Hill Climbing

Ammar Ahmad

Abstract

This report presents an overview of the advances around Hill Climbing Algorithms, focusing on the use cases and the variations. We discuss the theorems, proofs, complexities and use cases around it. It also includes Definitions and Proofs and Implications for future Research.

1 Hill climbing

The Hill Climbing algorithm is a heuristic search algorithm used for mathematical optimization problems. It belongs to the family of local search algorithms, which means it explores the solution space by making incremental changes to a current solution, moving towards the direction that maximizes or minimizes an objective function.

1.1 Basic Idea

: At each step, the algorithm evaluates the current solution and iteratively makes small adjustments to it, moving towards a better solution. It operates on the principle of "climbing" a metaphorical hill: it keeps moving uphill (towards a higher value for maximization problems or a lower value for minimization problems) until it reaches a peak where no further improvement is possible.

1.2 Steps of the Hill Climbing Algorithm:

1. Initialization: Start with an initial solution, which could be randomly generated or based on some predefined criteria.

2. Evaluation: Evaluate the quality of the current solution using an objective function. This function quantifies how "good" or "bad" the solution is with respect to the problem being solved.

3. Iteration: Repeat the following steps until a termination condition is met:

4. Neighbor Generation: Generate neighboring solutions by making small changes (mutations) to the current solution. These changes could involve tweaking parameters, adding or removing components, or any other modification that transforms the current solution into a new one.

5. Selection: Among the generated neighbors, select the one that optimally improves the objective function. This could involve selecting the neighbor with the highest increase (for maximization) or the lowest decrease (for minimization) in the objective function value.

6. Update: Update the current solution with the selected neighbor.

7. Termination: The algorithm stops when it reaches a predefined termination condition, such as reaching a maximum number of iterations, finding a solution that meets certain criteria, or when no further improvement is possible.

1.2.1 Complexity Analysis

: An heuristic search algorithm and local optimizer. One of the variant expands best nodes first, i.e. those that have min h(n) and forgets about the alternatives. Hill climbing is neither complete *nor* optimal, has a time complexity of $O(\infty)$ but a space complexity of O(b). The time and space complexity for hill climbing algorithm varies a lot according to the type of hill climbing algorithms.

1.3 Types of Hill Climbing:

- Stochastic Hill Climbing
- First Choice Hill Climbing
- Steepest Ascent Hill Climbing
- Random-restart hill climbing

1.3.1 Stochastic Hill Climbing

Stochastic Hill Climbing is a variant of the Hill Climbing algorithm that introduces randomness into the selection of neighbor solutions. Unlike deterministic Hill Climbing variants, which always select the best neighboring solution to move towards, Stochastic Hill Climbing probabilistically selects a neighbor based on certain criteria, allowing for exploration of a wider solution space. Here's an overview of Stochastic Hill Climbing:

1. Randomized Selection: Instead of deterministically choosing the best neighbor solution, Stochastic Hill Climbing randomly selects a neighbor solution based on a probability distribution.

2. Exploration-Exploitation Balance: Stochastic Hill Climbing balances exploration (searching for new solutions) and exploitation (improving the current solution) by introducing randomness into the selection process.

3. Probability Distribution: The probability distribution used for selecting neighbor solutions can vary depending on the specific implementation and problem domain. Common distributions

include uniform distribution, Boltzmann distribution, and others tailored to the problem characteristics.

4. Temperature Parameter: In some implementations, Stochastic Hill Climbing may use a parameter known as temperature to control the level of randomness in the selection process. Higher temperatures lead to more exploration, while lower temperatures favor exploitation.

Advantages:

- Enhanced exploration: Stochastic Hill Climbing can explore a broader range of solutions compared to deterministic variants, which may help avoid getting stuck in local optima.
- Flexibility: The use of probability distributions allows for adaptation to different problem domains and solution spaces.
- Robustness: Introducing randomness can make the algorithm more robust to noise and uncertainty in the objective function.

Disadvantages:

- Increased complexity: Introducing randomness adds complexity to the algorithm's implementation and analysis.
- Convergence issues: Depending on the selection mechanism and probability distribution, Stochastic Hill Climbing may converge more slowly or struggle to converge to an optimal solution.
- Tuning parameters: Choosing appropriate probability distributions and tuning parameters such as temperature can be challenging and may affect the algorithm's performance.

