Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP [⋆]

Panos M. Pardalos[1], L.Pitsoulis[1], T. Mavridou[1], and Mauricio G.C. Resende[2]

[1] Center for Applied Optimization and Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611-6595 USA
[2] AT&T Bell Laboratories, Room 2D-152, 600 Mountain Avenue, Murray Hill, NJ 07974-2070 USA

Abstract. In this paper, we review parallel search techniques for approximating the global optimal solution of combinatorial optimization problems. Recent developments on parallel implementation of genetic algorithms, simulated annealing, tabu search, and greedy randomized adaptive search procedures (GRASP) are discussed.
Key words: Parallel Search, Heuristics, Genetic Algorithms, Simulated Annealing, Tabu Search, GRASP, Parallel Computing.

1 Introduction

Search techniques are fundamental problem-solving methods in computer science and operations research. Search algorithms have been used to solve many classes of problems, including *path-finding problems*, *two-player games* and *constraint satisfaction problems*. Classical examples of path-finding problems include many combinatorial optimization problems (e.g. integer programming) and puzzles (e.g. Rubic's cube, Eight Puzzle). Chess, backgammon, and othello belong to the class of two player games, while a classic example of a constraint satisfaction problem is the eight-queens problem.

For $\mathcal{NP}$-hard combinatorial optimization problems, exact search algorithms, such as branch and bound, may degenerate to complete enumeration. For that reason, exact approaches limits us to solve only moderately sized problem instances, due to the exponential increase in CPU time when problem size increases. Therefore, in practice, heuristic [3] search algorithms are necessary to find sub-optimal solutions to these problems. For large-scale problems, one of the main limitations of heuristic search is its computational complexity. Efficient parallel implementation of search algorithms can significantly increase the size of the problems that can be solved. While there is a large

[⋆] invited paper, Workshop on "Parallel Algorithms for Irregularly Structured Problems" Lyon, France (September 4–6, 1995)

[3] Etymologically the word *heuristic* comes from the Greek word *heuriskein* to discover; the Greek mathematician and inventor Archimedes (287–212 B.C.) is known for the famous *Heureka!* when he discovered a method to verify the purity of gold.

body of work on search algorithms, work on parallel search algorithms is relatively sparse.

In this paper, we explore different approaches of parallel heuristic search for solving combinatorial optimization problems. We focus on issues of parallelizing genetic algorithms, simulated annealing, tabu (or taboo) search, and GRASP (Greedy randomized adaptive search procedures). These heuristic methods have been used to approximately solve a wide spectrum of combinatorial optimization problems.

2 Parallel Genetic Algorithms

In the 1960's, biologists began to use digital computers to perform simulations of genetic systems. Although these studies were aimed at understanding natural phenomena, some were not too distant from the modern notion of a genetic algorithm (GA). Genetic algorithms, as they are used today, were first introduced by John Holland [23]. Genetic algorithms try to imitate the development of new and better populations among different species during evolution, just as their early biological predecessors. Unlike most standard heuristic algorithms, GAs use information of a population of individuals (solutions) when they conduct their search for better solutions and not only information from a single individual. GAs have been applied to a number of problems in combinatorial optimization. In particular, the development of parallel computers has made this an interesting approach.

A GA aims at computing sub-optimal solutions by letting a set of random solutions undergo a sequence of unary and binary transformations governed by a selection scheme biased towards high-quality solutions. Solutions to optimization problems can often be coded to strings of finite length. The GAs work on these strings [23]. The encoding is done through the structure named *chromosomes*, where each chromosome is made up of units called *genes*. The values of each gene are binary, and are sometimes called *alleles*. The problem is encoded by representing all its variables in binary form and placing them together in a single chromosome. A fitness function evaluates the quality of a solution represented by a chromosome.

There are several critical parts which strongly affect the success of genetic algorithms:

– Representation of the solutions.
– Generation of the initial population.
– Selection of which individuals in an old population that will be allowed to affect the individuals of a new population. In terms of evolution, this relates to the selection of suitable parents in the new population.
– Genetic operators, such as crossover and mutation. That is, how to recombine the genetic heritage from the parents in the previous generation.

If $P(t)$ denotes the population at time t, the GA can be described as in Figure (1).

