Modern Iterative Algorithms and their Applications in Computer Engineering

Sadiq M. Sait and Habib Youssef

1 Introduction

Combinatorial optimization problems are encountered everywhere. The advent of the digital computer is credited to the explosion in the number of algorithmic solutions to combinatorial optimization problems. Such solution techniques were unthinkable before this magnificent invention.

In this chapter we are concerned with one class of combinatorial optimization algorithms: *general iterative non-deterministic algorithms.* The growing interest in this class of algorithms is attributed to their generality, ease of implementation, and mainly, the many success stories reporting very positive results. We shall limit ourselves to five dominant iterative non-deterministic algorithms, which, in order of popularity are: (1) Simulated Annealing (SA), (2) Genetic Algorithm (GA), (3) Tabu Search (TS), (4) Simulated Evolution (SimE), and (5) Stochastic Evolution (StocE). All five search heuristics have several important properties in common.

1. They are blind, in that they do not know when they reached the optimal solution. Therefore they must be told when to stop.
2. They are approximation algorithms, that is, they do not guarantee finding an optimal solution.
3. They have 'hill climbing' property, that is, they occasionally accept uphill (bad) moves.
4. They are easy to implement. All that is required is to have a suitable solution representation, a cost function, and a mechanism to traverse the search space.
5. They are all '*general*'. Practically they can be applied to solve any combinatorial optimization problem.
6. They all strive to exploit domain specific heuristic knowledge to bias the search toward "good" solution subspace. The quality of subspace searched depends to a large extent on the amount of heuristic knowledge used.

7. Although they asymptotically converge to an optimal solution, the rate of convergence is heavily dependent on the adequate choice of several parameters.

The last two properties are the hidden bone in the five combinatorial optimization strategies. Our goal in this chapter is to briefly introduce these five heuristics. The chapter is organized into 10 sections. In the following 5 sections, we introduce the five iterative algorithms. Only an intuitive discussion and an essence of the heuristic is given. The remaining sections deal with convergence aspects of the heuristics, their parallel implementation, applications, and conclusion.

2 Simulated Annealing

Simulated Annealing is one of the most well developed and widely used iterative techniques for solving optimization problems. It is a general *adaptive* heuristic and belongs to the class of *non-deterministic* algorithms [NSS89]. It has been applied to several combinatorial optimization problems from various fields of science and engineering.

One typical feature of simulated annealing is that, besides accepting solutions with improved cost, it also, to a limited extent, accepts solution with deteriorated cost. It is this feature that gives the heuristic the hill climbing capability. Initially the probability of accepting inferior solutions (those with larger costs) is large; but as the search progresses, only smaller deteriorations are accepted, and finally only good solutions are accepted.

A strong feature of the simulated annealing heuristic is that it is both effective and robust. Regardless of the choice of the initial configuration it produces high quality solutions. It is also relatively easy to implement.

The term *annealing* refers to heating a solid to a very high temperature (whereby the atoms gain enough energy to break the chemical bonds and become free to move), and then slowly cooling the molten material in a controlled manner until it crystallizes. By cooling the metal at a proper rate, atoms will have an increased chance to regain proper crystal structure with perfect lattices. During this annealing procedure the free energy of the solid is *minimized.*

In the early eighties, a correspondence between annealing and combinatorial optimization was established, first by Kirkpatrick, Gelatt and Vecchi [KCGV83] in 1983, and independently by Černy [Čer85] in 1985. These scientists observed that a solution in combinatorial optimization is equivalent to a *state* in the physical system and the cost of the solution is analogous to the *energy* of that state. As a result of this analogy they introduced a solution method in the field of combinatorial optimization. This method is thus based on the simulation of the physical annealing process, and hence the name *simulated annealing* [KCGV83, Čer85].

Every combinatorial optimization problem may be discussed in terms of a *state space*. A *state* is simply a configuration of the combinatorial objects involved. For example, consider the problem of partitioning a graph of $2n$ nodes into two equal sized subgraphs such that the number of edges with vertices in both subgraphs is minimized. In this problem, any division of $2n$ nodes into two equal sized blocks is a configuration. There is a large number of such configurations. Only some of these correspond to global optima, i.e., states with optimum cost.

An iterative improvement scheme starts with some given state, and examines a *local neighborhood* of the state for better solutions. A local neighborhood of a state S, denoted by $\aleph(S)$, is the set of all states which can be reached from S by making a small change to S. For instance, if S represents a two-way partition of a graph, the set of all partitions which are generated by swapping two nodes across the partition represents a local neighborhood. The iterative improvement algorithm moves from the current state to a state in the local neighborhood if the latter has a better cost. If all the local neighbors have larger costs, the algorithm is said to have *converged* to a *local optimum*. This is illustrated in Figure 1.

Here, the states are shown along the x-axis, and it is assumed that two consecutive states are local neighbors. It is further assumed that we are discussing a *minimization* problem. The cost curve is *non-convex*, i.e., it has multiple minima. A greedy iterative improvement algorithm may start off with an initial solution such as **S** in Figure 1, then slide along the curve and find a local minimum such as **L**. There is no way such an algorithm can find the global minimum **G** of Figure 1, unless it "climbs the hill" at the local minimum **L**. In other words, an algorithm which occasionally accepts inferior solutions can escape from getting trapped in a local optimum. Simulated annealing is such a hill-climbing algorithm.

During annealing, a metal is maintained at a certain temperature T for a pre-computed amount of time, before reducing the temperature in a controlled manner. The atoms have a greater degree of freedom to move at higher temperatures than at lower temperatures. *The movement of atoms is analogous to the generation of new neighborhood states in an optimization process.* In order to simulate the annealing process, much flexibility is allowed in neighborhood generation at higher "temperatures", i.e., many 'uphill' moves are permitted at higher temperatures. The temperature parameter is lowered gradually as the algorithm proceeds. As the temperature is lowered, fewer and fewer uphill moves are permitted. In fact, at absolute zero, the simulated annealing algorithm turns greedy, allowing only downhill moves.

The simulated annealing algorithm is shown in Figure 2. The core of the algorithm is the *Metropolis* procedure, which simulates the annealing process at a given temperature T (Figure 3) [M$^+$53]. The *Metropolis* procedure receives as input the current temperature T, and the current solution $CurS$ which it improves through local search. Finally, *Metropolis* must also be provided with the value M, which is the amount of time for which annealing must be applied at temperature T. The procedure *Simulated_annealing* simply invokes *Metropolis* at decreasing temperatures. Temperature is initialized to a value T_0 at the beginning of the procedure, and is reduced in a controlled manner (typically in a geometric progression); the parameter α is used to achieve this cooling. The amount of time spent in annealing at a temperature is gradually *increased* as temperature is lowered. This is done using the parameter $\beta > 1$. The variable *Time* keeps track of the time being expended in each call to the *Metropolis*. The annealing procedure halts when *Time* exceeds the allowed time.

The *Metropolis* procedure is shown in Figure 3. It uses the procedure *Neighbor* to generate a local neighbor $NewS$ of any given solution S. The function *Cost* returns the cost of a given solution S. If the cost of the new solution $NewS$ is better than the cost of the current solution $CurS$, then the new solution is accepted, and we do so by setting $CurS$=$NewS$. If the cost of the new solution is better than the best solution ($BestS$) seen thus far, then we also replace $BestS$ by $NewS$. If the new solution has a higher cost in comparison to the original solution $CurS$, *Metropolis* will accept the new solution on a *probabilistic* basis. A random number is generated in the range 0 to 1. If this random number is smaller than $e^{-\Delta Cost/T}$, where $\Delta Cost$ is the difference in costs, ($\Delta Cost$= $Cost(NewS) - Cost(CurS)$), and T is the current temperature, the uphill solution is accepted. This criterion for accepting the new solution is known as the *Metropolis criterion*. The *Metropolis* procedure generates and examines M solutions.

The probability that an inferior solution is accepted by the *Metropolis* is given by $P(RANDOM < e^{-\Delta Cost/T})$. The random number generation is assumed to follow a *uniform distribution*. Remember that $\Delta Cost > 0$ since we have assumed that $NewS$ is uphill from $CurS$. At very high temperatures, (when $T \to \infty$), $e^{-\Delta Cost/T} \simeq 1$, and hence the above probability approaches 1. On the contrary, when $T \to 0$, the probability $e^{-\Delta Cost/T}$ falls to 0.

In order to implement simulated annealing on a digital computer we need to formulate a suitable cost function for the problem being solved. In addition, as in the case of local search techniques we assume the existence of a neighborhood structure, and need *perturb* operation or *Neighbor* function to generate new states (neighborhood states) from current states. And finally, we need a control parameter to play the role of temperature and a random number generator. The actions of simulated annealing are best illustrated with the help of an example. For the solution of the two way partitioning problem using SA, please refer to [SY95c].

In practice, simulated annealing is only run for a finite amount of time. A finite time implementation can be realized by generating homogeneous Markov chains of finite lengths for a sequence of decreasing values of temperature. To achieve this, a set of parameters that govern the convergence of the algorithm must be specified. This set of parameters is commonly referred to as the "cooling schedule" [AK89, OvG89, KCGV83]. It is customary to determine the cooling schedule by trial and error. However, some researchers have proposed cooling schedules that rely on some mathematical rigor. For a discussion on cooling schedule, and SA requirements the reader is refered to [SY99].

3 Genetic Algorithms

Genetic Algorithm (GA), is a powerful, domain-independent, search technique that was inspired by Darwinian theory. It emulates the natural process of evolution to perform an efficient and systematic search of the solution space to progress toward the optimum. It is based on the theory of *natural selection* that assumes that individuals with certain characteristics are more able to survive, and hence pass their characteristics to their offsprings.

Genetic algorithm is an *adaptive* learning heuristic. Similar to simulated annealing, it also belongs to the class of general *non-deterministic* algorithms. Several variations of the basic algorithm (modified to adapt to the problem at hand) exist. We will henceforth refer to this set as genetic algorithms (in plural).

