Saed Alazamir, Steffen Rebennack and Panos M. Pardalos
Improving the Neighborhood Selection Strategy in Simulated Annealing using Optimal Stopping Problem
in *Global Optimization: Focus on Simulated Annealing*,
Cher Ming Tan (Ed.), I-Tech Education and Publication, pp. 363–382, 2008

Improving the Neighborhood Selection Strategy in Simulated Annealing using the Optimal Stopping Problem

Saed Alizamir, Steffen Rebennack, Panos M. Pardalos

University of Florida
Center of Applied Optimization
Gainesville, FL 32611, USA
{saed,steffen,pardalos}@ufl.edu

February 2, 2009

Abstract

The search method utilized by metaheuristic algorithms is in fact a local search in the solution space. Investigations of these methods show that their efficiency in solving different types of problems significantly depends on the applied strategy in searching the solution space. Traditional neighborhood selection methods have disregarded the essential characteristics embodied in the chosen neighborhood. In this research, we enhance the efficiency of solving combinatorial optimization problems using simulated annealing method by taking the advantages embodied in neighborhood structures. This new algorithm improves simulated annealing in two different aspects: First, by employing multiple neighborhood structures, it performs a more powerful search and second, using optimal stopping problem, it finds the best time to change the temperature which is a critical issue in simulated annealing. The novel algorithm is illustrated on the traveling salesman problem.

Key words: Metaheuristics; Simulated Annealing; Stochastic Dynamic Programming; Optimal Stopping Problem; Neighborhood Selection

1 Introduction

Simulated Annealing is one the most well known local search methods. In practice, it is often used to solve discrete optimization problems; especially very tough problems, [54, 37, 34, 3, 2, 41]. Global

optimization is computationally extremely challenging and for large instances, exact methods reach their limitations quickly. Hence, in practice, often local optimization methods are used. Simulated annealing provides a powerful tool for escaping local optima by allowing moves to lower quality solutions with a pre-specified probability. Another big plus of Simulated Annealing is its ease of implementation.

At each iteration of Simulated Annealing, the objective function value of the current solution and a new generated solution are compared. Improving moves are always accepted while only a fraction of non-improving moves are performed with the aim to escape local optimal solutions. The probability of accepting a non-improving move depends on the non-increasing parameter of temperature. This technique comes from annealing in metallurgy. In this process, a metal is heated and slowly cooled off, in order to increase the size of its crystals while reducing the number of defects. The heating dissolves out atoms from their initial positions which can be seen as a local minimum with respect to energy level. Such atoms can then freely move around. Slowly cooling off has the effect that the free atoms can end up in positions with lower energy level than the initial positions. The crucial factor is to choose the cooling procedure appropriately, as cooling off to fast may not enable atoms to find better energy levels and cooling off to slowly is very time consuming.

Simulated Annealing was introduced by Kirkpatrick et al. [33] in 1983 and independently by Černý [13] in 1985. It is an adaptation of a special Monte Carlo method generating sample states of a thermodynamic system which was introduced by Metropolis et al. in 1953, [38]. In 1986, Lundy and Mees were able to proof that under some technical assumptions Simulated Annealing converges with probability 1 to the global optimum, [36].

In this work we equip the simulated annealing algorithm with K neighborhood strategies and apply the Optimal Stopping Problem to determine the optimal time for changing the temperature. This study is organized as follows. In Section 2, we give an introduction to meta-heuristics in general and show the connection to Simulated Annealing. The concept of negative dynamic programming is introduced in Section 3. This provides the mathematical background for the modified Simulated Annealing algorithm provided in Section 4 using the optimal stopping problem. We illustrate this new method on the Traveling Salesman Problem in Section 5. We close with some conclusions in Section 6.

2 Metaheuristics and Simulated Annealing

Metaheuristic methods perform a more or less systematic search over the solution space by simultaneously employing processes to improve the solution quality as well as strategies to escape local optima. Metaheuristics got a lot of attention in the scientific community over the last decades. The significant advantages of metaheuristics over other solution methods have made them the favorable

practical solution method for solving complex combinatorial optimization problems, [24].

Although these methods cannot guarantee optimality of their solutions, they have shown a credible performance in solving real world problems compared to exact methods. Moreover, metaheuristics can improve their performances in some circumstances when they are equipped with mechanisms borrowed from the exact methods, *e.g.* bounding, [44, 9].

There is a wide range of metaheuristic methods, each one using its own approach [8, 22, 7]. However, all of these methods can be classified into a few groups based on their structures. One of the most basic heuristics which plays a critical role in almost all metaheuristic methods is called *local search*. Local search starts from an initial solution and tries to improve its quality by a sequence of moves toward a local optimum [32]. Many metaheuristics are based on local search techniques; among them are for instance Simulated Annealing (SA), Variable Neighborhood Search (VNS) [39, 28], Tabu Search (TS) [23, 25], Greedy Randomized Adaptive Search Procedures (GRASP) [19, 20], Genetic Algorithm (GA) [30, 26], Differential Evolution (DE) [43, 21] and Ant Colony Optimization (ACO) [17].

In general, all metaheuristics can be broken down into the following three main components [1]:

- **Initial Solution:** The quality of the initial solution affects the performance of the metaheuristics. To diminish this dependency, some methods start their search from multiple initial solutions.
- **Neighborhood Selection:** The way in which a metaheuristic moves from one solution to its neighbor is another critical component in all metaheuristics. In the early applications and theoretical developments of search methods in metaheuristics, the impact of the neighborhood selection has been underestimated. Instead, the neighbor selection has been limited to some factors such as ease of implementation, evaluation speed, usage of other researchers and empirical studies, [5]. In fact, mataheuristics often use the same neighborhood selection strategy during the course of their search. Using different neighborhood selection strategies in a systematic way can significantly improve their efficiency and effectiveness. Especially selecting the neighborhood in multiple ways can be a powerful tool in preventing the method from being trapped in local optima. To the best knowledge of the authors, the only metaheuristic focusing on different neighborhood structures is VNS, see [24, 29].
- **Optimization Strategy:** Optimization strategy is the policy deciding about the acceptance or rejection of a new solution. This is the most diverse aspect of metaheuristics and it varies from a simple procedure to a complex and sophisticated algorithm.