1.3.2 First Choice Hill Climbing

First Choice Hill Climbing is a variant of the Hill Climbing algorithm that aims to address one of the primary limitations of basic Hill Climbing methods: getting stuck in local optima due to always selecting the first neighboring solution encountered. In First Choice Hill Climbing, instead of blindly accepting the first neighboring solution, the algorithm evaluates multiple neighbors and selects the first one that improves the objective function. Here's how it works:

1. Randomized Selection: Like Stochastic Hill Climbing, First Choice Hill Climbing introduces randomness into the neighbor selection process. However, it differs in how it chooses among the generated neighbors.

2. Local Search with Random Restart: First Choice Hill Climbing combines local search with random restarts. It iteratively performs local search starting from different initial solutions, aiming to escape local optima and find better solutions.

3. Improved Exploration: By considering multiple neighboring solutions and selecting the first one that improves the objective function, First Choice Hill Climbing enhances exploration compared to basic Hill Climbing variants.

Advantages:

- Enhanced exploration: First Choice Hill Climbing can explore a broader range of solutions compared to basic Hill Climbing algorithms by considering multiple neighbors.
- Quick escape from local optima: By selecting the first improving neighbor, the algorithm can quickly escape local optima and continue searching for better solutions.
- Simplicity: First Choice Hill Climbing maintains the simplicity of basic Hill Climbing algorithms while offering improved exploration capabilities.

Disadvantages:

- Potential for premature convergence: Despite improved exploration, First Choice Hill Climbing may still converge prematurely, especially in complex solution spaces with many local optima.
- Computational overhead: Evaluating multiple neighbor solutions can increase the computational overhead, especially for problems with a large number of neighbors.
- Sensitivity to parameter settings: The performance of First Choice Hill Climbing may depend on the choice of parameters, such as the number of neighbors evaluated at each iteration.

1.3.3 Steepest Ascent Hill Climbing

Steepest Ascent Hill Climbing is a variant of the Hill Climbing algorithm that aims to mitigate one of its main drawbacks: the tendency to get stuck in local optima. Unlike basic Hill Climbing methods, which greedily select the first improving neighboring solution, Steepest Ascent Hill Climbing examines all neighboring solutions and selects the one that offers the most improvement in the objective function value. Here's how it works:

1. Exhaustive Neighbor Examination: Instead of selecting the first improving neighboring solution encountered, Steepest Ascent Hill Climbing evaluates all neighboring solutions and selects the one that maximally improves the objective function value (for maximization problems) or minimally decreases it (for minimization problems).
2. Enhanced Exploitation: By considering all neighboring solutions, Steepest Ascent Hill Climbing aims to exploit the local search space more effectively, potentially leading to quicker convergence towards better solutions.

Advantages:

- Better exploitation of the solution space: By considering all neighboring solutions, Steepest Ascent Hill Climbing can exploit the local search space more effectively than basic Hill Climbing methods, potentially leading to faster convergence towards optimal or near-optimal solutions.
- Reduced likelihood of getting stuck in local optima: The exhaustive examination of neighboring solutions reduces the risk of prematurely converging to suboptimal solutions trapped in local optima.

Disadvantages:

- Increased computational complexity: Evaluating all neighboring solutions at each iteration can significantly increase the computational overhead, especially for problems with a large number of neighbors or complex objective functions.
- Potential for slow convergence: The exhaustive examination of neighboring solutions may lead to slower convergence, particularly in large search spaces or when the objective function is expensive to evaluate.

1.3.4 Random Restart Hill Climbing

Random Restart Hill Climbing, also known as Random Restart Local Search, is a metaheuristic algorithm that combines the local search strategy with random restarts to overcome the limitations of getting stuck in local optima. This approach involves running a local search algorithm multiple times from different starting points (randomly selected initial solutions). Here's how it works:

1. Local Search: Random Restart Hill Climbing employs a local search algorithm, such as basic Hill Climbing or its variants like Steepest Ascent Hill Climbing or Simulated Annealing, to explore the solution space and improve solutions iteratively.
2. Random Restarts: Instead of relying on a single initial solution, Random Restart Hill Climbing starts the local search from multiple initial solutions, often randomly generated. After completing the local search from each starting point, the algorithm selects the best solution found among all runs.

Advantages:

- Overcoming local optima: Random Restart Hill Climbing mitigates the risk of getting stuck in local optima by exploring multiple regions of the solution space through random restarts.
- Increased exploration: By starting the local search from multiple initial solutions, the algorithm explores a broader range of solutions, increasing the chances of finding a global optimum.
- Simplicity: Random Restart Hill Climbing is relatively easy to implement and does not require sophisticated techniques compared to some other metaheuristic algorithms.