Genetic algorithms (GAs) were invented by John Holland and his colleagues [Hol75] in the early 1970s. Holland incorporated features of natural evolution to propose a *robust*, computationally simple, and yet powerful technique for solving difficult optimization problems.

Genetic algorithms (GAs) operate on a *population* (or set) of *individuals* (or solutions) encoded as strings. These strings represent points in the search space. In each iteration, referred to as a generation, a new set of strings that represent solutions (called offsprings) is created by crossing some of the strings of the current generation [Gol89]. Occasionally new characteristics are injected to add diversity. GAs combine information exchange along with *survival of the fittest* among individuals to conduct the search.

When employing GAs to solve a combinatorial optimization problem one has to find an efficient representation of the solution in the form of a chromosome (encoded string). Associated with each chromosome is its *fitness value*. If we simulate the process of natural reproduction, combined with the biological principle of survival of the fittest, then, as each generation progresses, better and better individuals (solutions) with higher fitness values are expected to be produced.

Genetic algorithms, are both effective and *robust* [Dav91, Gol89, SY99]. Their main characteristics are listed below:

They work with coding of parameters: Therefore, one requirement when employing GAs is to find an *efficient* representation of the solution in the form of a chromosome (encoded string).

They search from a set of points: GAs simultaneously work from a rich collection of points (a population of solutions). Therefore, the probability of getting trapped in false valleys (in case of minimization problem) is reduced.

They only require objective function values: GAs are not limited by assumptions about the search space (such as continuity, existence of derivatives, etc.,), and they do not need or use any auxiliary information. They **only** require *objective* (cost) *function* values.

They are non-deterministic: GAs use probabilistic transition rules, not deterministic rules. Mechanism for choice of parents to produce offsprings, or for combining of genes in various chromosomes are probabilistic.

They are blind: They are blind in the sense that they do not know when they hit the optimum, and therefore they must be told when to stop.

The structure that encodes how the organism is to be constructed is called a *chromosome*. One or more chromosomes may be associated with each member of the population. The complete set of chromosomes is called a *genotype* and the resulting organism is called a *phenotype*. Similarly, the representation of a solution to the optimization problem in the form of an encoded string is termed as a *chromosome*. In most combinatorial optimization problems a single chromosome is generally sufficient to represent a solution, that is, the genotype and the chromosome are the same. The symbols that make up a chromosome are known as *genes*. The different values a gene can take are called *alleles*.

The fitness value of an individual (genotype or a chromosome) is a *positive* number that is a measure of its goodness. When the chromosome represents a solution to the combinatorial optimization problem, the fitness value indicates the cost of the solution. In the case of a minimization problem, solutions with lower cost correspond to individuals that are more fit.

GAs work on a population of solutions. An initial population constructor is required to generate a certain predefined number of solutions. The quality of the final solution produced by a genetic algorithm depends on the size of the population and how the initial population is constructed. The initial population generally comprises random solutions.

In GAs common genetic operators are *crossover* (χ) and *mutation*. They are derived by analogy from the biological process of evolution. Crossover operator is applied to pairs of chromosomes. The two individuals selected for crossover are called *parents*. Mutation is another genetic operator that is applied to a single chromosome. The resulting individuals produced when genetic operators are applied on the parents are termed as *offsprings*.

The choice of parents for crossover from the set of individuals that comprise the population is probabilistic. In keeping with the ideas of natural selection, we assume that stronger individuals, that is those with higher fitness values, are more likely to mate than the weaker ones. One way to simulate this is to select parents with a probability that is directly proportional to their fitness values.

Larger the fitness, the greater is chance of an individual being selected as one of the parents for crossover [Gol89].

There are several crossover operators that have been proposed in the literature. Depending on the combinatorial optimization problem being solved some are more effective than others. One popular crossover that will also help illustrate the concept is the *simple crossover*. It performs the "cut-catenate" operation. It consists of choosing a *random* cut point and dividing each of the two chromosomes into two parts. The offspring is then generated by catenating the segment of one parent to the left of the cut point with the segment of the second parent to the right of the cut point.

Mutation (μ) produces incremental random changes in the offspring by randomly changing allele values of some genes. In case of binary chromosomes it corresponds to changing single bit positions. It is not applied to all members of the population, but is applied probabilistically only to some. Mutation has the effect of perturbing a certain chromosome in order to introduce *new* characteristics not present in any element of the parent population. For example, in case of binary chromosomes, toggling some selected bit produces the desired effect.

Inversion is the third operator of GA and like mutation it also operates on a single chromosome. Its basic function is to laterally invert the order of alleles between two randomly chosen points on a chromosome.

A *generation* is an iteration of GA where individuals in the current population are selected for crossover and offsprings are created. Due to the addition of offsprings, the size of population increases. In order to keep the number of members in a population fixed, a constant number of individuals are selected from this set which consists of both the individuals of the initial population, and the generated offsprings. If $\mathbf{M}$ is the size of the initial population and N_o is the number of offsprings created in each generation, then, before the beginning of next generation, $\mathbf{M}$ new parents from $\mathbf{M} + N_o$ individuals are selected. A greedy selection mechanism is to choose the best $\mathbf{M}$ individuals from the total of $\mathbf{M} + N_o$. The complete pseudo code of a simple GA is given in Figure 4.

4 Tabu Search

In the previous section we discussed simulated annealing, which was inspired by the cooling of metals, and genetic algorithms, which imitate the biological phenomena of evolutionary reproduction. In this section we present a more recent optimization method called Tabu Search (TS) which is based on selected concepts of artificial intelligence (AI).

Tabu search was introduced by Fred Glover [Glo89, Glo90b, GTdW93, GL97] as a general iterative heuristic for solving combinatorial optimization problems. Initial ideas of the technique were also proposed by Hansen [Han86] in his *steepest ascent mildest descent* heuristic.

Tabu search is conceptually simple and elegant. It is a form of local neighborhood search. Each solution $S \in \Omega$ has an associated set of neighbors $\aleph(S) \subseteq \Omega$. A solution $S' \in \aleph(S)$ can be reached from S by an operation called a *move* to S'. Normally, the neighborhood relation is assumed symmetric. That is, if S' is a neighbor of S then S is a neighbor of S'.

Tabu search is a generalization of local search. At each step, the local neighborhood of the current solution is explored and the best solution in that neighborhood is selected as the new current solution. Unlike local search which stops when no improved new solution is found in the current neighborhood, tabu search continues the search from the best solution in the neighborhood even if it is worse than the current solution. To prevent cycling, information pertaining to the most recently visited solutions are inserted in a list called *tabu list*. Moves to tabu solutions are not allowed. The tabu status of a solution is overridden when certain criteria (aspiration criteria) are satisfied. One example of an aspiration criterion is when the cost of the selected solution is better than the best seen so far, which is an indication that the search is actually not cycling back, but rather moving to a new solution not encountered before.

Tabu search is a *metaheuristic*, which can be used not only to guide search in complex solution spaces, but also to direct the operations of *other* heuristic procedures. It can be superimposed on any heuristic whose operations are characterized as performing a sequence of *moves* that lead the procedure from one trial solution to another. In addition to several other characteristics, the attractiveness of tabu search comes from its ability to escape local optima.

Tabu search differs from simulated annealing or genetic algorithm which are "memoryless", and also from branch-and-bound, A* search, etc., which are rigid memory approaches. One of its features is its systematic use of *adaptive* (flexible) memory. It is based on very simple ideas with a clever combination of components, namely [Glo90a, SM93]:

1. a short-term memory component; this component is the core of the tabu search algorithm,

2. an intermediate-term memory component; this component is used for regionally **intensifying** the search, and,

3. a long-term memory component; this component is used for globally **diversifying** the search.

The central idea underlying tabu search is the exploitation of the above three memory components. Using the short-term memory, a *selective history* $\mathbf{H}$ of the states encountered is maintained to guide the search process. Neighborhood $\aleph(S)$ is replaced by a modified neighborhood which is a function of the history $\mathbf{H}$, and is denoted by $\aleph(\mathbf{H}, S)$. History determines which solutions may be reached by a move from S, since the next state S is selected from $\aleph(\mathbf{H}, S)$. The short-term memory component is implemented through a set of *tabu* conditions and the associated *aspiration criterion*.

The major idea of the short-term memory component is to classify certain search directions as tabu (or *forbidden*). By doing so we avoid returning to previously visited solutions. Search is therefore forced away from recently visited solutions, with the help of a short-term memory (*tabu list* $\mathbf{T}$). This memory contains *attributes* of some k most recent moves. The size of the tabu list denoted by k is the number of iterations for which a move containing that attribute is forbidden after it has been made. The tabu list can be visualized as a window on accepted moves as shown in Figure 5. The moves which tend to undo previous moves within this window are forbidden.

A flow chart illustrating the basic short-term memory tabu search algorithm is given in Figure 6. Intermediate-term and long-term memory processes are used to intensify and diversify the search respectively, and have been found to be very effective in increasing both quality and efficiency [Glo95, Glo96, DV93].

We first introduce the basic tabu search algorithm based on the short-term memory component. Following this we present some discussion on uses of intermediate and long-term memories.

An algorithmic description of a simple implementation of the tabu search is given in Figure 7. Refering to Figure 7, initially the current solution is the best solution. Copies of the current solution are perturbed with moves to get a set of new solutions. The best among these is selected and if it is not tabu then it becomes the current solution. If the move is tabu its aspiration criterion is checked. If it passes the aspiration criterion then it becomes the current solution. If the move to the next solution is accepted, then the move or some of its attributes are stored in the tabu list. Otherwise moves are regenerated to get another set of new solutions. If the current solution is better than the best seen thus far, then the best solution is updated. Whenever a move is accepted the iteration number is incremented. The procedure continues for a fixed number of iterations, or until some pre-specified stopping criterion is satisfied.