2.1 Neighborhood Selection Analysis

There are several factors to consider when defining a neighborhood structure. The following are characterizations of a proper neighborhood structure, [1]:

- **Effectiveness:** the power of the neighborhood structure in covering the whole feasible space.
- **Efficiency:** the efficiency of a neighborhood structure which is the quality of its performance in covering the feasible region depends on several (contradictory) factors:
  - **Speed:** the number of moves needed to reach any arbitrary point in the feasible region.
  - **Computational Effort:** the computations needed for each movement.
  - **Size (Number of Neighbors):** the size of a neighborhood structure is defined as the number of solutions which are accessible in an immediate move from the current solution. A larger number is usually an advantage as any arbitrary solution can be reached in less number of moves.
  - **Information Volume:** the amount of information transformed. This information may be used to perform better moves through the feasible space. For instance, there are gradients, Hessian matrix, eigenvalues and convexity information for the continuous space and taboo list, function characteristics and lower & upper bounds for the discrete space.

2.2 Simulated Annealing

Here, we describe SA in a mathematical manner. We do not go into full details but brief all the concepts we need in the later sections. For further mathematical analysis including the convergence rate and analysis, please refer, for instance, to the book by Otten and Ginneken [40] or the book by Salamon et al. [49].

Suppose Ω is the set of all feasible solutions and $f : \Omega \to \mathbb{R}$ is the objective function defined over the solution space. The purpose is to solve the (nonconvex) optimization problem

$$\max f(\omega) \tag{1}$$
$$\text{s.t. } \omega \in \Omega \tag{2}$$

That is, we want to find the global maximum ω^* in the solution space Ω. The global maximum ω^* has the property that for every $\omega \in \Omega$ we have $f(\omega) \leq f(\omega^*)$.

Let us define $N(\omega)$ as the neighborhood function for any $\omega \in \Omega$ and consider Algorithm 2.1 for a generic SA. GSA starts from an initial solution $\omega_0 \in \Omega$. At each *iteration*, GSA chooses the

Algorithm 2.1 Generic Simulated Annealing (GSA)

Input: initial feasible solution ω_0, initial temperature t_0, number of iterations M_p for each p, neighborhood structure $N(\omega)$, overall stopping criteria and temperature decreasing rule

Output: feasible solution ω

1: initialize *step* counter: $p = 0$
2: set current solution ω to initial solution ω_0: $\omega = \omega_0$
3: **while** the stopping criteria are not met **do**
4: **for** M_p *iterations* **do**
5: compute a solution belonging to the neighborhood of the current solution: $\omega' \in N(\omega)$.
6: **if** $f(\omega') - f(\omega) \geq 0$ **then**
7: $\omega \leftarrow \omega'$
8: **else**
9: $\omega \leftarrow \omega'$ with probability $\exp\left(\frac{f(\omega')-f(\omega)}{t_p}\right)$
10: **end if**
11: **end for**
12: decrease temperature t_p
13: $p \leftarrow p + 1$
14: **end while**
15: **return** feasible solution ω.

next solution among the neighbors of the current one, stage 5. The thermodynamic behavior of the system is modeled through the *Metropolis function* such that the probability of accepting the new solution ω' is defined as

$$\begin{cases} \exp\left(\frac{f(\omega')-f(\omega)}{t_p}\right), & \text{if } f(\omega') - f(\omega) < 0 \\ 1, & \text{otherwise} \end{cases} ,$$

in which t_p is the temperature parameter defined in the *iteration* loop p such that $\lim_{p\to\infty} t_p = 0$, stages 4 to 11 of Algorithm 2.1. For each temperature t_p, this process is repeated M_p times, stage 4. After these M_p iterations are performed, the current temperature t_p is decreased, stage 12. One of the easiest cooling procedures is a geometric sequence

$$t_p = \gamma^p \cdot t_0 \quad , \qquad 0 < \gamma < 1 \quad ,$$

in which the temperature at each *step* is γ times the temperature of the previous step. GSA repeats these steps until some stopping criteria is met, stage 3, and returns its current solution ω. Recognize that this might not be the best solution found and hence, one could modify the generic algorithm to store in addition to the current solution also the best solution. We brief on possible stopping criteria in Section 2.2.1.

As already mentioned, the cooling procedure is a key element in the search mechanism of SA. If the cooling procedure is slow enough, then the system can reach its steady state at each temperature t_p. This steady state follows the Boltzmann machine which can be defined as the probability of system being in state ω with the objective function value of $f(\omega)$ at temperature t_p [35]. The

probability of the system being in state ω at temperature t_p is then given by

$$\frac{\exp\left(-f(\omega)/t_p\right)}{\sum_{\omega' \in \Omega} \exp\left(-f(\omega')/t_p\right)} \quad .$$

2.2.1 Implementation Issues of Simulated Annealing

In general, the implementation of simulated annealing can be analyzed from two different points of views [18]:

1. The generic choices, being the same among all problems.
2. The problem-specific choices, changing from one problem to another.

