Introduction to Optimization Using Metaheuristics

Thomas J. K. Stidsen

Outline

- General course information
- Motivation, modelling and solving
- Hill climbers
- Simulated Annealing

Large-Scale Optimization

Wellcome to the course ! Course homepage:

http://www.imm.dtu.dk/courses/02719/

Take a look at the course plan.

The Lecturer: Thomas Stidsen

- Name: Thomas Stidsen: tks@imm.dtu.dk
- Nationality: Danish.
- Languages: Danish and English.
- Education: Ph.D. OR/mathematics and Msc. in Computer Science
- Teaching: Teaching in Linear Programming (42112) and Large Scale Optimization using Decomposition (42132), Optimization using Meta-heuristics (42133) and Advanced OR (42134).
- Research: Optimization of telecommunication networks

Examination

Grades will be based on the project assignment:

- 1 large 8 week project, starting 8/3, pre-hand in 22/3 and final hand in 29/4, presentation 10/5 (in english). Oral examination 20/5.

The different Metaheuristics

In the book you will find 19 different chapters, each on a specific metaheuristic or topic. This is too much for this course and not all of it is relevant, but they are:

1. Introduction
2. Classical Techniques
3. Integer Programming
4. Genetic Algorithms*
5. Genetic Programming
6. Tabu Search*

Lectures

Lectures will be given once each week:

- Monday 13.00-17.00, in databar 43 building 303N

Litterature

The course litterature is:

- „Search Metholodogies“, Edmund K. Burke & Graham Kendall, ISBN0-387-23460-8.

7. Simulated Annealing*
8. Variable Neighborhood Search
9. Constraint Programming
10. Multi-Objective Optimization
11. Complexity Theory and the No Free Lunch Theorem*
12. Machine Learning
13. Artificial Immune Systems
14. Swarm Intelligence*
15. Fuzzy Reasoning
16. Rough Set Based Decision Support
17. Hyper Heuristics
18. Approximation Algorithms
19. Fitness Landscapes

The different Metaheuristics II

- You may be confused by the different methods, but there are **many** more out there !
- I only present a selection of the metods which I consider essential.
- **There is no such thing as a best metaheuristic !!!**. This has actually been proven mathematically (chapter 11) ! Select the best metaheuristic for the problem at hand.

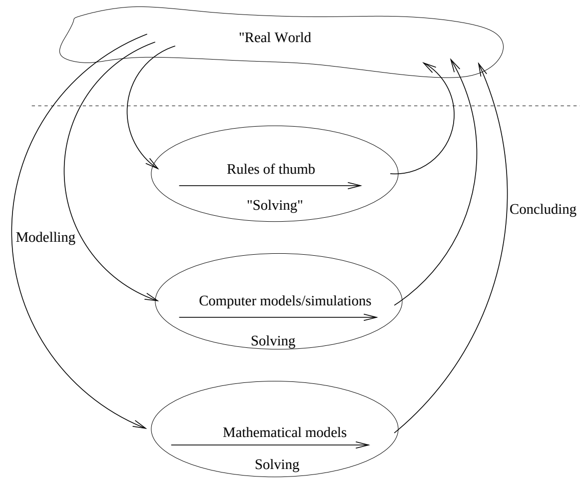
Determined by counting hits on Google.

	2003	2004	2005	2006	2008	2010
SA	81100	143000	342000	266000	654000	534.000
TS	20240	43500	88100	64200	36300	33.300
GLS	688	1540	895	114000	163000	70.600
ILS	-	2250	4000	775	3010	81.800
EA	-	-	988000	250000	547000	203.000
AC	-	-	105000	2170000	216000	622.000
GRASP	-	-	4370	22600	26400	14.900
SI	-	-	-	110000	26100	32.600
HI	-	-	-	12600	30.700	19.400
GP	-	-	-	267000	253.000	423.000
AIS	-	-	-	32100	11.200	59.100

AIS = <''artificial immune systems''>

- SA and EA are known and used outside the computer science/enginering/OR domain.