Tabu restrictions and aspiration criterion have a symmetric role. The order of checking for tabu status and aspiration criterion may be reversed, though most applications check if a move is tabu before checking for aspiration criterion [Glo90c]. For more discussion on move attributes, types of tabu Lists and the various tabu restrictions, the data structure to handle tabu-lists, and other aspiration criteria, the reader is referred to [SY99].

In many applications, the short-term memory component by itself has produced solutions superior to those found by alternative procedures, and usually the use of intermediate-term and long-term memory is bypassed. However, several studies have shown that intermediate and long-term memory components can improve solution quality and/or performance [MGPO89, Rya89, IE94, DV93].

The basic role of the intermediate-term memory component is to *intensify* the search. By its incorporation, the search becomes more aggressive. As the name suggests, memory is used to intensify the search. Intermediate-term memory component operates as follows. A selected number $m \gg |\mathbf{T}|$ (recall that $|\mathbf{T}|$

is the size of tabu list) of best trial solutions generated during a particular period of search are chosen and their features are recorded and compared. These solutions may be m consecutive best ones, or m local optimal solutions reached during the search. Features common to most of these are then taken and new solutions that contain these features are sought. One way to accomplish this is to restrict/penalize moves that remove such attributes. For example, in the TSP problem with moderately dense graphs, the number of different edges that can be included into any tour is generally a fraction of the total available edges (Why?). After some number of initial iterations, the method can discard all edges not yet incorporated into some tour. The size of the problem and the time per iteration now become smaller. The search therefore can focus on possibilities that are likely to be attractive, and can also examine many more alternatives in a given span of time.

The goal of long-term memory component is to *diversify* the search. The principles involved here are just the *opposite* of those used by the intermediate-term memory function. Instead of more intensively focusing the search with regions that contain previously found good solutions, the function of this component is to drive the search process into new regions that are different from those examined thus far.

Diversification using long-term memory in tabu search can be accomplished by creating an evaluator whose task is to take the search to new starting points [Glo89]. For example, in the TSP, a simple form of long-term memory is to keep a count of the number of times each edge has appeared in the tours previously generated. Then, an evaluator can be used to penalize each edge on the basis of this count, thereby favoring the generation of "other hopefully good" starting tours that tend to avoid those edges most commonly used in the past. This sort of approach is viewed as a **frequency** based tabu criterion in contrast to the **recency** based (tabu list) discussed earlier. Such a long-term strategy can be employed by means of a long-term tabu list (or any other appropriate data structure) which is periodically activated to employ tabu conditions of increased stringency, thereby forcing the search process into new territory [KLG94].

It is easy to create and test the short-term memory component first, and then incorporate the intermediate/long components for additional refinements.

Let a matrix entry $Freq(i,j)$ (i and j be movable or swappable elements) store the number of times swap (i,j) was made to take the solution from current state S to a new state S^*. We can then use this information to define a move evaluator $\mathcal{E}(\mathbf{H}, S)$, which is a function of both the cost of the solution, and the frequency of the swaps stored. Our objective is to diversify the search by giving more consideration to those swaps that have not been made yet, and to penalize those that frequently occured, that is given them less consideration [LG93]. Taking the above into consideration, the evaluation of the move can be expressed as follows:

$$\mathcal{E}(\mathbf{H}, S^*) = \begin{cases} Cost(S^*) & Cost(S^*) \leq Cost(S) \\ Cost(S^*) + \alpha \times \text{Freq}(i,j) & Cost(S^*) > Cost(S) \end{cases}$$

α is constant which depends on the range of the objective function values, the number of iterations, the span of history considered, etc. Its value (α's) is such that cost and frequency are appropriately balanced.

5 Simulated Evolution (SimE)

The Simulated Evolution algorithm (SimE) is a general search strategy for solving a variety of combinatorial optimization problems. The first paper describing Simulated Evolution (SimE) appeared in 1987 [KB87b]. Other papers by the same authors followed [KB89, KB90, KB91]. The SimE algorithm is usually confused with the Stochastic Evolution algorithm that will be the subject of the next section. However, as shall be seen, each algorithm has its own distinctive features. Distinctions among these two heuristics are the result of differences in the way they mimic the biological processes of evolution as well as in the way they adapt their parameters during the search.

The SimE algorithm starts from an initial assignment, and then, following an evolution-based approach, it seeks to reach better assignments from one generation to the next. SimE assumes that there exists a population P of a set M of n (movable) elements. In addition, there is a cost function $Cost$ that is used to associate with each assignment of movable element m a cost C_m. The cost C_m is used to compute the goodness (fitness) g_m of element m, for each $m \in M$. Furthermore, there are usually additional constraints that must be satisfied by the population as a whole or by particular elements. A general outline of the SimE algorithm is given in Figure 8.

SimE algorithm proceeds as follows. Initially, a population[1] is created at random from all populations satisfying the environmental constraints of the problem. The algorithm has one main loop consisting of three basic steps, *Evaluation*, *Selection*, and *Allocation*. The three steps are executed in sequence until the population average *goodness* reaches a maximum value, or no noticeable improvement to the population *goodness* is observed after a number of iterations. Another possible stopping criterion could be to run the algorithm for a prefixed number of iterations (see Figure 8). Below we present some details of the steps of the SimE algorithm.

Evaluation: The *Evaluation* step consists of evaluating the goodness of each individual i of the population P. The *goodness* measure must be a single number

[1] In SimE terminology, a population refers to a single solution. Individuals of the population are components of the solution; they are the movable elements.

expressible in the range $[0,1]$. *Goodness* is defined as follows:

$$g_i = \frac{O_i}{C_i} \tag{1}$$

where O_i is an estimate of the optimal cost of individual i, and C_i is the actual cost of i in its current location. The above equation assumes a minimization problem (maximization of goodness). Notice that, according to the above definition, the O_i's **do not** change from generation to generation, and therefore, are computed only once during the initialization step. Hence only the C_i's have to be recomputed at each call to the *Evaluation* function. Empirical evidence [Kli90] shows that the accuracy of the estimation of O_i is not very crucial to the successful application of SimE. However, the *goodness* measure must be strongly related to the target objective of the given problem.

Selection: The second step of the SimE algorithm is *Selection*. *Selection* takes as input the population P together with the estimated *goodness* of each individual, and partitions P into two disjoint sets, a selection set P_s and a set P_r of the remaining members of the population (see Figure 9). Each member of the population is considered separately from all other individuals. The decision whether to assign individual i to the set P_s or set P_r is based solely on its *goodness* g_i. The operator uses a selection function *Selection*, which takes as input g_i and a parameter B, which is a *selection bias*. Values of B are recommended to be in the range $[-0.2, 0.2]$. In many cases a value of $B = 0$ would be a reasonable choice.

The *Selection* function returns *true* or *false*. The higher the *goodness* value of the element, the higher is its chance of staying in its current location, i.e., unaltered in the next generation. On the other hand, the lower the *goodness* value, the more likely the corresponding element will be selected for alteration (mutation) in the next generation (will be assigned to the selection set P_s).

The *Selection* operator has a nondeterministic nature. An individual with a high *fitness* (*goodness* close to one) still has a non zero probability of being assigned to the selected set P_s. It is this element of nondeterminism that gives SimE the capability of escaping local minima.

For most problems, it is always beneficial to alter the elements of the population according to a deterministic order that is correlated with the objective function being optimized. Hence, in SimE, prior to the *Allocation* step, the elements in the selection set P_s are sorted. The sorting criterion is problem specific. Usually there are several criteria to choose from [SY99].

Allocation: *Allocation* is the SimE operator that has most impact on the quality of solution. *Allocation* takes as input the two sets P_s and P_r and generates a new population P' which contains all the members of the previous population P, with the elements of P_s mutated according to an allocation function *Allocation* (see Figure 10).

The choice of a suitable *Allocation* function is problem specific. The decision of the *Allocation* strategy usually requires more ingenuity on the part of the designer than the *Selection* scheme. The *Allocation* function may be a nondeterministic function which involves a choice among a number of possible mutations (moves) for each element of P_s. Usually, a number of *trial-mutations* are performed and rated with respect to their *goodnesses*. Based on the resulting goodnesses, a final configuration of the population P' is decided. The goal of *Allocation* is to favor improvements over the previous generation, without being too greedy.

The *Allocation* operation is a complex form of genetic *mutation* which is one of the genetic operations thought to be responsible for the evolution of the various species in biological environments. However, there is no need for a *crossover* operation as in GA since only one parent is maintained in all generations. However, since *mutation* is the only mechanism used by SimE for inheritance and evolution, it must be more sophisticated than the one used in GA.

Allocation alters (mutates) all the elements in the selected set P_s one after the other in a predetermined order. The order as well as the type of mutation are problem specific. For each individual e_i of the selected set P_s, W distinct trial alterations are attempted. The trial that leads to the best configuration (population) with respect to the objective being optimized is accepted and made permanent. Since the *goodness* of each individual element is also tightly coupled with the target objective, superior alterations are supposed to gradually improve the individual goodnesses as well. Hence, *Allocation* allows the search to progressively converge toward an optimal configuration where each individual is optimally located.

Initialization Phase: This step precedes the iterative phase. In this step, the various parameters of the algorithm are set to their desired values, namely, the maximum number of iterations required to run the main loop, the selection bias B, and the number of trial alterations W per individual. Furthermore, like any iterative algorithm, SimE requires that an initial solution be given. The convergence aspects of SimE are not affected by the quality of the initial solution. However, starting from a randomly generated solution usually increases the number of iterations required to converge to a near-optimal solution.

6 Stochastic Evolution (StocE)

Stochastic Evolution is a powerful general and randomized iterative heuristic for solving combinatorial optimization problems [SR89, SR90, SR91]. StocE algorithm is an instance of the class of general iterative heuristics discussed in [NSS85]. It is stochastic because the decision to accept a move is a probabilistic decision. Good moves, i.e., moves which improve the cost function are accepted with probability one, and bad moves may also get accepted with a non-zero probability. This feature gives Stochastic Evolution hill-climbing property. The word *evolution* is used in reference to the alleged evolution processes of biological species. Like simulated annealing and simulated evolution, stochastic evolution is conceptually simple and elegant. Actually stochastic evolution is somehow inspired in part by both simulated annealing and simulated evolution.