$P(0)$ is usually generated at random. The evaluation of a population $P(t)$ involves computing the fitness of the individuals and checking if the current population satisfies certain termination conditions. Types of termination rules include:

```
Layout of Genetic Algorithm

Input: A problem instance

Output: A (sub-optimal) solution

1. $t = 0$, initialize $P(t)$, and evaluate the fitness of the individuals in $P(t)$

2. while (termination condition is not satisfied) do
   (a) $t = t + 1$
   (b) Select $P(t)$, recombine $P(t)$ and evaluate $P(t)$

3. Output the best solution among all the population as
   the (sub-optimal) solution.
```

Fig. 1. Layout of Genetic Algorithm

- A given time limit which is exceeded.
- The population is dominated by a few individuals.
- The best objective function value of the populations is constant over a given number of generations.

Due to their inherent parallel properties, GAs have been successfully implemented on parallel computers, introducing this way a new group of GAs, *Parallel Genetic Algorithms* (PGAs). In a PGA the population is divided into subpopulations and an independent GA is performed on each of these subpopulations. Furthermore, the best individuals of a local population are transferred to the other subpopulations. Communication among the subpopulations is established to facilitate the operations of selection and recombination. There are two types of communication [34]: (1) *among all nodes* where the best string in each subpopulation is broadcasted to all the other subpopulations, and (2) *among the neighboring nodes*, i.e. only the neighboring subpopulations receive the best strings.

The most important aspects of PGAs, which result in a considerable speedup relative to sequential GAs, are the following [25]:

- *Local selection*, i.e. a selection of an individual in a neighborhood is introduced, in contrast with the selection in original GAs which is performed by considering the whole population.
- *Asynchronous behavior* which allows the evolution of different population structures at different speeds, possibly resulting in an overall improvement of the algorithm in terms of CPU time.
- *Reliability* in computation performance, i.e. the performance of one processor does not affect the performance of the other processors.

Jog et al. [25] consider two basic categories of *parallel genetic algorithms*:

1. The *coarse-grained* PGAs, where subpopulations are allocated to each processor of the parallel machine. The selection and recombination steps are performed within a subpopulation.
2. The *fine-grained* PGAs, where a single individual or a small number of individuals are allocated to each processor.

In the same reference a review of parallel genetic algorithms applied to the *traveling salesman problem* (TSP) is presented. A PGA developed by Suh and Gucht [51], has been applied to TSPs of growing size (100-1000 cities) problems. PGAs without selection and crossover, so called *independent strategies* were used. These algorithms consist of runing an "unlimited" number of independent sequential local searches in parallel. PGAs with *low amount of local improvement* were used and performed better in terms of quality solution than the independent strategies. In terms of computational time, the PGA showed nearly a linear-speedup for various TSPs, using up to 90 processors. The algorithms were run on a BBN Butterfly.

Another implementation of a PGA applied to the TSP can be found in [21], where an asynchronous PGA, called ASPARAGOS, has is presented in detail. An application of ASPARAGOS was also presented by Muhlenbein [33] for the *quadratic assignment problem* (QAP) using a polysexual voting recombination operator. The PGA was implemented on QAPs of size 30 and 36 and for TSPs of size 40 and 50 with known solutions. The algorithm found a new optimum for the Steinberg's problem (QAP of size 36). The numbers of processors used to run this problem were 16, 32, and 64. The 64 processor implementation (on a system with distributed memory) gave by far the best results in terms of computational time.

Furthermore, Battiti and Tecchiolli [5] presented parallelization schemes of genetic algorithms and tabu search for combinatorial optimization problems and in particular for *quadratic assignment* and *N-k problems* giving indicative experimental results.

Tanese [53] presents a parallel genetic algorithm implemented on a 64-processor NCUBE/six hypercube. Each processor runs the algorithm on each own subpopulation (coarse-grained PGA), and sends in an adjustable frequency a portion of good individuals to one of its neighboring processors. Each exchange takes place along a different dimension of the hypercube. The PGA is applied to a function optimization problem and its performance is compared with the corresponding GA, which is found to be similar. In addition, the PGA achieves comparable results with near linear-speedup.

Pettey et al. [43] proposed an "island model" that restricts the communication among the subpopulations to some adjacent interconnected subpopulations [30]. The PGA was tested on four of DeJong's testbed of functions [14]. Population sizes of 50 to 800 were used, with the best answer in terms of quality solution, corresponding to size 50, which is approximately the theoretical optimal population size. The algorithm was implemented on an Intel *iPSC*, a message-based multiprocessor system with a binary *n-cube* interconnection network.

