In The Name Of God

University of Tehran

Faculty of Engineering

Department of Surveying and Geomatics Engineering

Geographical Information System (GIS) division

Global Optimization Algorithms
– Theory and Application –

Instructor : Dr.Samadzadegan

Name : Mehran Ghandehary

Student Number : 810389010

Email : m.ghandhary@yahoo.com

Date Delivered :12.11.2010

SUMMARY

Optimization algorithms are search methods, where the goal is to find a solution to an optimization problem, such that a given quantity is optimized, possibly subject to a set of constraints. Although this definition is simple, it hides a number of complex issues. For example, the solution may consist of a combination of different data types, nonlinear constraints may restrict the search area, the search space can be convoluted with many candidate solutions, the characteristics of the problem may change over time, or the quantity being optimized may have conflicting objectives.

CONTENTS

Introduction

One of the most fundamental principles in our world is the search for an optimal state.
It begins in the microcosm where atoms in physics try to form bonds[1] in order to minimize the energy of their electrons .When molecules form solid bodies during the process of freezing, they try to assume energy-optimal crystal structures. These processes, of course, are not driven by any higher intention but purely result from the laws of physics.The same goes for the biological principle of survival of the fittest which, together with the biological evolution ,leads to better adaptation of the species to their environment .
Here, a local optimum is a well-adapted species that dominates all other animals in its surroundings. Homo sapiens have reached this level, sharing it with ants, bacteria, flies, cockroaches, and all sorts of other creepy creatures. As long as humankind exists, we strive for perfection in many areas. We want to reach a maximum degree of happiness with the least amount of effort. In our economy, profit and sales must be maximized and costs should be as low as possible. Therefore, optimization is one of the oldest of sciences which even extends into daily life .
If something is important, general, and abstract enough, there is always a mathematical discipline dealing with it. Global optimization[2] is the branch of applied mathematics and numerical analysis that focuses on, well, optimization. The goal of global optimization is to find the best possible elements $x^\star$ from a set X according to a set of criteria F = {f1, f2, .., fn}.These criteria are expressed as mathematical functions[3], the so-called objective functions.

1.1 A Classification of Optimization Algorithms

1.1.1 Classification According to Method of Operation

Figure 1 sketches a rough taxonomy of global optimization methods. Generally, optimization algorithms can be divided in two basic classes: deterministic and probabilistic algorithms. Deterministic algorithms are most often used if a clear relation between the characteristics of the possible solutions and their utility for a given problem exists. Then, the search space can efficiently be explored using for example a divide and conquer scheme. If the relation between a solution candidate and its “fitness” are not so obvious or too complicated, or the dimensionality of the search space is very high, it becomes harder to solve a problem deterministically. Trying it would possible result in exhaustive enumeration of the search space, which is not feasible even for relatively small problems.

Then, probabilistic algorithms come into play.They trade in guaranteed correctness of the solution for a shorter runtime. This does not mean that the results obtained using them are incorrect they may just not be the global optima. On the other hand, a solution a little bit inferior to the best possible one is better than one which needs 10100 years to be found. . .

Heuristics used in global optimization are functions that help decide which one of a set of possible solutions is to be examined next. On one hand, deterministic algorithms usually employ heuristics in order to define the processing order of the solution candidates.

Probabilistic methods, on the other hand, may only consider those elements of the search space in further computations that have been selected by the heuristic.

Definition (Heuristic). A heuristic is a part of an optimization algorithm that uses the information currently gathered by the algorithm to help to decide which solution candidate should be tested next or how the next individual can be produced. Heuristics are usually problem class dependent.

Definition (Metaheuristic). A metaheuristic is a heuristic method for solving a very general class of problems. It combines (problem dependent) objective functions or heuristics in an abstract and hopefully efficient way.

This combination is often performed statistically by using population as sample from the search space or based on a model of some natural phenomenon or physical process. Simulated annealing for example decides which solution candidate to be evaluated according to the Boltzmann probability factor of atom configurations of solidifying metal melts. Evolutionary algorithms copy the behavior of natural evolution and treat solution candidates as individuals that compete in a virtual environment.