An Operations Researchers Work



Search terms:

SA = <''simulated annealing''>

TS = <''tabu search'' OR {}''taboo search''>

GLS = <''guided local search''>

ILS = <''iterated local search''>

EA = <''evolutionary algorithms'' OR {}''genetic

AC = <''ant colony'' OR {}''ant system''>

GRASP = <''greedy randomized adaptive search''>

SI = <''swarm intelligence''>

HH = <''hyper heuristics''>

GP = <''genetic programming''>

Large-Scale Optimization

What is this course about ?

- Methods to solve problems which cannot be solved by standard solvers.

Hill Climbers and Simulated Annealing

Optimisation as search

What are hard problems ?

Problems can be hard for two reasons:

- Large
- Complex:
  - Complex objective functions/constraints: Non-linear, non-continous
  - Complex domains for optimisation variables: Integers !

How many of you have heard about complexity theory ? (computer science).

Computer based optimisation

Assume we have:

- A welldefined problem domain, i.e. complete identification of design variables (endogenous variables)
- A computer model (program) which can calculate the costs of a certain choice of variables

Then we can use a computer to search the problem domain.

How to search ?

There are several ways to search the possible solutions:

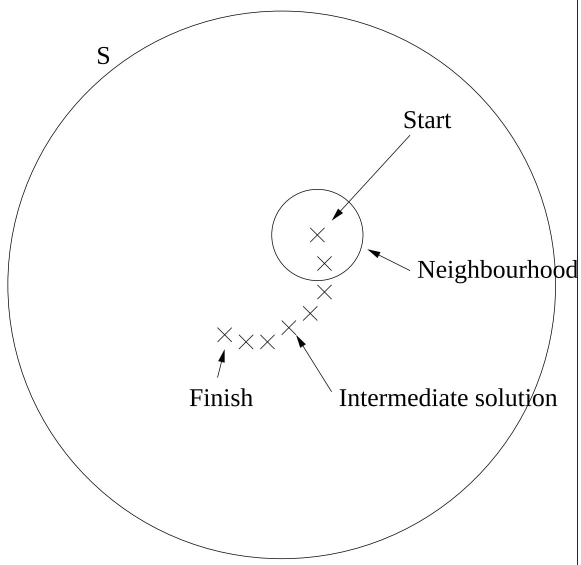
- Systematic search, i.e. complete enumeration (branch and bound)
- Random search, just try as many random samples as possible
- Smart search, maybe we can create a smart algorithm to solve the particular problem, eventhough the number of possible solutions is huge (P problems .....)
- Local search: Start somewhere and then iteratively improve

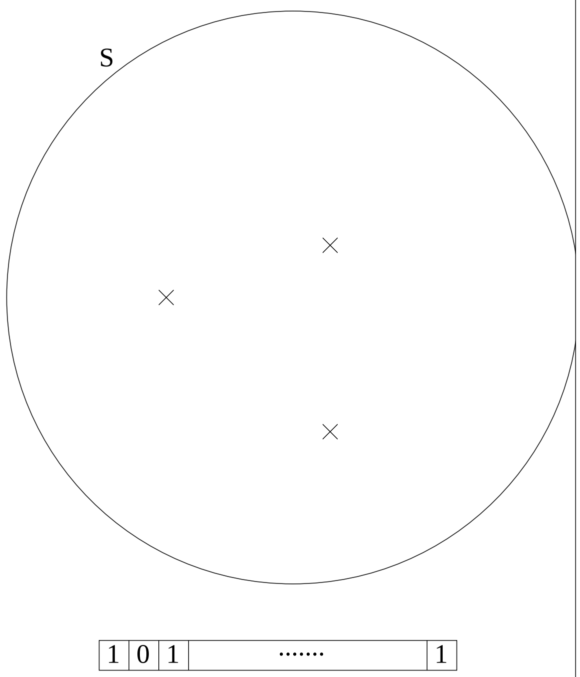
Local Search Terminology

When working with local search we use a number of terms:

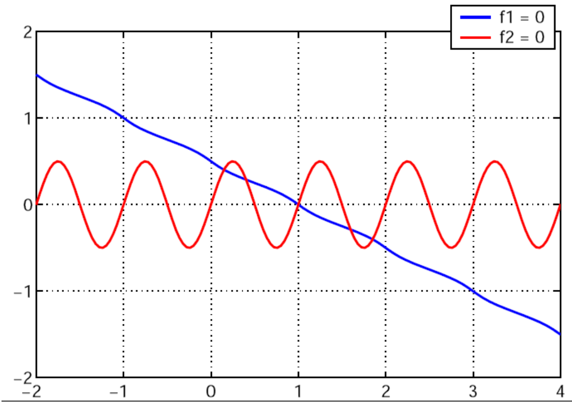
- A **solution**, a possible but not necessary optimum to a optimisation problem, i.e. an assignment of variable values
- A **neighbourhood** is a number of solutions within a certain "distance" of a solution in parameter space
- An **evaluation function** which maps parameter space into objective space

Local Search Terminology

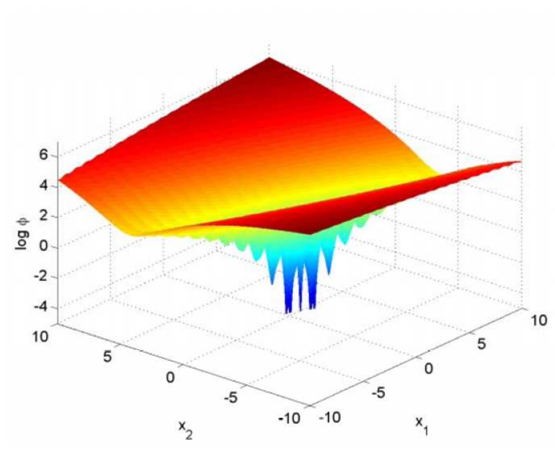




Zero Points



3D-plot

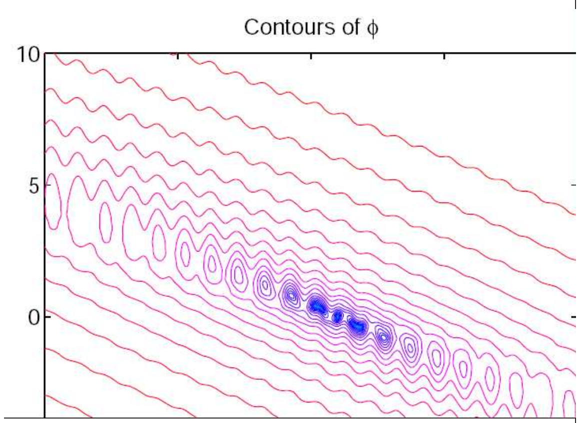


Example: Branin function

Lets look at a simple example: Branin's function:

- $f_1(x, y) = 1 - 2y + \frac{1}{20}sin(4\pi y) - x$
- $f_2(x, y) = y - \frac{1}{2}sin(2\pi x)$
- $\phi(x, y) = \frac{1}{2}(f_1(x, y)^2 + f_2(x, y)^2)$

Contour



Hill Climbers

Assuming minimisation we can use a so-called hill climber (actually hill descenter):

select initial solution s_0
repeat
    select $s \in N(s_i)$
    if $f(s) < f(s_i)$ **then**
        $s_{i+1} = s$
until stopping criterion is true

Notice that we could view the simplex algorithm as a local hill climber.

Hill climber parts

The main parts of the hillclimber are:

- Generation of initial solution
- Definition of neighbourhood
- Selection from neighbourhood:
  - Select best
  - Search entire neighbourhood, but switch when better
- Stopping criteria

Problems with hill climbers

There are two main problems with hill climbers:

- Local optima, means no optimality guarantee
- Needs adaptation to the problem

But there are many examples where simple hillclimbers perform well.

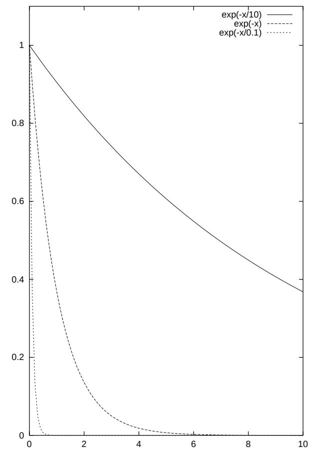
Hill climbing on Branin

So lets try to hill climb (ok, hill descent) the Branin function.