More recently, Lee and Kim [30] developed PGAs, based on the island model, for solving job scheduling problems with generally weighted earliness and tardiness penalties (GWET), satisfying specific properties, such as the *V-shaped schedule* around the due date. A binary representation scheme is used to code the job schedules into chromosomes. A GA is developed by parallelizing the population into subgroups, each of which keeps its distinct feasible schedules. The initial population is constructed so that the resulting genotype sequence satisfies the *V-shaped schedule*. Two different reproduction methods are employed, the *roulette wheel selection* and the *N-best selection method*. Also, the *crossover* and *mutation* operators are implemented in parallel. Several instances of problems with subpopulations of 30 chromosomes (jobs) and total population size of 100 to 200 where used to evaluate the efficiency of the PGA. The authors report that the roulette wheel selection scheme performs far better than the N-best selection in terms of quality solution, though the N-best selection gives good results in terms of CPU time. The mutation operator seems to improve the performance of the PGA. The paper concludes, showing the superior performance of the parallel algorithm when compared to the sequential GA. In terms of CPU time, the parallelization of the population into several groups speeds up the convergence of the GAs as the size of the problem increases.

Parallel genetic algorithms have been applied to the graph partition problem [29], scheduling problems [11], and global optimization problems [49].

3 Parallel Simulated Annealing

Since the simulated annealing method (SA) was proposed several years ago by Kirpatrick et al. [27], based on the pioneering work of Metropolis et al. [32], much research has been accomplished regarding its implementation (sequential and parallel) to solve a variety of difficult combinatorial optimization problems. Simulated annealing is based on the analogy between statistical mechanics and combinatorial optimization. The term *annealing* refers to the process of a thermal system by first melting at high temperatures and then lowering the temperature slowly based on an annealing schedule, until the vicinity of the solidification temperature is reached, where the system is allowed to reach the *ground state* (the lowest energy state of the system). Simulated annealing is a simple Monte Carlo approach to simulate the behavior of this system to achieve thermal equilibrium at a given temperature in a given annealing schedule. This analogy has been applied in solving combinatorial optimization problems. Given an objective function $f(x)$ over a feasible domain D, a generic simulated annealing algorithm for finding the global minimum of $f(x)$ is given in Figure 2.

An introduction to general concepts of parallel simulated annealing techniques can be found in Aarts and Korst [1]. Several parallel algorithms based on SA have been implemented for a variety of combinatorial optimization problems. In the context of annealing, a parallel implementation can be presented in two forms [47]: (1) *functional parallelism*, which uses multiple processors to evaluate different phases of a single move, (2) *data parallelism*, which uses different processors or group of processors to propose and evaluate moves independently. The second form has the advantage of "easily scaling the algorithm to large ensembles of processors."

The *standard-cell approach* is a semi-custom designing method in which functional building blocks called *cells* are used to construct a part of, or an entire, VLSI chip [47]. Two basic approaches have been applied to the placement problem [47] – *constructive*

```
Layout of Simulated Annealing

Input: A problem instance

Output: A (sub-optimal) solution

1. Generate an initial solution at random and initialize the temperature $T$.

2. while ($T > 0$) do
    (a) while (thermal equilibium not reached) do
        (i) Generate a neighbor state at random and evaluate the change in
            energy level $\Delta E$.

        (ii) If $\Delta E < 0$ update current state with new state.

        (iii) If $\Delta E \geq 0$ update current state with new state
              with probability $e^{\frac{-\Delta E}{K_B T}}$.

    (b) Decrease temperature $T$ according to annealing schedule.

3. Output the solution having the lowest energy.
```

Fig. 2. Layout of Simulated Annealing

methods and *iterative* methods. The SA technique belongs to the second group, as an approach that uses probabilistic hill climbing to avoid local minima. The `TimberWolf` program has been proposed as a version of SA implemented for the cell placement problem. It has been shown to provide enormous chip area savings compared to the already existed standard methods of cell layout. Jones and Banerjec [26] developed a parallel algorithm based on `TimberWolf`, where multiple cell moves are proposed and evaluated by using pairs of processors. The algorithm was implemented on a iPSC/1 Hypercube. Rose et al. [45] presented two parallel algorithms for the same problem. The *Heuristic Spanning* approach replaces the high temperature portion of simulated annealing, assigning cells to fixed sub-areas of the chip. The *Section Annealing* approach is used to speed-up the low temperature portion of simulated annealing. The placement here is geographically divided and the pieces are assigned to separate processors. Other parallel algorithms based on SA and implemented for the cell placement problem can be found in [8, 47, 55].

