JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **37,** 207–212 (1996)
ARTICLE NO. 0121

Parallel Simulated Annealing Algorithms

D. JANAKI RAM,[1] T. H. SREENIVAS, AND K. GANAPATHY SUBRAMANIAM

Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India

Simulated annealing (SA) has been considered a good tool for complex nonlinear optimization problems. The technique has been widely applied to a variety of problems. However, a major disadvantage of the technique is that it is extremely slow and hence not suitable for complex optimization problems such as scheduling. There are many attempts to develop parallel versions of the algorithm. Many of these algorithms are problem dependent in nature. We present, in this paper, two general algorithms for SA. The algorithms have been applied to job shop scheduling problem (JSS) and the traveling salesman problem (TSP) and it has been observed that it is possible to achieve superlinear speedups using the algorithm. © 1996 Academic Press, Inc.

I. INTRODUCTION

Simulated annealing (SA) has been considered a good tool for complex nonlinear optimization problems [5, 2]. The technique has been widely applied to a variety of problems. One of the major drawbacks of the technique is its very slow convergence.

There have been many attempts to develop parallel versions of the algorithm. Some parallel systems exist for achieving close to ideal speedup on small processor arrays [6]. There are also special purpose architectures for implementing the annealing algorithm [9]. Two different approaches to parallelization of SA exist in literature—single-trial parallelism and multiple-trial parallelism [5]. Very often these approaches are highly problem-dependent and the speedup achievable depends on the problem characteristics. In all these cases there is need to divide the problem into subproblems and subsequently distribute these problems among the nodes or processors.

We present, in this paper, two distributed algorithms for simulated annealing. The first algorithm is named the clustering algorithm (CA) and the second algorithm the genetic clustering algorithm (GCA). In the clustering algorithm a cluster of nodes work on the search space of the simulated annealing algorithm. The cluster of nodes assist each other by exchanging their partial results and this helps the nodes to converge to a good initial solution to start with. In the second algorithm, the genetic algorithm is applied to find the initial n good solutions required as the starting point for the simulated annealing algorithm on the n different nodes of the network. The two algorithms have been applied to the job shop scheduling problem (JSS) and the traveling salesman problem (TSP). Both algorithms showed very good performance in terms of solution time and solution quality.

The rest of the paper is organized in the following fashion. Section II describes the simulated annealing technique. Section III presents the cluster algorithm. Section IV presents the genetic cluster algorithm. Section V gives implementations of both the algorithms on a network of Sun workstations. Section VI deals with the application and performance studies of the algorithms to the cases of JSS and TSP.

II. SIMULATED ANNEALING TECHNIQUE

Often the solution space of an optimization problem has many local minima. A simple local search algorithm proceeds by choosing random initial solution and generating a neighbor from that solution. The neighboring solution is accepted if it is a cost decreasing transition. Such a simple algorithm has the drawback of often converging to a local minimum. The simulated annealing algorithm, though by itself it is a local search algorithm, avoids getting trapped in a local minimum by also accepting cost increasing neighbors with some probability. In SA, first an initial solution is randomly generated, and a neighbor is found and is accepted with a probability of $\min(1, \exp(-\delta/T))$, where δ is the cost difference and T is the control parameter corresponding to the temperature of the physical analogy and will be called temperature. On slow reduction of temperature, the algorithm converges to the global minimum, but the time taken increases drastically.

Simulated Annealing is inherently sequential and hence very slow for problems with large search spaces. Several attempts have been made to speed up this process, such as development of parallel simulated annealing techniques and special purpose computer architectures.

A. Parallel Versions of SA

Parallelism in SA can be broadly classified into two approaches—single trial parallelism and multiple trial parallelism [5]. But these methods are highly problem dependent and speed up achieved depends wholly on the problem at hand. Another taxonomy divides parallel annealing

[1] To whom correspondence should be addressed. E-mail: djram@iitm.ernet.in.

0743-7315/96 $18.00

techniques into three major classes; (1) serial-like algorithms, (2) altered generated algorithms, and (3) asynchronous algorithms [1]. Each class of the algorithm makes some tradeoff among cost function accuracy, state generation, parallelism, and communication overhead. High performance special purpose architectures show promise for solving computationally expensive applications without expensive super computers and include specially designed computer architectures to suit annealing algorithm [11].

III. CLUSTERING ALGORITHM FOR SIMULATED ANNEALING

Experiments on the SA technique have shown that a good initial solution results in faster convergence. Similar observations were made in [13].