The generic options for the implementation of SA can be stated as

- **Generation Probability Function:** determines the probability of choosing each of the current solution's neighbors. It is often set as a uniform function. However, sometimes other distributions are preferable, based on smarter mechanisms.
- **Acceptance Probability:** the probability of accepting a solution with a lower quality.
- **Cooling Schedule:** the rate in which the temperature decreases may heavily affect the performance of SA. In fact the convergence of the algorithm is proven when the temperature decreases with a logarithmic rate [42]. A low cooling rate may increase the running time of the algorithm while a high cooling rate may cause being trapped in a local optimum.
- **Stopping Rule:** the stopping criteria of the algorithm. It can be defined in many different ways. For instance, a simple stopping rule can be an upper limit on the number of iterations or an upper limit on the number of decreases in the temperature. The following two rules are among the most common stopping rules [42]:
  - If the best value found for the objective function so far does not increases by at least $\varepsilon_1\%$ after performing p_1 number of temperature decreases, the algorithm stops.
  - If the number of accepted moves in p_2 temperature decreases is less than $\varepsilon_2\%$, then the algorithm stops.

The problem-specific choices should be chosen carefully based on the nature of each problem. They are:

- **Objective Function and Solution Space:** their structure is very important in implementing the simulated annealing method. While the soft constraints can be added to the objective

function with a penalty function, the hard constraints should be considered in defining the feasible region [52].

- **Neighborhood Selection Strategy:** the way in which the algorithm travels through the feasible space. This is the main focus of this study.

More about implementation issues of SA can be found in the book [34, Chapter 5]. For a discussion of the advantages and pitfalls of SA refer, for instance, to [31].

3 Negative Dynamic Programming

A *stochastic process*

$$X = \{X_t, t \in T\}$$

is a collection of random variables. That is, X_t is a random variable for each t belonging to index set T. Here, t refers to time and X_t is the state of the process at time t, [48, Chapter 1.9]. Consider a stochastic process that is observed at the beginning of a discrete time period to be in a particular state X_0. After observation of the state, an action must be chosen. Based only on the state at that time and the action chosen, an expected reward is earned and the probability distribution for the next state is determined. We call the set of all possible actions to be chosen at each state the *action space*, [47, Chapter 3].

Now consider a stochastic process in which the state space is defined as a countable subset of non-negative integer numbers and with the action space of A. In addition, assume that if we are in state i and the action a is taken, then the cost of

$$C(i, a) \geq 0 \tag{3}$$

is imposed to the system. As this problem is equivalent to a system with non-positive reward function $R(i, a) = -C(i, a)$, we call this problem *Negative Dynamic Programming*. For a comprehensive introduction to Negative Dynamic Programming, see [53].

A policy is defined as a procedure introducing the action to be chosen as a function of the state of the process. For each policy π, we define the expected profit function as

$$V_\pi(i) = \mathbb{E}_\pi \left[\sum_{n=0}^{\infty} C(X_n, a_n) \,|\, X_0 = i \right] \qquad \forall i \geq 0 \quad . \tag{4}$$

Equation (4) is derived from dynamic programming techniques [6] and indicates that the expected profit, if we start in state i and choose our action based on policy π, is equal to the expected value

of the summation of all rewards (costs) obtained in each single state.

Since $C(i,a) \geq 0$, V_π may become infinite. If

$$V(i) = \inf_\pi V_\pi(i) < \infty \quad ,$$

then π^* is the optimal policy if

$$V_{\pi^*}(i) = V(i)$$

for each $i \geq 0$. If $V(i)$ is infinite then all policies are optimal with infinite expected profit. For those instances where there is at least one state i with the property $V(i) < \infty$, the following two theorems can be stated.

Theorem 1 ([47]). *The optimality equation for an infinite horizon can be defined as*

$$V(i) = \min_a \left[C(i,a) + \sum_j P_{ij}(a) V(j) \right]$$

for each $i \geq 0$.

Theorem 2 ([47]). *Let f be a stationary policy defined by*

$$C(i, f(i)) + \sum_j P_{ij}(f(i)) V(j) = \min_a \left[C(i,a) + \sum_j P_{ij}(a) V(j) \right] \qquad \forall i \geq 0 \quad , \tag{5}$$

then, for each $i \geq 0$, we have $V_f(i) = V(i)$. In other words, f is an optimal stationary policy.

In fact, this theorem indicates that if the policy f at each state chooses that action minimizing the expected cost of the system if it starts from that particular state and continues to infinity, then this policy is optimal. For more references on the application of dynamic programming in stochastic processes and more advanced topics, see [45, 50].

3.1 Optimal Stopping Problem

The *Optimal Stopping Problem* is a classic problem in Negative Dynamic Programming and has been widely studied, [47, 16]. It has a broad range of applications in different areas including statistics and finance. For advanced topics and applications, see [15]. The reader is referred to [51, 14, 11, 12, 55] for different settings of the problem. We use its solution approach to develop a new method in neighborhood search for Simulated Annealing.

The Optimal Stopping Problem can be stated as follows. Given is a system with non-negative integer states. Suppose, in each state the decision maker has the opportunity to either stop in the

current state i and gain the corresponding reward of $R(i)$, or pay a cost of $C(i)$ and continue the process for one more step. If she decides to continue, she will be in state j in the next iteration with probability P_{ij}. All values of $R(i)$ and $C(i)$ are non-negative. Let us define action 1 as the stop decision and state 2 as the continue decision. Theoretically, we can decide to continue at each state and never stop. Therefore, for technical reasons, we define state ∞ to represent the stop state. From these definitions, the following transition probabilities are immediate

$$P_{i,\infty}(1) = 1, \quad P_{i,\infty}(2) = 0, \quad P_{i,j}(1) = 0, \quad P_{i,j}(2) = P_{ij}, \quad P_{\infty,\infty}(a) = 1 \quad , \tag{6}$$

for all states i, j and all actions $a \in \{1, 2\}$. Moreover, the cost function is given by

$$C(\infty, a) = 0, \quad C(i, 1) = -R(i), \quad C(i, 2) = C(i) \quad , \tag{7}$$

for all states i.