Combinatorial optimization problems can be modeled in a number of ways. StocE adopts the following generic model[SR90]:

Given a finite set M of movable elements and a finite set L of locations, a state is defined as a function $S : M \rightarrow L$ satisfying certain constraints.

Many of the combinatorial optimization problems can be formulated according to this model.

The Stochastic Evolution (StocE) algorithm seeks to find a suitable location $S(m)$ for each movable element $m \in M$, which eventually leads to a lower cost of the whole state $S \in \Omega$, where Ω is the state space. A general outline of the StocE algorithm is given in Figure 11.

The inputs to the StocE algorithm are:

1. an initial state (solution) S_0,
2. an initial value p_0 of the control parameter p, and
3. a stopping criterion parameter R.

Throughout the search, S holds *the current state (solution)*, while $BestS$ holds *the best state*. If the algorithm generates *a worse state*, a uniformly distributed random number in the range $[-p, 0]$ is drawn. The new uphill state is accepted if the magnitude of the loss is greater than the random number, otherwise the current state is maintained. Therefore, p is a function of the average magnitude of the uphill moves that the algorithm will tolerate. The parameter R represents the expected number of iterations the StocE algorithm needs until an improvement in the cost with respect to the best solution seen so far takes place, that is, until $CurCost \leq BestCost$. If R is too small, the algorithm will not have enough time to improve the initial solution, and if R is too large, the algorithm may waste too much time during the later generations. Experimental studies indicate that a value of R between 10 and 20 gives good results [Saa90].

Finally, the variable ρ is a counter used to decide when to stop the search. ρ is initialized to zero, and $R - \rho$ is equal to the number of remaining generations before the algorithm stops.

After initialization, the algorithm enters a **Repeat** loop **Until** the counter ρ exceeds R. Inside the **Repeat** body, the cost of the current state is first calculated and stored in $PrevCost$. Then, the **PERTURB** function (see Figure 12) is invoked to make a compound move from the current state S. **PERTURB** scans the set of movable elements M according to some apriori ordering and attempts to move every $m \in M$ to a new location $l \in L$. For each trial move, a new state S' is generated, which is *a unique* function $S' : M \to L$ such that $S'(m) \neq S(m)$ for some movable object $m \in M$. To evaluate the move, the gain function $Gain(m) = Cost(S) - Cost(S')$ is calculated. If the calculated gain is greater than some randomly generated integer number in the range $[-p, 0]$, the move is accepted and S' replaces S as the current state[2]. Since the random number is ≤ 0, moves with positive gains are always accepted. After scanning all the movable elements $m \in M$, the **MAKE_STATE** routine makes sure that the final state satisfies the state constraints. If the state constraints are not satisfied then **MAKE_STATE** *reverses* the fewest number of latest moves until the state constraints are satisfied. This procedure is required when perturbation moves that violate the state constraints are accepted.

The new state generated by **PERTURB** is returned to the main procedure as the current state, and its cost is assigned to the variable $CurCost$. Then the routine **UPDATE** (Figure 13) is invoked to compare the previous cost ($PrevCost$) to the current cost ($CurCost$). If $PrevCost = CurCost$, there is a good chance that the algorithm has reached a local minimum and therefore, p is increased by p_{incr} to tolerate larger uphill moves, thus giving the search the possibility of escaping from local minima. Otherwise, p is reset to its initial value p_0.

At the end of the loop, the cost of the *current state* S is compared with the cost of the *best state* $BestS$. If S has a lower cost, then the algorithm keeps S as the best solution ($BestS$) and decrements R by ρ, thereby rewarding itself by increasing the number of iterations (allowing the search to live R generations more). This allows a more detailed investigation of the neighborhood of the newly found best solution. If S, however, has a higher cost, ρ is incremented, which is an indication of no improvements.

7 Convergence Aspects

One of the desirable properties that a stochastic iterative algorithm should possess is the convergence property, i.e., the guarantee of converging to one of the global optima if given enough time. The convergence aspects of the simulated annealing algorithm have been the subject of extensive studies. For a thorough discussion of simulated annealing convergence we refer the reader to [AK89, AL85, OvG89].

For convergence properties of the GA heuristic based on Markovian analysis, the reader is referred to [GS87, NV93, EAH90, DP91, DP93, Mah93, Rud94]. Fogel [Fog95] provides a concise treatment of the main GA convergence results.

[2] Here, we assume that we are dealing with a minimization problem

The tabu search algorithm as described in this article is known as *ordinary* or *deterministic tabu search*. Because of its deterministic nature, ordinary tabu search may never converge to a global optimum state. The incorporation of a nondeterministic element within tabu search allows the algorithm to lend itself to mathematical analysis similar to that developed for simulated annealing, making it possible to establish corresponding convergence properties. Tabu search with nondeterministic elements is called *probabilistic tabu search* [FK92, Glo89]. Probabilistic tabu search has been shown to converge in the limit to a global optimum state. The proof is analogous to that of simulated annealing.

The proof that SimE algorithm converges to a global optimum can be found in [KB91]. Another proof has been reported in[MH96].

In StocE, because of the way the parameter p is updated, it was stated that StocE may never converge [Saa90]. The behavior of the stochastic evolution algorithm can be modeled by a non-homogeneous Markov chain. The non-homogeneity is due to the fact that the transition probabilities between neighboring states depend on the parameter p, which gets updated during the course of the search. Were the parameter p fixed, the algorithm would be guaranteed to converge to a global optimum state since the behavior of the algorithm could in that case be modeled by a homogeneous ergodic Markov chain [SY99, Kle75]. For complete convergence analysis the reader may refer to [SY99].

8 Parallelization/Accelaration

Due to their iterative and blind nature, the heuristics discussed in this chapter require large runtime, especially on large problems, and CPU-intensive cost functions. Substantial amount of work has been done to parallelize or design accelarators to run these time consuming heuristics. With respect to Simulated annealing, some ingenuity is required on the part of the designer to cleverly anneal the problem in question. Several accelerations techniques employing parallelization have been reported in [AK89, HRSV86, SH87, GS86, LA87, CDV88, SSV86, Gro86]. Hardware acceleration which consists of implementing time consuming parts in hardware [IKB83]; and parallel acceleration, where execution of the algorithm is partitioned on several concurrently running processors is reported in [KR87, DRKN87, DKN87]. Other approaches that have been applied to parallelize simulated annealing are found in [Dur89, AK89, KR87, Yao97].

The Genetic Algorithm is highly parallel. The reported GA parallelization strategies fall into three general categories: (a) the *island model* [SWM91, Tan89] (b) the *stepping stone model*, [GS91, EG72, Eld85, CHMR87, CHMR91] and (c) the *neighborhood model*, also called the *cellular model* [GW94, GS89].

Work on parallelization of the tabu search heuristic can be found in [Tai91, Tai90, GPR94, FBTV94, Tai93, SY99]. The heuristic has also been parallelized and executed on a network of workstations using PVM [?].

Techniques to accelarate the execution of SimE by implementing on Vector

Processors [Kli90] and on a network of workstations [KB87a] are elaborated in [SY99].

To speed up StocE execution, acceleration techniques used with previous algorithms are resorted to. The acceleration of the stochastic evolution algorithm did not receive any attention from the research community. A tentative discussion appears in [SY99].

9 Applications

The first applications of simulated annealing was on placement [KCGV83]. Furthermore, the largest number of application of simulated annealing was on digital design automation problems. [SY95c]. A popular package that uses simulated annealing for VLSI standard-cell placement and routing is the TimberWolf3.2 package [SSV86]. In addition to placement, there are several other problems to which SA has been applied successfully. These include classical problems such as the TSP [KCGV83], graph partitioning, matching problem, Steiner problems [Dow91], linear arrangement [NSS89], clustering problem [SAS91], quadratic assignment [Con90], various scheduling problems [OP89, OS90], graph coloring [CHdW87], etc. In the area of engineering SA has been applied extensively to solve various hard VLSI physical design automation problems [SY95a]. In addition, it has been applied with success in other areas such as topology design of computer networks [EP93], image processing [GG84], test pattern generation, code design, etc. A comprehensive list of bibliography of some of the above applications and some details of their implementation such as cost function formulation, move set design, parameters, etc., is available in [AK89, SY99, Dow95, CEG88, Egl90].

In addition to their application to classical optimization problems such as the knapsack problem [Spi95], TSP [WSF89, T^+94], Steiner tree problem [HMS89], set covering problem [ASHN96], N-queens problem[HTA92], clustering problem[ASK96], graph partitioning [PR94], etc., GAs have also been applied to several engineering problems. Some examples of these applications include job shop and multiprocessor scheduling [WSF89, HAH94, BS94], discovery of maximal distance codes for data communications [DJ90], bin-packing [FD92], design of telecommunication (mesh) networks [K^+97], test sequence generation for digital system testing [RPGN94], VLSI design (cell placement [CP87, SM91, SM90, SY95b], floorplanning [S^+95], routing [H^+95]), pattern matching [ACH90], technology mapping [KP93], PCB assembly planning [LWJ92], and high-level synthesis of digital systems [ASB94, RS96]. The books by Goldberg (1989) [Gol89], Davis (1991) [Dav91], recent conference proceedings on evolutionary computation, and on applications of genetic algorithms discuss in detail the various applications of GAs in science and engineering. These range from optimization of pipeline systems and medical imaging to applications such as 'robot trajectory generation' and 'parametric design of aircraft' [Gol89, Dav91].