The distributed algorithms proposed take advantage of this observation. Initially, the n nodes of the network run SA algorithm using different initial solutions. After a fixed number of iterations, they exchange their partial results to get the best one. All the nodes accept the best partial solution and start applying the SA technique for that best partial result. They again exchange their partial results after some fixed number of iterations. After repeating this process for a predefined number of times, each node works independently on its partial result. The complete algorithm is given in Table I.

IV. COMBINATION OF GENETIC ALGORITHM AND SIMULATED ANNEALING ALGORITHM

Experiments showed that a good initial solution for SA improves both the quality of the solution and also the execution time. Genetic algorithms try to improve a set of solutions rather than a single solution. Since we require n initial solutions for distributing among n nodes, we chose to combine SA with GA.

TABLE I
Cluster Algorithm for Simulated Annealing

Input to the algorithm:
- n = Number of the nodes in the network.
- p = Exchange parameter for partial results.
- r = Reduction parameter for the number of iterations before exchange of partial results.
- i = Input graph for scheduling.

Co-ordinator node algorithm:
1. Distribute the n random initial solutions to the n nodes and wait.
2. Upon receiving the first converged result from any of the nodes stop simulated annealing on other nodes.

Worker node algorithm:
1. Accept initial solutions from the co-ordinator.
2. **repeat**
   - 2.1. Execute Simulated annealing for p iterations. Exchange partial results among the worker nodes. Accept the best partial result.
   - 2.2. $p = p - r^*$ (loop iteration number).

   until ($p \Leftarrow 0$).
3. Execute simulated annealing using the best solution found as the initial solution.
4. Send the converged value to the coordinator.

TABLE II
Genetic Clustering Algorithm (GCA)

(1) Central node generates n initial solutions using GA. It runs GA for fixed number of iterations, t.
- (1.1) Choose initial population of fixed size and set $i = 1$.
- (1.2) **while** ($i \Leftarrow t$)
  begin
  - (1.2.1) Apply the operator on the two parent schedules chosen randomly to produce two offspring and replace the parents by the best two out of the four schedules.
  - (1.2.2) $i = i + 1$

  end

(2) Central node sends n best solutions chosen to the n remote worker nodes.
(3) Each worker node runs the SA algorithm using the initial state received.
(4) Upon receiving a converged result from one of the worker nodes, the central node stops execution.

A. Genetic Algorithm

In GA [11], an initial population consisting of a set of solutions is chosen and then solutions are evaluated. Relatively more effective solutions are selected to have more offsprings, which are in some way related to the original solutions. If the genetic operator is chosen properly, the final population will have better solutions. GA improves the whole population. SA aims at producing one best solution. For the distributed SA implementation, we require several good initial solutions to ensure fast convergence of SA. We chose GA for obtaining the required number of good initial solutions. The operator used for generating offsprings in JSS is related to the processing order of jobs on the different machines of the two parent solutions. Let PO11, PO12, ... , PO1m be the processing orders of jobs on machines 1, 2, ... , m in parent1 and PO21, PO22, ... , PO2m be the processing order on machines 1, 2, ... , m in parent2. If random $(1, m) = i$, then processing orders in child1 and child2 are PO11, ... , PO1i, PO2i + 1, ... , PO2m and PO21, ... , PO2i, PO1i + 1, ... , PO1m respectively. After getting the offspring, a check is made to see if there are any cycles in the offsprings and if there is one, the operation is performed once again by generating another random number. A cycle in a state indicates an invalid schedule. The pseudo code for the Genetic Clustering algorithm (GCA) is given in Table II.

V. IMPLEMENTATION OF THE ALGORITHMS

Both the above algorithms have been implemented using a platform called DiPS (Distributed Problem Solver) [3] running on a network of 18 Sun workstations. It is built on a communication kernel. Using the kernel, it is possible

TABLE III
Clustering Algorithm for the Central Node

1. Initialize ( ).
2. Generate n random initial states and assign to the n nodes of the network.
3. Wait for results.
4. Output_Results.

to send task award messages, task result messages, configure messages, and partial result messages, among the various nodes of the DiPS network. The full implementation details of both algorithms are given in subsequent sections.

A. Implementation of Clustering Algorithm

In the clustering algorithm, the central node executes the code in Table III and the worker nodes the code in Table IV. The algorithm for simulated annealing is given in Table V.