As we have seen in equation (3), Negative Dynamic Programming requires non-negative cost. In fact, the rewards $R(i)$ can be considered as negative costs, $C(i, 1) = -R(i) \leq 0$. Hence, the Optimal Stopping Problem cannot be embedded in the Negative Dynamic Programming framework in its current form. Thus, we need a method to transform it to a Negative Dynamic Programming problem. To do so, we make the following two assumptions

$$\inf_i C(i) > 0 \quad \text{and} \tag{8}$$

$$\sup_i R(i) < \infty \quad . \tag{9}$$

Equation (8) means that the cost of continuing the process is strictly positive. That is, it is bounded from below by a positive value $\varepsilon > 0$. Similarly, equation (9) ensures that the reward in each state is bounded from above.

Using assumptions (8) and (9), we set

$$R = \sup_i R(i)$$

and re-define the Optimal Stopping Problem in the following way. Everything stays the same with the only difference that the reward earned in state i is $R(i) - R$, when deciding to stop. Equivalently, the cost of $R - R(i) \geq 0$ is paid. Similar to equation (4), we define $\overline{V}_\pi$ as the expected cost for each policy π. It can be shown that for such a policy we have

$$\overline{V}_\pi(i) = V_\pi(i) + R \qquad \forall i \geq 0 \quad .$$

The proof is straightforward and can be found, for instance, in [47, Chapter 3]. As a consequence, any optimal policy for the Optimal Stopping Problem is optimal for the re-defined version, and vice versa.

A *Markovian stochastic process* [48, 45] is a stochastic process in which the state of the system in the future given the past and present states, is independent of the past states and depends only on the present state.

Since the re-defined problem is a Markovian decision process with non-negative costs, it can be considered as a Negative Dynamic Programming problem and can be treated using the above cited theorems. Theorem 1 implies

$$\overline{V}(i) = \min \left[R - R(i), C(i) + \sum_j P_{ij} \overline{V}(j) \right] \qquad i \geq 0 \quad .$$

As $V(i) = \overline{V}(i) - R$, we have

$$V(i) = \min \left[-R(i), C(i) + \sum_j P_{ij} V(j) \right] \quad .$$

Hence, the optimal policy is the same in both problems. Now, let $V_0(i) = -R(i)$ and define $V_n(i)$ as the minimum expected cost if the decision maker starts in state i and has at most n decision opportunities before stopping. Then, for $n > 0$ we have

$$V_n(i) = \min \left[-R(i), C(i) + \sum_j P_{ij} V_{n-1}(j) \right] \qquad i \geq 0 \quad .$$

Since adding one more decision opportunity will increase the chance to gain more, the following inequality holds

$$V_n(i) \geq V_{n+1}(i) \geq V(i) \quad .$$

Finally, we can conclude that

$$\lim_{n \to \infty} V_n(i) = V(i) \quad . \tag{10}$$

Equation (10) is called *Stability condition*. A system satisfying this condition is called *stable*. It can be shown that the two assumptions (8) and (9) result in stability, [47, Chapter 3].

Let us define set B as

$$\begin{aligned} B &= \left\{ i : -R(i) \leq C(i) - \sum_{j=0}^{\infty} P_{ij} R(j) \right\} \\ &= \left\{ i : R(i) \geq \sum_{j=0}^{\infty} P_{ij} R(j) - C(i) \right\} \quad . \end{aligned}$$

Set B is in fact the set of all states where stopping is at least as good as continuing for exactly one more step and then stopping. The policy which stops at the first time when the system enters a state i belonging to set B is called *One-Stage Look-Ahead policy*.

Next, we see that if set B is a closed set over the state space, *i.e.*, when the system enters a state in B then the probability leaving B is 0, then the One-Stage Look-Ahead policy is optimal, assuming the stability assumptions.

Theorem 3 ([47]). *If a process is stable and $P_{ij} = 0$ for $i \in B$ and $j \notin B$, then the optimal policy stops at state i, if and only if $i \in B$. In other words, the One-Stage Look-Ahead policy is optimal.*

Note that the assumption we have made for set B is in fact equivalent to the closeness of this set over the state space.

4 Simulated Annealing with Optimal Stopping Time

The use of multiple neighborhood structures has not received too much attention in the literature of SA. In fact, the only paper in which SA is implemented using several neighborhood structure is by Bouffard and Ferland, [10]. In that paper, SA annealing is integrated with VNS to solve the resource-constrained scheduling problem. Three nested neighborhood structures have been employed along with a separately designed one. Their numerical studies confirms the advantage of multiple neighborhood selection strategies in SA. However, their algorithm is different than ours in the way in which the number of iteration at each temperature is determined and how the change in neighborhood structure is proposed. We are using Optimal Stopping Problem while their algorithm is based on Variable Neighborhood Search Method.

In this section, we introduce a novel variation of SA using the Optimal Stopping Problem for solving the combinatorial optimization problem P. Without loss of generality, we can assume P to be a maximization problem in the form stated in equations (1) and (2). The main difference of the new algorithm with the generic SA is the usage of K different neighborhood structures. These structures can be defined based on any heuristic approach and are mostly problem-specific. We have to decide which of the K structures to use at each step. In fact, in each step we have to solve a decision problem with K different alternatives. After choosing the neighborhood structure in the current temperature t_p, in each iteration, we are encountered with a two-alternative decision problem, either to continue or to stop. When we decide to stop, we may either switch to another neighborhood structure, if there is a profitable one, or decrease the temperature and repeat the process. These decisions are made using stochastic dynamic programming with a small amount of computational effort. Next to the K neighborhood structures, compared to the generic Simulated Annealing, the parameter of M_p iterations is determined by an optimal stopping decision.

Assume that the objective function f can take n different values. In the case that it is continuous, we can divide the range of changes in the value of the objective function into n smaller intervals; each of them is represented by their mid-point. Such a range can be obtained by lower and upper bounds on the value of the objective function f. As we have a maximization problem, any feasible solution determines a lower bound. To obtain an upper bound, we may use different kinds of relaxation such as LP relaxation or Lagrangian Relaxation.