Disadvantages:

- Computational cost: Running the local search algorithm multiple times from different initial solutions increases the computational overhead, especially for complex optimization problems.
- Redundant exploration: Random restarts may lead to redundant exploration of similar regions of the solution space, particularly if the initial solutions are generated without much diversity.
- Parameter tuning: The performance of Random Restart Hill Climbing may depend on parameters such as the number of restarts and the termination conditions for individual local search runs, which may require tuning.

2 Extras

2.0.1 Exploration-Exploitation Trade-Off

The exploration-exploitation trade-off is a fundamental concept in optimization and decision-making algorithms, including Hill Climbing. It refers to the dilemma of balancing two conflicting objectives:

1. Exploration: This involves seeking out new or unexplored regions of the solution space. In the context of optimization algorithms like Hill Climbing, exploration entails trying out different solutions to discover potentially better ones. Exploration is essential for discovering diverse solutions and avoiding premature convergence to suboptimal solutions. However, excessive exploration may lead to inefficiency or failure to exploit promising regions of the solution space.
2. Exploitation: Exploitation focuses on maximizing the utility or benefit of known solutions. In optimization algorithms, exploitation involves intensively searching the vicinity of promising solutions to further improve them. Exploitation is crucial for maximizing the algorithm's progress towards the optimal solution and achieving convergence. However, excessive exploitation may cause the algorithm to get stuck in local optima or miss out on better solutions in other parts of the solution space.

Balancing exploration and exploitation is challenging because increasing one often comes at the expense of the other. Here's how this trade-off manifests in Hill Climbing algorithms:

- Exploitation Bias: Basic Hill Climbing methods tend to prioritize exploitation by greedily selecting the best neighboring solution available at each iteration. While this approach may lead to quick progress towards local optima, it also increases the risk of getting stuck in suboptimal solutions, especially if the algorithm converges prematurely.
- Exploration Enhancement: Variants of Hill Climbing, such as Stochastic Hill Climbing or Random Restart Hill Climbing, introduce mechanisms to enhance exploration. Stochastic methods

Figure 1: Algorithm

introduce randomness in neighbor selection, allowing the algorithm to explore different regions of the solution space. Random restarts enable the algorithm to escape local optima by restarting the search from different initial solutions. These techniques aim to balance exploitation with exploration, improving the algorithm's ability to find high-quality solutions.

- Adaptive Strategies: Some advanced Hill Climbing algorithms employ adaptive strategies to dynamically adjust the balance between exploration and exploitation based on the algorithm's performance and the problem characteristics. Adaptive techniques allow the algorithm to adapt its behavior over time, increasing exploration when progress is slow or exploiting promising regions when significant improvements are observed.

3 Definitions and Notations

Define a discrete optimization minimization problem as a two-tuple (Ω,c) where

- 1.Ω is a finite space composed of (Ω,c) solutions,
- 2.c:Ω→R+ is an non-negative objective function.

Define a *neighborhood function* η:Ω→2Ω, which provides connections between the elements of Ω. Define $\Delta ij{=}cj{=}ci$ to be the change in objective function value between two distinct solutions i,j∈Ω. Define G⊂ Ω to be the set of globally optimal solutions, with objective function value copt=mini∈Ω{ci}. Define \L⊂ Ω\G to be the set of locally (but not globally) optimal solutions (i.e., $i{\in}L$ if $\Delta i{,}j{\geq}0$ for all \i∈Ω\G and $j{\in}\eta(i)$). Finally, define \H=Ω\(L∪G) to be the set of all other solutions in Ω.A GHC algorithm is initialized with a solution i∈Ω having objective function value ci. The total number of outer loop iterations K, the total number of inner loop iterations M, the *solution generation probabilities* $gi{,}j(k)$, non-negative random variables $Rk(i{,}j)$, and a stopping criterion must all be specified. The GHC algorithm is depicted in pseudocode in Fig. 1. For $i \in \Omega$, $j \in \eta(i)$, and all k, the one-step transition probability $P_{i,j}(k)$ of accepting the neighboring solution as the new current solution is expressed as:

$$P_{i,j}(k) = g_{i,j}(k) \cdot \Pr(R_k(i,j) \geq \delta_{i,j}) \quad \text{for all } i \in \Omega, j \in \eta(i), j \neq i,$$

$$1 - \sum_{q \in \eta(i), q \neq i} P_{i,q}(k) \quad \text{if } j = i,$$

$$0 \quad \text{otherwise.}$$

Note that $\Pr(R_k(i,j) \geq \delta_{i,j})$ defines the *solution acceptance probability.* Moreover, the generation probabilities $g_{i,j}(k)$ must be non-negative, and satisfy:

$$\sum_{j \in \eta(i)} g_{i,j}(k) = 1.$$

The GHC algorithm can be modeled as an inhomogeneous Markov chain, or as a sequence of K homogeneous Markov chains, where each chain is of length M, and each state is a solution in Ω. When certain conditions (as described in Section 3) are placed on the transition matrix associated with each homogeneous Markov chain, then as M approaches infinity, each associated Markov chain approaches its unique equilibrium distribution $\pi(k)$. Additional conditions (also described in Section 3) on $Rk(i,j)$ ensure that as K approaches infinity, the sequence of equilibrium distributions converges to a form where all the probability mass is concentrated on the set of globally optimal solutions.