TS has also been applied to solve combinatorial optimization problems appearing in various fields of science, engineering, and business. Results reported indicate superior performance to other previous techniques. Examples of some hard problems to which tabu search has been applied with success include graph partitioning [LC91], clustering [AS95], TSP (traveling salesman problem) [MHKM89], maximum independent set problem [FHdW90], graph coloring [DdW93, HdW87], maximum clique problem [GSS93], and quadratic assignment problem [Tai91, SK90] to name a few. In the area of engineering, tabu search has been applied to machine sequencing [Ree93], scheduling [MR93, DT93, IE94, WH89, BC95, Wid91], fuzzy clustering [ASF98], multiprocessor scheduling [HG94], vehicle routing [Osm93, ST93, RLB96], general fixed charge problem [SM93], bin-packing [GH91], bandwidth packing [LG93], VLSI placement [SV92], circuit partitioning [AV93], global routing [YS98], high-level synthesis of digital systems [SAB96, AK94], etc. A good summary of most recent applications of tabu search can be found in [Glo96, GL95, SY99].

The SimE algorithm has also been used to solve a wide range of combinatorial optimization problems. Kling and Banerjee published their results with respect to SimE in design automation conferences [KB87b, KB90] and journals [KB89, KB91]. This explains the fact that most published work on SimE has been originated by researchers in the area of design automation of VLSI circuits [LHT89, LM93, MH94, Mao94, MH96]. The first problem on which SimE was first applied is standard cell placement [KB87b, Kli90]. A number of papers describe SimE-based heuristics as applied to the routing of VLSI circuits [LHT89, CLAHL95, TSN93, CLH89, LHT88, LAH95, HHYLH91]. SimE was also successfully applied in high level synthesis [TJ90, KL92a, KL92b]. Other reported SimE applications are in micro-code compaction [IMK95], automatic synthesis of Gate Matrix [Mao94], and the synthesis of cellular architecture Field Programmable Gate Arrays (FPGAs) [ANM93].

A formulation and discussion on how the StocE algorithms can be used to solve eight NP-Hard problems: network bisection, vertex cover, set partition, Hamiltonian circuit, traveling salesman, linear ordering, standard cell placement, and multi-way circuit partitioning is found in [SY99, Saa90]. Details ion how StocE can be used to solve technology mapping problem in FPGAs, a hard combinatorial optimization problem encountered in FPGA-design methodologies is given in [SY99].

10 Conclusion

This chapter has introduced the reader to five effective heuristics that belong to the class of *general iterative algorithms*, namely, simulated annealing, genetic algorithm, tabu search, simulated evolution, and, stochastic evolution. From the immense literature that is available it is evident that for a large variety of applications, in certain settings, these heuristics produce excellent results.

All algorithms discussed in this chapter are iterative in nature, and operate on design solutions generated at each iteration. A value of the objective function is used to compare results of consecutive iterations and a solution is selected based on its value.

All algorithms incorporate domain specific knowledge to dictate the search strategy. They also tolerate some element of non-determinism that helps the search escape out of local minima. They all rely on the use of a suitable cost function which provides feedback to the algorithm as the search progresses. The principle difference among these heuristics is how and where domain specific knowledge is used. For example, in simulated annealing such knowledge is mainly included in the cost function. Elements involved in a perturbation are selected randomly, and perturbations are accepted or rejected according to the Metropolis criterion which is a function of the cost. The cooling schedule has also a major impact on the algorithm performance and must be carefully crafted to the problem domain as well as the particular problem instance.

For the three evolutionary algorithms discussed in the chapter, genetic algorithms, simulated evolution, and stochastic evolution, domain specific knowledge is exploited in all phases. In the case of genetic algorithms, the fitness of individual solutions incorporates domain specific knowledge. Selection for reproduction, the genetic operations, as well as generation of the new population also incorporate a great deal of heuristic knowledge about the problem domain. In simulated evolution, each individual element of a solution is characterized by a goodness measure that is highly correlated with the objective function. The perturbation step (selection followed by allocation) affects mostly low goodness elements. Therefore, domain specific knowledge is included in every step of the simulated evolution algorithm. In stochastic evolution, at each iteration the algorithm actually attempts to move each element. The acceptance/rejection of the move is based on a gain measure and a parameter p. Both the gain measure and p are tuned to the problem domain.

Tabu search is different from the above heuristics in that it has an explicit memory component. At each iteration the neighborhood of the current solution is partially explored, and a move is made to the best non-tabu solution in that neighborhood. The neighborhood function as well as tabu list size and content are problem specific. The direction of the search is also influenced by the memory structures (whether intensification or diversification is used).

It is also possible to make hybrids of these algorithms. The basic idea of hybridization is to enhance the strengths and compensate for the weaknesses of two or more complementary approaches. For the details about the hybridization the readers are refered to [SY99].

In this chapter, it has not been our intention to demonstrate the superiority of one algorithm over the other. Actually it would be unwise to rank such algorithms. Each one of them has its own merits. Recently, an interesting theoretical study has been reported by Wolpert and Macready in which they proved a number of theorems stating that the average performance of any pair of iterative (deterministic or non-deterministic) algorithms across all problems is identical. That is, if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the remaining set of problems [WM97]. However, it should be noted that the reported theorems assume that the algorithms do not include domain specific knowledge of the problems being solved. Obviously, it would be expected that a well engineered algorithm would exhibit superior performance to that of a poorly engineered one.

Acknowledgement

Authors acknowledge King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for all support. This work is carried out under university funded project number COE/ITERATE/221.

References

[ACH90] N. Ansari, M-H Chen, and E. S. H. Hou. Point pattern matching by genetic algorithm. In *16th Annual Conference on IEEE Industrial electronics*, pages 1233–1238, 1990.

[AK89] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley & Sons, 1989.

[AK94] S. Amellal and B. Kaminska. Functional synthesis of digital systems with TASS. *IEEE Transactions on Computer-Aided Design*, 13(5):537–552, May 1994.

[AL85] E. H. L. Aarts and P. J. N. Van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problem. *Philips Journal of Research*, 40(4):193–226, January 1985.

[ANM93] Dasari A.K., Song N., and Chrzanowska-Jeske M. Layout-driven factorization and fitting for cellular-architecture fpgas. In *Proceedings of NORTHCON'93 Electrical and Electronics Convention*, pages 106–111, New York, NY, USA, Oct 1993. IEEE.

[AS95] K. S. Al-Sultan. A tabu search approach to the clustering problem. *Pattern Recognition*, 28(9):1443–1451, 1995.

[ASB94] S. Ali, Sadiq M. Sait, and M. S. T. Benten. GSA: Scheduling and allocation using genetic algorithms. In *Proceeding of EURO-DAC'94: IEEE European Design Automation Conference*, pages 84–89, September 1994.

[ASF98] K. S. Al-Sultan and C. A. Fedjki. A tabu search based algorithm for the fuzzy clustering problem. *To appear in Pattern Recognition*, 1998.

[ASHN96] K. S. Al-Sultan, M. F. Hussain, and J. S. Nizami. A genetic algorithm for the set covering problem. *Journal of the Operational Research Society*, 47:702–709, 1996.

[ASK96] K. S. Al-Sultan and M. Maroof Khan. Computational experience on four algorithms for the hard clustering problem. *Pattern Recognition Letters*, 17:295–308, 1996.

[AV93] S. Areibi and A. Vannelli. Circuit partitioning using a tabu search approach. In *1993 IEEE International Symposium on Circuits and Systems*, pages 1643–1646, 1993.

[BC95] J. W. Barnes and J. B. Chambers. Solving the job shop scheduling problem with tabu search. *IIE Transactions*, 27:257–263, 1995.

[BS94] M. S. T. Benten and Sadiq M. Sait. Genetic scheduling of task graphs. *International Journal of Electronics*, 77(4):401–415, 1994.

[CDV88] F. Catthoor, H. DeMan, and J. Vandewalle. Samurai: a general and efficient simulated-annealing schedule with fully adaptive annealing parameters. *Integration*, 6:147–178, 1988.

[CEG88] N. E. Collins, R. W. Eglese, and B. L. Golden. Simulated annealing - - an annotated bibliography. *AJMMS*, 8:209–307, 1988.

[Čer85] V. Černy. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Application*, 45(1):41–51, January 1985.

[CHdW87] M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32:260–266, 1987.

[CHMR87] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. S. Ricichards. Punctuated equilibria: A parallel genetic algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, 1987.

[CHMR91] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. S. Ricichards. Distributed genetic algorithms for the floorplan design proboblem. *IEEE Transactions on Computer-Aided Design*, CAD-10:483–492, April 1991.

[CLAHL95] Ching-Dong Chen, Yuh-Sheng Lee, Wu A.C.-H., and Youn-Long Lin. Tracer-fpga: a router for ram-based fpga's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):371–374, March 1995.

[CLH89] Yirng-An Chen, Youn-Long Lin, and Yu-Chin Hsu. A new global router for asic design based on simulated evolution. In *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*, pages 261–265, New York, NY, USA, May 1989. IEEE.

[Con90] D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46:93–100, 1990.

[CP87] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design*, CAD-6:956–964, November 1987.

[Dav91] L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, NY, 1991.

[DdW93] N. Dubois and D. de Werra. EPCOT: An efficient procedure for coloring optimally with tabu search. *Computers Math Applic.*, 25(10/11):35–45, 1993.

[DJ90] K. Dontas and K. De Jong. Discovery of maximal distance codes using genetic algorithms. *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pages 805–811, 1990.

[DKN87] F. Darema, S. Kirkpatrick, and V. A. Norton. Parallel techniques for chip placement by simulated annealing on shared memory systems. *Proceedings of International Conference on Computer Design: VLSI in Computers & Processors, ICCD-87*, pages 87–90, 1987.

[Dow91] K. A. Dowsland. Hill climbing, simulated annealing, and the Steiner problem in graphs. *Eng. Opt.*, 17:91–107, 1991.

[Dow95] K. A. Dowsland. Simulated annealing. In C. R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Optimization Problems*. McGraw-Hill Book Co., Europe, 1995.

[DP91] T. E. Davis and J. C. Principe. A Simulated Annealing Like Convergence Theory for the Simple Genetic Algorithm. *Proceedings of the 4th International Conference on Genetic Algorithm*, pages 174–181, 1991.

