Simulation and Optimization for Decision Support (G54SOD)

Semester 2 of Academic Session 2017-2018

Dr Dario Landa-Silva
dario.landasilva@nottingham.ac.uk
http://www.cs.nott.ac.uk/~pszjds

Workshop 6 – Understanding Local Search Heuristics

- Local Search Heuristics
  To describe different variants of local search heuristic methods
- Some Design Issues for Heuristics
  To explain some of the most important design issues to consider when implementing local search heuristics

Additional Reading

Heuristic Local Search Tutorial
https://rdmc.nottingham.ac.uk/handle/internal/168#

Sections 2.3 to 2.8 of the book (Talbi, 2009)

Section 5.1 of the book (Rothlauf, 2011)

Three Methods to Automate the Space Allocation Process in UK Universities. E.K. Burke, P. Cowling, J.D. Landa-Silva, B. McCollum. *Practice and Theory of Automated Timetabling III (PATAT 2000)*, E. Burke, W. Erben (eds.), Lecture Notes in Computer Science, Vol. 2079, pp. 254-272, Springer, 2001.
http://www.cs.nott.ac.uk/~pszjds/research/files/jdls_patat2000.pdf

Metaheuristics - the Metaphor Exposed. Sörensen, Kenneth. *International Transactions in Operational Research*. Vol. 22, No. 1, pp. 3-18, 2015.
http://onlinelibrary.wiley.com/doi/10.1111/itor.12001/full

Local Search Heuristics

General Iterative Improvement (also hill-climbing or descent)

1. Generate initial solution x
2. Select x’ ∈ N(x)
3. If f(x’) is better than f(x) then x = x’
4. If stop condition true then finish, otherwise go to 2

Factors that influence the overall search strategy:

- Quality of initial solution
  - Random, Greedy, Peckish
- Quality of the selected neighbour solution
  - Random, First improving, Best improving, Random improving, Least improving, Probabilistic improving
- The neighbourhood explored
  - Size, Quality, Single/Multiple
- Stopping condition
  - Time Limit, Idle Iterations, Max Iterations, etc.

Steepest (Greedy) Iterative Improvement

```
1. Generate initial solution x
2. Set best known solution $x_{best} = x$
3. local_opt = *false*
4. Select the best $x' \in N(x)$
    a) If $f(x')$ is better than $f(x)$ then $x = x'$
    b) Else local_opt = *true*
5. If local_opt = *false* then go to 4
6. If $f(x)$ is better than $f(x_{best})$ then $x_{best} = x$
7. If stop condition true then finish, otherwise go to 2
```

In steepest iterative improvement (also called greedy iterative improvement), the best neighbour solution is selected every time.

Although simple to implement, like other local search heuristics, steepest iterative improvement has limitations in terms of solution quality and computation time.

Some Strategies For Escaping Local Optima

- Re-initialise the search from a different starting solution.
- Accept non-improving candidate solutions to replace the current solution in deterministic or probabilistic manner.
- Explore only a fraction of the neighbourhood (e.g. pruning) for peckish instead of greedy search.
- Use more than one neighbourhood to generate candidate solutions.
- Design composite (independent or not) neighbourhoods.
- Maintain a memory of already visited solutions or regions.
- Modify the fitness function to change the search landscape.
- Perturb local optima in deterministic or probabilistic manner.
- Improve (evolve) more than one solution simultaneously.

Intensification **vs.** **Diversification**

Meta-heuristics

Greedy heuristics and simple local search have some limitations with respect to the quality of the solution that they can produce.

A good balance between intensification and diversification is crucial in order to obtain high-quality solutions for difficult problems.

A meta-heuristic can be defined as "an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method"(Voss et al. 1999).

A meta-heuristic method is meant to provide a generalised approach that can be applied to different problems.

Types of Meta-heuristics

Single-solution method vs. Population-based methods

Nature-inspired method vs. Non-nature inspired methods

'Pure' methods vs. Hybrid methods

Memory-based vs. Memory-less methods

Deterministic vs. Stochastic methods

Iterative vs. Greedy methods

Examples of Meta-heuristics

The following algorithms are examples of meta-heuristics:

- Iterated local search
- Threshold acceptance
- Great deluge
- Simulated annealing
- Greedy randomised search procedure
- Guided local search
- Variable neighbourhood search
- Tabu search
- Evolutionary algorithms
- Particle swarm optimisation
- Artificial immune systems
- Etc.

