A SIMULATED ANNEALING ALGORITHM FOR MULTI-AGENT SYSTEMS: A JOB-SHOP SCHEDULING APPLICATION

M. Emin Aydin, Terence C. Fogarty
South Bank University, School of Computing, Information Systems and Mathematics, 103 Borough Road, London, SE1 0AA, UK
{aydinme,fogarttc}@sbu.ac.uk

ABSTRACT

In this paper, a parallel implementation of a modular simulated annealing (MSA) algorithm, a shortened simulated annealing (SA) algorithm, applied to classical job-shop scheduling (JSS) problems is presented. The implementation has been done as a multiple island system suitable to run on the Distributed Resource Machine (DRM) environment, which is a novel, scalable, distributed virtual machine developed based on Java technology. The JSS problems tackled are very well known difficult benchmarks, which are considered to measure the quality of such systems. The support of the DRM environment was very effective with respect to message passing, having collaboration with a remote machine. The empirical results show that the method proposed is quite successful compared to the ordinary MSA and other systems described in literature.

KEY WORDS

Parallel Simulated Annealing, Multi-Agent Systems, Job-Shop Scheduling, DRM.

1. INTRODUCTION

SA is a stochastic heuristic algorithm in which the solutions are searched for in hill climbing processes constantly commenced by random moves. Because of its ease of use, SA is an extremely popular method for solving large-sized and practical problems like job-shop scheduling, timetabling and travelling salesman. However, for various reasons, like many other search algorithms, SA may become trapped by any local minima, which does not allow moving up or down, or take a long time to find a reasonable solution, which sometimes makes the method unpreferrable. For these reasons, many SA implementations have been done as part of a hybrid method, [1][7][8][15]. In this work, we investigated a parallel version of modular simulated annealing (MSA), a new SA algorithm that works like an evolutionary process as an operator with a population of solutions [1]. In order to reach a better result by SA algorithms, it is necessary to give sufficient time to SA. This makes the process longer which could be much more time consuming, if SA works with a population. Although MSA manages to reach good result in a shorter time, it may require the longer time needed for very hard problems. The idea of this work is to parallelise MSA to improve its performance in job shop scheduling problems by partitioning a bigger population into small subparts each to be operated by a separate MSA agent so that each individual takes more opportunities to be seen. Ordinarily, SA is not so flexible as to be parallelised. But since MSA works with a population of solutions, the parallelisation can be done via partitioning the population into sub-parts, so as SA can be parallelised as multi-island models.

On the other side, we have tackled classical job shop scheduling problems known as static scheduling too. The benchmark problems undertaken are very hard problems collected in the OR library [4], a collection of benchmark problems for OR studies. The implementation has been done as a multiple island model to run on distributed resource machine (DRM), which is

a novel scalable distributed problem-solving environment [6]. Each island has run a separate MSA algorithm in parallel with other peers.

The organisation of the rest of the paper is as follows. We give an introduction to DRM environment and then the classical job-shop scheduling problems in Section 2 and 3. Afterwards, we briefly described MSA algorithm and the proposed parallel version of MSA in Section 4 and 5. We discuss the empirical results yielded by parallel MSA comparing them with those obtained from ordinary MSA in Section 6. The superiority of our results over the other related works are discussed in Section 7. The paper finishes with the conclusion in Section 8.

2. DISTRIBUTED RESOURCE MACHINE (DRM)

DRM is the framework of a distributed problem solving environment that is one of two main part of DREAM [11] project [5] which deals with creating a multi island based evolutionary computation environment running on a scalable network on Internet. The main idea is to have a peer-to-peer network of nodes spreading on physically distributed computers. Each node has incomplete knowledge of the rest of the network and works as the container of all the agents running on that computer. Since DRM is an autonomous agent environment, the applications are implemented as multi agent systems. The environment has very good functionalities to develop applications such that the agents would have good communication and limited mobility. (See [6] for more information on DRM).

3. JOB SHOP SCHEDULING

Job Shop Scheduling (JSS) problems have been studied for a long time. It is difficult to reach the optimal solution in a short time, since the problems have a very wide solution space and there is no guarantee to reach a better state after a feasible state [2]).