The *state* of the system is defined as the best value of the objective function found so far. For instance, if the best found value for the objective function happens to be in an interval with mid-point r, we assert that the system is in state r. The justification for this definition is provided later in this section when we associate the state of the system with SA. Also, we define $P_{ij}^{(k)}$ as the transition probability which is the probability of going from state i to state j if the neighborhood structure k is employed. We explain how these transition probabilities can be obtained in Section 4.1. Since each neighborhood structure incurs a different amount of computational time, we define $C^{(k)}$ as the cost of using the neighborhood structure k for one iteration. In general, this cost can also be state-dependent to represent the bigger effort needed as the value of objective function improves. But in our case, we keep it constant for each neighborhood structure.

As is Section 3.1, we define ∞ as the stop state, 1 as the stopping decision and 2 as the continuing decision. Let us define now the cost and transition probabilities obtained in formulae (6) and (7) for our case. Obviously, we have

$$C(i,2) = C^{(k)} \quad .$$

The cost of the system when stopping at a solution with the objective function value of i is equal to the negation of the highest value of the objective function that we have found so far. Assume we stop at a solution with the objective function of i with a current temperature of t_p after doing M_p iterations. If O_m represents the value of the objective function at iteration m, then we have

$$C(i,1) = -\max\{O_1, O_2, O_3, \ldots, O_{M_p} = i\} \quad .$$

This means, if we found a solution with the objective function value of $j > i$ at iteration m, then our system has the *recall property*, that is, it remembers this solution as the best solution found until a solution with a better value of the objective function is found or the system stops. In the case of stop, if j is still the best solution found, then it is reflected as the state of the system. Hence, we get the following transition probabilities

$$P_{ij}^{(k)} = \begin{cases} g^{(k)}(i,j), & \text{if } j > i \\ \sum_{j \le i} g^{(k)}(i,j), & \text{if } j = i \\ 0, & \text{if } j < i \end{cases} \quad .$$

where $g^{(k)}(i,j)$ is the probability of moving from a solution with the objective value in range i to a solution with the objective value in range j using the kth neighborhood structure; regardless of any other decision. It is easy to see that the above probabilities sum up to 1.

Similar to Section 3.1, where we transformed the Optimal Stopping Problem to a Negative Dynamic Programming problem, we define a second problem with non-negative costs and the same optimal policy as the original problem. This enables us to use the Optimal Stopping Problem principles to find the optimal policy for the original problem. Now, redefine the cost of each state as

$$C(i,1) = U - \max\{O_1, O_2, O_3, \ldots, O_{M_p}\} \quad , \tag{11}$$

where U is a known upper bound for the value of the objective function. By definition (11), it is obvious that $C(i,1) \geq 0$.

Comparing this problem with the Optimal Stopping Problem defined in the Section 3.1, bears in mind that the optimal policy for this problem has the following structure. If the highest value of the objective function found so far is greater than or equal to a threshold value i_k^*, then stop and otherwise continue. Hence, the stopping criterion is defined as

$$\max\{O_1, O_2, O_3, \ldots, O_{M_p}\} \geq i_k^* \quad .$$

Next, let us find the optimal value of i_k^*. Define set B as

$$B = \{i : i \geq i_k^*\} \quad .$$

It is clear that B is a closed set; *i.e.*, if the system enters a state belonging to B, then the probability that the next states of the system are not in B is equal to 0. The reasoning can be as follows. If we reach a value of the objective function which is in B, then it remains the highest value so far or it improves, but it is not possible that the best found value of the objective function declines. Hence, we can summarize this in the following formula

$$\left.\begin{array}{lll} i \in B & \rightarrow & i \geq i_k^* \\ j \notin B & \rightarrow & j < i_k^* \end{array}\right\} \quad \longrightarrow \quad j < i \quad \longrightarrow \quad P_{ij}^{(k)} = 0 \quad .$$

Regarding this new definition of our transition probabilities which implies the closeness of B, it follows that all the assumption of Theorem 3 are satisfied and the One-Stage Look-Ahead policy is optimal.

Now, let us restate set B as

$$\begin{aligned} B &= \left\{ i : i \geq i \sum_{j \leq i} g^{(k)}(i,j) + \sum_{j>i} j g^{(k)}(i,j) - C(k) \right\} \\ &= \left\{ i : C_{(k)} \geq i \sum_{j \leq i} g^{(k)}(i,j) - i + \sum_{j>i} j g^{(k)}(i,j) \right\} \\ &= \left\{ i : C_{(k)} \geq \sum_{j>i} (j-i) g^{(k)}(i,j) \right\} \\ &= \{ i : i \geq i^* \} \end{aligned}$$

So at each temperature t_p, while implementing the kth neighborhood structure, in iteration 0, it is enough to find the value of i_k^* and continue until reaching a value of the objective function which is greater than or equal to i_k^*. Then, we stop and investigate whether proceeding with a new neighborhood structure, or reducing the temperature, is more beneficial. This decision can be made by comparing the updated values of i_k^* for each neighborhood structure with the cost of its implementation. In other words, after the system stops with respect to the current neighborhood structure, all the values of i_k^* are updated. These values are in fact the expected value of objective function when implementing this neighborhood structure. It is up to the decision maker to decide if it is worthy to continue with a new neighborhood structure knowing the value of i_k^* and $C^{(k)}$.

The only missing step in the chain of our proposed algorithm is which neighborhood structure has to be chosen after decreasing the temperature; *i.e.*, which neighborhood structure is the best to start with in the new temperature. Of course, that neighborhood structure is desirable maximizing the expected value of the objective function. For example, consider the two neighborhood structures k and k'. By definition, i_k^* is the expected profit before stopping in the case of using k, and $i_{k'}^*$ is the expected profit before stopping in the case of using k'. So at each temperature, we compute the values of i_k^* before starting. Afterwards, we implement the neighborhood structure having the highest value of i_k^*.