[DP93] T. E. Davis and J. C. Principe. A Markov Chain Framework for the Simple Genetic Algorithm. *Proceedings of the 4th International Conference on Genetic Algorithm*, 13:269–288, 1993.

[DRKN87] F. Darema-Rogers, S. Kirkpatrick, and V. A. Norton. Parallel algorithms for chip placement by simulated annealing. *IBM Journ. of Research and Development*, 31:391–402, May 1987.

[DT93] M. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 1993.

[Dur89] M. D. Durand. Accuracy vs. speed in placement. *IEEE Design & Test of Computers*, pages 8–34, June 1989.

[DV93] F. Dammeyer and Stefan Voβ. Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research*, 41:31–46, 1993.

[EAH90] A. H. Eiben, E. H. L. Aarts, and K. M. Van Hee. Global convergence of genetic algorithms: A markov chain analysis. In H. P. Schwefel and Männer, editors, *In Paralle Problem Solving from Nature*, pages 4–12. Berlin: Springer-Verlag, 1990.

[EG72] N. Eldredge and S. J. Gould. *Punctuated Equilibrea: An alternative to phyletic gradualism. Models of Paleobiology, T.J. M. Schopf, Ed. San Fransisco: CA, Freeman.* Cooper and Co., 1972.

[Egl90] R. W. Eglese. Simulated annealing: a tool for operational research. *European Journal of Operational Research*, 46:271–281, 1990.

[Eld85] N. Eldredge. *Time Frames.* New York: Simon and Schuster, 1985.

[EP93] C. Ersoy and S. S. Panwar. Topological design of interconnected LAN/MAN networks. *IEEE Journal on Selected Areas in Communications*, 11(8):1172–1182, 1993.

[FBTV94] I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro. Improving search by incorporating evolution principles in parallel tabu search. In *Proc. of the first IEEE Conference on Evolutionary Computation- ICEC'94*, pages 823–828, June 1994.

[FD92] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. *Proceedings of International Conference on Robotics and Automation*, pages 1186–1192, May 1992.

[FHdW90] C. Friden, A. Hertz, and D. de Werra. TABARIS: An exact algorithm based on tabu search for finding a maximum independent set in a graph. *Computers and Operations Research*, 19(1–4):81–91, 1990.

[FK92] U. Faigle and W. Kern. Some Convergence Results for Probabilistic Tabu Search. *ORSA Journal on Computing*, 4(1):32–37, Winter 1992.

[Fog95] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* . IEEE Press, 1995.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:721–741, 1984.

[GH91] F. Glover and R. Hübscher. Binpacking with a tabu search. *Technical Report, Graduate School of Business Administration, University of Colorado at Boulder*, 1991.

[GL95] F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems.* McGraw-Hill Book Co., Europe, 1995.

[GL97] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, MA, 1997.

[Glo89] F. Glover. Tabu search- Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Glo90a] F. Glover. Artificial intelligence, heuristic frameworks and tabu search. *Managerial and Decision Economics*, 11:365–375, 1990.

[Glo90b] F. Glover. Tabu search- Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[Glo90c] F. Glover. Tabu search: A tutorial. *Technical Report, University of Colorado, Boulder*, February 1990.

[Glo95] F. Glover. Tabu search fundamentals and uses. *Technical Report, Graduate School of Business Administration, University of Colorado at Boulder*, June 1995.

[Glo96] F. Glover. Tabu search and adaptive memory programming – advances, applications and challenges. *Technical Report, College of Business, University of Colorado at Boulder*, 1996.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc., 1989.

[GPR94] Bruno-Laurent Garica, Jean-Yves Potvin, and Jean-Marc Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research*, 21(9):1025–1033, November 1994.

[Gro86] L. K. Grover. A new simulated annealing algorithm for standard cell placement. In *IEEE International Conference of Computer-Aided Design*, pages 378–380, 1986.

[GS86] J. W. Greene and K. J. Supowit. Simulated annealing without rejected moves. *IEEE Tansactions on Computer-Aided Design*, 5:221–228, 1986.

[GS87] D. E. Goldberg and P. Segrest. Finite markov chain analysis of genetic algorithms. *Genetic Algorithms and Their Applications: Proceedings of 2nd International Conference on GAs*, pages 1–8, 1987.

[GS89] M. Gorges-Schleuter. ASPARAGOS–An asynchronous parallel genetic optimization strategy. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, pages 422–427. Morgan-Kaufmann, San Maeto, CA, 1989.

[GS91] M. Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In H. P. Schwefel and R. Männer, editors, *Problem Solveing from Nature*, pages 150–159. Springer Verlag, New York, 1991.

[GSS93] M. Gendreau, P. Soriano, and L. Salvail. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, 41:385–404, 1993.

[GTdW93] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.

[GW94] V. Scott Gordon and Darrell Whitley. A machine-independent analysis of parallel genetic algorithms. *Complex Systems*, 8:181–214, 1994.

[H$^+$95] H. I. Han et al. GenRouter: a genetic algorithm for channel routing problems. In *Proceeding of TENCON 95, IEEE Region 10 International Conference on Microelectronics and VLSI*, pages 151–154, November 1995.

[HAH94] E. S. H. Hou, N. Ansari, and R. Hong. genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, February 1994.

[Han86] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. *Congress on Numerical Methods in Combinatorial Optimization*, 1986.

[HdW87] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[HG94] R. Hübscher and F. Glover. Applying tabu search with influential diversification to multiprocessor scheduling. *Computers & Operations Research*, 21(8):877–884, 1994.

[HHYLH91] Yung-Ching Hsich, Chi-Yi Hwang, Youn-Long, and Yu-Chin Hsu. Lib: a cmos cell compiler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(8):994–1005, Aug 1991.

[HMS89] J. Hesser, R. Männer, and O. Stucky. Optimization of steiner trees using genetic algorithms. In *ICGA'89*, pages 231–236, 1989.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[HRSV86] M. D. Huang, F. Romeo, and A. L. Sangiovanni-Vincentelli. An efficient general cooling schedule for simulated annealing. In *IEEE International Conference on Computer-Aided Design*, pages 381–384, 1986.

[HTA92] A. Homaifar, J. Turner, and Samia Ali. The N-queens problem and genetic algorithms. In *IEEE Proceedings of Southeastcon'92*, pages 262–267, April 1992.

[IE94] O. Icmeil and S. Selcuk Erenguc. A tabu search procedure for the resource constrained project scheduling problem with discounted cash flows. *Computers & Operations Research*, 21(8):841–853, 1994.

[IKB83] A. Iosupovici, C. King, and M. Breuer. A module interchange placement machine. *Proceedings of 20th Design Automation Conference*, pages 171–174, 1983.

[IMK95] Ahmad I., Dhodhi M.K., and Saleh K.A. An evolutionary-based technique for local microcode compaction. In *Proceedings of ASP-DAC'95/CHDL'95/VLSI with EDA Technofair*, pages 729–734, Tokyo, Japan, Sept 1995. Nihon Gakkai Jimu Senta.

[K+97] King-Tim Ko et al. Using genetic algorithms to design mesh networks. *Computer*, pages 56–60, 1997.

[KB87a] Raplph Michael Kling and Prithviraj Banerjee. Concurrent ESP: A placement algorithm for execution on distributed processors. *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 354–357, 1987.

[KB87b] Raplph Michael Kling and Prithviraj Banerjee. ESP: A new standard cell placement package using simulated evolution . *Proceedings of 24th Design Automation Conference*, pages 60–66, 1987.

[KB89] R. M. Kling and P. Banerjee. ESP: Placement by Simulated Evolution. *IEEE Transactions on Computer-Aided Design*, 8(3):245–255, March 1989.

[KB90] R. M. Kling and P. Banerjee. Optimization by Simulated Evolution with Applications to Standard Cell Placement . *Proceedings of 27th Design Automation Conference*, pages 20–25, 1990.

[KB91] R. M. Kling and P. Banerjee. Empirical and Theoretical Studies of the Simulated Evolution Method Applied to Standard Cell Placement. *IEEE Transactions on Computer-Aided Design*, 10(10):1303–1315, October 1991.

[KCGV83] S. Kirkpatrick, Jr. C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):498–516, May 1983.

[KL92a] Yau-Hwang Kuo and Shaw-Pyng Lo. Automated synthesis of asynchronous pipelines. In *Custom Integrated Circuits Conference, Proceedings of the IEEE 1992*, volume 2, pages 685–688, New York, NY, USA, May 1992. IEEE.

[KL92b] Yau-Hwang Kuo and Shaw-Pyng Lo. Partitioning and scheduling of asynchronous pipelines. In *Computer Systems and Software Engineering, CompEuro 1992 Proceedings.*, pages 574–579, Loss Alamitos, CA, USA, May 1992. IEEE Comput. Soc. Press.

[Kle75] Leonard Kleinrock. *Queueing Systems, Volume I: Theory.* John Wiley & Sons, 1975.

[KLG94] J. P. Kelly, M. Laguna, and F. Glover. A study of diversification strategies for the quadratic assignment problem. *Computers Ops Research*, 21(8):885–893, 1994.

[Kli90] R. M. Kling. *Optimization by Simulated Evolution and its Application to cell placement.* Ph.D. Thesis, University of Illinois, Urbana, 1990.

[KP93] V. Kommu and I. Pomeranz. GAFPGA: Genetic algorithms for FPGA technology mapping. In *Proceeding of EURO-DAC'93: IEEE European Design Automation Conference*, pages 300–305, September 1993.

[KR87] Saul A. Kravitz and Rob A. Rutenbar. Placement by simulated annealing on a multiprocessor. *IEEE Transactions on Computer-Aided Design*, CAD-6(4):534–549, July 1987.

[LA87] P. J. M. Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Reidel, Dordrecht, 1987.

[LAH95] Yuh-Sheng Lee and Wu A.C.-H. A performance and routability driven router for fpgas considering path delays. *ANSI/IEEE Std 802.lb-1995*, pages 557–561, March 1995.