We are given a set of jobs *(J)* to be processed on a set of machine (*M*) subject to a number of technological constraints. Each job consists of *m* operations (*O_j*) that must be processed on a particular specified machine and each job visits each machine exactly once. There is a predefined restricted order for the operations of each job in which the operations are processed after their predecessors (*PJ_j*) and before their successor (*SJ_j*). At the same time, there is another order being built up among the operations of different jobs that must be processed on particular machines: this is to put *n* operations in an order on each machine during the scheduling time. Therefore, each operation processed on the *M_i* has a predecessor (*PM_i*) and a successor (*SM_i*), which are not restricted so as to optimise that order to have a shorter schedule. A machine can process only one operation at a time. There are no set-up times, no release dates and no due dates.

Each operation has a processing time (*p_{ij}*) on related machine starting on *r_{ij}*. The completion time of *o_{ij}* is:- $c_{ij} = r_{ij} + p_{ij}$ where *i = (1,...,m), j=(1,...,n)* and $r_{ij} = \max(c_{iPJ_j}, c_{PM_i j})$. Each machine and job have particular completion times which are calculated in the following manner:- $C_{M_i} = \sum_{j=1}^{n} c_{ij}$ and $C_{J_j} = \sum_{i=1}^{m} c_{ij}$. The overall objective is to minimise the completion time of whole schedule, which is the supremum of machine's completion times. $C_{\max} = \max(C_{M_1}, ..., C_{M_m})$

A job-shop scheduling problem can be represented by a disjunctive graph as given in Figure 1. The tasks are the nodes in the graph, each with a single attribute representing the duration. Two dummy nodes are introduced: the start node and the end node, each with duration 0. Each precedence constraint is represented by a directed arc, from a task to its successor. Additional arcs are added from the start node to the first task in each job, and from the last task in each job to the end node. Each resource constraint is represented by a bi-directional arc (or *disjunctive* arc). Selecting one of the two directions for a disjunctive arc imposes an ordering on the two tasks concerned. Selecting an orientation for every disjunctive arc such that there are no cycles in the graph reduces the disjunctive resource constraints to precedence constraints. Given a fully oriented graph, the minimum makespan for that graph can be found by computing the longest path (given by double line in Figure 1) from start node to the end node, where the length of an arc is equal to the duration of the task that starts the arc. The scheduling problem thus reduces to one of finding orientations for all the disjunctive arcs such that the least makespan can be obtained [3]. Local search methods can operate by changing the orientation of some of the disjunctive arcs, and re-computing the minimum makespan. It has been shown [3] that the makespan can only be reduced by changing the orientation of one of the disjunctive arcs on the longest path.
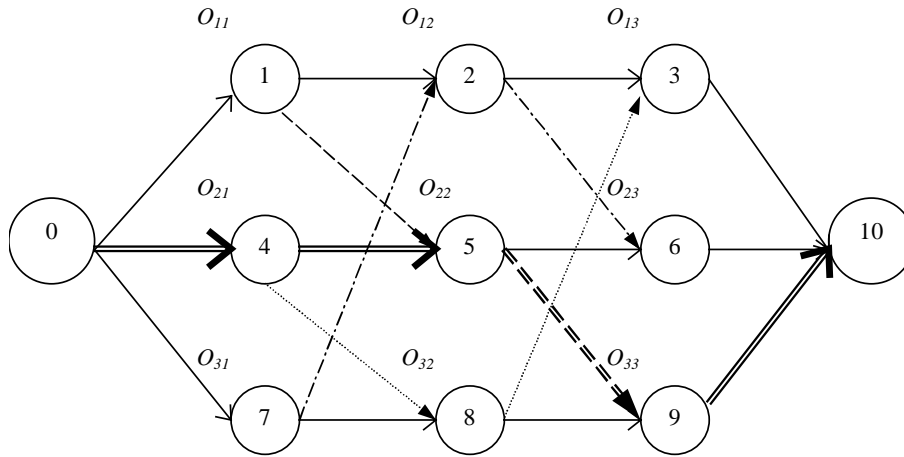

Figure 1; a disjunctive graph for a 3x3 job-shop problems

4. MODULAR SIMULATED ANNEALING (MSA)

The modular simulated annealing (MSA) algorithm is the partitioning of the SA algorithm into shorter slices to be implemented in various configurations together with different methods and environments. The idea behind modular SA is to have a more uniform distribution of random moves along the SA procedure. In fact, SA provides the solution process by a logarithmic distribution of random moves such that each random move starts a new hill climbing process to reach the global minimum. However, the logarithmic nature of this distribution may not help to rescue the solution from local minimum as in the case, when SA is applied to very difficult combinatorial optimisation problems like some of the hard benchmark job shop scheduling problems tackled in this work. Such problems need more random moves even in the latter part of the optimisation process. But the probability of having a random move at that stage is so low as to make it longer to reach the global optimum. On the other hand, modular SA algorithm takes such a short time that it can be considered an operation when applied with a context of evolutionary processes, and it can be

constantly applied to a particular solution as well as a population of solutions. Aydin and Fogarty [1] have applied a modular SA to some job shop scheduling benchmarks that have moderate difficulty. The idea of that work was to evolve a population of solutions by applying a modular SA constantly to selected solutions.