Greening [22] examines the *asynchronous parallel* SA algorithm in relation with the effects of calculation errors resulting from the parallel implementation or an approximate cost function. More analytically, the relative work analyzes the effects of *instantaneous* and *accumulated* errors. The first category contains the errors which result as the difference between the true and inaccurate costs computed at a given time. An accumulated error is the sum of a stream of instantaneous errors. The author proves a direct connection between the accumulated errors measured in previous research work, and annealing properties.

Most recently, Boissin and Lutton [6] proposed a new SA algorithm that can be efficiently implemented on a massively parallel computer. A comparison to the sequential algorithm has been presented by testing both algorithms on two classical combinatorial optimization problems: (1) the *quadratic sum assignment problem* and (2) the *minimization of an unconstrained* $0-1$ *quadratic function*. The numerical results showed that the parallel algorithm converges to high-quality suboptimal solutions. For the problem of minimization of quadratic functions with 1000 variables, the parallel implementation on a 16K Connection Machine was 3.3 times faster than the sequential algorithm implemented on a SPARC 2 workstation, for the same quality of solution.

4 Parallel Tabu Search

Tabu search (TS), first introduced by Glover [18, 19], is a heuristic procedure to find good solutions to combinatorial optimization problems. A *tabu list* is a set of solutions determined by historical information from the last t iterations of the algorithm, where t is fixed or is a variable that depends on the state of the search, or a particular problem. At each iteration, given the current solution x and its corresponding neighborhood $N(x)$, the procedure moves to the solution in the neighborhood $N(x)$ that most improves the objective function. However, moves that lead to solutions on the tabu list are forbidden, or are tabu. If there are no improving moves, TS chooses the move which least changes the objective function value. The tabu list avoids returning to the local optimum from which the procedure has recently escaped. A basic element of tabu search is the *aspiration* criterion, which determines when a move is admissible despite being on the tabu list. One *termination* criterion for the tabu procedure is a limit in the number of consecutive moves for which no improvement occurs. A more detailed description of the main features, aspects, applications and extensions of tabu search, can be found in [20]. Several implementations of parallel algorithms based on TS have been developed for classical optimization problems, such as TSP and QAP, which will be discussed below. Given an objective function $f(x)$ over a feasible domain D, a generic tabu search for finding an approximation of the global minimum of $f(x)$ is given in Figure 3.

Taillard [52] presents two implementations of parallel TS for the quadratic assignment problem. Computational results on instances of size up to 64 are reported. The form of TS considered is claimed to be more robust than earlier implementations, since it requires less computational effort and uses only the basic elements of TS (moves to neighboring solutions, tabu list, aspiration function). The neighborhood used is a *2-exchange neigborhood*. The tabu list is made up of pairs (i, j) of interchanged units both of which are placed at locations they had occupied within the last s iterations, where s is the size of tabu list. The size s changes its value randomly in an interval during the search. The *aspiration criterion* is introduced to allow the tabu moves to be chosen, if both interchanged units are assigned to locations they have not occupied within the t most recent iterations. In the first method, the neighborhood is divided into p parts of

```
Layout of Tabu Search

Input: A problem instance

Output: A (sub-optimal) solution

1. Initialization:
    (a) Generate an initial solution $x$ and set $x^* = x$;
    (b) Initialize the tabu list $T = \emptyset$;
    (c) Set iteration counters $k = 0$ and $l = 0$;

2. while $(N(x) \setminus T \neq \emptyset)$ do
    (a) $k = k + 1$; $l = l + 1$;
    (b) Select $x$ as the best solution from set $N(x) \setminus T$;
    (c) If $f(x) < f(x^*)$ then update $x^* = x$ and set $l = 0$;
    (d) If $k = \overline{k}$ or if $l = \overline{l}$ go to step 3;

3. Output the best solution found $x^*$;
```

Fig. 3. Layout of Tabu Search

approximately the same size. Each part is distributed for evaluation to one of p different processors. Using a number of processors proportional to the size of the problem (precisely $p = n/10$), the complexity is reduced by a factor of n, where n is the size of the problem. Moreover, the computational results showed improvement to the quality of solution for many large problems and the best published solutions of other problems have been found. The second method performs independent searches, each of which starts with a different initial solution. The parallel algorithm was implemented on a ring of 10 transputers (T800C-G20S). The computational complexity of this algorithm is less than that of earlier TS implementations for QAP by a factor n.