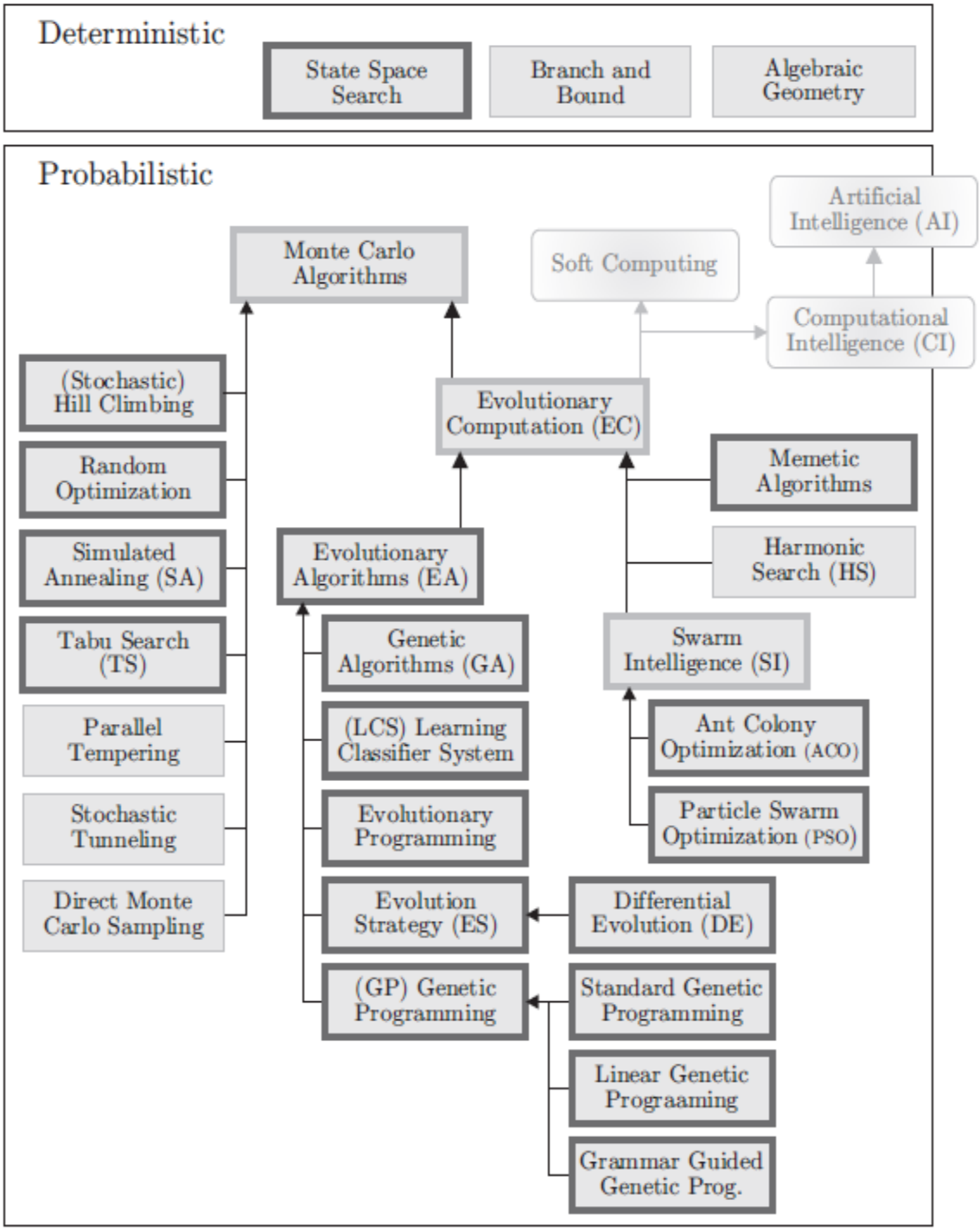

Figure 1 : The taxonomy of global optimization algorithms.