A typical instance of modular SA algorithm is presented in Figure 2. In this case, the algorithm is implemented to evolve a population of solutions running modular SA constantly up to a predefined number of iterations. First of all, a population of solutions is randomly initialised, and then, the number of iterations is set. After that, modular SA starts with a highest temperature (100), which is being cooled by cooling coefficient (0.955) iteration by iteration. When the temperature cooled to 0.01 short-term SA finishes with 200 iterations, which are counted to completion. The selected and optimised solution obtained through a modular SA is put back into the population. That is the end of one modular SA process. The succeeding cycle of evolution starts by selecting another solution randomly from the population. This process repeats until that total number of iterations is completed.

Begin
⇨ *Initialise* the **population**,
  Repeat:
    • *pick* one **completed schedule (*old*)**,
    • *set* the **highest temperature** *(t=100),*
    repeat:-
      ▪ *select* a particular **task**, *conduct* a move by **neighbourhood function**
      ▪ *repair* the **new schedule (*new*)**
      ▪ *if* (***new-old***)<0 then *replace **old*** with ***new***
      ▪ *else*
        – *generate* a **random number *(r)***
        – *if exp(-(new-old)/t)>r* then *replace **old*** with ***new***
        – *endif*
      ▪ *endif*
      ▪ *t=t*0.955*
    until *t<0.01*
    • *put* the **schedule** back into the **population**
  *Until **pre-defined number** of **iterations***
End.

Figure 2: An instance of modular simulated annealing algorithm for evolution of a population

5. A PARALLEL IMPLEMENTATION OF MSA

As it is well known, there are two main ways to implement a system as a parallel computation. One is by partitioning a whole data set into subparts and running the same algorithm on each of those subparts on multiple machines or processes. This could be called as the physical parallelism. The second one is more complicated in which the parallelisation is done on the algorithm itself rather then partitioning the data. That is called as the algorithmic parallelism. Since it is difficult to parallelise an ordinary SA in the sense of algorithmic parallelism, we have parallelised the system in the sense of physical parallelism.

As discussed in the previous section, MSA gives new opportunities to commence new valuable hill climbing processes in which the considered particular solution may have chances to change to better situation. Therefore, the more time to see a particular solution for MSA, the easier to reach global optimum. However, operating on a single solution is not preferable for MSA, because of the special local minimum of solutions. It is better to let MSA operate on a population of solutions to utilise the diversity of population, which causes longer time. These two constraints make MSA to work on a rather small-sized population. Unlike genetic algorithms, we need to work on better-designed small-sized populations to have the advantages of both the diversity of population and having more consideration by MSA run. However, it is difficult to have a good spectrum of solutions in small sized populations. One of the possible solutions for this can be the consideration of parallel computing opportunities. The idea of this parallel application of MSA is to distribute a rather bigger-sized population over more islands to create more opportunities for letting MSA manipulate solutions for more times even within a shorter time.

The MSA algorithm has been implemented to run on Distributed Resource Machine (DRM) [6], which is a virtual, scalable distributed environment based on Java technology. Since MSA has a modular nature, we easily designed this implementation for running on DRM as a multiple island framework. Problem solving with DRM requires partitioning of the problem into subparts to be applied as a multi-island model. For this reason, we designed our islands with repeated MSA algorithm and a small population of solutions where MSA operates on that population to evolve it towards an optimum value. The population uses a simulated annealing based replacement rule to promote new solutions over the old. The solution tackled per iteration is selected randomly, operated by MSA algorithm once and then is assessed to be replaced with its parent. One randomly selected solution attempts to migrate to another randomly determined island by a predefined period. This cycle is repeated for a predefined number of iterations.

