Proceedings, W. E. Nagel, W. V. Walter, and W. Lehner (Eds.): Euro-Par 2006, Parallel Processing, 12th International Euro-Par Conference, Springer LNCS 4128, Aug. 29 - Sep. 1, Dresden, Germany, 2006.

Load Balanced Parallel Simulated Annealing on a Cluster of SMP Nodes

Agnieszka Debudaj-Grabysz[1] and Rolf Rabenseifner[2]

[1] Silesian University of Technology, Department of Computer Science
Akademicka 16, 44-100 Gliwice, Poland
agrabysz@polsl.pl,
[2] High-Performance Computing-Center (HLRS), University of Stuttgart
Nobelstr. 19, D-70550 Stuttgart, Germany
rabenseifner@hlrs.de, www.hlrs.de/people/rabenseifner

Abstract. The paper focuses on a parallel implementation of a simulated annealing algorithm. In order to take advantage of the properties of modern clustered SMP architectures a hybrid method using a combination of OpenMP nested in MPI is advocated. The development of the reference implementation is proposed. Furthermore, a few load balancing strategies are introduced: time scheduling at the annealing process level, clustering at the basic annealing step level and suspending—inside of the basic annealing step. The application of the algorithm to VRPTW—a generally accepted benchmark problem—is used to illustrate their positive influence on execution time and the quality of results.

Key words: Simulated annealing, parallel processing, load balancing, MPI, OpenMP, hybrid parallelization.

1 Introduction

The paper presents a time scheduled algorithm for parallel simulated annealing—a heuristic method of optimization—that is intended to run on modern clusters of shared-memory (SMP) nodes. While clusters of SMPs with numbers of processors ranging into hundreds are becoming more and more popular, the question of how to use them efficiently for parallel simulated annealing, knowing its sequential character, is still open. One of popular programming styles for clustered systems uses different communication environments for their separate components, combining the benefits of both shared and distributed memory systems at the same time. The communication method discussed in the paper adopts such a hybrid approach for simulated annealing and is called a *hybrid communication method* (HC).

The research described in this work is a continuation of the efforts reported in [9], where the reference HC method was introduced. It proved to be the most effective compared with the other tested methods, when solving a bicriterion optimization problem. The paper presents a modification of the reference method, namely the *hybrid communication method with a single data exchange*, which is the way to improve quality of results for the second optimized criterion. The time scheduling aspects for both methods are discussed. A constrained cost is assumed, which means that searching for the optimal solution is performed with a given pool of processors available for a specified period of time. This approach produces linear speed-up.

Simulated annealing (SA) is a heuristic optimization method used when the solution space is too large to explore all possibilities within a reasonable amount of time. The vehicle routing problem with time windows (VRPTW) is an example of such a problem. Other examples are school bus routing, newspaper and mail distribution or delivery of goods to department stores. Optimization of routing lowers distribution costs and parallelization allows a better route to be found within given time constraints.

The SA bibliography focuses on the sequential version of the algorithm (e.g., [2, 17]), however parallel versions are investigated too, as the sequential method is considered to be slow when compared with other heuristics [18]. In [1, 3, 10, 12, 13] and many others, directional recommendations for parallelization of SA can be found. VRPTW, formally formulated by Solomon [16], who also proposed a suite of tests for benchmarking, also has a rich bibliography [18]. Additionally, a few works discussing parallel SA to solve the VRPTW are known, namely [6, 7, 4, 8]. Nevertheless, in contrast to the constraints applied in the current research, i.e., limited time, the first two take advantage of the parallel algorithm to achieve higher accuracy of solutions, while the others define different stopping criteria for the algorithm.

The plan of the paper is as follows: section 2 presents the theoretical basis of the sequential and parallel SA algorithm. Section 3 describes the two variants of the hybrid communication method, while section 4 presents practical issues, leading to load balanced execution. The results of the experiments are described in section 5. Conclusions follow.