[LC91] A. Lim and Yeow-Meng Chee. Graph partitioning using tabu search. In *1991 IEEE International Symposium on Circuits and Systems*, pages 1164–1167, 1991.

[LG93] M. Laguna and F. Glover. Bandwidth packing; a tabu search approach. *Management Science*, 39(4):492–500, 1993.

[LHT88] Youn-Long Lin, Yu-Chin Hsu, and Fur-Shing Tsai. A detailed router based on simulated evolution. In *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*, pages 38–41, New York, NY, USA, Nov 1988. IEEE Comput. Soc. Press.

[LHT89] Y. L. Lin, Y. C. Hsu, and F. H. S. Tsai. SILK: A Simulated Evolution Router. *IEEE Transactions on Computer-Aided Design*, 8(10):1108–1114, October 1989.

[LM93] . A. Ly and Jack T. Mowchenko. Applying Simulated Evolution to High Level-Synthesis. *IEEE Transactions on Computer-Aided Design*, 12(3):389–409, March 1993.

[LWJ92] M. C. Leu, H. Wong, and Z. Ji. Genetic algorithm for solving printed circuit board assembly planning problems. *Proceedings of Japan-USA Symposium on Flexible Automation*, pages 1579–1586, July 1992.

[M+53] N. Metropolis et al. Equation of state calculations by fast computing machines. *Journal of Chem. Physics*, 21:1087–1092, 1953.

[Mah93] S. W. Mahfoud. Finite markov chain models of an alternative selection strategy for the genetic algorithm. *Complex systems*, 7:155–170, 1993.

[Mao94] C. Y. Mao. *Simulated Evolution Algorithms for Gate Matrix layouts.* Ph.D. Thesis, University of Wisconsin, Madison, 1994.

[MGPO89] M. Malek, M. Guruswamy, M. Pandya, and H. Owens. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, 21:59–84, 1989.

[MH94] C. Y. Mao and Y. H. Hu. SEGMA: A Simulated Evolution Gate Matrix Layout Algorithm. *VLSI Design*, 2(3):241–257, 1994.

[MH96] C. Y. Mao and Y. H. Hu. Analysis of Convergence Properties of Stochastic Evolution Algorithm. *IEEE Transactions on Computer-Aided Design*, 15(7):826–831, July 1996.

[MHKM89] M. Malek, M. Heap, R. Kapur, and A. Mourad. A fault tolerant implementation of the traveling salesman problem. *Research Report, Department of EE and Computer Engineering, The University of Texas-Austin*, May 1989.

[MR93] E. L. Mooney and R. L. Rardin. Tabu search for a class of scheduling problems. *Annals of Operations Research*, 41:253–278, 1993.

[NSS85] S. Nahar, S. Sahni, and E. Shragowitz. Experiments with Simulated Annealing. *Proceedings of 22nd Design Automation Conference*, pages 748–752, 1985.

[NSS89] S. Nahar, S. Sahni, and E. Shragowitz. Simulated annealing and combinatorial optimization. *International Journal of Computer-Aided VLSI Design*, 1(1):1–23, 1989.

[NV93] A. E. Nix and M. D. Vose. Modeling genetic algorithms with markov chains. *Annals of Mathematics and Artificial Intelligence*, pages 79–88, 1993.

[OP89] I. H. Osman and C. N. Potts. Simulated annealing for permutation flow-shop annealing. *OMEGA*, 17:551–557, 1989.

[OS90] F. A. Ogbu and D. K. Smith. The application of the simulated annealing algortihm to the solution of the $\mathrm{n/m/c_{max}}$ flowshop problem. *Computers & Operations Research*, 17:243–253, 1990.

[Osm93] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[OvG89] R.H.J.M. Otten and L.P.P.P. van Ginneken. *The Annealing Algorithm.* Kluwer Academic Publishers, MA, 1989.

[PR94] H. Pirkul and E. Rolland. New heuristic solution procedures for uniform graph partitioning problem: Extensions and evaluation. *Computers & Operations Research*, 21(8):895–907, October 1994.

[Ree93] C. R. Reeves. Improving the efficiency of tabu search for machine sequencing problems. *Journal of Operational Research Society*, 44:375–382, 1993.

[RLB96] J. Renaud, G. Laporte, and F. F. Boctor. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers Ops Research*, 23:229–235, 1996.

[RPGN94] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann. Sequential circuit test generation in a genetic algorithm framework. In *Proceedings of the 31st Design Automation Conference*, pages 698–704, 1994.

[RS96] C. P. Ravikumar and V. Saxena. TOGAPS: a testability oriented genetic algorithm for pipeline synthesis. *VLSI Design*, 5(1):77–87, 1996.

[Rud94] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5:1:96–101, 1994.

[Rya89] J. Ryan, editor. *Heuristics for Combinatorial Optimization.* June 1989.

[S+95] Sadiq M. Sait et al. Timing influenced general-cell genetic floorplanner. In *ASP-DAC'95: Asia ad South-Pacific Design Automation Conference*, pages 135–140, 1995.

[Saa90] Youssef G. Saab. *Combinatorial Optimization by Stochastic Evolution with applications to the physical design of VLSI circuits.* Ph.D. Thesis, University of Illinois, Urbana, 1990.

[SAB96] Sadiq M. Sait, S. Ali, and M. S. T. Benten. Scheduling and allocation in high-level synthesis using stochastic techniques. *Microelectronics Journal*, 27(8):693–712, October 1996.

[SAS91] S. Selim and K. S. Al-Sultan. A simulated annealing algorithm for the clustering problem. *Pattern Recognition*, 24(10):1003–1008, 1991.

[SH87] H. Szu and R. Hartley. Fast simulated annealing. *Physics Letters, A*, 122:157–162, 1987.

[SK90] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.

[SM90] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer-Aided Design*, 9(5):500–511, May 1990.

[SM91] K. Shahookar and P. Mazumder. VLSI cell placement techniques. *ACM Computing Surveys*, 23(2):143–220, June 1991.

[SM93] Minghe Sun and P. G. McKeown. Tabu search applied to the general fixed charge problem. *Annals of Operations Research*, 41:405–420, 1993.

[Spi95] R. Spillman. Solving large knapsack problems with a genetic algorithm. In *International Conference on Systems, Man and Cybernetics*, pages 632–637, 1995.

[SR89] Y. Saab and V. Rao. An Evolution-Based Approach to Partitioning ASIC Systems. In *26th ACM/IEEE Design Automation Conference*, pages 767–770, 1989.

[SR90] Y. Saab and V. Rao. Stochastic Evolution: A Fast Effective Heuristic for some Generic Layout Problems. In *27th ACM/IEEE Design Automation Conference*, pages 26–31, 1990.

[SR91] Youssef G. Saab and Vasant B. Rao. Combinatorial Optimization by Stochastic Evolution. *IEEE Transactions on Computer-Aided Design*, 10(4):525–535, April 1991.

[SSV86] C. Sechen and A. L. Sangiovanni-Vincentelli. Timberwolf3.2: A new standard cell placement and global routing package. *Proceedings of 23rd Design Automation Conference*, pages 432–439, 1986.

[ST93] F. Semet and E. Taillard. Solving real-life vehicle routing problems efficiently using tabu search. *Annals of Operations Research*, 41:469–488, 1993.

[SV92] L. Song and A. Vannelli. VLSI placement using tabu search. *Microelectronics Journal*, 17(5):437–445, 1992.

[SWM91] T. Starkweather, D. Whitley, and K. Mathias. Optimization using distributed genetic algorithm. In *Parallel problem solving from nature*, 1991.

[SY95a] Sadiq M. Sait and Habib Youssef. *VLSI Design Automation: Theory and Practice*. McGraw-Hill Book Co., Europe (Also co-published by IEEE Press), 1995.

[SY95b] Sadiq M. Sait and Habib Youssef. *VLSI Design Automation: Theory and Practice*. McGraw-Hill Book Co., Europe, 1995.

[SY95c] Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation: Theory and Practice.* McGraw-Hill Book Company, Europe, 1995.

[SY99] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.* IEEE Computer Society Press, 1999.

[T^+94] H. Tamaki et al. A comparison study of genetic codings for the traveling salesman problem. In *Proceedings of the Ist IEEE Conference on Evolutionary Computation*, pages 1–6, 1994.

[Tai90] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 417:65–74, 1990.

[Tai91] E. Taillard. Robust tabu search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

[Tai93] E. Taillard. Parallel iterative search methods for the vehicle routing problem. *Networks*, 23:661–673, 1993.

[Tan89] M. Tanese. Distributed genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 434–439. Morgan-Kaufmann, San Maeto, CA, 1989.

[TJ90] Ly T.A. and Mowchenko J.T. Applying simulated evolution to scheduling in high level synthesis. In *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*, volume 1, pages 172–175, New York, NY, USA, Aug 1990. IEEE Comput. Soc. Press.

[TSN93] Koide T., Wakabayashi S., and Yoshida N. An integrated approach to pin assignment and global routing for vlsi building-block layout. In *1993 European Conference on Design Automation with the European Event in ASIC Design*, pages 24–28, Loss Alamitos, CA, USA, Feb 1993. IEEE Computer Society Press.

[WH89] M. Widmer and A. Hertz. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41:186–193, 1989.

[Wid91] M. Widmer. Job shop scheduling with tooling constraints: a tabu search approach. *Journal of Operational Research Society*, 42(1):75–82, 1991.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[WSF89] D. Whitley, T. Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, pages 133–140, 1989.

[Yao97] Xin Yao. Global optimization by evolutionary algorithms. In *Proceedings of IEEE International Symposium on parallel algorithms architecture synthesis*, pages 282–291, 1997.

[YS98] Habib Youssef and Sadiq M. Sait. Timing driven global router for standard cell design. *International Journal of Computer Systems Science and Engineering (to appear)*, 1998.

