http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (2): 222-226, 2007
ISSN 1812-5638

Modular Simulated Amealing in Classical Job Shop Scheduling

[1]S. Jayalakshmi and [2]S.P. Rajagopalan
[1]Department of Computer Science, S.D.N.B. Vaishnav College For Women,
Chromepet, Chennai-600 044, India
[2]Mohammed Sathak Group of Educational Institutions, Chennai-600 005, India

Abstract: In this research, a parallel implementation of a Modular Simulated Annealing (MSA) 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 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: Modular simulated annealing, job shop scheduling, parallel implementation, distributed resource machine, multi island systems

INTRODUCTION

Simulated Annealing (SA) is a stochastic heuristic algorithm in which the solutions are searched in hill climbing processes constantly commenced by random moves. SA is an extremely popular method for solving large-sized and practical problems like job-shop scheduling, timetabling and traveling salesman problem. However, 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. For this reason, many SA implementations have been done as part of a hybrid method. In this study, 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 (Aydin and Fogarty, 2002). 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 for very hard problems. The idea of this work is to parallelize MSA to improve its performance in job shop scheduling problems. Parallelization can be done by partitioning the population in to sub parts, so that SA can be parallelized as multi island models.

Here, we have tackled classical job shop scheduling problems which are otherwise known as static scheduling. The benchmark problems undertaken are very hard problems collected in the OR library (Beasley, 1990), 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 (Jelasity *et al*., 2002). Each island has run a separate MSA algorithm in parallel with other peers. The empirical results of ordinary MSA and parallel MSA are compared to show that parallel implementation of MSA is quite successful.

DISTRIBUTED RESOURCE MACHINE (DRM)

The DRM is a Peer-to-Peer overlay network on the Internet forming an *autonomous agent environment*. The applications are implemented as multi-agent applications. The exact way an application is implemented in the multi-agent framework is not restricted in any way. In the DRM every node is completely equivalent. There are no nodes that possess special information or have special functions. The nodes must be able to know enough about the rest of the network in order to be able to remain connected to it and to provide information about it to the agents. 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.

Corresponding Author: S. Jayalakshmi, Department of Computer Science, S.D.N.B. Vaishnav College For Women, Chromepet, Chennai-600 044, India Tel: 044-22235840

JOB SHOP SCHEDULING

Job Shop Scheduling (JSS) problems have been attended for a long time. It is difficult to reach the optimal solution in short time, since the problems have a very wide solution space and there is no guarantee to transmit a better state after a feasible state (Baker, 1974). We are given a set jobs (J) to be processed in a set of machine (M) subject to a number of technological constraints. Each job consists of m operations Oj, which must be processed on a single specified machine and each job visits each machine exactly once. There is a predefined ordering of the operations within a job. Because of this order, each operation has to be processed after its predecessor (PJj) and before its successor (SJj). Each machine processes n operations in an order that is determined during the scheduling time, although there is no such an order initially.

Therefore, each operation processed on the Mi has a predecessor (PMi) and a successor (SMi). 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 (pij) on related machine starting on rij. The completion time of oij is:- cij = rij + pij

where i = (1, …, m), j = (1, …, n) and rij = max (ciPJ j, cPMi j).

Each machine and job have particular completion times which are calculated in the following manner:-

$$C_{Mi} = \sum_{j=1}^{n} cij \quad \text{and} \quad C_{ji} = \sum_{i=1}^{m} cij$$

The overall objective is to minimize the completion time of whole schedule, which is the supremum of machine's completion times.

$C_{max} = \max(C_{M1}, C_{M2}, \ldots, C_{mm})$. A job-shop scheduling problem can be represented by a disjunctive graph. 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

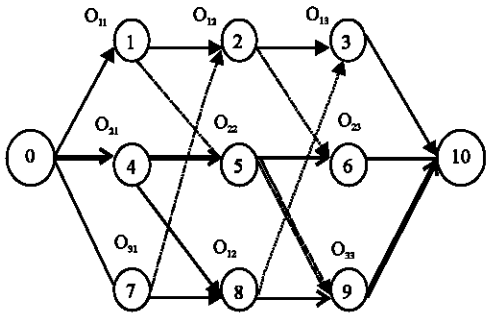

Fig. 1: A disjunctive graph for a 3×3 job-shop problems

makespan for that graph can be found by computing the longest path (given by double line in Fig. 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 (Balas, 1969). 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 (Balas, 1969) that the makespan can only be reduced by changing the orientation of one of the disjunctive arcs on the longest path.