2 Sequential and Parallel Simulated Annealing

In simulated annealing, one searches for the optimal state, i.e., the state that gives either the minimum or maximum value of the *cost function*. It is achieved by comparing the current solution with a random solution from a specific *neighbourhood*. With some probability, worse solutions could be accepted as well, which can prevent convergence to local optima. However, the probability of accepting a worse solution decreases over the process of annealing, in synchronisation with the parameter called *temperature*. An outline of the SA algorithm is presented in Figure 1, where a single execution of the innermost loop step is called a *trial*. The final solution which is returned is the best one ever found. Simulated annealing can be also modelled by using the theory of Markov chains. The algorithm is formed by a sequence of Markov chains where each chain consists of a sequence of trials for which the acceptance criterion with a fixed value of temperature was applied.

```
$S \leftarrow$ GetInitialSolution();
$T \leftarrow$ $InitialTemperature$;
for $i \leftarrow 1$ to $NumberOfTemperatureReduction$ do
    for $j \leftarrow 1$ to $EpochLength$ do
        $S' \leftarrow$ GetSolutionFromNeighbourhood();
        $\Delta C \leftarrow$ CostFunction($S'$) $-$ CostFunction($S$);
        if ($\Delta C < 0$ or AcceptWithProbabilityP($\Delta C, T$))
            $S \leftarrow S'$;     {i.e., the trial is accepted}
        end if;
    end for;
    $T \leftarrow \lambda T$;     {with $\lambda < 1$}
end for;
```

Fig. 1. SA algorithm

Since in SA each new state contains modifications to the previous state, the process is often considered to be inherently sequential and its parallelization is not trivial. However, a few strategies for designing a parallel SA algorithm exist, e.g., based on different types of applied decomposition. In the research the creation of trials is decomposed among processors. Additionally, the chain length is fixed, meaning that the number of trials performed within the chain is the same for both the sequential and parallel algorithms.

3 Hybrid Communication—Nesting OpenMP in MPI

Clustered SMP systems support two parallelization levels: the outer parallelization for communication between SMP nodes and the inner parallelization for the shared memory environment within nodes. The HC method tries to exploit the features of the parallel SA approach that can be supported by the architecture: intensively communicating parts can be realised inside the inner level with OpenMP [15], while parts with infrequent communication can be realised at the outer level with MPI [11, 14].

3.1 The Reference Method

Outer-level parallelization. Following previous research [9], in the algorithm for the outer level each Markov chain of SA optimization is divided into sub-chains. Their length is equal to the length of the original chain divided by the number of sub-chains. The main idea is to assign a separate sub-chain to each individual cluster node and thus to let nodes generate different sub-chains simultaneously. In this way the computation for generating a Markov chain is divided over all the available nodes. After generating the first Markov chain, the process of generating every consecutive chain is performed without communication between nodes. For each node the outcome of the last trial of the preceding sub-chain is the starting point for the subsequent sub-chain. At the end, the best solution found is picked up as the final one. The usage of multiple sub-chains allows intensive exploration of the search space. However, excessive shortening of the sub-chain length negatively affects the quality of results, so the maximum number of nodes used is limited by reasonable shortening of the sub-chain length.

Inner-level parallelization. Within a node a few threads can communicate to build one sub-chain of the length determined at the outer level. Negligible deterioration of quality is a key requirement for the inner-level algorithm. The idea of parallelization is to divide the total number of trials of each sub-chain into short *sets of trials*. The size of the sets equals the number of threads, so each thread generates one trial at a time, independently of the others. After completing a set, the master thread selects one solution among all the accepted ones and the others are discarded. The selected solution is common for all threads and becomes the starting point for further computation.

3.2 The Method with a Single Data Exchange

Tracing the process of finding solutions one can conclude, that incorporating lightweight communication between nodes could improve the quality of results. During the optimization all processes working on cluster nodes explore the search space, but after the first stage, which is characterized by "long jumps" and large changes of position, it is likely that only a few processes will be working in the "right" area of the global minimum. The rest of them may perform useless computations. One can speculate that global selection of the best result found during the stage of heavy exploration would let all the processes move into these "right" areas, leading to significant improvement of the quality of results. The ratio between durations of the two, above mentioned stages should be carefully selected. The optimization process should be able to use an adequate period of time during the first stage to explore the search space precisely enough to reach the area of the global minimum. On the other hand, the duration of the second stage should be long enough to let the processes exploit the promoted area and further approach the minimum.