GHC algorithms traverse the solution space Ω in search of a globally optimal solution. To understand this process, the concept of a *path* between solutions must be defined.

3.0.1 Definition 2.1

A *path* from i to j, depicted as $i \to j$, for all $i, j \in L \cup G$ and all k, is a sequence of solutions $l_0, l_1, \ldots, l_d \in \Omega$ with $l_0 = i$, $l_d = j$, $l_1, l_2, \ldots, l_{d-1} \in H$, and $g_{lm,lm+1}(k) > 0$ for $m = 0, 1, \ldots, d-1$.

Note that a local or global optimum cannot be an intermediate solution on any path. However, the GHC algorithm can move from i to j via an intermediate solution $l \in L \cup G$, but the trajectory would not be defined as a path. A path can be *equivalent* to some other path, or *distinct.* These concepts are formally defined.

3.0.2 Definition 2.2

A path between solutions $i,j \in L \cup G$ is said to be *equivalent* to another path between i and j, if

- all the solutions visited along both paths are identical;
- he order in which each solution is visited along both paths is identical.

If a path between solutions $i,j \in L \cup G$ is not equivalent to any other path between i and j, then the path is said to be *distinct.* Using these definitions, the probability of transitioning via a path between solutions $i, j \in L \cup G$ can be defined. Suppose that the distinct paths from i to j are labeled $s = 1, 2, \ldots, S(i,j)$. Note that $S(i,j)$ can be infinite if one or more solutions in H can be visited infinitely often along a path. Let $P_k(i \to s_j)$ be the probability of transitioning along the s-th (distinct) path between $i, j \in L \cup G$ at iteration k. Therefore, $P_k(i \to s_j)$ is the product of all one-step transition probabilities between adjacent solutions along the s-th path; for example, for $i, j \in \Omega$, the s-th path $i, l_1, \ldots, l_{d-1}, j \in \Omega$ occurs with probability $P_k(i \to s_j) = \prod_{m=0}^{d-1} P_{lm,lm+1}(k)$. Note that path distinctness is sufficient for the probability of the union of all distinct paths between $i, j \in \Omega$ to be equal to the sum of the probabilities of the paths [**?**].

3.0.3 Definition 2.3

The *path* probability between any two solutions $i, j \in L \cup G$ is defined as:

$$P_k(i \to s_j) \equiv \begin{cases} \sum_{s=1}^{S(i,j)} P_k(i \to j), & \text{if } i \neq j, \\ 1 - \sum_{t \in (L \cup G) \setminus \{i\}} P_k(i \to j), & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Note that $P_k(i \to j)$ can also be viewed as the transition probability from i to j at iteration k. The example depicted in Fig. 2 illustrates how the path probabilities are defined, where $i, l \in L$, $j \in G$, and $p, q, r, s \in H$. The neighborhood structure (indicating all positive one-step transition probabilities) is indicated by the lines connecting the nodes. Therefore, $P_k(i \to j) > 0$, since i and j are separated only by q. Similarly, $P_k(l \to j) > 0$, since l and j are separated only by r. However, the only way to reach solution l from solution i is to pass through the global optimum j, and so $P_k(i \to l) = 0$ from the "otherwise" case of (3).

Recall that the GHC algorithm is composed of an outer loop, indexed on k, and an inner loop, indexed on m. Furthermore, when the sufficient conditions of Theorem 3.1 (see Section 3) are satisfied, then $\pi(k)$ is the equilibrium (long-run) probability vector for all solutions i∈Ω, for each k, as M approaches infinity. Definition 2.4 introduces $\delta(k)$ as the vector of equilibrium probabilities $\delta i(k)$ for all solutions $i{\in}L{\cup}G$. Each probability $\delta i(k)$ is obtained by scaling the corresponding equilibrium probability $\pi i(k)$ over the total equilibrium probability $\Omega(k)$ of all solutions in $L{\cup}G$.