Another parallel tabu search algorithm for solving the quadratic assignment problem has been developed by Chakrapani and Skorin-Kapov [9]. The algorithm includes elements of TS, such as aspiration criterion, dynamically changing tabu list sizes and long-term memory [18]. A new intensification strategy is proposed, based on the intermediate term memory, restricting the searches in a neighborhood. This results in less amount of computational effort during one iteration since the procedure does not examine the entire neighborhood. A massively parallel implementation was tested on the Connection Machine CM-2 for large QAPs of size ranging from $n = 42$ to 100, using n^2 processors. The new tabu strategy gave good results in terms of quality of solutions. For problems up to size 90, it obtained the best known solutions or close to those. For size $n = 100$, every previously best solution found, was improved upon.

Battiti and Tecchiolli [5] describe a parallelization scheme for tabu search called *reactive* tabu scheme, and apply it to the quadratic assignment problem and the *N*-*k* problem with the objective of achieving a speedup in the order of the number of processors. In the reactive tabu search, each processor executes and independent search. Furthermore, the same problems are solved by a parallel genetic algorithm in which the interaction between different search processes is strong because the generation of new candidate points depends on the consideration of many members of the population.

A tabu search for the traveling salesman problem has been proposed by Fiechter [17], and was tested on instances having 500 to 10000 vertices. A new estimation of the asymptotic normalized length of the shortest tour through points uniformly distributed in the unit square is given. Numerical results and speedups obtained by the implementation of the algorithm on a network of transputers show the efficiency of the parallel algorithm.

5 Parallel GRASP

A GRASP [16] is an iterative process for finding approximate solutions to combinatorial optimization. Let, as before, $f(x)$ denote the objective function to be minimized over a feasible domain D. A generic layout of GRASP is given in Figure 4. The GRASP

```
Layout of GRASP

Input: A problem instance

Output: A (sub-optimal) solution

1. Initialization: set $x^* = \infty$;
2. while (stopping criterion not satisfied) do
     (a) Construct a greedy randomized solution $x$;
     (b) Find local minimum $\tilde{x}$ in neighborhood $N(x)$ of $x$;
     (c) If $f(\tilde{x}) < f(x^*)$ then update $x^* = \tilde{x}$;
3. Output the best solution found $x^*$;
```

Fig. 4. Layout of GRASP

iterations terminate when some stopping criterion, such as maximum number of iterations, is satisfied. Each iteration consists of a construction phase, and a local search phase. If an improved solution is found, then the incumbent is updated. A high-level description of these two phases is given next.

In the construction phase, a feasible solution is built up, one element at a time. The choice of the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function. An element from the candidate list is randomly chosen from among the best candidates in the list, but is not necessarily the top candidate. Figure 5 displays a generic layout for the construction phase of a GRASP. The solution set S is initialized empty, and the construction iterations are repeated until

```
Layout of GRASP Construction Phase

Input: A problem instance and pseudo random number stream

Output: A (sub-optimal) solution

1. Initialization: set solution $S = \emptyset$;
2. while (solution construction not done) do
    (a) Using greedy function, make restricted candidate list (RCL);
    (b) At random, select element $s$ from RCL;
    (c) Place $s$ in solution, i.e. $S = S \cup \{s\}$;
    (d) Change greedy function to take into account updated $S$;
3. Output the solution $x$ corresponding to set $S$;
```

Fig. 5. Layout of GRASP Construction Phase

the solution is built. To do this, the restricted candidate list is setup. This candidate list contains a subset of candidates that are well ordered with respect to the greedy function. A candidate selected from the list at random is added to the solution. The greedy function is adapted to take into account the selected element.

The solutions generated in the construction are not guaranteed to be locally optimal and therefore local search is applied to produce a locally optimal solution. Figure 6 illustrates a generic local search procedure.