1.1.2 Classification According to Properties

The taxonomy just introduced classifies the optimization methods according to their algorithmic structure and underlying principles, in other words, from the viewpoint of theory. A software engineer or a user who wants to solve a problem with such an approach is however more interested in its “interfacing features” such as speed and precision.
Speed and precision are conflicting objectives, at least in terms of probabilistic algorithms. A general rule of thumb is that you can gain improvements in accuracy of optimization only by investing more time.

Optimization Speed

When it comes to time constraints and hence, the required speed of the optimization algorithm, we can distinguish two main types of optimization use cases.

Definition (Online Optimization). Online optimization problems are tasks that need to be solved quickly in a time span between ten milliseconds to a few minutes. In order to find a solution in this short time, optimality is normally traded in for speed gains.
Optimization Speed When it comes to time constraints and hence, the required speed of the optimization algorithm, we can distinguish two main types of optimization use cases.

Definition (Online Optimization). Online optimization problems are tasks that need to be solved quickly in a time span between ten milliseconds to a few minutes. In order to find a solution in this short time, optimality is normally traded in for speed gains.

1.2 What is an optimum?

We have already said that global optimization is about finding the best possible solutions for given problems. Thus, it cannot be a bad idea to start out by discussing what it is that makes a solution optimal .

1.2.1 Basic Ingredients of Optimization Problems

Each optimization problem consists of the following basic ingredients:
• An **objective function**, which represents the quantity to be optimized, that is, the quantity to be minimized or maximized. Let f denote the objective function. Then a maximum of f is a minimum of $-f$. Some problems, specifically constraint-satisfaction problems (CSP), do not define an explicit objective function. Instead, the objective is to find a solution that satisfies all of a set of constraints.
• A **set of unknowns or variables**, which affects the value of the objective function. If $\mathbf{x}$ represents the unknowns, also referred to as the independent variables, then $f(\mathbf{x})$ quantifies the quality of the candidate solution, $\mathbf{x}$.

• A **set of constraints**, which restricts the values that can be assigned to the unknowns. Most problems define at least a set of boundary constraints, which define the domain of values for each variable. Constraints can, however, be more complex, excluding certain candidate solutions from being considered as solutions.
The goal of an optimization method is then to assign values, from the allowed domain,to the unknowns such that the objective function is optimized and all constraints are satisfied. To achieve this goal, the optimization algorithm searches for a solution in a search space, *S*, of candidate solutions. In the case of constrained problems, a solution is found in the feasible space, $F \subseteq S$.

1.2.2 Single Objective Functions

In the case of optimizing a single criterion f, an optimum is either its maximum or minimum, depending on what we are looking for. If we own a manufacturing plant and have to assign incoming orders to machines, we will do this in a way that miniminzes the time needed to complete them. On the other hand, we will arrange the purchase of raw material, the employment of staff, and the placing of commercials in a way that maximizes our profit. In global optimization, it is a convention that optimization problems are most often defined as minimizations and if a criterion f is subject to maximization, we simply minimize its negation (−f).
Figure 2 illustrates such a function f defined over a two-dimensional space X = (X1,X2). As outlined in this graphic, we distinguish between local and global optima. A global optimum is an optimum of the whole domain X while a local optimum is an optimum of only a subset of X.