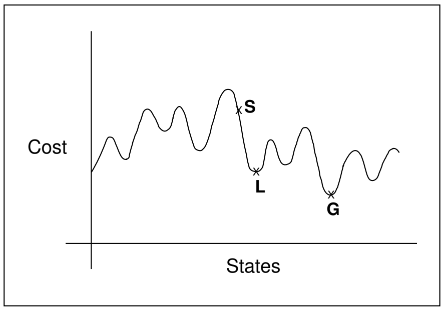

Figure 1: Local versus global optima.

Algorithm Simulated_annealing($S_0, T_0, \alpha, \beta, M, Maxtime$);
(*S_0 is the initial solution *)
(*BestS is the best solution *)
(*T_0 is the initial temperature *)
(*α is the cooling rate *)
(*β a constant *)
(*$Maxtime$ is the total allowed time for the annealing process *)
(*M represents the time until the next parameter update *)

Begin
$T = T_0$;
$CurS$=S_0;
$BestS$=$CurS$;/* $BestS$ is the best solution seen so far */
$CurCost$=$Cost(CurS)$;
$BestCost$=$Cost(BestS)$;
$Time = 0$;
Repeat
Call Metropolis($CurS$, $CurCost$, $BestS$, $BestCost$, T, M);
$Time = Time + M$;
$T = \alpha T$;
$M = \beta M$
Until ($Time \geq MaxTime$);
Return ($BestS$)
End. (*_of Simulated_annealing_*)

Figure 2: Procedure for simulated annealing algorithm.

Algorithm Metropolis($CurS$, $CurCost$, $BestS$, $BestCost$, T, M);
Begin
Repeat
$NewS$= $Neighbor(CurS)$; /* Return a neighbor from $aleph(CurS)$ */
$NewCost$=$Cost(NewS)$;
$\Delta Cost$=($NewCost - CurCost$);
If ($\Delta Cost < 0$) **Then**
$CurS$=$NewS$;
If $NewCost < BestCost$ **Then**
$BestS$= $NewS$
EndIf
Else
If ($RANDOM < e^{-\Delta Cost/T}$) **Then**
$CurS$=$NewS$;
EndIf
EndIf
$M = M - 1$
Until ($M = 0$)
End. (*_of Metropolis_*)

Figure 3: The Metropolis procedure.

Procedure $(Genetic\_Algorithm)$

M= Population size.	(*# Of possible solutions at any instance.*)
N_g= Number of generations.	(*# Of iterations.*)
N_o= Number of offsprings.	(*To be generated by crossover.*)
P_μ= Mutation probability.	(*Also called mutation rate M_r.*)
$\mathcal{P} \leftarrow \Xi(\mathbf{M})$	(*Construct initial population $\mathcal{P}$. Ξ is population constructor.*)
For $j = 1$ to $\mathbf{M}$	(*Evaluate fitnesses of all individuals.*)
Evaluate $f(\mathcal{P}[j])$	(*Evaluate fitness of $\mathcal{P}$.*)
EndFor	
For $i = 1$ to N_g	
For $j = 1$ to N_o	
$(x, y) \leftarrow \phi(\mathcal{P})$	(*Select two parents x and y from current population.*)
offspring$[j] \leftarrow \chi(x, y)$	(*Generate offsprings by crossover of parents x and y.*)
Evaluate $f($offspring$[j])$	(*Evaluate fitness of each offsprings.*)
EndFor	
For $j = 1$ to N_o	(*With probability P_μ apply mutation.*)
mutated$[j] \leftarrow \mu(y)$	
Evaluate $f($mutated[j]$)$	
EndFor	
$\mathcal{P} \leftarrow Select(\mathcal{P}$, offsprings$)$	(*Select best $\mathbf{M}$ solutions from parents & offsprings.*)
EndFor	
Return highest scoring configuration in $\mathcal{P}$.	

End

Figure 4: Structure of a simple genetic algorithm.

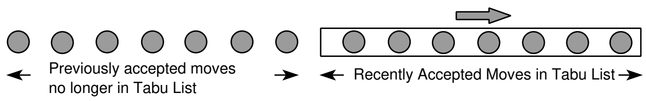

Figure 5: The tabu list can be visualized as a window over accepted moves.

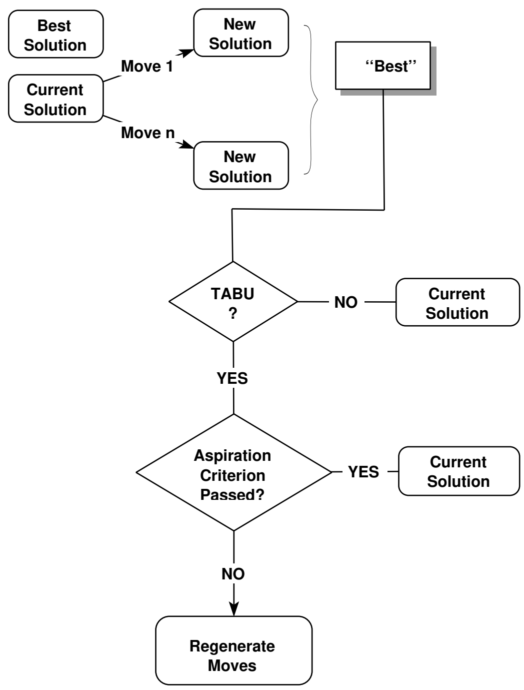

Figure 6: Flow chart of the tabu search algorithm.

Ω	:	Set of feasible solutions.
S	:	Current solution.
S^*	:	Best admissible solution.
$Cost$	:	Objective function.
$\aleph(S)$	:	Neighborhood of $S \in \Omega$.
$\mathbf{V}^*$	:	Sample of neighborhood solutions.
$\mathbf{T}$	:	Tabu list.
$\mathbf{AL}$	:	Aspiration Level.

Begin
Start with an initial feasible solution $S \in \Omega$.
Initialize tabu lists and aspiration level.
For fixed number of iterations **Do**
Generate neighbor solutions $\mathbf{V}^* \subset \aleph(S)$.
Find best $S^* \in \mathbf{V}^*$.
If move S to S^* is not in $\mathbf{T}$ **Then**
Accept move and update best solution.
Update tabu list and aspiration level.
Increment iteration number.
Else
If $Cost(S^*) < \mathbf{AL}$ **Then**
Accept move and update best solution.
Update tabu list and aspiration level.
Increment iteration number.
EndIf
EndIf
EndFor
End.

Figure 7: Algorithmic description of short-term Tabu Search (TS).

ALGORITHM *Simulated_Evolution(M, L)*;
/* M: Set of movable elements; */
/* L: Set of locations; */
/* B: Selection bias; */
/* Stopping criteria and selection bias can be automatically adjusted; */
$INITIALIZATION$;
Repeat
    $EVALUATION$:
        ForEach $m \in M$ **Do** $g_m = \frac{O_m}{C_m}$ **EndForEach**;

    $SELECTION$:
        ForEach $m \in M$ **Do**
            If *Selection(m,B)* **Then** $P_s = P_s \cup \{m\}$
                Else $P_r = P_r \cup \{m\}$
            EndIf;
        EndForEach;
    Sort the elements of P_s;

    $ALLOCATION$:
        ForEach $m \in P_s$ **Do** $Allocation(m)$ **EndForEach**;
Until *Stopping-criteria are met*;
Return $(BestSolution)$;
End *Simulated_Evolution*.

Figure 8: Simulated Evolution Algorithm.

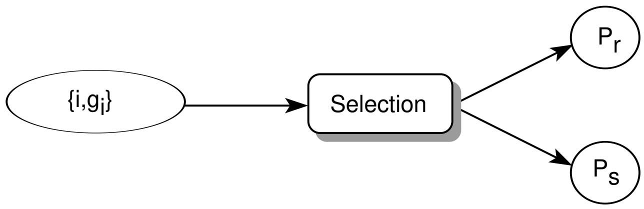

Figure 9: Selection.

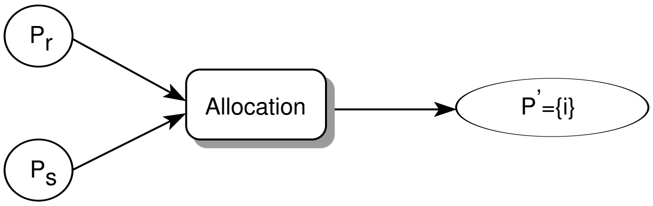

Figure 10: Allocation.

AlgorithmStocE($S_0,\ p_0,\ R$**)**;
Begin
    $BestS$= $S = S_0$;
    $BestCost$= $CurCost$= $Cost(S)$;
    $p = p_0$;
    $\rho = 0$;
    Repeat
        $PrevCost$= $CurCost$;
        $S = PERTURB(S, p)$; /* perform a search in the neighborhood of s */
        $CurCost$= $Cost(S)$;
        $UPDATE(p,\ PrevCost,\ CurCost)$; /* update p if needed */
        If ($CurCost$< $BestCost$) **Then**
            $BestS$=S;
            $BestCost$= $CurCost$;
            $\rho = \rho - R$; /* Reward the search with R more generations */
        Else
            $\rho = \rho + 1$;
        EndIf
    Until $\rho > R$
    Return ($BestS$);
End

Figure 11: The Stochastic Evolution algorithm.

```
FUNCTION PERTURB(S, p);
Begin
    ForEach (m ∈ M) Do   /* according to some apriori ordering */
        S' = MOVE(S, m);
        Gain(m) = Cost(S) − Cost(S');
        If (Gain(m) > RANDINT(−p, 0)) Then
                S = S'
        EndIf
    EndFor;
    S =MAKE_STATE(S); /* make sure S satisfies constraints */
    Return (S)
End
```

Figure 12: The PERTURB function.

```
PROCEDURE UPDATE(p, PrevCost, CurCost);
Begin
    If (PrevCost=CurCost) Then /* possibility of a local minimum */
        p = p + p_incr; /* increment p to allow larger uphill moves */
    Else
        p = p_0; /* re-initialize p */
    EndIf;
End
```

Figure 13: The UPDATE procedure.