Single-solution Meta-heuristics

These methods maintain one current solution at a time and conduct the search moving from one solution to another while building a single-point trajectory.

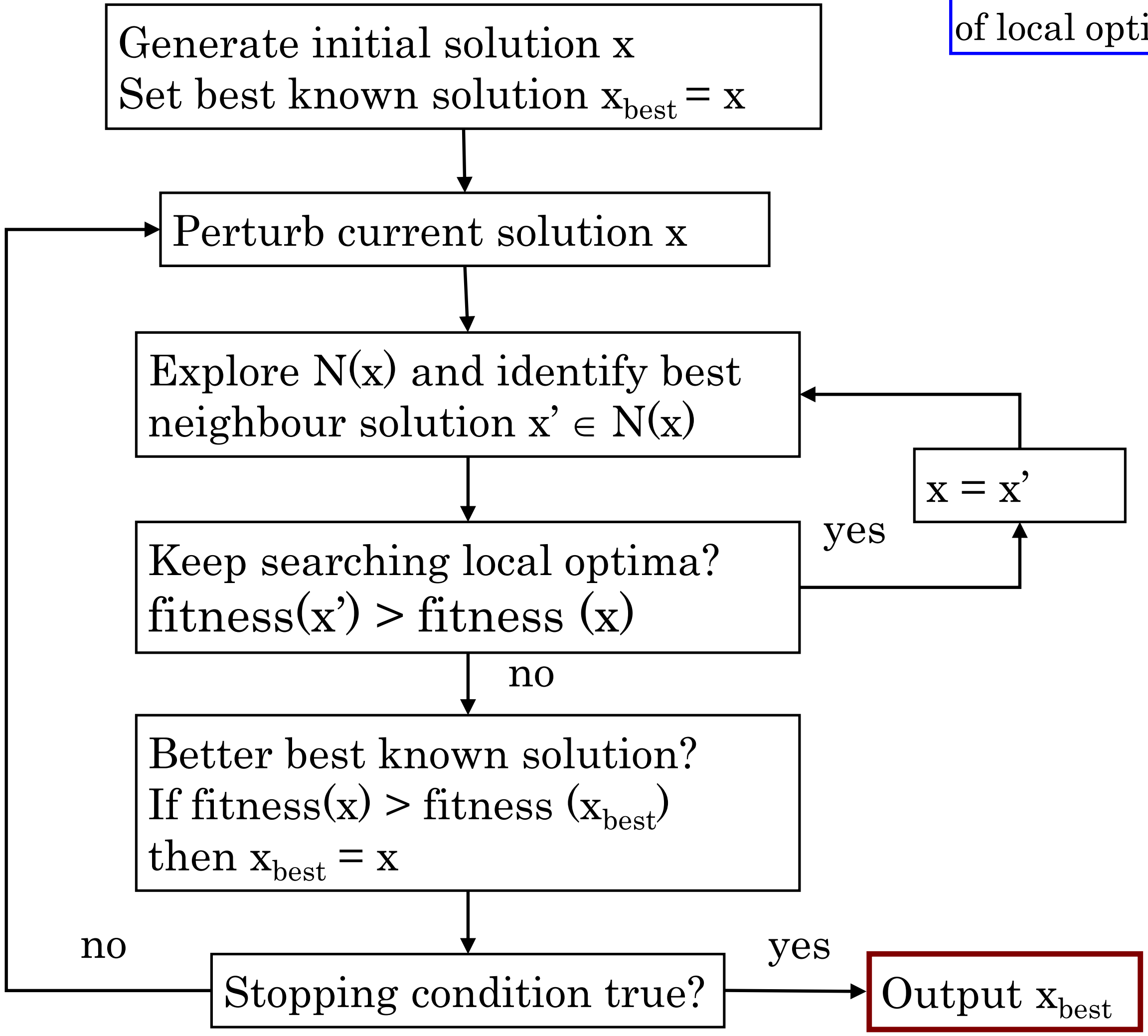
Important common components in single-solution meta-heuristics:

- Generation of candidate solutions (random, greedy, peckish, etc.)
- Definition of neighbourhoods (exchange, swaps, insertion, inversion, very large, ejection chains, cyclic, etc.)
- Computation of objective function (exact, delta, approximate or surrogate, etc.)
- Replacement of current solution (deterministic, probabilistic, only improving, non-improving, etc.)
- Termination criteria (execution time, total iterations, total function evaluations, idle iterations, etc.)
- Parameter tuning (temperature, water level, memory length, etc.)