In this application, we have a group of islands consisting of 5 islands each evolving a 10-sized population for 2 million iteration and one of them is the root that performs collecting the bests and providing the islands with relevant data to initiate their populations. The idea is presented in Figure 3.
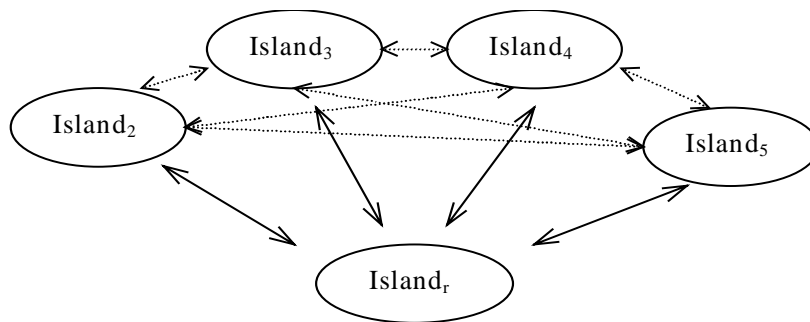

Figure 3; inter-islands relationships for parallel MSA

The islands communicate with one another by letting the solutions migrate from one to another as well as reporting their bests to the Root Island at the end of every period. The experiments are launched on DRM by creating the Root Island first on the root node. The Root Island creates other islands and randomly dispatches them on randomly selected living nodes of DRM by providing them with completely different population, which is of size 10

solutions. The whole size of population operated within an experiment is thus 50. By this application, we have got more chances for each particular solution as well as a more divers population, which provides different landscape of solutions to search on.

6. EMPIRICAL RESULTS

In order to illustrate the efficiency of parallel implementation of modular SA, we have done a series of experiments of job shop scheduling problems. The problems tackled are very well known difficult benchmarks, which have been solved by various researchers to show the goodness of their methods. In Table 1, the results of both ordinary and parallel MSA implementations are shown as the average value, standard deviations, best and worst values and then the time taken per experiment in average. The optimal and/or lower bound of each problem has been given in the second column adjacent to the problem names, where the values given with asterisks (*) are for the optimum and the others are lower bounds.

The general considerations for both cases are as follows:-
- the algorithm has done 10 million moves at the end (each cycle of MSA takes 200 iterations that means the population has been operated 50000 times) ,
- the size of population is 50 (which means that each individual has been operated 1000 times),
- a solution to manipulate is selected randomly,
- the new results are replaced only if they satisfy a simulated annealing like stochastic rule.

The parallel implementation of MSA is a partitioned version of the ordinary one, cutting the population into 5 parts and dispatching each to a particular parallel island. The islands have done 2 million of iterations over the 10-sized population. Thus, each individual in this case has got exactly the same number of manipulation as the individuals in the case of Ordinary MSA. Since the MSA operates on an individual 200 times per cycle, each individual has been seen and operated 1000 times. The migration is set up to a period, which is 100000, with a low probability. That means that every 100000 iteration a randomly selected individual attempts to migrate to another randomly defined island, if the random number generated is greater then 0.50. Therefore, every island totally allows 10 individuals to migrate to another island, and accepts in. Experiments have been repeated 5 times per problem.

Comparing both cases, we can easily see that there is statistically no difference between the results of the parallel version and the ordinary one, apart from LA38. Best and worst results are very close to each other. On the other hand, the time taken is very different. The ordinary case has taken more or less 4 times then the parallel one. This is the significant aspect of our method. The worse results are caused by the low diversity of the population. We suspected that, if we increase the diversity of island's populations by changing the rate of migration, the inter-population effect might make our method much better. In fact, we applied two more strategies for migration beside the first one: one does not allow any migration at all, the other works in such a way that every 100000 iterations the island allows an individual to migrate with 0.75 migration rate. That changes the number of migrated individuals from 10 to 15 per island. The results for all situations are shown in Figure 4 presenting % error between the average of found values and optimal ones. On the horizontal axis, the benchmarks are given in number representing ABZ7, ABZ8, ABZ9, LA21, LA24, LA25, LA27, LA29, LA38 and LA40, respectively.   As it is, there is no significant difference among those three strategies. It helps very little to keep the populations of islands diverse.