3.0.4 Definition 2.4

Let Ω(k)≡Σi∈L∪Gπi(k). Define the equilibrium probability δi(k)≡πi(k)/Ω(k)for all i∈L∪G and all k. Note that $\delta i(k){\geq}\pi i(k)$ for all k, since $\Omega(k){\leq}1$.

Definition 2.3, Definition 2.4 allow the primary convergence proof (Theorem 3.2) to focus only on the sets of local and global optima, which will be shown to be the only solutions of significance for a GHC algorithm. This is done purely for theoretical development. In practice, a GHC algorithm can visit any element in Ω, including elements in H.

4 Proofs of Convergence

Theorem 3.1 provides the sufficient conditions for a unique equilibrium distribution $\pi(k)$ to exist for each iteration k. Corollary 3.1 shows that the GHC algorithm converges to the set of solutions $L \cup G$ as k approaches infinity. Theorem 3.2 provides the additional sufficient conditions for the GHC algorithm to converge to the set of globally optimal solutions G, as k approaches infinity.

Theorem 3.1

Let (Ω,c) denote an instance of a discrete optimization problem with neighborhood function η. Let the GHC transition probabilities $P_{i,j}(k)$ be defined by (1). Assume that the generation probabilities $g_{i,j}(k)$ satisfy

1. For all $i, j \in \Omega$ and all iterations k, there exists an integer $d \geq 1$ and a corresponding sequence of solutions $l_0, l_1, l_2, \ldots, l_d \in \Omega$, with $l_0 = i$, $l_d = j$, and $g_{lm,lm+1}(k) > 0$, $m = 0, 1, \ldots, d-1$,
2. For all $i, j \in \Omega$, $j \in \eta(i)$, $\lim_{k\to\infty} g_{i,j}(k)$ exists and is strictly positive. Moreover, assume that the acceptance probabilities satisfy
3. $\Pr(R_k(i,j) \geq \Delta_{i,j}) > 0$ for all $i \in \Omega$, $j \in \eta(i)$, and all iterations k.
4. $c_i < c_j \Rightarrow \lim_{k\to\infty} \Pr(R_k(i,j) \geq \Delta_{i,j}) = 0$.

Then $\pi(k)$ exists for each k. Proof: See Appendix 1.

Corollary 3.1 show that the equilibrium probability of all non-optimal solutions (i.e., solutions that are neither local nor global optima) approach zero as k approaches infinity.

Corollary 3.1

If $\lim_{k\to\infty} \pi(k)$ *exists, then under the conditions and assumptions of Theorem* 3.1,

$$\lim_{k\to\infty} \pi_i(k) = 0 \quad \text{for all } i \in H.$$

Proof: See Appendix 2.

Corollary 3.2 shows that for all local and global optima $i \in L \cup G$, the probabilities $\Delta_i(k)$ approach the equilibrium probabilities $\pi_i(k)$ in value, as k approaches infinity.

Corollary 3.2

If $\lim_{k\to\infty} \pi(k)$ *exists, then under the conditions and assumptions of Theorem* 3.1, *for all* $i \in L \cup G$,

$$\lim_{k\to\infty} \Delta_i(k) = \lim_{k\to\infty} \pi_i(k).$$

Proof: See Appendix 3.

The following definitions of minimal and maximal path probabilities are used to obtain the sufficient conditions presented in Theorem 3.2.

Definition 3.1

The minimum positive path probability between any local (but not global) optimum and any (local or global) optimum at iteration k is $P_k(\text{Min\_Path}) \equiv \min\{P_k(j \to i) \mid j \in L, i \in L \cup G \text{ and } P_k(j \to i) > 0\}$.

Definition 3.2

The maximum path probability between any global optimum and any local (but not global) optimum at iteration k is $P_k(\text{Max\_Path}) \equiv \max\{P_k(i \to j) \mid i \in G, j \in L\}$.

Definition 3.3

The maximal product of locally (but not globally) optimal solution equilibrium distribution probabilities and their associated path probabilities to other local (but not global) optima at iteration k is $P_k(\text{Max\_Prod}) \equiv \max\{\Delta_j(k) P_k(j \to q) \mid j, q \in L, q \neq j\}$.

Theorem 3.2 provides sufficient conditions for the equilibrium probability of all local (but not global) optima to approach zero, as k approaches infinity.

Theorem 3.2

Under the conditions and assumptions of Theorem 3.1 *and Corollary* 3.2, *if*

1. $\sum_{k=1}^{\infty} P_k(\text{Min\_Path}) = +\infty$,
2. $\sum_{k=1}^{\infty} P_k(\text{Max\_Path}) < +\infty$,
3. $\sum_{k=1}^{\infty} P_k(\text{Max\_Prod}) < +\infty$,

then

$$\lim_{k\to\infty} \Delta_j(k) = 0 \quad \text{for all } j \in L.$$

Proof

See Appendix 4.

