorithm for ATSP

Local optimization is a well known and widely used general purpose heuristic. A generic procedure for local search optimization is shown in Figure 2 [17]. Step 1 is the *initialization step* which produces the initial solution S. Step 2 is the *optimization step* which attempts to improve the existing solution through the local search.

```
1      find initial solution S
2      while not done do
2.1      transform S into S'
2.2      if S' is better than S then S = S'
3      output S
```

Figure 2: Generic local search procedure

Finding the initial solution may be random or can be obtained by some algorithm. The result in step 2 is usually something like "no improvement for some (long) time". Transformation in 2.1 should be cheap in comparison to finding the initial solution in step 1.

This section presents a heuristic for asymmetric traveling salesman problems [1]. Preliminary results reported in [1] were promising.

The main idea of our heuristic is based on the *cheapest insertion algorithm* [8].

Algorithm RCIA:

1. Start with a tour consisting of a given vertex and self–loop.
2. Randomly choose a vertex not on the tour.
3. Insert this vertex between neighboring vertices on the tour in the cheapest possible way. If the tour is still incomplete, go to step 2.
4. Keep this tour solution, say S.
5. Repeat n^2-times steps 6 through 10.
6. Randomly choose i and j (i, j in $N = \{1, \ldots, n\}$ and $1 \leq i \leq j \leq n$ is satisfied.
7. From the circuit with all vertices remove a path beginning with vertex i through vertex j, and connect the vertex $i-1$ with vertex $j+1$.
8. Randomly choose a vertex from the removed path.
9. Insert this vertex between two neighboring vertices on the tour in the cheapest possible way. If the tour is still incomplete go to step 8.
10. Compare current solution with the solution S. Keep the better one.

First four steps generate an initial circuit. In the main loop – steps 6 through 10 – an optimization is performed. Some vertices are removed from the circuit and later they are randomly reinserted into the circuit once more in the cheapest possible way.

There is an interesting question how many times the optimization should be performed (step 5). We repeated it n^2-times (n is the number of cities). We have no theoretical arguments for this choice; it turned out to give good results in a reasonably short time.

Our heuristic algorithm is very simple and it can be implemented to run on a sequential computer or can be parallelized.

In the rest of this paper we discuss the parallelization of the ATSP algorithm on a heterogeneous computing system.

4 Task Scheduling and Load Balancing

The problem of load partitioning and scheduling in a multiple-processor system has been an area of active and sustained research over the past two decades [5].

The most critical aspect of a task-scheduling algorithm is the strategy used to allocate problems to slaves. Generally, the chosen strategy will represent a compromise between the conflicting requirements for independent operation (to reduce communication costs) and global knowledge of computation state (to improve load balance).

4.1 Master/Slave

Figure 3 illustrates a simple task scheduling scheme that is nevertheless effective for moderate numbers of processors. The central master task is given responsibility for problem allocation. Each slave repeatedly requests and executes a problem from the master. Slaves can also send new tasks to the master for allocation to other slaves.

In the so-called embarrassingly parallel problem, a computation consists of a number

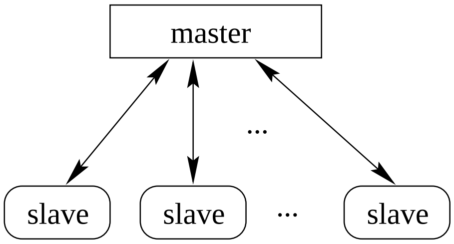

Figure 3: A master/slave scheme

of tasks that can be executed more or less independently, without communication [6]. These problems are usually easy to adapt for parallel execution. The same computation must be performed using a range of different input parameters. The parameter values are read from an input file. The different computation results are written in an output file.

4.2 Arbitrarily Divisible Jobs

The problem of heterogeneous load balancing and task scheduling is examined for two practical workload paradigms [11]: indivisible-task jobs and arbitrarily divisible jobs.

This section deals with load balancing and task scheduling in the context of arbitrarily divisible jobs. The arbitrarily divisible load model [4, 11] can be used in an application where the load consists of a large number of small elements with identical processing requirements. Examples can be found in applications for image and signal processing.

If the execution time per task is constant and each processor has the same computational power, then it is a good idea to decompose available problems into equal-sized sets and allocate one such set to each processor. In other situations, each worker task repeatedly requests parameter values from the input task, computes using these values, and sends the results to the output task. The execution time can vary. The input and the output task cannot expect to receive messages from various workers in any particular order. This nondeterminism affects only the allocation of problems to workers and the ordering of results in the input file, but not the actual results computed.

4.3 Our Method

As mentioned before, we have modeled the computing capacity of each computer with a single parameter: it's response time needed for the execution of the task.