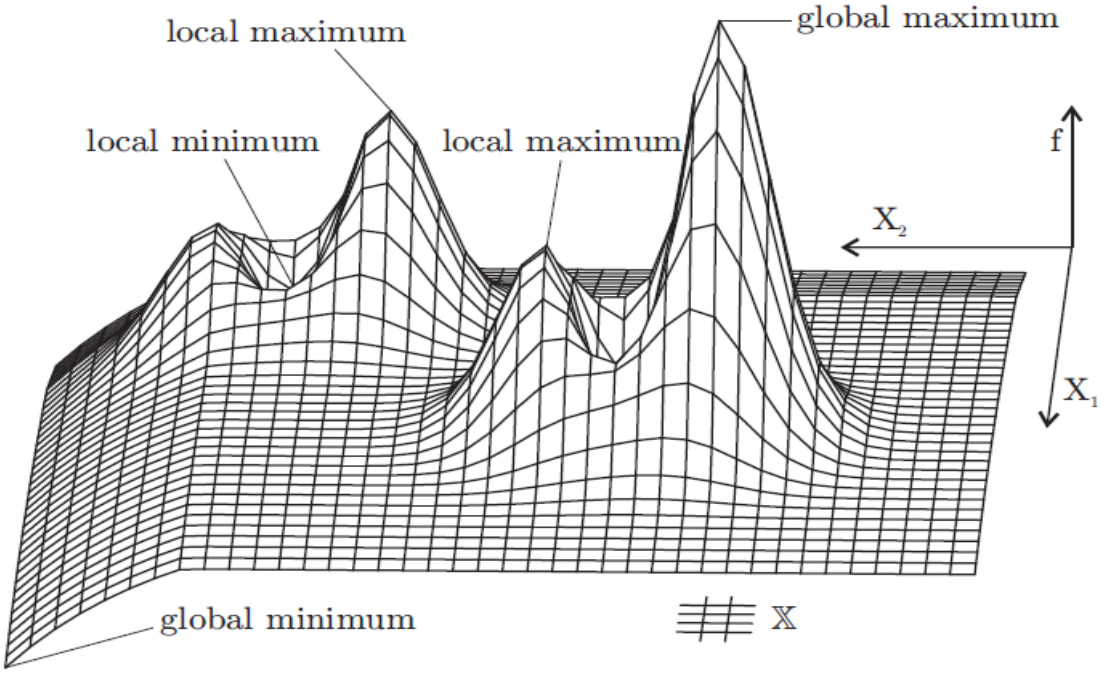

Figure 2 : Global and local optima of a two-dimensional function.

1.2.3 Multiple Objective Functions

Global optimization techniques are not just used for finding the maxima or minima of single functions f. In many real-world design or decision making problems, they are rather applied to sets F consisting of n = |F| objective functions fi, each representing one criterion to be optimized .
Algorithms designed to optimize such sets of objective functions are usually named with the prefix multi-objective .

1.3 Optimization Method Classes

An optimization algorithm searches for an optimum solution by iteratively transforming a current candidate solution into a new, hopefully better, solution. Optimization methods can be divided into two main classes, based on the type of solution that is located. Local search algorithms use only local information of the search space surrounding the current solution to produce a new solution. Since only local information is used, local search algorithms locate local optima (which may be a global minimum).
A global search algorithm uses more information about the search space to locate a global optimum. It is said that global search algorithms explore the entire search space, while local search algorithms exploit neighborhoods. Optimization algorithms are further classified into deterministic and stochastic methods. Stochastic methodsuse random elements to transform one candidate solution into a new solution. The new point can ntherefore not be predicted. Deterministic methods, on the other hand, do not make use of random elements.
Based on the problem characteristics, optimization methods are grouped in the following classes (within each of these classes further subdivision occurs based on whether local or global optima are located, and based on whether random elements are used to investigate new points in the search space):
• **unconstrained methods**, used to optimize unconstrained problems;
• **constrained methods**, used to find solutions in constrained search spaces;
• **multi-objective optimization methods** for problems with more than one objective to optimize;
• **multi-solution (niching) methods** with the ability to locate more than one solution; and
• **dynamic methods** with the ability to locate and track changing optima.

1.4 Optimization Algorithms

Many optimization algorithms have been developed to solve unconstrained problems. This section summarizes only a few **Algorithms .**

1.4.1 General Local Search Procedure

In Local search methods A starting point, **x**(0), is selected, and its quality evaluated. Then, iteratively a search direction is determined and a move is made in that direction.