B. Implementation of Genetic Clustering Algorithm

In the case of Genetic Clustering Algorithm (GCA), first the genetic algorithm is run on the central node to get the required n initial solutions. These initial solutions are used by the n client nodes of the distributed systems as a starting solution for the simulated annealing algorithm.

The code that is executed on the central node is the same as the code in Table III except that in step 3 the n schedules are the best n solutions chosen from the population after applying GA. The genetic algorithm starts with an initial population. It then performs the crossover operation and the population is updated. This is repeated a number of times.

VI. CASE STUDIES

The algorithms have been applied to the job shop scheduling problem and the traveling salesman problem.

A.1. Job Shop Scheduling

Job Shop Scheduling involves scheduling of various operations relating to a number of jobs on a number of machines [2, 8, 4]. different techniques exist in the literature for solving JSS [15]. Since the main focus of this paper is parallel SA, we consider the SA algorithm for solving JSS. Each machine can process only one operation at a time. Each job consists of a sequence of operations in a predefined precedence order. The problem is to find a schedule having minimum total time (cost), often called ''make span'' of the schedule. An initial schedule is obtained for a given set of n jobs to be scheduled on m machines. The simulated annealing algorithm is applied to the initial schedule. The algorithm improves on the initial schedule by generating neighborhood schedules and evaluating them. As the temperature is gradually reduced, the algorithm converges to a near optimal solution.

One of the major drawbacks of the simulated annealing algorithm is that it is very slow, especially when the search space is large. The algorithm's behavior is also greatly influenced by the choice of the initial schedule. Attempts to parallelize the SA algorithm for JSS resulted in three different algorithms, namely the temperature modifier algorithm, the locking edges algorithm, and the modified locking edge algorithm [10, 12]. The temperature modifier algorithm, though is problem independent, has not shown much improvement in speedup. However, the quality of results produced is better. In the other two algorithms, the search space is divided using a special technique called the locking edge technique. These algorithms are highly problem dependent and hence are applicable only to the JSS problem. It is for the above reasons that a problem-independent true distributed simulated annealing algorithm is attempted for development. One of the basic problems for distributed simulated annealing algorithms is that SA is inherently sequential. Hence the option is to distribute the search space and let the algorithm run on a reduced search space on each node of the network. Since the division of the problem into subproblems depends largely on

TABLE IV
Clustering Algorithm for the Worker Node

1. Get the subtask from the central node and p, the exchange parameter.
2. **while** ($p > 0$)
   begin
   2.1. Simulated_annealing (n).
   2.2. Send the best solution obtained to the central node.
   2.3 $p = p -$ (loop_iteration_value)*r.
   end.
3. Run SA.
4. Send the converged value to the central node.

TABLE V
Simulated Annealing

Simulated annealing (n)
begin
1. Set t = Initial_Temperature.
2. **repeat**
   2.1 Counter = 0.
   2.2 **repeat**
      2.2.1 Compute the cost of the schedule ($f[i]$).
      2.2.2 Find the critical path schedule.
      2.2.3 Generate a neighbor and compute the cost of the Neighbor ($f[j]$).
      2.2.4 Accept or reject the neighbor with a probability of min(1, $e^{-(f[i]-f[j])/t}$).
      2.2.5 increment counter.
      until (Counter = Number of Iterations at t).
3. $t = t^*$ temp_modifier.
4. After every n iterations exchange results and accept the best schedule found.
   until (shopping criteria)
end.

TABLE VI
Effect of Initial Solution on Simulated Annealing

Initial makespan	Final makespan	Time taken (s)
1076	896	498
1120	896	510
1284	905	610
1328	905	815

the characteristics of the problem, such as algorithm cannot be general in nature. Also, it is likely that one node gets into other nodes' search space, which is termed a collision. Such intrusions have to be treated separately. For these reasons a static division of the search space among the nodes is not ideal.

A.2. Performance Study of the Algorithms

The performance of the CA and GCA are compared with that of sequential simulated annealing (SSA) for different sizes of job shop scheduling problem (from 10 jobs on 10 machines to 20 jobs on 15 machines). The annealing algorithm used an initial temperature of 2000, temperature modifier of 0.95. Annealing was frozen when the best solution found did not change for the last 25 temperature changes or the temperature was reduced to zero. Table VI shows performance of SSA.