4.1 How to Obtain the Transition Probabilities

As mentioned in Section 4, to find the transition probabilities $P_{ij}^{(k)}$ we need the values of $g^{(k)}(i,j)$. They are defined as the probability of moving from range i to range j using the neighborhood structure k regardless to whether we accept or reject the new solution. To obtain these values, we have to find out how the neighborhood structure affects the change in the value of objective function in general. As a neighborhood structure changes more components of the current solution, the range of changes in the value of objective function for the resulted solution increases. Roughly

speaking, a more complicated neighborhood structure may cover a wide range of the feasible region and has the potential of moving far away in a few number of iterations while a simple neighborhood structure needs far more iterations to move from one part of feasible region to another. Thus, the probability of a specific change in the value of objective function plays a critical role in finding the transition probabilities.

Define $\gamma_d^{(k)}$ as the probability of a change equal to d in the objective function f using neighborhood structure k. Then the transition probabilities $P_{ij}^{(k)}$ can be computed by considering the difference between i and j compared to the value of d. For an illustration, see Section 5 where it has been applied to the traveling salesman problem.

The only missing part in this procedure is that $\gamma_d^{(k)}$ actually represents the probability of change and does not provide any information about the direction of this change. In fact, this depends strongly on the value of the objective function in the current solution. For instance, if the value of the objective function in the current solution is in range i, then the bigger the value of i, the higher is the possibility that the change will happen in the negative direction. This implies that, in addition to $\gamma_d^{(k)}$, we need a quantitative measure to evaluate the direction of change. To do so, we use the lower bound L and upper bound U of the objective function and define the *improvement index* as

$$\alpha_i = \frac{i - L}{U - L} \quad .$$

The closer the value of α is to 0, the higher is the possibility of moving in the positive direction. So, if i is the current value of the objective function then

$$(1 - \alpha_i)\gamma_d^{(k)}$$

is the probability of moving to $i + d$ in the next iteration and

$$\alpha_i \gamma_d^{(k)}$$

is the probability of moving to $i - d$ in the next iteration. Finally, $\gamma_d^{(k)}$ can be defined based on the problem information. The way to do so depends heavily on the nature of the problem under study. We illustrate this procedure in Section 5 for the traveling salesman problem.

4.2 Summary of the Algorithm

Please consider Algorithm 4.1. Algorithm SAOST is similar to Algorithm 2.1, but having k different neighborhood structures instead of one. These neighborhood structure are defined beforehand based on any heuristic approach. In stage 3 of Algorithm 4.1, the cost of using each neighborhood

structure for one iteration $C^{(k)}$ are computed, regarding to the computational burden imposed by each neighborhood structure and all other impacting factors. After that, in stage 5, the initial values for the transition probabilities are computed; *i.e.*, the values for $\gamma_d^{(k)}$ with respect to each neighborhood structure are found and used to determine $P_{ij}^{(k)}$. In case we do not have enough information to compute these probabilities, we can start with a uniform distribution and update the values as the algorithm proceeds. In the outer while loop, stage 5, the the optimal value of i_k^* for each neighborhood structure is computed in stage 6. This allows us to find the neighborhood structure

$$k^* = \text{argmax}\{i_k, k = 1, 2, \ldots, K\}$$

having the highest value. The inner while loop, stages 8 to 15, is basically the same as stages 4 to 11 of Algorithm 2.1 but replacing parameter M_p by the dynamically updated criteria

$$f(\omega) < i_{k^*}^* \quad .$$

After leaving the inner while loop, update the lower bounds and, if applicable, also update the upper bounds; stage 16. Using the new lower and upper bound, the current value of the objective function and the change probabilities $\gamma_d^{(k)}$, the transition probability matrix is determined in stage 17. Afterwards, one can decided to continue in the current temperature with a new neighborhood structure or to decrease the temperature according to a predefined cooling scheme.

4.3 Advantages and Disadvantages

Despite the variety of heuristic algorithms, there are some measures that enable us to evaluate each method and decide about its properness for a specific problem. Following Glover and Kochenberger, a good meta-heuristic must have the followings properties [24, Chapter 6]:

1. **Simplicity:** A good meta-heuristic algorithm should be based on some simple principles which are applicable in all cases.
2. **Precision:** Different steps of a meta-heuristic should be defined mathematically in an exact way and ambiguous terms should be avoided.
3. **Coherence:** All steps of a meta-heuristic for any specific problem have to follow the main principles of that heuristic.
4. **Efficiency:** A good meta-heuristic should have the ability of finding an optimal or near-optimal solution for a large group of real world problems.
5. **Effectiveness:** A good meta-heuristic should find its final solution in a reasonable amount of computational time.
6. **Robustness:** The performance of a meta-heuristic need to be verified on a large variety of problems. A good meta-heuristic is the one which is consistent with a wide range of problems.

Algorithm 4.1 Simulated Annealing with Optimal Stopping Time (SAOST)

Input: initial feasible solution ω_0, initial temperature t_0, lower bound L and upper bound U for the objective function f, k neighborhood structures N_k, overall stopping criteria and temperature decreasing rule

Output: feasible solution ω

initialize *step* counter: $p = 0$
set current solution ω to initial solution ω_0: $\omega = \omega_0$
compute the cost of using each neighborhood structure: $C^{(k)}$
compute the initial values for the transition probability matrix: $P_{ij}^{(k)}$
while the stopping criteria are not met **do**
  find the value of i_k^* for each neighborhood structure
  choose the neighborhood structure k^* having the highest value
  while $f(\omega) < i_{k^*}^*$ **do**
    compute a solution belonging to neighborhood structure k^* of the current solution: $\omega' \in N_{k^*}(\omega)$
    if $f(\omega') - f(\omega) \geq 0$ **then**
      $\omega \leftarrow \omega'$
    else
      $\omega \leftarrow \omega'$ with probability $\exp\left(\frac{f(\omega')-f(\omega)}{t_p}\right)$
    end if
  end while
  update the lower and upper bounds
  construct the transition probability matrix for each neighborhood structure: $P_{ij}^{(k)}$
  if decided to decrease temperature **then**
    decrease temperature t_p
    $p \leftarrow p + 1$
  end if
end while
return feasible solution ω.