The assumption that $\lim_{k\to\infty} \pi(k)$ exists is not a severe restriction, and is satisfied by any GHC algorithm (that also meets conditions (a)–(d)) whose hill climbing probabilities become monotone nonincreasing as k approaches infinity. This asymptotic monotone behavior is satisfied by the hill climbing acceptance functions typically used in practice. This assumption does preclude, for example, pathological acceptance function formulations that could lead to oscillating $\pi(k)$ vector values for k even versus k odd. Moreover, in practice, it is difficult to find GHC formulations that satisfy both (e) and (f) unless $P_k(\text{Max\_Path}) < P_k(\text{Min\_Path})$. Examples 4.1 and 4.3 illustrate two different approaches of formulating algorithms that satisfy this inequality.

Note that Anily and Federgruen [3] provide the most general sufficient conditions in the literature. However, while condition (g) requires that the equilibrium probability distribution $\pi(k)$ be explicitly known only for the set of local and global optima, Anily and Federgruen's [3] convergence theorem requires that the equilibrium distribution $\pi(k)$ be known for *all* solutions in Ω. Hence Theorem 3.2 is a relaxation of the Anily and Federgruen [3] result, in that Theorem 3.2 requires equilibrium distribution information for only a (presumably small) subset of the solution space. Note also that other convergence theorems (e.g. [9]) use either the exponential acceptance function formulation or reversible Markov chain theory to ensure that the solution probability distributions (e.g., each $\pi(k)$) are explicitly known. Therefore, Theorem 3.2 allows the use of acceptance functions that previously were unable to be proven convergent. Section 4 provides several examples.

Corollary 3.3 shows that all probability mass is concentrated on the set of optimal solutions G, as k approaches infinity.

Corollary 3.3

Under the assumptions of Theorem 3.2, $\lim_{k\to\infty} \left(\sum_{i\in G} \pi_i(k)\right) = 1$.

Proof: See Appendix 5.

5 Simulated Annealing

Simulated annealing is a method for solving unconstrained and bound-constrained optimization problems. The method models the physical process of heating a material and then slowly lowering the

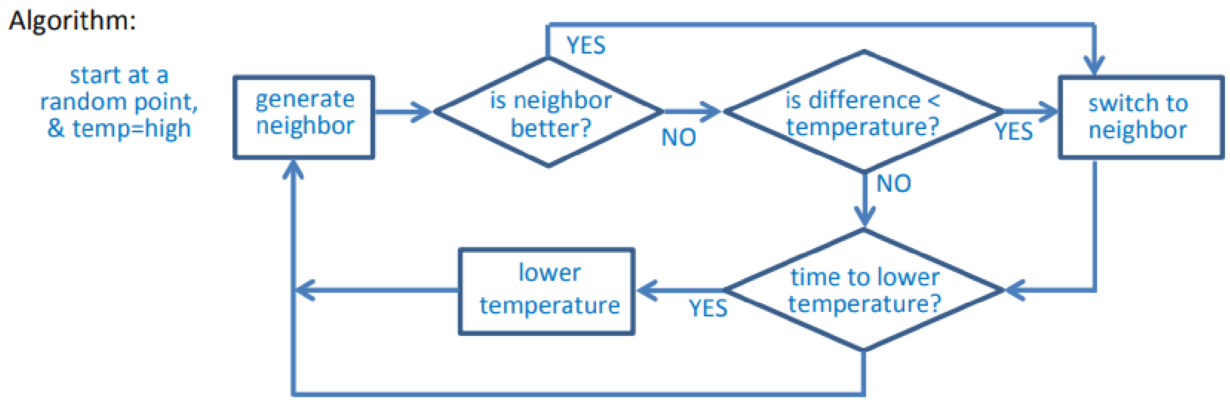

Figure 2: SA Algorithm

temperature to decrease defects, thus minimizing the system energy.

At each iteration of the simulated annealing algorithm, a new point is randomly generated. The distance of the new point from the current point, or the extent of the search, is based on a probability distribution with a scale proportional to the temperature. The algorithm accepts all new points that lower the objective, but also, with a certain probability, points that raise the objective. By accepting points that raise the objective, the algorithm avoids being trapped in local minima, and is able to explore globally for more possible solutions. An *annealing schedule* is selected to systematically decrease the temperature as the algorithm proceeds. As the temperature decreases, the algorithm reduces the extent of its search to converge to a minimum.