Algorithm General Local Search Algorithm
Find starting point **x**(0) $\in S$;
$t = 0$;
repeat
Evaluate $f(\mathbf{x}(t))$;
Calculate a search direction, **q**(t);
Calculate step length $\eta(t)$;
Set **x**(t + 1) to **x**(t) + $\eta(t)$**q**(t);
$t = t + 1$;
until *stopping condition is true*;
Return **x**(t) as the solution;

Search directions and step lengths can be determined using steepest gradient descent, conjugate gradients, or Newton methods (amongst many others).

1.4.2 Tabu Search

Tabu search (TS) is an iterative neighborhood search algorithm, where the neighborhood changes dynamically. TS enhances local search by actively avoiding points in the search space already visited. By avoiding already visited points, loops in search trajectories are avoided and local optima can be escaped. The main feature of TS is the use of an explicit memory. A simple TS usually implements two forms of memory:

• A **frequency-based memory**, which maintains information about how often a search point has been visited (or how often a move has been made) during a specified time interval.

• A **recency-based memory**, which maintains information about how recently a search point has been visited (or how recently a move has been made). Recency is based on the iteration at which the event occurred.

If, for example, the frequency count of a search point exceeds a given threshold, then that point is classified as being tabu for the next cycle of iterations. Positions specified in the tabu list are excluded from the neighborhood of candidate positions that can be visited from the current position. Positions remain in the tabu list for a specified

time period.
The following may be used to terminate TS:
• the neighborhood is empty, i.e. all possible neighboring points have already been visited, or
• when the number of iterations since the last improvement is larger than a specified threshold.

1.4.3 Simulated Annealing

Annealing refers to the cooling process of a liquid or solid, and the analysis of the behavior of substances as they cool . As temperature reduces, the mobility of molecules reduces, with the tendency that molecules may align themselves in a crystalline structure. The aligned structure is the minimum energy state for the system. To ensure that this alignment is obtained, cooling must occur at a sufficiently slow rate. If the substance is cooled at a too rapid rate, an amorphous state may be reached. Simulated annealing is an optimization process based on the physical process described above. In the context of mathematical optimization, the minimum of an objective function represents the minimum energy of the system. Simulated annealing is an algorithmic implementation of the cooling process to find the optimum of an objective function .
Simulated annealing (SA) uses a random search strategy, which not only accepts new positions that decrease the objective function (assuming a minimization problem), but also accepts positions that increase objective function values. The latter is accepted probabilistically based on the Boltzmann–Gibbs distribution. If *Pij* is the probability of moving from point **x***i* to **x***j*, then *Pij* is calculated using

$$P_{ij} = \begin{cases} 1 & \text{if } f(\mathbf{x}_j) < f(\mathbf{x}_i) \\ e^{-\frac{f(\mathbf{x}_j) - f(\mathbf{x}_i)}{c_b T}} & \text{otherwise} \end{cases}$$

where $c_b > 0$ is the Boltzmann constant and T is the temperature of the system.

The algorithm requires specification of the following components:
• A **representation of possible solutions**, which is usually a vector of floatingpoint values.
• A **mechanism to generate new solutions** by adding small random changes to current solutions. For example, for continuous-valued vectors,
$\mathbf{x}(t + 1) = \mathbf{x}(t) + \mathbf{D}(t)\mathbf{r}(t)$
where $\mathbf{r}(t) \sim U(-1, 1)^{nx}$, and **D** is a diagonal matrix that defines the maximum change allowed in each variable. When an improved solution is found,
$\mathbf{D}(t + 1) = (1 - \alpha)\mathbf{D}(t) + \alpha\omega\mathbf{R}(t)$
where **R**(*t*) is a diagonal matrix whose elements are the magnitudes of the successful changes made to each variable, and α and ω are constants. For integer problems,
$\mathbf{x}(t + 1) = \mathbf{x}(t) + \mathbf{r}(t)$