4 Load Balancing in the Hybrid Communication Methods

4.1 Outer Level Load Balancing

The major drawbacks to obtain balanced computational load and acceptable speed-up are differences in the execution times of the trials, because the effort of performing them depends on the current configuration. This leads to substantial idle times when stopping the algorithm after generating a number of sub-chains. This is shown in Figure 2, where the times for generating 8 separate sub-chains are presented, based on an example run of the investigated problem (see section 5). To overcome this difficulty a real time limit is set for computation. It derives

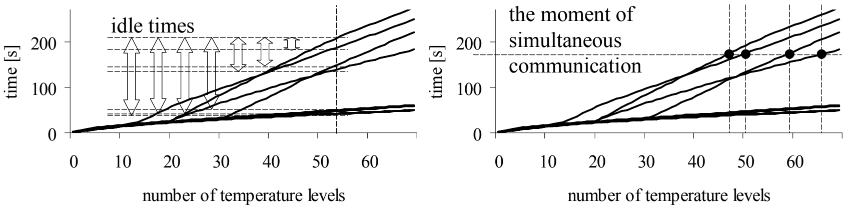

Fig. 2. The times for generating 8 sub-chains based on a run for the investigated problem. Left: scheduling by a fixed number of temperature levels with idle times marked with arrows. Right: time-based scheduling, where communication is scheduled after the same amount of time on all processes

from the average time needed by the sequential algorithm and is calculated so as to assure linear speed-up.

A time-based scheduling is also suitable when defining the way for announcing the moment of a single data exchange. Specifying the time limit for the computation by measurements of the elapsed time, gives a new opportunity to determine the exact moment of data exchange. Setting the number of data exchanges is straightforward as well. Therefore the proposed method forces one data exchange when a specific percentage of time limit (e.g., after 50%, 70% or 90%) elapses. After selecting and broadcasting the best solution found so far, all processes starts their computation from this agreed solution. The method results in much better balancing than an alternative with a fixed number of temperature levels (i.e., sub-chains) (Figure 2 (left)) and makes more efficient use of the given time limit. In Figure 2 (right) the moment of simultaneous communication is marked on the time axis. This can be individually determined by working processes irrespectively of the number of performed trials.

4.2 Inner Level Load Balancing

To achieve an acceptable inner-level speed-up a few optimization stages were necessary for the OpenMP parallelization with loop worksharing. The need for optimization stems from extremely varying execution time for each trial. In the presented example these differences were within a factor of 100. Consequently, where the number of trials equals the number of threads, i.e., each thread generates one trial at a time, a theoretically calculated speed-up, based on a comparison of the execution times, does not exceed 2 with the use of 4 threads. The average execution times of trials within a set, accumulated throughout the whole example run, are presented in Figure 3 (left). A case of one trial per a thread is marked white. The distance from the average is visible. The white bars show the average execution time of the fastest (left most bar) trial in a set of 4 trials, up to the slowest trial (right most bar). In Figure 3 (right) the histogram of trials with timings that falls into 25 ranges is presented. It also proves their imbalanced distribution.

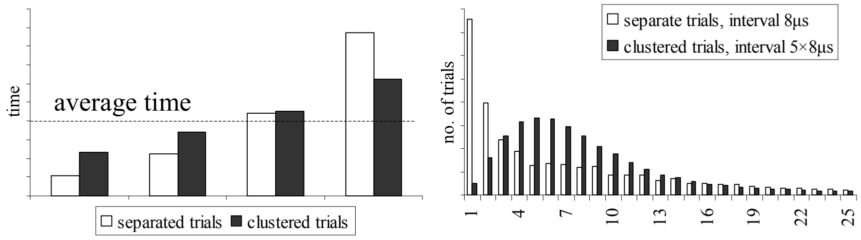

Fig. 3. Trials before the reconfiguration. Left: average execution times of the trials within a set. Right: the histogram of execution times. Example: Solving VRPTW, test R108 from Solomon's benchmark set

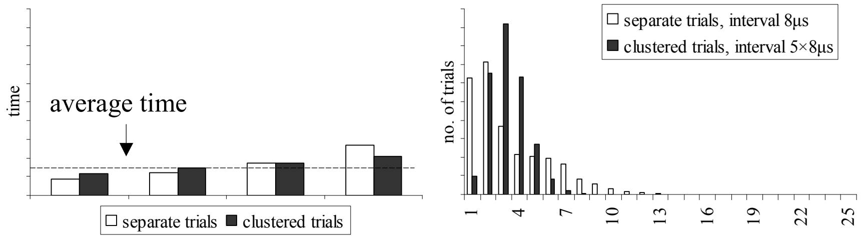

Fig. 4. Trials after the reconfiguration. Left: average execution times of the trials within a set. Right: the histogram of execution times. Example: Solving VRPTW, test R108 from Solomon's benchmark set