Simulated Annealing

select initial solution s_0
$t = T_{start}$
repeat
    select $s \in N(s_i)$
    $\delta = f(s) - f(s_i)$
    if $\delta < 0$ **or** with probability $p(\delta, t_i)$ **then**
        $s_{i+1} = s$
    $t_{i+1} = t_i \cdot \alpha$
until stopping criterion is true

Probability function



I admit

Ok, I admit, I am cheating:

- I have discretized the variables, i.e. instead of two continuous variables in the interval $[-10, 10]$ they have been replaced with two discrete variables between 0 and 100. These are then rescaled into the interval.

- Actually I am using the Simulated Annealing algorithm instead of a hill climber, but setting the *parameters* in a stupid fashion, simulating a hill climber.

Acceptance probability

The analogy to statistical thermodynamics gives us the following acceptance probability $p(n)$:

$$p(\delta, t_i) = \exp(-\frac{1}{t_i}\delta)$$

where t_i is the "temperature" at step i. Typically $p(i)$ decreases with time i and the size of the deteriation (δ).

The laws of thermodynamics state that at temperature t, the probability of an increase in energy of magnitude δE is given by:

$$p(\delta E) = \exp(-\frac{\delta E}{kt})$$

Book: E. Aarts & J. Korst “imulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing ”, 1989. (chapter 11 !!!)

More on the temperature

- The start temperature should be high enough to make a large fraction of the neighbours acceptable.
- The temperature should not be so high that we spend time working with to high temperatures.
- The *accept ratio* is the ratio between the number of selected solutions and the number of accepted solutions.
  - Aim to get the accept ratio around 0.5 in the beginning of the search.
  - Can depend on the structure of the problem.

General Comments

Simulated Annealing is:

- Invented in 1983 by Kirkpatrick, Gellat and Vecchi (based on earlier work by Metropolis).
- The simplest of the metaheuristic algorithms
- Have only few parameters: T_{start}, α and termination parameter ...
- Is invented based on the annealing process of metals (but the analogy is weak) ....
- Delta evaluation is usually possible

Controlling the temperature

- The tempeature should allow (almost) all moves to be accepted in the beginning of the execution of the algorithm,
- then gradually the temperature is decreased, and
- towards the end of the execution of the algorithm almost only improving solutions should be accepted.
- Methods for controlling the temperature are called *cooling schedules* or *temperature schedules*.

Stopping criterion

- If f^* was not improved at least ϵ % after K iterations.
- If the number of accepted moves is less than ϵ % of the last K iterations.
- After K_{max} iterations.
- When the temperature gets below a predefined constant τ.

- As the temperature tends to zero the distributions tends to a uniform distribution over the set of optimal solutions.
- As the temperature is not constant the process can be seen as a number of different homogenour Markov chains.

  - Now we need to determine t_0 on the basis of the accept ratio.
- Try different values and see what happens.
- If time is not a critical faktor "just choose t_0 high enough".
- If we are using a "good" start solution (generated by a construction heuristic) high temperatures might "destroy it". Therefore start lower.

A little theory

- The behaviour of simulated annealing can be modelled using Markov chains.
- At **constant** temperature the probability p_{ij} of moving from one solution i to another solution j depends only on i and j.
- This type of transition give rise to a homogenous Markov chain.
- This process converges towards a stationary distribution which is independent of the starting solution.

But,....

However, their usefulness in practical applications is limited as they imply solutions which are exponential in problem size and often require more iterations than exhaustive search.

What about the neighbourhood?

- A difference to Tabu Search is that we do not have to check the whole neighbourhood. Size is not that critical anymore.
- Is the change easy to compute?
- Allow non-feasible solutions to be part of the solution space.

Tips and tricks

- Consider making some of the constraints into "soft constraints".
- Consider the number of **accepted moves** rather than the number of trials when deciding when to change the temperature. This number is denoted the *cut-off parameter*). Short time will be spend at high temperatures. Be careful about not using too much time at low temperatures.
- Replace the computation of exp$(-\delta/t_n)$ by an approximation or precomputed exponentials.
- Use *cyclic sampling* instead of random sampling (requires enumeration of the neighbourhood).

- Reheating? (non-monotonic SA).
- Many short runs instead of one long run.
- Look at pictures or traces of the execution of the algorithm.

Problems

Problems with the algorithm:

- The theory is weak ...
- Very little direction, means long runs ...
- Difficult to parallelize
- Difficult to use for Multi Criteria optimisation