where each element of **r**(t) is randomly selected from the set $\{-1, 0, 1\}$.
• A **method to evaluate solutions**, which is usually just the objective function in the case of unconstrained problems.
• An **annealing schedule**, which consists of an initial temperature and rules for lowering the temperature with increase in number of iterations. The annealing schedule determines the degree of uphill movement (objective function increase) allowed during the search. An initial high temperature is selected, which is then incrementally reduced using, for example,
– Exponential cooling: $T(t + 1) = \alpha T(t)$, where $\alpha \in (0, 1)$.
– Linear cooling: $T(t+1) = T(t) - \Delta T$, where, e.g. $\Delta T = (T(0) - T(nt))/nt$;
$T(0)$ is the initial large temperature, and $T(nt)$ is the final temperature at the last iteration, nt.

Algorithm : Simulated Annealing Algorithm

Create initial solution, $\mathbf{x}(0)$;
Set initial temperature, $T(0)$;
$t = 0$;
repeat
  Generate new solution, $\mathbf{x}$;
  Determine quality, $f(\mathbf{x})$;
  Calculate acceptance probability using equation (A.5);
  if $U(0, 1) \leq$ *acceptance probability* **then**
    $\mathbf{x}(t) = \mathbf{x}$;
  end
until *stopping condition is true*;
Return $\mathbf{x}(t)$ as the solution;

1.4.4 Hill Climbing

hill climbing is a mathematical optimization technique which belongs to the family of local search. It is relatively simple to implement, making it a popular first choice. Although more advanced algorithms, e.g., Simulated annealing or tabu search, may give better results, in some situations hill climbing works just as well. Especially for some real-time systems, hill climbing can get a relatively better solution in a limited time.

Hill climbing can be used to solve problems that have many solutions, we call it a search space. In the search space, different solutions usually have different values. Hill climbing starts with a random (potentially poor) solution, and iteratively makes small changes to the solution to generate a neighborhood solution. If the neighborhood solution is better than the current solution, then we use the neighborhood solution to substitute the current solution. When the current solution can no longer be improved, it terminates. Ideally, at that point the current solution is close to optimal, but it is not

guaranteed that hill climbing will ever come close to the optimal solution. However, it may obtain the local optimal solution.

For example, hill climbing can be applied to the traveling salesman problem. It is easy to find a solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much better route is obtained.

Hill climbing is used widely in artificial intelligence, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms.

Pseudocode

```
---------------------------------------------------------------------------------------------------
Discrete Space Hill Climbing Algorithm
  currentNode = startNode;
  loop do
    L = NEIGHBORS(currentNode);
    nextEval = -INF;
    nextNode = NULL;
    for all x in L
      if (EVAL(x) > nextEval)
          nextNode = x;
          nextEval = EVAL(x);
    if nextEval <= EVAL(currentNode)
      //Return current node since no better neighbors exist
      return currentNode;
    currentNode = nextNode;
---------------------------------------------------------------------------------------------------
Continuous Space Hill Climbing Algorithm
  currentPoint = initialPoint;    // the zero-magnitude vector is common
  stepSize = initialStepSizes;    // a vector of all 1's is common
  acceleration = someAcceleration // a value such as 1.2 is common
  candidate[0] = -acceleration;
  candidate[1] = -1 / acceleration;
  candidate[2] = 0;
  candidate[3] = 1 / acceleration;
  candidate[4] = acceleration;
  loop do
    before = EVAL(currentPoint);
    for each element i in currentPoint do
      best = -1;
      bestScore = -INF;
      for j from 0 to 4        // try each of 5 candidate locations
        currentPoint[i] = currentPoint[i] + stepSize[i] * candidate[j];
```

```
        temp = EVAL(currentPoint);
        currentPoint[i] = currentPoint[i] - stepSize[i] * candidate[j];
        if(temp > bestScore)
            bestScore = temp;
            best = j;
      if candidate[best] is not 0
        currentPoint[i] = currentPoint[i] + stepSize[i] * candidate[best];
        stepSize[i] = stepSize[i] * candidate[best]; // accelerate
    if (EVAL(currentPoint) - before) < epsilon
      return currentPoint;
```