At time $t+2$ the new task size $s_p(t+2)$ for processor p is a function of two previous values of response times $\tau_p(t+1), \tau_p(t)$ and task sizes $s_p(t+1), s_p(t)$:

$$s_p(t{+}2) = f(s_p(t{+}1),\ \tau_p(t{+}1),\ s_p(t),\ \tau_p(t)),\qquad t = 0,\ 1,\ 2,\ \ldots \tag{1}$$

This single parameter includes more aspects of heterogeneity of each computer during the given operating conditions.

4.4 Algorithm

The algorithm for dynamic scheduling on a heterogeneous computing system is presented in this section.

There is one node that represents *master* or *manager*, all other nodes are *slaves* or *workers* (see Fig. 3). Master does not compute, but it

- collects global status information,
- performs the dynamic scheduling algorithm that also distributes tasks into the process unit on a heterogeneous computing system,
- collects the results.

Additionally, master reads data from an input file, distributes data, makes comparisons between temporary solutions to find the best one, etc.

Slaves do not have information about the global status of the heterogeneous system such as system load and program execution progress. Each slave has only two tasks: to make a computation to find a solution, i.e. task execution, and make time measurements, i.e. local information. Local information is sent from slaves to master that has an overview of all activities in the heterogeneous system. Master uses the function (1) to calculate the new task size. Master can store all values $s(t)$ and $\tau(t)$, but because of time locality and space assumption for storing those values, we used only the last two values. Given a parameter $\epsilon > 0$, the function f is defined as follows. Let

$$k = \frac{\frac{s(t+1)}{\tau(t+1)}}{\frac{s(t)}{\tau(t)}}. \qquad (2)$$

If

1. $k < 1 - \epsilon$: the new task size can be increased by factor 2 (the load execution time will be better if the new task size is greater than the current task size),

2. $k > 1 + \epsilon$: the new task size can be decreased by factor 2,

3. $1 - \epsilon \leq k \leq 1 + \epsilon$: task size does not need to be changed.

Parameter ϵ should not be too small or too large. In the first case the function for calculating the new task size is too sensitive to all small changes in the system load. In the second case the dynamic scheduling strategy is rigid. In both cases lower levels of performance were observed. Of course, there are lower and upper limits of minimum and maximum task size. We have obtained the highest performance for $0.05 \leq \epsilon \leq 0.1$.

Algorithms MASTER and SLAVE in pseudo code:

```
Module MASTER
begin
 /* performs task scheduling and load balancing */
 Initialization
 Read data from input files
 Use heuristic method to compute an initial solution
 Send input data to slaves
 Send to slaves:
   the initial solution
   the size of the subtask
 while (program application not finished) do begin
   Receive temporary solution from any slave
   if this solution is the best known then begin
     Receive all data from a slave with the best solution
     Store this temporary best solution and other data
   end
   Receive measurements from slave
   Use current and previous measurements to
     compute new size of the next subtask
   Send the new subtask to slave
 end
 Receive data at the finish of the program application
   from all slaves
 Print solution, results,...
 Finalization
end master

Module SLAVE /* worker */
begin
 /* Slave works and sends measured data to master */
 Initialization
 Receive input data from master
 while (something to do) do begin
   Receive initial solution and size of subtask work and
     compute to find solution /* RCIA is used */
   Send the local best solution and time measurements
     to master
 end
 Send data at the end
 Finalization
end slave
```

5 Results

We have chosen the traveling salesman problem [15, 18] to test the dynamic scheduling on a heterogeneous system.

If computers are heterogeneous they have different powers, so it is difficult to measure the *speedup* of the parallel computation, since it is not clear which computer to use as a reference of the sequential computation. In our measurement we used 41 equal computers.

Another metric that can be useful is *utilization*. The total execution time of program application is the sum of communication time and work time. Utilization can be defined as the ratio between the work time and the total execution time.

Table 1 shows the utilizations of slaves, which were about 0.9 (the total execution time of program application was $700.437s$). Slaves have been mapped to computers connected in a local–area network using Ethernet.

Table 1: Utilization.

Slave	Work time [s]	Utilization [%]
1	646.719	92.33
2	626.861	89.50
3	632.048	90.24
4	627.857	89.64
5	662.287	94.55
6	612.457	87.44
7	659.365	94.14
8	659.221	94.12
9	660.629	94.32

We have tested dynamic scheduling on a homogeneous computing system. The results of the execution times (one node, 41 nodes – computers connected using ATM) and speedup for two asymmetric TSP instances with 358 and 443 are presented in Table 2.

Table 2: Speedup

Problem	T_1 [s]	T_{41} [s]	Speedup $\frac{T_{41}}{T_1}$
rbg358	1359.21	48.08	28.87
rbg443	3591.50	112.80	31.84

Some results obtained in our experiments are shown in Table 3. The second indicates the ATSP instances and the number of cities. The shortest tours are known for all of them and are shown in the third column - Opt. The optimal solution values can be found in TSPLIB library[1] [18, 19]. For each of the problem instances, 30 independent runs were performed. In the last two columns the best tour solutions - min, and the averages of the runs - avg are shown. The results of other 22 problem instances are very similar to the results obtained by sequential algorithms (see [1], page 247).