GRASP can be easily implemented on an MIMD multi-processor environment. Each processor can be initialized with its own copy of the procedure, the problem data, and independent pseudo-random number sequences. The GRASP iterations are then independently executed in parallel and the best solution found over all processors is the GRASP solution. This approach was implemented by Pardalos, Pitsoulis, and Resende [39] to approximately solve instances of the QAP of dimension up to $n = 128$. On a Kendall Square KRS-1 parallel computer with 128 processors (of which only 64 were utilized), the authors were able to achieve on average a speedup factor of 62 and speedup factors of 130 and 221 on single instances of dimensions $n = 30$ and $n = 22$, respectively.

Another approach to implement GRASP parallel was proposed by Feo, Resende, and Smith [15], where the problem being optimized is decomposed into many smaller problems, and each processor is given a set of small problems, that are solved with

```
Layout of GRASP Local Search Phase

Input: A problem instance, solution x, neighborhood $N(x)$

Output: A locally-optimal solution $\tilde{x}$

1. while ($x$ not locally optimal) do
    (a) Find $\tilde{x} \in N(x)$ with $f(\tilde{x}) < f(x)$;
    (b) Update $x = \tilde{x}$;
3. Output the locally optimal solution $\tilde{x}$;
```

Fig. 6. Layout of GRASP Local Search Phase

GRASP one after the other. Using this approach to approximately solve large instances of the maximum independent set problem, the authors were able to achieve almost linear speedup on an eight processor Alliant FX/80 MIMD computer.

6 Concluding Remarks

In the last ten years, we have witnessed an explosion and availability of parallel, multiprocessing computers. Since most of the interesting combinatorial optimization problems are $\mathcal{NP}$-hard, it is quite natural to consider implementing algorithms on such environments. In this brief survey, we summarized recent developments in parallel implementations of several classes of new heuristics for combinatorial optimization.

References

1. E. AARTS AND J. KORST, *Simulated Annealing and Boltzmann Machines – A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley and Sons, 1989.
2. S.G. AKL, D.T. BARNARD AND R.J. DORAN, *Design, Analysis, and Implementation of a Parallel Tree Search Algorithm*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-4 (1982), pp. 192–203.
3. I. ALTH, *A Parallel Game Tree Search Algorithm with a Linear Speedup*, Journal of Algorithms 15 (1993), pp. 175–198.
4. G.Y. ANANTH, V. KUMAR AND P. M. PARDALOS, *Parallel Processing of Discrete Optimization Problems*, In `Encyclopedia of Microcomputers` Vol. 13 (1993), pp. 129–147, Marcel Dekker Inc., New York.
5. R. BATTITI AND G. TECCHIOLLI, *Parallel Biased Search for Combinatorial Optimization: Genetic Algorithms and TABU*, Microprocessors and Microsystems 16 (1992), pp. 351–367.
6. N. BOISSIN AND J.-L. LUTTON, *A Parallel Simulated Annealing Algorithm*, Parallel Computing, 19 (1993), pp. 859–872.

7. R.J. Brouwer, P. Banerjee, *A Parallel Simulated Annealing Algorithm for Channel Routing on a Hypercube Multiprocessor*, Proceedings of 1988 IEEE International Conference on Computer Design, pp. 4–7.

8. A. Casotto and A. Sanngiovanni-Vincentelli, *Placement of Standard Cells Using Simulated Annealing on the Connection Machine*, Proceedings ICCAD, 1987, pp. 350–352.

9. J. Chakrapani and J. Skorin-Kapov, *Massively Parallel Tabu Search for the Quadratic Assignment Problem*, Annals of Operation Research, 41 (1993), pp.327–341.

10. R.D. Chamberlain, M.N. Edelman, M.A. Franklin, E.E. Witte, *Simulated Annealing on a Multiprocessor*, Proceedings of 1988 IEEE International Conference on Computer Design, pp. 540–544.

11. G.A. Cleveland and S.F. Smith, *Using Genetic Algorithms to Schedule Flow Shop Releases*, Proceeding of the Third International Conference on Genetic Algorithms, (1990), Morgan Kaufmann, Los Altos, CA.

12. J.P. Cohoon, S.U. Hegde, W.N. Martin and D. Richards, *Punctuated Equilibria: A Parallel Genetic Algorithm*, Proceedings of the Second International Conference on Genetic Algorithms and their Applications, J.J. Grefenstette (editor), July 1987, pp. 148–154.