The first optimization step was to increase the size of a set of trials to make each thread generate a few trials at a time without any communication. In this way the load imbalance was substantially decreased. Nevertheless, in order to maintain quality, the size of the set of trials should be as small as possible, because it affects the number of accepted but discarded trials. The average execution times of so called "clustered" trials within a set of the size 20 (= 4 threads × 5 trials/thread), as well as the histogram of execution times are also presented in Figure 3, but marked black. The scale for "clustered" trials was normalised to the scale of "separate" trials to indicate more clear their smaller deviations from the average value and changed distribution.

The second optimization step was the redefinition of a trial, in order to improve load balancing and simultaneously to decrease the number of discarded trials. Hence, the speed-up as well as a quality of results can be increased. As the execution time of each trial is determined by the time for finding a new valid solution S' in the neighbourhood of S, one can limit the number of actions taken within GetSolutionFromNeighbourhood() (see Figure 1). If after only a few disturbances of the current configuration no new solution can be created, the algorithm produces a transitional status "no answer" and suspends the process of completing a trial. After the selection has been made over the set by the master thread, all uncompleted trials are continued. The influence of the redefinition can be seen in Figure 4, where the time scale is the same as in Figure 3. The average execution time of a trial is shortened but better balanced than before the redefinition.

The third optimization step was to choose an appropriate moment for forking as well as for joining parallel threads. As the execution time for a set of trials can be short compared to the OpenMP fork-joined overhead, the parallel loop should comprise a wider region, i.e., the whole temperature step.

5 Experimental Results

In the vehicle routing problem with time windows it is assumed that there is a warehouse, centrally located to customers. There is a road between each pair of customers and between each customer and the warehouse. The objective is to supply goods to all customers at the minimum cost. The solution with fewer route legs (the first goal of optimization) is better then a solution with smaller total distance travelled (the second goal of optimization). Each customer as well as the warehouse has a time window. Each customer has its own demand level and should be visited only once. Each route must start and terminate at the warehouse and should preserve maximum vehicle capacity. As already mentioned, previous work [9] focused on the first goal of optimization, while this paper focuses on the second one, i.e., optimizing the final distance when the minimum number of route legs is already achieved. The sequential algorithm from [5] was the basis for parallelization.

Experiments were carried out on a NEC Xeon EM64T Cluster installed at the High Performance Computing Center, Stuttgart (HLRS). Additionally, for tests of the OpenMP algorithm, a NEC TX-7 (ccNUMA) system was used. The numerical data were obtained by running the program 100 times for Solomon's [16] R108, R111, RC105 and RC108 tests with 100 customers and the same set of parameters. The number of OpenMP threads was 4 and the size of the set of trials was 20, this giving the best combination of efficiency and quality. Due to the lack of access to a genuine clustered SMP machine with 4 CPUs per each node (the NEC Xeon EM64T consists of dual CPU nodes), the usage of 4 OpenMP threads per cluster node was emulated. The emulation was carried out by extending the applied time limit by the speed-up factor coming from a separate set of experiments. Such an extension can be thought as undoing the speed-up to be observed on a cluster of nodes having 4 CPUs instead of 2.

Time results. At the outer level both versions of the hybrid algorithm give linear speed-up, since a real time limit is applied. However, in case of the method with a single data exchange one should consider the additional communication overhead. In investigated examples the time needed for selecting and broadcasting the best solution between nodes was between 0.4ms (2 nodes) to 0.8ms (30 nodes), which is substantially shorter than the execution time. At the inner level the average speed-up factor obtained empirically was 2.7, which gives the efficiency of OpenMP parallelization as 67%.