MODULAR SIMULATED ANNEALING (MSA)

The modular simulated annealing (MSA) algorithm is the partitioned 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 optimization 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 optimization 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.

A typical instance of modular SA algorithm is presented in the following. 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 initialized 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 complete. The selected and optimized solution 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.

An instance of modular simulated annealing algorithm:

```
Begin
Initialize 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.
```

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 and known as physical parallelism. The second one is more complicated in which the parallelization is done on the algorithm itself rather than partitioning the data and is known as algorithmic 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 better 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 utilize 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.

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 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 the following Fig. 2.

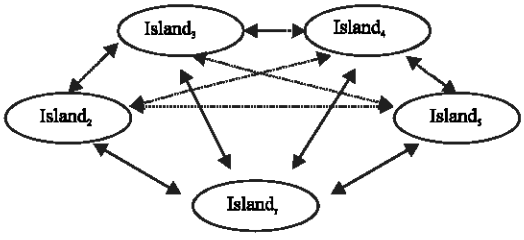

Fig. 2: Inter-islands relationships for parallel MSA

The islands communicate with one another by letting the solutions migrate from one to another as well as to report 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 diverse population, which provides different landscape of solutions to search on.

RESULTS AND DISCUSSION

In order to illustrate the efficiency of parallel implementation of modular SA a series of experiments were done on 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 and then the time taken per experiment in average and test statistic value t to test the significant difference between means. 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 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. 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. 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. On the other hand, the time taken is very different. The ordinary case has taken more or less 4 times than the parallel one. This is the significant aspect of the MSA method. The results for all situations are shown in Table 2 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 two strategies.

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

Problem		Ordinary MSA, Pop-50			Parallel MSA, Pop-50 (10 per Island)			Test statistic
Name	Optimum	Mean	SD	Time	Mean	SD	Time	t-value
ABZ7	655	675.5	2.12	10542	675.2	2.8	1345.2	0.2416
ABZ8	638	686.8	4.95	10491	687.2	4.9	1802.6	0.1624
ABZ9	656	699.0	1.41	10465	703.0	2.9	1478.4	0.8772
LA21	1046*	1049.4	2.88	5052	1048.4	2.7	738.2	0.7165
LA24	935*	939.2	2.68	4933	936.6	1.5	470.2	0.3684
LA25	977*	978.4	2.19	5161	977.8	1.8	935.2	0.5987
LA27	1235*	1244.4	4.56	6997	1245.4	3.9	882.2	0.4714
LA29	1130	1177.4	7.54	6830	1182.6	6.1	1047.8	1.5165
LA38	1196	1201.8	3.77	7802	1214.6	3.0	1043.4	7.5155
LA40	1222*	1230.8	3.03	7849	1229.2	2.68	1794.6	1.1187

*Optimum, Others- Lower bounds

Table 2: Error percentage of results for ordinary MSA and Parallel MSA

Problem		Ordinary MSA, Pop-50		Parallel MSA, Pop-50 (10 per Island)	
Name	Optimum	Mean	Error percentage	Mean	Error percentage
ABZ7	655	675.5	0.030	675.2	0.0308
ABZ8	638	686.8	0.076	687.2	0.0771
ABZ9	656	699.0	0.0655	703.0	0.0716
LA21	1046*	1049.4	0.0032	1048.4	0.0023
LA24	935*	939.2	0.0045	936.6	0.0017
LA25	977*	978.4	0.0014	977.8	0.0008
LA27	1235*	1244.4	0.0076	1245.4	0.0084
LA29	1130	1177.4	0.0419	1182.6	0.0465
LA38	1196	1201.8	0.0048	1214.6	0.0156
LA40	1222*	1230.8	0.0072	1229.2	0.0631

*Optimum, Others- Lower bounds

CONCLUSION

In this study, 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 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 empirical results show that compared to the ordinary MSA the Parallel Implementation of MSA is quite successful.

REFERENCES

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

Baker, K.R., 1974. Introduction to Sequencing and Scheduling, John Wiley and Son.

Balas, E., 1969. Machine sequencing via disjunctive graphs: An implicit enumeration algorithm. Operations Research, 17: 941-957.

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

Jelasity, M., M. Preu and B. Peachter, 2002. A scalable and robust framework for distributed applications. CEC'02: The 2002 World Congress on Computational Intelligence, May 2002: Honolulu, HI, USA., pp: 12-17.