13. D. Cvijovic and J. Klinowski, *Taboo Search: An Approach to the Multiple Minima Problem*, Science, 267 (1995), pp. 664–666.

14. K.A. De Jong, *An Analysis fo the Behavior of a Class of Genetic Adaptive Systems*, Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan, 1975.

15. T.A. Feo and M.G.C. Resende and S.H. Smith, *A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set*, Operations Research, 42 (1994), pp. 860–878.

16. T.A. Feo and M.G.C. Resende, *Greedy Randomized Adaptive Search Procedures*, Journal of Global Optimization, 6 (1995), pp. 109–133.

17. C.-N. Fiechter, *A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems*, Discrete Applied Mathematics, 51 (1994), pp. 243–267.

18. F. Glover, *Tabu Search. Part I*, ORSA J. Comput., 1 (1989), pp. 190–206.

19. F. Glover, *Tabu Search. Part II*, ORSA J. Comput., 2 (1990), pp. 4–32.

20. F. Glover, E. Taillard and D. de Werra, *A User's Guide to Tabu Search*, Annals of Operation Research, 41 (1993), pp. 3–28.

21. M. Gorges-Schleuter, *ASPARAGOS: A Parallel Genetic Algorithm and Population Genetics*, Lecture Notes on Computer Science, Vol. 565 (1989), pp. 407–518.

22. D.R. Greening, *Asynchronous Parallel Simulated Annealing*, Lectures in Complex Systems, Vol. 3 (1990), pp. 497–505.

23. J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

24. S. R. Huang and L.S. Davis, *Speedup Analysis of Centralized Parallel Heuristic Search Algorithms*, Proceedings of the International Conference on Parallel Processing, Vol. 3. Algorithms and Applications (1990), pp. 18–21.

25. P. Jog, J.Y. Suh and D. Van Gucht, *Parallel Genetic Algorithms Applied to the Traveling Salesman Problem*, SIAM Journal of Optimization, 1 (1991), pp.515–529.

26. M. Jones and P. Banerjee, *Performance of a Parallel Algorithm for Standard Cell Placement on the Intel Hypercube*, Proceedings of the 24th Design Automation Conference, 1987, pp. 807–813.

27. S. Kirkpatrick, C.D. Gellat Jr. and M.P. Vecchi, *Optimization by Simulated Annealing*, Science, 220 (1983), pp. 671–680.