- switching to an inferior neighbor accepted with decreasing probability over time.
- temperature usually modeled as a value that starts at 1 ("high") and gradually reduces to 0.
- temperature decrease can be computed as a percentage, such as TEMP' = 0.99 (TEMP).
- Boltzmann distribution is commonly used: P(ΔE,T) = 1/(1+exp(E/CT)) E=energy, T=temp, C=constant.

SA has been successfully applied to a wide range of optimization problems, such as TSP, protein folding, graph partitioning, and job-shop scheduling. **The main advantage of SA is its ability to escape from local minima and converge to a global minimum**. SA is also relatively easy to implement and does not require a priori- knowledge of the search space.

5.1 Algorithm

5.1.1 Defining the problem

First, we need to define the problem to optimize. **This involves defining the energy function, i.e., the function to minimize or maximize**. For example, if we want to minimize a real-valued function of two variables, e.g., , the energy corresponds to the function itself. In the case of the TSP, the energy related to a sequence of cities is represented by the total length of the travel.

Once the energy function is defined, we need to set the initial temperature value and the initial candidate solution. The latter can be generated randomly or using some other heuristic method. Then we compute the energy of the initial candidate solution.

5.1.2 Define the Perturbation Function

A perturbation function is defined to generate new candidate solutions. This function should generate solutions that are close to the current solution but not too similar. For example, if we want to minimize a function , we can randomly perturb the current solution by adding a random value between -0.1 and 0.1 to both and . In the case of the TSP, a new candidate solution can be generated by swapping two cities in the traveling order of the current solution.

5.1.3 Acceptance Criterion

The acceptance criterion determines whether a new solution is accepted or rejected. The acceptance depends on the energy difference between the new solution and the current solution, as well as the current temperature. **The classic acceptance criterion of SA comes from statistical mechanics, and it is based on the** Boltzmann probability distribution. A system in thermal equilibrium at temperature can be found in a state with energy with a probability proportional to exp(-E/kT) where k is the Boltzmann constant. Hence, at low temperatures, there is a small chance that the system is in a high-energy state. This plays a crucial role in SA because an increase in energy allows escape from local minima and find the global minimum.

Based on the Boltzmann distribution, the following algorithm defines the criterion for accepting an energy variation at temperature . A candidate solution with lower energy is always accepted. Conversely, a candidate solution with higher energy is accepted randomly with probability (for our purpose, we can set). The latter case can be implemented by comparing the probability with a random value generated in the range .

5.1.4 Temperature Schedule

The temperature schedule determines how the temperature of the system changes over time. In the beginning, the temperature is high so that the algorithm can explore a wide range of solutions, even if they are worse than the current solution. As the iterations increase, the temperature gradually decreases, so the algorithm becomes more selective and accepts better solutions with higher probability. **A simple scheduling can be obtained by dividing the current temperature by a factor , which is less than 1.**

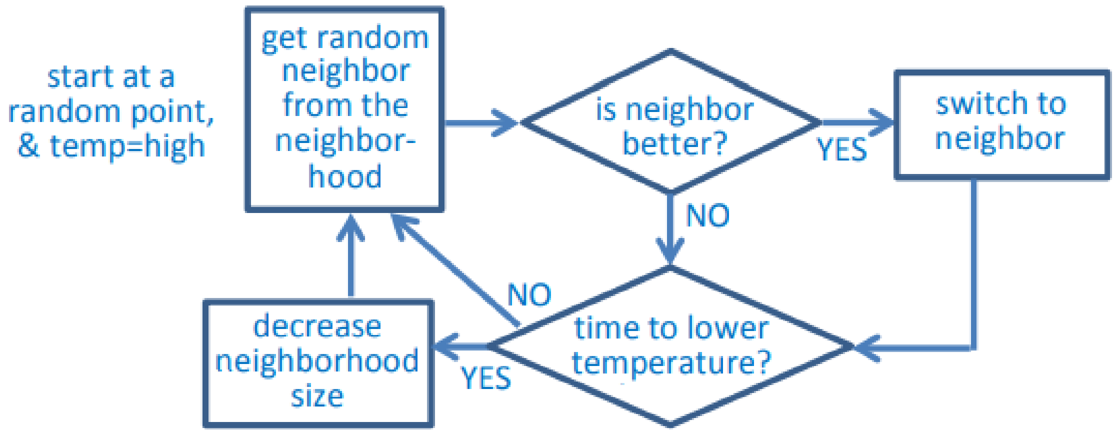
start at a random point, & temp=high
get random neighbor from the neighbor-hood
is neighbor better?
YES
switch to neighbor
NO
time to lower temperature?
NO
YES
decrease neighborhood size

Figure 3: Neighbour Annealing

5.1.5 Run the SA Algorithm

Finally, run the algorithm by iteratively applying the perturbation function and acceptance criterion to the current solution. The algorithm terminates when the temperature has cooled to a certain level or when the energy of the current solution is lower than a fixed threshold .