Designing good components is perhaps more important than tuning parameters

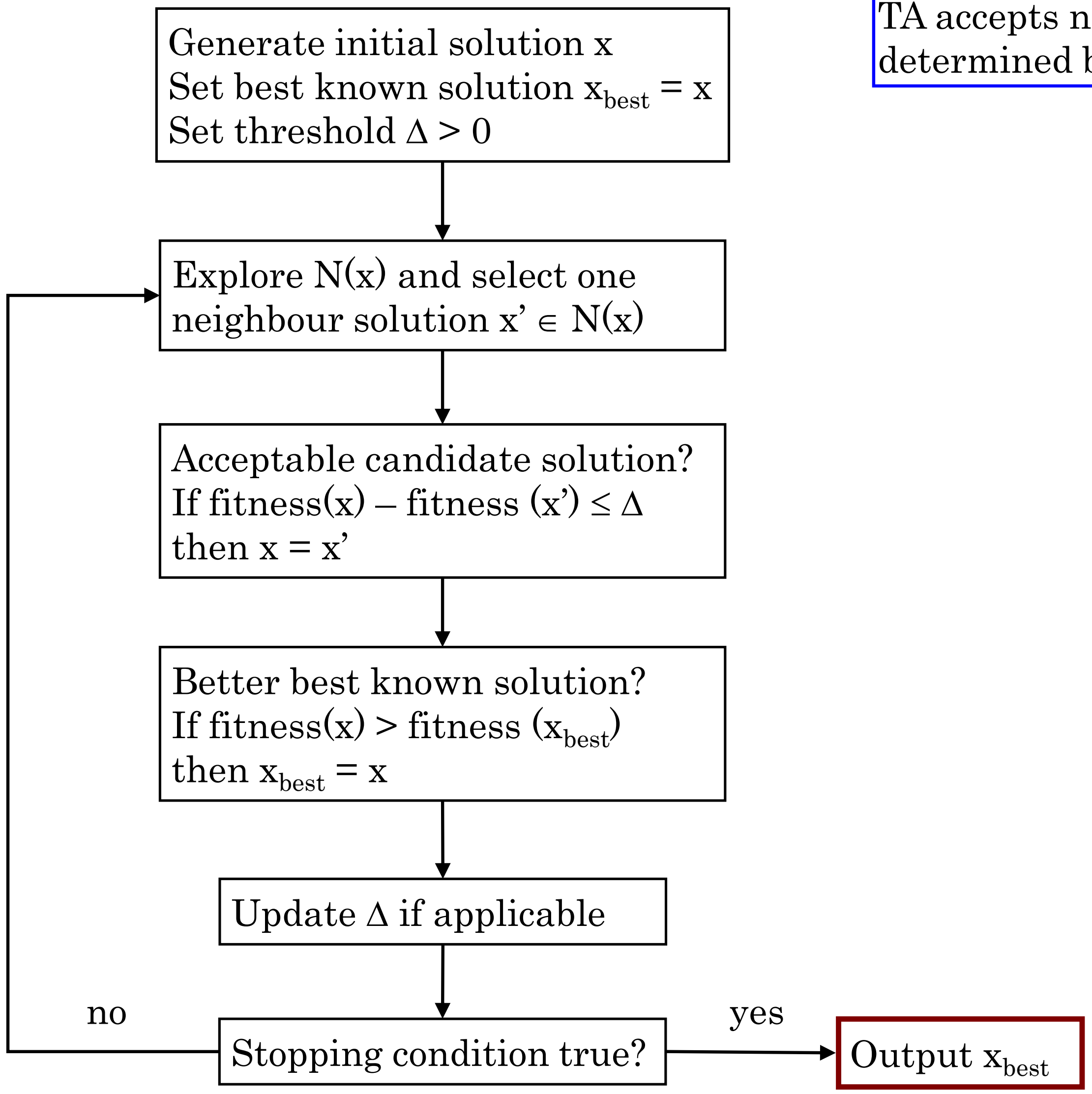
Iterated Local Search

ILS works on perturbations of local optima



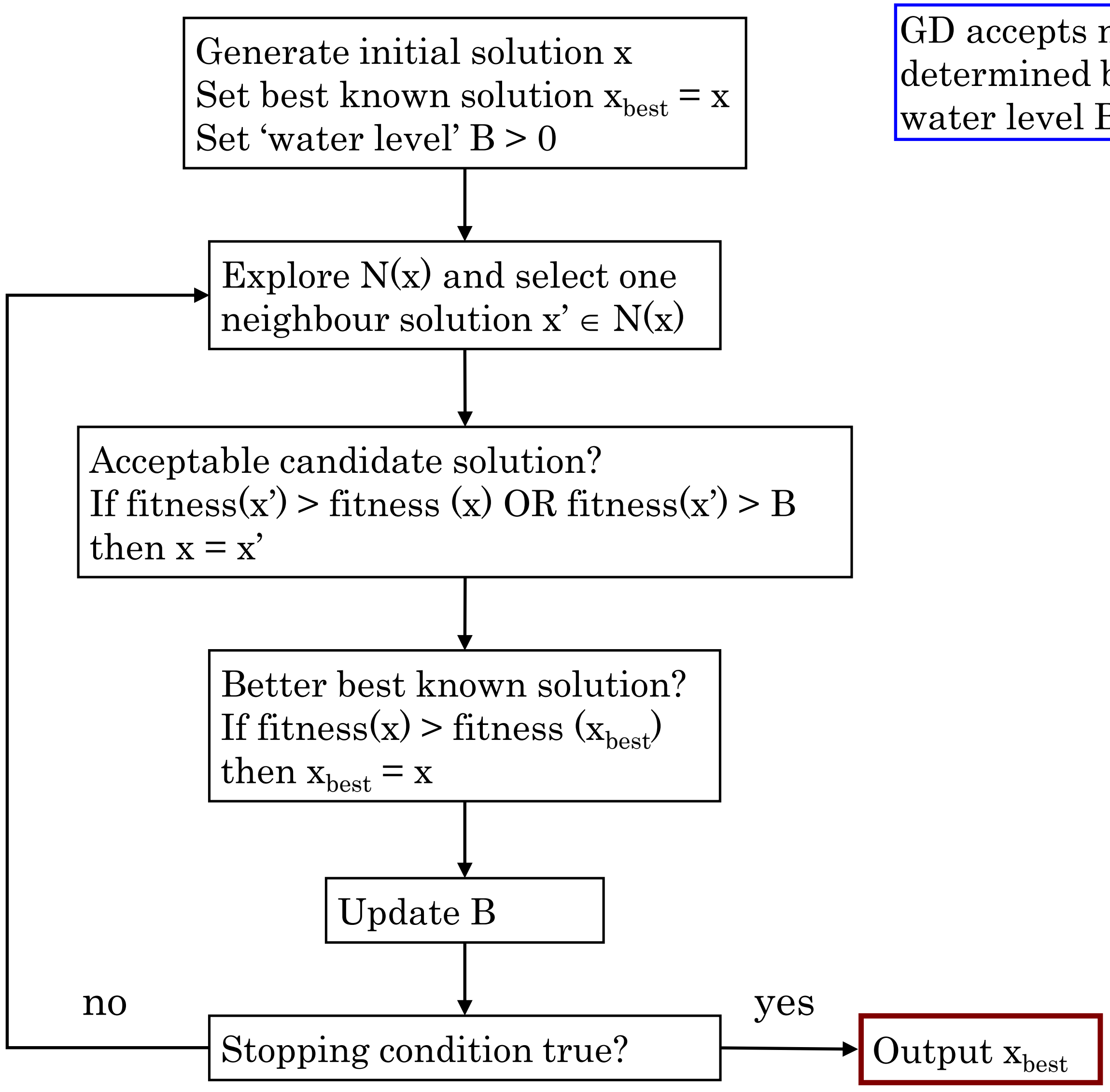
Generate initial solution x
Set best known solution x_{best} = x
Perturb current solution x
Explore N(x) and identify best neighbour solution x' $\in$ N(x)
Keep searching local optima? fitness(x') > fitness (x)
yes
x = x'
no
Better best known solution? If fitness(x) > fitness (x_{best}) then x_{best} = x
no
Stopping condition true?
yes
Output x_{best}

Threshold Acceptance


Generate initial solution x
Set best known solution x_{best} = x
Set threshold $\Delta > 0$
Explore N(x) and select one neighbour solution $x' \in N(x)$
Acceptable candidate solution? If fitness(x) – fitness (x') $\leq \Delta$ then x = x'
Better best known solution? If fitness(x) > fitness (x_{best}) then x_{best} = x
Update Δ if applicable
Stopping condition true?
no
yes
Output x_{best}