6 Conclusion

First we present the heuristic used to incorporate the domain knowledge into an approximation algorithm which supports the computation near-optimum solutions for the asymmetric traveling salesman problem (ATSP).

[1] http://softlib.rice.edu/softlib/tsplib/

Table 3: Results for the asymmetric TSP instances

Problem	Opt	min	%	avg	%
p43	5620	5620	100.00	5620.65	99.99
rbg323	1326	1335	99.33	1348.95	98.30
rbg358	1163	1166	99.74	1170.85	99.33
rbg403	2465	2465	100.00	2466.01	99.96
rbg443	2720	2720	100.00	2720.70	99.97

Secondly, the dynamic scheduling on a heterogeneous computing system has been presented. Our method decomposes the program workload into computationally homogeneous subtasks, which may be of different size, depending on the current load of each machine in the heterogeneous computer system.

The feasibility of the proposed heuristic has been demonstrated by presenting the performance results for all ATSP instances from the TSP library containing up to 443 cities. The heuristic is simple but the results are comparable to the results of other approaches described in the literature.

In our future research, we intend to extend the power of the method for dynamic scheduling on a heterogeneous computing system, and to apply it to a wider range of practical problems. The proposed heuristic for ATSP can be used in similar combinatoric problems, such as sequential ordering problem, etc.

References

[1] J. Brest and J. Žerovnik. An approximation algorithm for the asymmetric traveling salesman problem. In N. Callaos, M. Bennamoun, and J. Aguilar, editors, *World Multiconference on System, Cybernetics and Informatics*, volume 3, pages 244–249, July 1997.

[2] Gregory D. Burns, Raja B. Daoud, and James R. Vaigl. LAM: An Open Cluster Environment for MPI. In *Supercomputing Symposium '94*, Toronto, Canada, June 1994.

[3] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Finding Cuts in the TSP (A preliminary report). Technical report, DIMACS Technical Report, 95-05, March, 1995.

[4] Maciej Drozdowski. *Selected Problems of Scheduling Tasks in Multiprocessor Computer System.* PhD thesis, Institute of Computing Science, Poznan University of Technology, 1997.

[5] Mary. M. Eshagian, editor. *Heterogeneous Computing.* Artech House, Inc., Norwood, MA 02062, ISBN 0-89006-552-7, 1996.

[6] Ian Foster. *Designing and Building Parallel Programs.* Addison-Wesley, ISBN 0-201-57594-9, 1995.

[7] A. M. Frieze, G. Galbiati, and F. Maffioli. On the Worst–Case Performace of some Algorithms for the Asymmetric Traveling Salesman Problem. *Networks*, 12:23–39, 1982.

[8] G. A. P. Kindervater, J. K. Lenstra and D. B. Shmoys. The Parallel Complexity of TSP Heuristics. *Journal of Algorithms 10*, pages 249–270, 1983.

[9] G. Dueck, T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys. 90, No.1, 161-175*, 1990.

[10] Gregory D. Burns. The Local Area Multicomputer. In *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications.* ACM Press, March, 1989.

[11] Emile Haddan. Load Balancing and Scheduling in Network Heterogeneous Computing. In Mary. M. Eshagian, editor, *Heterogeneous Computing*, pages 224–276, Norwood, MA 02062, ISBN 0-89006-552-7, 1996. Artech House, Inc.

[12] D. S. Johnson and C. H. Papadimitriou. Performace Guarantees for Heuristics, The Traveling Salesman Problem. *The Traveling Salesman Problem*, pages 145–180, 1985.

[13] Samir Khuller. Problems. *Journal of Algorithms*, 28(1):192–195, July 1998.

[14] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[15] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy, and D. B. Shmoys, editors. *The Traveling Salesman Problem.* John Wiley & Sons Ltd., 1985.

[16] Min Tan, Howard Jay Siegel, John K. Antonio and Yan Alexander Li. Minimizing the Application Execution time Through Schudeling of Subtasks and Communication Traffic in a Heterogeneous Computing system. *IEEE Transaction on Parallel and distributed systems*, Vol. 8(No. 8):173–186, August, 1997.

[17] R. Sosič and G. D. Wilby. Using the Quality-Time Tradeoff in Local Optimization. In *Proceedings of IEEE Second ANZIIS Conference, Brisbane*, pages 253–257, Dec, 1994.

[18] G. Reinelt. TSPLIB - A traveling salesman problem library. Technical Report 250, Universität Augsburg, Institut für Mathematik, Augsburg, 1990.

[19] Gerhard Reinelt. TSPLIB— A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[20] S. Kirkpatrick, C. D. Jr. Gelatt, M. P. Vecchi. Optimization by Simulated Annealing. *Science 220*, (No. 4598):671–680, 1983.