28. V. KUMAR AND L.N. KANAL, *Parallel Branch-and-Bound Formulations for AND/OR Tree Search*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-6 (1984), pp. 768–778.
29. G. VON LASZEWSKI, *Intelligent Structural Operators for K-Way Graph Partitioning Problem*, Proceeding of the Fourth International Conference on Genetic Algorithms, (1991), Morgan Kaufmann, San Mateo, CA.
30. C.Y. LEE AND S.J. KIM, *Parallel Genetic Algorithms for the Earliness-Tardiness Job Scheduling Problem with General Penalty Weights*, Computers & Ind. Eng., 28 (1995), pp. 231–243.
31. A. MAHANTI AND C.J. DANIELS, *A SIMD Approach to Parallel Heuristic Search*, Artificial Intelligence, 10 (1993), pp. 243–282.
32. N. METROPOLIS AND A. ROSENBLUTH AND M. ROSENBLUTH AND A. TELLER AND E. TELLER, *Equation of State Calculations by Fast Computing Machines*, Journal of Chemical Physics, 21 (1953), pp. 1087–1092.
33. H. MUHLENBEIN, *Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization*, `Lecture Notes in Computer Science`, Vol. 565 (1989), pp. 398–406.
34. H. MUHLENBEIN, M. SCHOMISCH AND J. BORN, *The Parallel Genetic Algorithm as Function Optimizer*, Proceedings on an International Conference on Genetic Algorithms, (1991).
35. T.A. MARSLAND AND F. POPOWICH, *Parallel Game-Tree Search*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-7 (1985), pp. 442–452.
36. P. M. PARDALOS, Y. LI AND K. A. MURTHY, *Computational Experience with Parallel Algorithms for Solving the Quadratic Assignment Problem*, In `Computer Science and Operations Research: New Developments in their Interface`, O. Balci, R. Sharda, S.A. Zenios (eds.), Pergamon Press, pp. 267–278 (1992).
37. P. M. PARDALOS, AND G. GUISEWITE, *Parallel Computing in Nonconvex Programming*, `Annals of Operations Research` 43 (1993), pp. 87–107.
38. P. M. PARDALOS, A. T. PHILLIPS AND J. B. ROSEN, *Topics in Parallel Computing in Mathematical Programming*, Science Press, 1993.
39. P. M. PARDALOS, L. S. PITSOULIS AND M.G.C. RESENDE, *A Parallel GRASP Implementation for the Quadratic Assignment Problem*, In Proceedings of *Parallel Algorithms for Irregularly Structured Problems (Irregular '94)*, Kluwer Academic Publishers (1995).
40. P.M. PARDALOS, M.G.C. RESENDE, AND K.G. RAMAKRISHNAN (Editors), *Parallel Processing of Discrete Optimization Problems*, DIMACS Series Vol. 22, American Mathematical Society, (1995).
41. P.M. PARDALOS AND H. WOLKOWICZ (Editors), *Quadratic Assignment and Related Problems*, DIMACS Series Vol. 16, American Mathematical Society (1994).
42. C. PETERSON, *Parallel Distributed Approaches to Combinatorial Optimization: Benchmark Studies on Traveling Salesman Problem*, Neural Computation, Vol. 2, (1990), pp. 261–269.
43. C.B. PETTEY, M.R. LEUZE AND J.J. GREFENSTETTE, *A Parallel Genetic Algorithm*, Proceedings of the Second International Conference on Genetic Algorithms and their Applications, J.J. Grefenstette (editor), July 1987, pp. 155–161.
44. C. POWLER, C. FERGUSON AND R. E. KORF, *Parallel Heuristic Search: Two Approaches*, In `Parallel Algorithms for Machine Intelligence and Vision`, V. Kumar, P.S. Gopalakrishnan and L.N. (eds.), Springer-Verlag, pp. 42–65 (1990).
45. J.S. ROSE, W.M. SNELGROVE AND Z.G. VRANESIC, *Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing*, IEEE Transactions on Computer-Aided Design, Vol. 7 (1988), pp. 387–396.
46. sc A.V. Sannier and E.D. Goodman, *Genetic Learning Procedures in Distributed Environments*, Proceedings of the Second International Conference on Genetic Algorithms and their Applications, J.J. Grefenstette (editor), July 1987, pp. 162–169.

47. J.S. SARGENT, *A Parallel Row-Based Algorithm with Error Control for Standard-Cell Placement on a Hypercube Multiprocessor*, Thesis, University of Illinois, Urbana-Illinois, 1988.
48. B. SHIRAZI, M. WANG AND G. PATHAK, *Analysis and Evaluation of Heuristic Methods for Static Task Scheduling*, Journal of Parallel and Distributed Computing 10 (1990), pp. 222–232.
49. P.S. DE SOUZA, *Asynchronous Organizations for Multi-Algorithm Problems*, Ph.D. Thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1993.
50. R. SHONKWILER AND E.V. VLECK, *Parallel Speed-Up of Monte Carlo methods for Global Optimization*, Journal of Complexity 10 (1994), pp. 64–95.
51. J. SUH AND D. VAN GUCHT, *Distributed genetic Algorithms*, Tech. Report 225, Computer Science Department, Indiana University, Bloomington, IN, July 1987.
52. E. TAILLARD, *Robust Taboo Search for the Quadratic Assignment Problem*, Parallel Computing, 17 (1991), pp. 443–445.
53. R. TANESE, *Parallel Genetic Algorithm for a Hypercube*, Proceedings of the Second International Conference on Genetic Algorithms and their Applications, J.J. Grefenstette (editor), July 1987, pp. 177–183.
54. N.L.J. ULDER, E.H.L. AARTS, H. -J. BANDELT, P.J.M. VAN LAARHOVEN AND E. PESCH, *Genetic Local Search Algorithms for the Traveling Salesman Problem*, In Lecture Notes in Computer Science, Parallel Problem Solving from Nature-Proceedings of 1st Workshop, PPSN 1, Vol. 496 (1991), pp. 109–116.
55. C.-P. WONG AND R.-D. FIEBRICH, *Simulated Annealing-Based Circuit Placement on the Connection Machine System*, Proceedings of International Conference on Computer Design (ICCD '87), 1987, pp. 78–82.