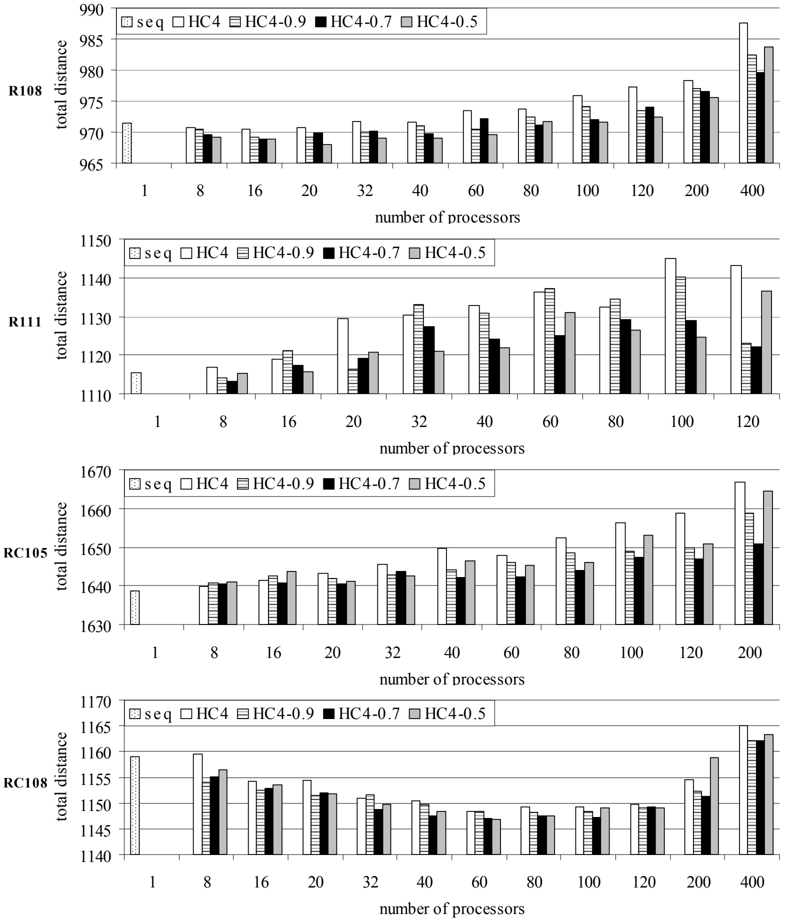

Fig. 5. Comparison of quality results for hybrid communication methods

Quality results. A few parameters for controlling the data exchange were investigated, namely after reaching 50%, 70% and 90% of the time limit. The results of experiments are presented in Figure 5. Generally, selecting a common solution after 50% (HC4-0.5) or 70% (HC4-0.7) of the time limit was much better than other tested possibilities. It should be noted that both HC4-0.5 and HC4-0.7 give better results than the reference method (HC4) almost for all investigated numbers of processors (with only one exception). Nevertheless, when compared to the sequential results (SEQ) it can be observed that the quality of the hybrid parallelization depends on a test. E.g., for R108 up to 40 processors, R111 with 8 processors and RC108 (excluding 400 processors) the results of HC4-0.7 are better than for the sequential version, but in other cases, i.e., RC105, R108 with more than 40 processors and R111 (excluding 8 processors) they are worse.

To verify these observations one can incorporate test statistics. Statistical hypotheses $H_0 : x_i = x_j$ versus alternative hypotheses $H_1 : x_i < x_j$ or $H_1' : x_i \neq x_j$ can be tested, where x denotes the mean value of the total travel distance, i, j - populations that are compared (HC4-0.7 with HC4, HC4-0.7 with SEQ, HC4-0.5 with SEQ, respectively). Let s denote the standard deviation, and n, the population size, then u, the test statistic is given by:

$$u = \frac{x_i - x_j}{\sqrt{\frac{s_i^2}{n_i} + \frac{s_j^2}{n_j}}}$$

The significance level is set as 0.05. When comparing HC4-0.7 with HC4 the calculated values u indicate that H_0 should be rejected in favour of H_1 in 64% of tested cases. This means HC4-0.7 gave statistically shorter total distance than HC4. Besides, when comparing HC4-0.7 with SEQ, although for the tests R111 and RC105 H_0 can not be rejected in favour of H_1' up to 20 processors, for R108 it can not be rejected up to 100 processors. In other words there is no evidence that HC4-0.7 gave statistically different results for these cases, compared to the sequential algorithm. Additionally, application of similar reasoning indicates, that for test RC108, statistically HC4-0.7 allowed to achieve solutions with smaller travel distances than its sequential equivalent for numbers of processors up to 200. For test R108, HC4-0.5 compared favourably to the sequential version with up to 40 processors.