Table 1. Empirical results for both serial and parallel MSA implementations

| Problem		Ordinary MSA , Pop-50, 10 million iterations					Parallel MSA, Pop- 50 (10 per Island), 10 million (2 per Island) iterations				
Name	Optimum	Mean	SD	Best	Worst	Time	Mean	SD	Best	Worst	Time
ABZ7	655	675.5	2.12	673	678	10542	675.2	2.8	672	678	1445.2
ABZ8	638	686.8	4.95	683	692	10491	687.2	4.9	681	692	1902.6
ABZ9	656	699	1.41	698	700	10465	703.0	2.9	699	706	1578.4
LA21	*1046	1049.4	2.88	1046	1051	5052	1048.4	2.7	1046	1053	838.2
LA24	*935	939.2	2.68	935	941	4933	939.6	1.5	938	941	570.2
LA25	*977	978.4	2.19	977	982	5161	977.8	1.8	977	981	1035.2
LA27	*1235	1244.4	4.56	1238	1250	6997	1245.4	3.9	1240	1250	982.0
LA29	1130	1177.4	7.54	1173	1188	6830	1182.6	6.1	1176	1190	1147.8
LA38	*1196	1201.8	3.77	1196	1204	7802	1214.6	3.0	1211	1219	1143.4
LA40	*1222	1230.8	3.03	1228	1235	7849	1229.2	2.68	1228	1234	1894.6
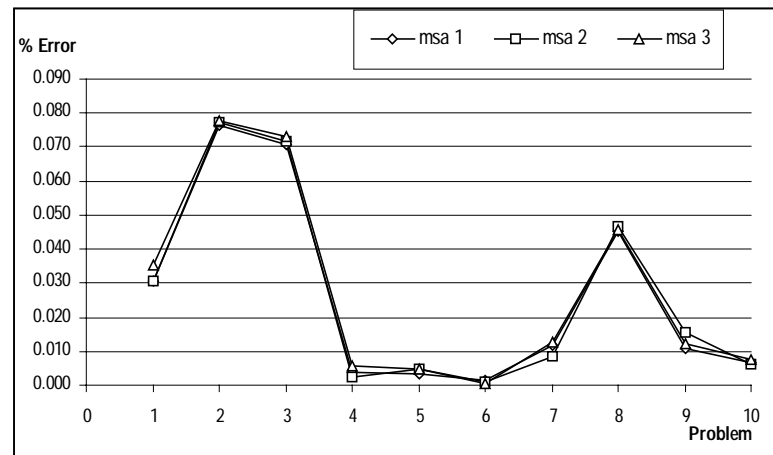
Figure 4; error percentage of found results respecting each migration rate.

7. RELATED WORKS

There are enormous studies done on classical job-shop scheduling problems. We have filtered the most relevant ones in terms of publication date, the problems tackled, and the methods employed to make comparison with our results where SA, GA and their combinations have been considered. Wang & Zheng [15] developed a hybrid optimisation strategy by embedding a SA into GA for job shop scheduling problems. The undertaken benchmarks in their paper are moderately difficult problems on which our system very quickly find their optimum. Steinhofel et al [13] discusses some parallel heuristics for simulated annealing-based algorithms to be applied to job shop scheduling problems. They theoretically criticise the calculation of longest path of schedules by clarifying the dependency on numbers of jobs and machines in an algorithmic parallelism point of view. This is an advanced version of their previous work, which discusses two SA heuristics for job shop scheduling problems [14]. They presented computational results for very few benchmarks that are worse then ours with respect to time.

Table 2. Some experimental results from the literature to be compared to ours

Problem		Satake et al.			KTM			Kolonko			Parallel MSA		
Name	Opt	Mean	Best	Time	Mean	Best	Time	Mean	Best	Time	Mean	Best	Time
ABZ7	655	684.8	679	5991.0	-	-	-	-	-	-	675.2	672	1445.2
ABZ8	638	698.0	684	5905.4	-	-	-	-	-	-	687.2	681	1902.6
ABZ9	656	706.6	698	5388.9	-	-	-	-	-	-	703.0	699	1578.4
LA21	*1046	1062.0	1046	1516.1	1050.0	1047	1720.2	1051.0	1047	594.2	1048.4	1046	838.2
LA24	*935	949.0	936	1422.8	942.0	941	1170.4	940.4	938	569.6	939.6	938	570.2
LA25	*977	989.6	980	1605.0	980.5	979	1182.8	979.0	977	644.4	977.8	977	1035.2
LA27	*1235	1263.2	1248	3761.0	1247.3	1241	919.8	1244.8	1236	3650.6	1245.4	1240	982.0
LA29	1130	1196.4	1167	4028.6	1173.3	1165	3042.9	1169.2	1167	4496.0	1182.6	1176	1147.8
LA38	*1196	1218.4	1202	3004.0	1210.5	1202	3044.0	1202.4	1201	5049.4	1214.6	1211	1143.4
LA40	*1222	1243.0	1233	2812.2	1233.7	1228	6692.8	1228.6	1226	4544.0	1229.2	1228	1894.6