Performance of the algorithms has been compared based on two factors, namely, the execution time of the algorithm and the cost of the solution. It can be observed from the Table VII that CA performed very well in terms of both execution time and quality of the solution compared to the sequential simulated annealing algorithm. CA sometimes showed superlinear speedups. It can also be observed from Table VII that at low problem sizes, GCA performed better compared to CA. This can be explained from the fact that at low problem sizes, GCA is able to give good initial solutions with a small overhead, whereas in the case of large problem sizes, the quality of the population is not appreciably improved by running GA for a fixed time period. In case of GCA, the time spent on GA can be increased to select good initial solutions. But the overhead of GA increases correspondingly. Hence the optimal time to be spent on GA in the case of GCA can be found by conducting experiments by varying this time.

TABLE VII
Comparison of SSA, CA, and GCA Performance (with 3 Client Nodes)

Problem size (jobs × m/cs)	SSA		CA		GCA	
	Time (s)	Cost	Time (s)	Cost	Time (s)	Cost
10 × 10	577	968	307	931	199	971
10 × 15	1402	1603	803	1565	551	1683
10 × 17	2864	1548	128	1542	890	1538
20 × 10	7439	3309	3137	3309	3266	3434
20 × 15	6701	3391	2862	3391	1907	3391

TABLE VIII
Performance of CA and CGA for a Specific Problem Instance (Size = 10 × 15)

No. of nodes	CA		GCA	
	Time (s)	Speedup	Time (s)	Speedup
3	835	2.44	695	2.93
4	656	3.10	551	3.70
5	593	3.75	494	4.12
6	419	4.86	502	4.05
7	445	4.57	321	6.31
8	317	6.42	296	6.88
9	226	9.01	288	7.07

Table VIII shows the relative performance of CA and GCA for a fixed problem size as the number of nodes is varied. As the number of nodes is increased, CA performed better compared to GCA. This can be explained from the fact that GCA requires n good initial solutions generated by GA. As n increases, the quality of the initial solution decreases as GA is run only for a fixed initial time.

Table VII shows relative performance of CA and GCA as the problem size is increased keeping the number of nodes fixed. At low problem sizes GCA performed better compared to CA. This can be explained by the fact that as the problem size increases, the quality of the initial solutions generated by GA by running it for a fixed amount of time is not good.

B.1. Traveling Salesman Problem

The TSP is that of finding the minimum weight Hamiltonian cycle in a given undirected graph. The quality of solutions obtained for TSP using simulated annealing is good, though obtaining them is time consuming. It is in this context that parallel algorithms for SA for solving TSP are of practical interest. We have applied the genetic clustering algorithm for the TSP and compared its performance with that of the sequential algorithm. The graph is represented as an adjacency matrix of intercity distances. A valid initial solution for SA and the initial population for

TABLE IX
Comparison of GCA and SA Performance for the TSP

Problem size (No. of cities)	GCA		SA		
	Time (s)	Cost	Time (s)	Cost	Nodes
50	107	717	112	767	3
100	139	1136	308	1134	3
150	497	1336	1568	1218	3

TABLE X
Comparison of Dynamic and Static Switching Performance for the TSP (with 3 Nodes)

	Dynamic switching		Static switching	
Problem size	Time (s)	Cost	Time (s)	Cost
50	55	707	57	725
100	80	1286	95	1276
150	100	1802	128	1862

GA are obtained by a depth first search. The neighboring solution for SA is generated by the method suggested in [6], where a section of the path chosen is traversed in the opposite direction. The genetic operator *crossover* called *edge recombination* [14] is applied to a pair of parent solutions to generate two offspring and the best two out of the four solutions replace original ones. The required n initial solutions for the parallel SA are obtained by the genetic algorithm as explained earlier.

B.2. Performance Study of the Algorithms

Three sizes of the TSP, namely 50, 100, and 150, have been considered for the performance analysis of the GCA algorithm. The cooling schedule for SA employed an initial temperature of 1000 and a temperature modifier value of 0.99, and that for GCA an initial temperature of 1 and a temperature modifier value of 0.99. GA was stopped when the solutions did not change over a fixed number of iterations. SA also was stopped when there was no improvement in the quality of the solution over a fixed number of iterations.

The relative performance between GCA and SA for TSP is given in Table IX. Three trials were taken on three problem instances and the average value of the cost and time are tabulated in each case. It is evident from the table that GCA performed very well as the problem size increased and showed superlinear speedups at higher problem sizes. At lower problem sizes the algorithm did not perform well, as the overhead of the genetic algorithm is high on a smaller search space. We also experimented with the dynamic switching from GA to SA. Instead of running the GA for a fixed amount of time depending on the search space size, the improvement in the quality of the n best solutions in the population has been used as the criterion for switching from GA to SA. When the n best solutions in the population do not change for a fixed number of iterations, GA is stopped. These n initial solutions are taken and fed to the parallel SA algorithms on n nodes of the network. It has been observed that there is an optimum time up to which GA can be run before switching to SA. If GA is run for less than this optimum time, n good initial solutions are not found for SA. If it is run for more time, time is wasted in finding more than n good initial solutions. Thus switching based on the improvement of the n initial solutions can be seen to perform best (refer to Table X).