6 Conclusions

In this study the implementation of parallel SA algorithm that is intended to run on clusters of SMP nodes is considered. The development of the reference method, based on performing time scheduled data exchange was proposed. Additionally, the paper provides detailed analyses of factors influencing the speed-up and efficiency, e.g.: defining the moment for terminating the optimization process, as well as time dependencies between randomly generated trials of SA algorithm. A few optimization strategies were introduced, that resulted in better balancing of the algorithm.

Based on experiments one can conclude that the quality of results for the modified method outperforms the reference one. When compared to sequential results, it needs to be stated that with proposed load balancing strategies it is possible in many cases to achieve better or comparable quality, always with linear speed-up. This observation is valid even up to 200 processors.

Acknowledgement

This work was supported by the EC-funded project HPC-Europa (contract No RII3-CT-2003-506079) and by the State Committee for Scientific Research grant 3T 11F 00429. Computing time was also provided within the framework of the HLRS-NEC cooperation.

References

1. Aarts, E., de Bont, F., Habers, J., van Laarhoven, P.: Parallel implementations of the statistical cooling algorithm. Integration, the VLSI journal (1986) 209–238
2. Aarts, E., Korst, J.: Simulated Annealing and Boltzman Machines, John Wiley & Sons (1989)
3. Azencott, R. (ed): Simulated Annealing Parallelization Techniques. John Wiley & Sons, New York (1992)
4. Arbelaitz, O., Rodriguez, C., Zamakola, I.: Low Cost Parallel Solutions for the VRPTW Optimization Problem, Proceedings of the International Conference on Parallel Processing Workshops, IEEE Computer Society, Valencia–Spain, (2001) 176–181
5. Czarnas, P.: Traveling Salesman Problem With Time Windows. Solution by Simulated Annealing. MSc thesis (in Polish), Uniwersytet Wrocławski, Wrocław (2001)
6. Czech, Z.J., Czarnas, P.: Parallel simulated annealing for the vehicle routing problem with time windows. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands–Spain, (2002) 376–383
7. Czech, Z.J., Wieczorek, B.: Frequency of cooperation of parallel simulated annealing processes, Proceedings of the 6th International Conference on Parallel Processing and Applied Mathemetics (PPAM'05), Poland (in print)
8. Debudaj-Grabysz, A., Czech, Z.J.: A concurrent implementation of simulated annealing and its application to the VRPTW optimization problem, in Juhasz Z., Kacsuk P., Kranzlmuller D. (ed), Distributed and Parallel Systems. Cluster and Grid Computing. Kluwer International Series in Engineering and Computer Science, Vol. 777 (2004) 201–209
9. Debudaj-Grabysz, A., Rabenseifner, R.: Nesting OpenMP in MPI to implement a hybrid communication method of parallel simulated annealing on a cluster of SMP nodes, in Di Martino B., Kranzlmuller D., Dongarra J.,(ed.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag Berlin Heidelberg, LNCS 3666, (2005) 18–27
10. Greening, D.R.: Parallel Simulated Annealing Techniques. Physica D, 42, (1990) 293–306
11. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing 22(6) (1996) 789–828
12. Lee, F.A.: Parallel Simulated Annealing on a Message-Passing Multi-Computer. PhD thesis, Utah State University (1995)
13. Lee, K.–G., Lee, S.–Y.: Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains, IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 10 (1996) 993–1008
14. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Rel. 1.1, June 1995, www.mpi-forum.org
15. OpenMP C and C++ API 2.5 Specification, from www.openmp.org/specs/
16. Solomon, M.: Algorithms for the vehicle routing and scheduling problem with time windows constraints, Operation Research 35 (1987) 254–265, see also http://w.cba.neu.edu/˜msolomon/problems.htm
17. Salamon, P., Sibani, P., Frost, R.: Facts, Conjectures and Improvements for Simulated Annealing, SIAM (2002)
18. Tan, K.C, Lee, L.H., Zhu, Q.L., Ou, K.: Heuristic methods for vehicle routing problem with time windows. Artificial Intelligent in Engineering, Elsevier (2001) 281–295