Table 2 shows some comparable results from different works, where our results are generally better than them. Satake et al. [12] present a rescheduling based SA approach in their paper with the results given in the second part of Table 2, running the experiments on an IBM-PC Pentium 133 MHz., while KTM used SUN SPARC workstation 2 for a taboo search based approach as reported by Satake et. al[12]. Our results are better then them with respect to the length of schedules and CPU time. However, Kolonko [10] has better results herding from a combined GA-SA system for the last four benchmarks respecting length of schedules, but not respecting CPU time. Our CPU times are superior, but our computers may be better as well (Pentium III). The main advantage of our method is to allow solving the problems in a partitioned and distributed way. The results reported in this paper are gained by a system consisting of 5 subsystem working distributedly and in parallel. If we increase the number of subparts (islands), we will definitely get shorter schedules then we have so far.

8. CONCLUSION

In this paper, a parallel implementation of MSA algorithm, a shortened SA algorithm, applied to classical job-shop scheduling problems has been presented. This parallel implementation has been done by running the system on DRM environment as a multi island system. The tackled JSS problems are very well known difficult benchmarks, which are considered to measure the quality of such works. The support of DRM environment was very effective with respect to message passing, having collaboration with a remote machine. The empirical results show that the method proposed is quite successful comparing to the ordinary MSA and some other works in literature. The only problem is the uncertainty in the size of populations, and the possibility of shortening the process time of the method. These points are still not clear and need to be investigated. In the future, these would be the main direction of this research.

9. ACKNOWLEDGEMENT

REFERENCES

[1] Aydin M. E.., Fogarty, T. C., "Simulated annealing with evolutionary processes in job shop scheduling", in Evolutionary Methods for Design, Optimisation and Control, (Proc. of EUROGEN 2001, Athens, 19-21 Sept.) CIMNE, Barcelona, 2002

[2] Baker, K. R., Introduction to Sequencing and Scheduling, John Wiley & Son: 1974.

[3] Balas, E., "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm", Operations Research, 17, 1969, 941-957.

[4] Beasley, J. E., OR Library, Imperial College, Management School, 1990. http://mscmga.ms.ic.ac.uk/info.html.

[5] Dream Web Site, http://www.world-wide-dream.org.

[6] Jelasity M., Preuβ M., Peachter B., "A scalable and robust framework for distributed applications", CEC'02: The 2002 World Congress on Computational Intelligence, *12-17 May 2002:* Honolulu, HI, U.S.A.

[7] Jeong, I.K, Lee J.J., "Adaptive Simulated Annealing Genetic Algorithm for System Identification", Engineering Applications of Artificial Intelligence, 9(5), 1996, 523-532.

[8] Huang, H.C., Pan, J.S., Lu, Z.M., Sun, S.H., Hang, H.M., " Vector quantization based on genetic simulated annealing", Signal Processing, 81, 2001, 1513-1523.

[9] Kirkpatrick, S., Gelatt, C. D. Jr. and Vecchi, M. P., "Optimisation by simulated annealing", Science, 220(4598), 1938, 671-679.

[10] Kolonko, M., "Some new results on simulated annealing applied to job shop scheduling problem", European Journal of Operational Research, 113, 1999, 123-136.

[11] Peachter B., Back T., Schoenauer M., Sebag M., Eiben A. E., Merelo J. J., Fogarty T. C., " A distributed resource evolutionary algorithm machine (DREAM)" in Proc. of the Congress of Evolutionary Computation 2000 (CEC200). IEEE, 2000, pp. 951-958, IEEE Press.

[12] Satake, T., Morikawa, K., Takahashi, K., Nakamura, N., "Simulated annealing approach for minimising the makespan of the general job-shop", International Journal of Production Economics, 60-61, 1999, 515-522.

[13] Steinhofel, K., Albrecht, A., Wong, C. K., "Fast parallel heuristics for the job shop scheduling problem", Computers & Operations Research, 29, 2002, 151-169.

[14] Steinhofel, K., Albrecht, A., Wong, C. K., "Two simulated annealing-based heuristics for the job shop scheduling problem", European Journal of Operational Research , 118, 1999, 524-548.

[15] Wang, L., Zheng, D.Z., "An effective hybrid optimisation strategy for job-shop scheduling problems", Computers & Operations Research, 28, 2001, 585-596.

[16] Wong, S.Y.W., "Hybrid simulated annealing/genetic algorithm approach to short term hydro-thermal scheduling with multiple thermal plants", Electrical Power & Energy Systems, 23 2001, 565-575.