VII. CONCLUSIONS

Two distributed algorithms for simulated annealing have been developed and have been applied to the JSS and TSP. The algorithms showed very good results as the problem space and the number of nodes employed increased.

REFERENCES

1. Greening, Daniel R., Parallel simulated annealing techniques, *Physica D* **42** (1990), 293–306.
2. Van Laarhoven, P. J. M., Aarts, E. H. L., and Lenstra, Jan Karel. Job shop scheduling by simulated annealing. In *Operation Research,* Vol. 40. 1992, pp. 113–125.
3. Janaki Ram, D., and Ganeshan, K. DiPS: A system for implementing and analyzing distributed algorithms. Tech. Rep. IITM-CSE-93-001, Indian Institute of Technology, Madras.
4. French, Simon. *Sequencing and Scheduling: An Introduction to the Mathematics of Job Shop.* Wiley, New York.
5. Eglese, R. W. Simulated annealing: A tool for operation research. In *Eur. J. Oper. Res.* **46** (1990), 271–281.
6. Allwright, James R. A., and Carpenter, D. B., A distributed implementation of simulated annealing for the traveling salesman problem. In *Parallel Comput.* **10** (1989), 335–338.
7. Decker, Keith S., Distributed problem solving techniques: A survey. In *IEEE Trans. Systems Man Cybernet.* **17** (1987), 729–739.
8. Adams, J., Balas, E., and Zawack, D. The shifting bottleneck procedure for job shop scheduling. *Mgmt. Sci.* **34** (1988), 391–401.
9. Abramson, David. A very high speed architecture for simulated annealing. *IEEE Comput.* (May 1992), 27–36.
10. Ganeshan, K. Designing and implementing flexible distributed problem solving systems. M.S. Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, 1993.
11. Syswerda, Gilbert. Schedule optimization using genetic algorithms. In Davis, L. (Ed.). *Handbook of Genetic Algorithms.* pp. 332–349.
12. Krishan, K. Ganeshan, and Ram, D. Janaki. Distributed simulated annealing algorithms for job shop scheduling. *IEEE Trans. Systems Man Cybernet.* **25,** 7 (July 1995), 1102–1109.
13. Gu and Huang, X. Efficient local search with search space smoothing. *IEEE Trans. Systems Man Cybernet.* **24,** 5 (May 1994), 728–735.
14. Whitley, Darrel, Starkweather, Timothy, and Shaner, Daniel. Schedule optimization using genetic algorithms. In Davis, Lawrence (Ed.). pp. 351–357.
15. Nowicki, E., and Smutnicki, C. A fast tabu search algorithm for the job shop problem. Report 8/93, Institute of Engineering Cybernetics, Technical University of Wroclaw, 1993. To appear in *ORSA J Comput.*

D. JANAKI RAM is currently working as an assistant professor in the Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India. He coordinates research activities of the Distributed and Object Systems Group at I.I.T. Madras. He obtained his Ph.D. degree from the Indian Institute of Technology, Delhi in 1989. He has taught courses on distributed systems, object oriented software development, operating systems, programming languages, and artificial intelligence at graduate and undergraduate levels. He is also a consulting engineer in the area of software development for various organizations.

His research interests include distributed and heterogeneous computing, distributed and object databases, object technology, software engineering, and CAD/CAM.

T. H. SREENIVAS is a faculty member in the Department of Computer Science and Engineering, National Institute of Engineering at Mysore, India. He received his M. Tech degree from the Indian Institute of Technology, Kanpur. He is presently carrying out his doctoral work at the Department of Computer Science and Engineering at the Indian Institute of Technology, Madras. His areas of interest include distributed algorithms, artificial intelligence, and expert systems.

GANAPATHY SUBRAMANIAM received his bachelor's degree in computer science and engineering from the Indian Institute of Technology, Madras in 1996. He is presently pursuing his graduate studies in the Department of Computer Science at the University of Maryland, College Park. His areas of interest include object databases and distributed algorithms.

Received August 30, 1994; revised May 2, 1996; accepted May 15, 1996.