7. **User friendliness:** A meta-heuristic algorithm need to be easy to understand and more importantly, easy to implement.
8. **Innovation:** It is expected for a new meta-heuristic to have some kind on innovation in its principles and applications.

The following advantages can be stated for our proposed method:

- Its ability to use the past information of the problem in making future decisions (tuning the search strategies)
- Less computational effort at each temperature because the best number of iteration in known
- Less algorithm parameters
- The ability to cover a wide range of the feasible space via appropriate selection of neighborhood structures
- Its flexibility in employing different neighborhood structure
- Considering different computational time for each neighborhood structure using the cost parameter

5 TSP as an Illustrative Example

In this section, we apply the proposed algorithm to the traveling salesman problem (TSP) just like the first application of SA by Černý [13]. The purpose is to illustrate the new algorithm. Hence, we only consider two basic neighborhood structures and do not discuss how to derive bounds or initial solutions. For a detailed discussion of the TSP, refer to [27, 4]. Different neighborhood structures are discussed in detail in [46]. From now on, we assume that the TSP has n nodes and the graph is complete.

In the first neighborhood structure, two adjacent nodes are randomly selected in the current solution and their positions are swapped. This is shown in Figure 1. The chain of nodes illustrate the order in which the nodes are visited in a tour starting from node 1 and returning after the nth node to node 1 again. As shown, two edges need to be removed and two new edges have to be created.

This neighborhood structure has the following characterizations:

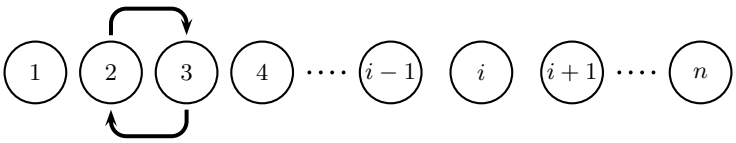

Figure 1: The edges $(1,2)$ and $(3,4)$ should be substituted with the edges $(1,3)$ and $(2,4)$ in the new solution.

- For every arbitrary solution, exactly n neighbor solutions exist for $n \geq 5$. These solutions are obtained by exchanging the position of every two consecutive nodes in the current solution.
- The *speed* of a neighborhood structure is defined as the number of moves needed to go from a solution to any other arbitrary solution. This number is $O(n^2)$ for this neighborhood structure. Considering a sequence of n distinct numbers and a desired order, the first number may need $n-1$ moves to reach its appropriate position. This is $n-2$ for the second number and so on for others. This clearly gives us $O(n^2)$ number of moves. The only difference in this case with a TSP tour is that in a TSP tour, since we have loop and elements can move in both directions, we may need fewer number of moves to seat every node. But this at most may give us half of what we get above for the sequence of numbers which keeps the complexity still $O(n^2)$.
- The computational effort needed for each move is four units, as two edges need be removed and two new edges have to be created.

The second neighborhood structure used in our numerical analysis is defined by exchanging the position of every two arbitrary nodes. This neighborhood structure is shown in Figure 2.

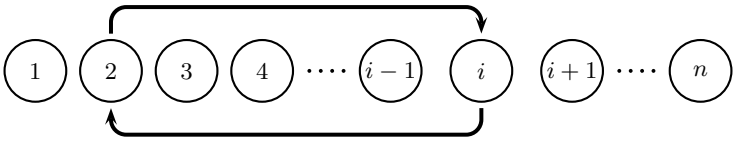

Figure 2: The edges between $(1,2),(2,3),(i-1,i)$ and $(i,i+1)$ should be substituted with the edges $(1,i),(i,3),(i-1,2)$ and $(2,i+1)$ in the new solution.

The following characterizations can be stated for this neighborhood structure:

- There exist $\binom{n}{2} = \frac{n(n-1)}{2}$ neighbors for each solution. These solutions are obtained by exchanging the position of every two arbitrary nodes.
- The speed of this neighborhood structure is $O(n)$: Note that in this case, the number of moves needed to go from a solution to any arbitrary solution in the worst case would be $O(n)$. This is as in the worst case, each node needs at most one move to be seated in its appropriate position.
- The computational effort is eight units. As it is clear from Figure 2, four existing edges need to be substituted with four new edges.

Next, we define the transition probabilities $P_{ij}^{(k)}$ of going from state i to state j using neighborhood structure k. To find the values of $\gamma_d^{(k)}$, the probability of an objective function change of d using neighborhood structure k, 1000 random moves are generated for each neighborhood structure k. For example, for the first neighborhood structure two edges are randomly picked and substituted with two other edges. The difference between the summation of the first two and the second two edges is recorded as the value of d for this experiment. Note that all of this process is performed regardless to feasibility. Recognize that we have not produced any feasible or infeasible solution. The purpose is to get a measure of how much the value of the objective function on average changes if a random move is implemented. For the second neighborhood structure, a similar process is performed but by exchanging four edges instead of two. Finally the values of d for each neighborhood structure are sorted in a frequency histogram and a discrete probability distribution is obtained. These probability distributions are in fact the values of $\gamma_d^{(k)}$.

The remaining parts of the procedure are based on the algorithm itself. Problem-specific are only the definition of the neighborhood structures and the computation of the transition probabilities, next to the usual variables of SA as initial solution, stopping criteria or cooling scheme.