5.2 Neighbour Annealing

- Requires defining a "neighborhood" – neighbors are selected randomly from neighborhood.
- When temperature is high, neighborhood size is large (neighbors can be far away).
- As temperature decreases, neighborhood size gets smaller (neighbors are only close by).
- Switch to the neighbor ONLY if it is better.

6 Applications of Hill Climbing

Hill Climbing algorithms find application in various fields due to their simplicity, efficiency, and effectiveness in optimizing certain types of problems. Here are some common applications of Hill Climbing algorithms:

1. Function Optimization: Hill Climbing algorithms are widely used for optimizing mathematical functions, particularly in single-objective optimization problems. They are employed in fields such as engineering, physics, economics, and operations research to find optimal values of parameters or variables that maximize or minimize an objective function.

2. Machine Learning: Hill Climbing algorithms are utilized in certain machine learning tasks, such as feature selection and model parameter tuning. In feature selection, Hill Climbing can iteratively select subsets of features that improve the performance of a machine learning model. In parameter tuning, Hill Climbing can optimize hyperparameters of models to improve their performance on validation data.

3. Data Clustering: Hill Climbing algorithms can be employed in data clustering tasks to partition data into groups based on similarity measures. They can iteratively adjust cluster centroids or boundaries to optimize clustering criteria such as within-cluster variance or silhouette score.

4. Game Playing: Hill Climbing algorithms are employed in game playing scenarios to optimize strategies or decision-making processes. They can be used to iteratively improve player strategies in games such as chess, checkers, and board games by adjusting move sequences or evaluating alternative moves.

5. Natural Language Processing: In certain natural language processing tasks, such as language generation and text summarization, Hill Climbing algorithms can be used to optimize language models or generation processes to produce more coherent and semantically meaningful output.

7 Implications for future Research

Future research in Hill Climbing algorithms could focus on several avenues to address existing limitations, enhance performance, and expand their applicability to a wider range of optimization problems. Here are some potential directions for future research:

1. Hybridization with other Metaheuristic Techniques: Investigate the integration of Hill Climbing algorithms with other metaheuristic techniques such as genetic algorithms, simulated annealing, ant colony optimization, or particle swarm optimization. Hybrid approaches could combine the strengths of different algorithms to improve solution quality, convergence speed, and robustness to problem characteristics.

2. Dynamic Adaptation: Develop adaptive Hill Climbing algorithms that dynamically adjust their behavior and parameters based on the problem landscape, search progress, or other environmental factors. Adaptive techniques could enhance exploration and exploitation strategies, leading to improved performance across a wider range of optimization problems.

3. Multi-objective Optimization: Extend Hill Climbing algorithms to handle multi-objective optimization problems, where the goal is to optimize multiple conflicting objectives simultaneously. Future research could explore techniques for balancing trade-offs between different objectives and identifying Pareto-optimal solutions using Hill Climbing approaches.

4. Large-scale and Continuous Optimization: Investigate strategies for applying Hill Climbing algorithms to large-scale and continuous optimization problems, where the solution space is high-dimensional and continuous. Research efforts could focus on efficient neighbor generation techniques, adaptive step size adjustment, and scalable optimization strategies.

5. Parallel and Distributed Implementations: Explore parallel and distributed implementations of Hill Climbing algorithms to leverage the computational power of modern parallel and distributed computing architectures. Parallelization techniques could accelerate convergence and enable the optimization of large-scale problems by distributing computation across multiple processing units.

6. Dynamic Neighborhood Structures: Investigate dynamic neighborhood structures that adaptively adjust the set of neighboring solutions considered at each iteration based on problem characteristics, search progress, or other criteria. Dynamic neighborhood structures could enhance exploration and exploitation capabilities while reducing computational overhead.

7. Incorporation of Machine Learning Techniques: Explore the integration of machine learning techniques, such as reinforcement learning or surrogate modeling, with Hill Climbing algorithms to improve solution quality, accelerate convergence, and handle complex optimization problems with noisy or uncertain objective functions.

8. Applications in Real-world Domains: Apply Hill Climbing algorithms to address optimization challenges in real-world domains such as healthcare, transportation, finance, energy, and manufacturing. Research efforts could focus on developing domain-specific optimization strategies and validating the effectiveness of Hill Climbing approaches in practical applications.

8 Conclusion

References