6 Conclusion

We developed a new SA-based algorithm for solving combinatorial optimization problems by modifying its search process. This algorithm takes advantage of multiple neighborhood structures at the same time to reduce the chance of being trapped in the local optima. Unlike the Variable Neighborhood Search method which selects the neighborhood structure in a deterministic way, the idea behind this new method is based on a well-known problem in Stochastic Processes called Optimal Stopping Problem. At each iteration, the proposed algorithm chooses that neighborhood structure which has a higher expected improvement in the objective function. The generality of this method allows us to adapt and apply it to a wide range of complex problems with combinatorial nature.

References

[1] *An Analysis Framework for Metaheuristics: A Mathematical Approach.* York 14 Conference, Bath, England, 2005.

[2] *Simulated Annealing Algorithm for Daily Nursing Care Scheduling Problem.* Proceedings of the 3rd Annual IEEE Conference on Automation Science and Engineering, 2007. Scottsdale, AZ, USA.

[3] Emile Aarts, Jan Korst, and Wil Michiels. *Simulated Annealing*, chapter 7. Springer, 2005. Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques.

[4] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook, editors. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, 2007.

[5] J. W. Barnes, N. Colletti, and D. Neuway. Using group theory and transition matrices to study a class of metaheuristic neighborhoods. *European Journal of Operational Research*, 138:531–544, 2002.

[6] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 3rd edition, 2007.

[7] L. Bianchi, M. Dorigo, L. Gambardella, and W. Gutjahr. Metaheuristics in stochastic combinatorial optimization: a survey. Technical report, IDSIA, Dalle Molle Institute for Artificial Intelligence, 2006.

[8] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[9] A. Bolte and U. W. Thonemann. Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research*, 92:402–416, 1996.

[10] V. Bouffard and J. A. Ferland. Improving simulated annealing with variable neighborhood search to solve the resource-constrained scheduling problem. *J Sched*, 10:375–386, 2007.

[11] W. M. Byoce. Stopping rules for selling bonds. *Bell J. Econ. Manage Sci.*, 1:27–53, 1970.

[12] W. M. Byoce. On a simple optimal stopping problem. *Discrete Math.*, 5:297–312, 1973.

[13] V. Černý. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and applications*, 45(1):41–51, 1985.

[14] R. W. Chen and F. K. Hwang. On the values of an (m,p) urn. *Congr. Numer*, 1:75–84, 1984.

[15] Yuan Shih Chow, Herbert Robbins, and David Siegmund. *Great expectations: The theory of optimal stopping*. Houghton Mifflin, 1971.

[16] Morris H. DeGroot. *Optimal Statistical Decisions*. Wiley Classics Library. Wiley-Interscience, 2004.

[17] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

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

[19] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[20] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.

[21] V. Feoktistov. *Differential Evolution - In Search of Solutions*, volume 5 of *Optimization And Its Applications*. Springer, 2006.

[22] Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213, 2005.

[23] F. Glover. Future paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[24] F. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, New York, 2003.

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

[26] D.E Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[27] Gregory Gutin and Abraham P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Combinatorial Optimization. Springer, 2007.

[28] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-heuristics, Advances and trends in local search paradigms for optimization*, pages 433–458. Kluwer Academic Publishers, 1998.

[29] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[30] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press, 1975.

[31] A. L. Ingber. Simulated annealing: Practice versus theory. *J Mathl. Comput. Modelling*, 18(11):29–57, 1993.

[32] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 17:79–100, 1988.

[33] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[34] P. J. M. Laarhoven and E. H. L. Aarts, editors. *Simulated annealing: theory and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[35] F. T. Lin, C. Y. Cao, and C. C. Hsu. Applying the genetic approach to simulated annealing in solving some np-hard problems. *IEEE Transactions on Systems, Man. and Cybernetics*, 23(6):1752–1767, 1993.

[36] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34:11–124, 1986.

[37] Thelma D. Mavridou and Panos M. Pardalos. Simulated annealing and genetic algorithms for the facility layout problem: A survey. *Computational optimization and Applications*, 7:111–126, 1997.

[38] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[39] N. Mladenović. A variable neighborhood algorithm – a new metaheuristic for combinatorial optimization. In *Abstracts of papers presented at Optimization Days, Montréal*, page 112. 1995.

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

[41] D. T. Pham and D. Karaboga. *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer, 2000.

[42] M. Pirlot. General local search methods. *European Journal of Operational Research*, 92:493–511, 1996.

[43] Kenneth V. Price. Genetic annealing. *Dr. Dobb's Journal*, 220:127–132, 1994.

[44] Jakob Puchinger and G"unther R. Raidl. *Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification*, pages 41–53. Springer, 2005. Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach.

[45] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley-Interscience, 2005.

[46] Gerhard Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Lecture Notes in Computer Science. Springer, 1991.

[47] Sheldon M. Ross. *Introduction to Stochastic Dynamic Programming*. Probability and Mathematical Statistics. Academic Press, 1995.

[48] Sheldon M. Ross. *Stochastic Processes*. Wiley Series in Probability and Mathematical Statistics. Johnson Wiley & Sons, Inc., 2nd edition, 1996.

[49] Peter Salamon, Paolo Sibani, and Richard Frost. *Facts, Conjectures, and Improvements for Simulated Annealing*. SIAM Monographs on Mathematical Modeling and Computation, 2002.

[50] Linn I. Sennott. *Stochastic Dynamic Programming and the Control of Queueing Systems*. Wiley Series in Probability and Statistics. Wiley-Interscience, 1998.

[51] L. A. Shepp. Explicit solutions to some problems of optimal stopping. *Annals of Mathematical Statistics*, 40:993–1010, 1969.

[52] J. M. Stern. simulated annealing with a temperature dependent penalty function. *ORSA Journal on Computing*, 4:311–319, 1992.

[53] Ralph E. Strauch. Negative dynamic programming. *Annals of Mathematical Statistics*, 37(4):871–890, 1966.

[54] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57:1143–1160, 2006.

[55] David D. Yao and Shaohui Zheng. Sequential inspection under capacity constraints. *Operations Research*, 47(3):410–421, 1999.