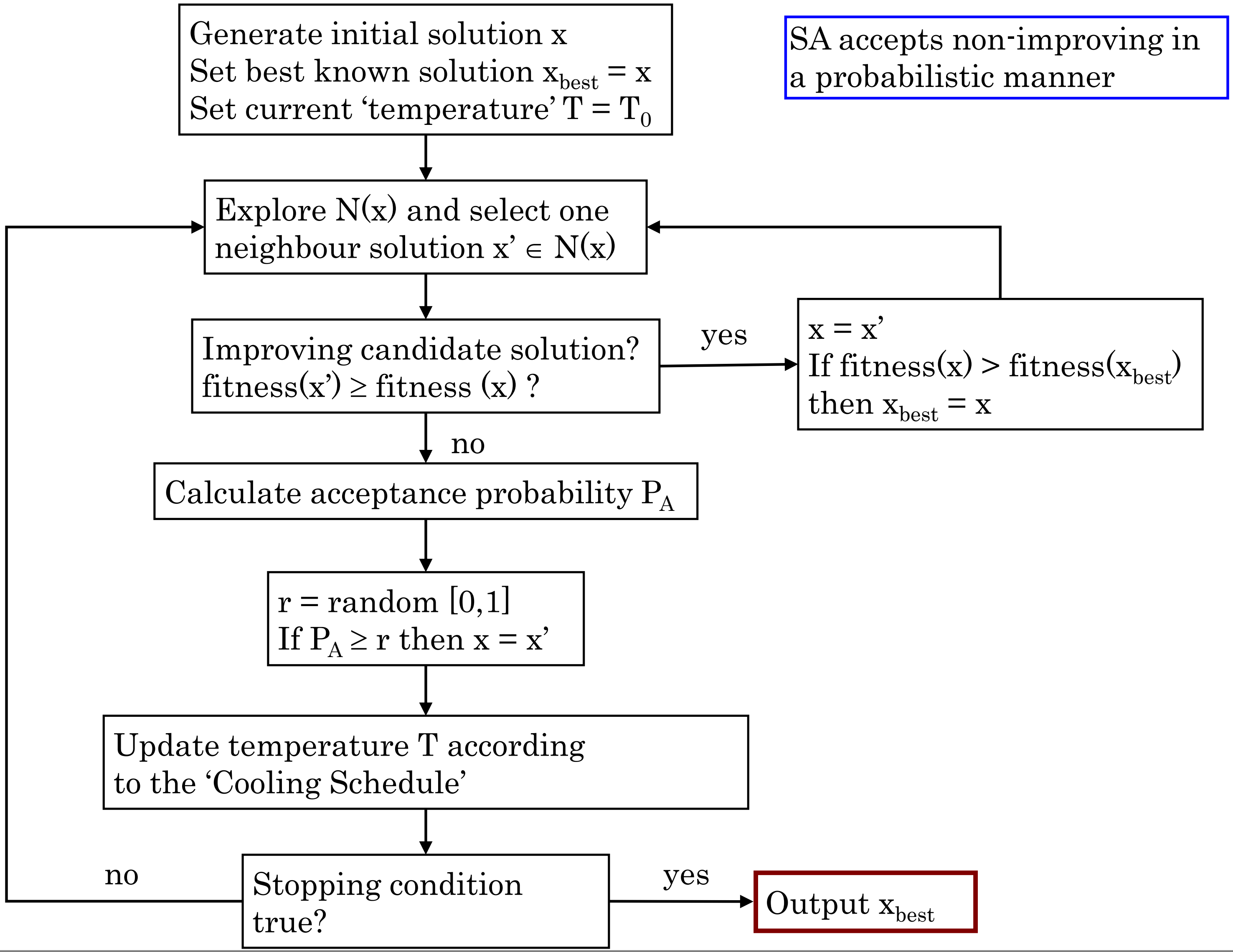
TA accepts non-improving as determined by a threshold

Great Deluge


Generate initial solution x
Set best known solution x_{best} = x
Set 'water level' B > 0
Explore N(x) and select one neighbour solution x' ∈ N(x)
Acceptable candidate solution?
If fitness(x') > fitness (x) OR fitness(x') > B then x = x'
Better best known solution?
If fitness(x) > fitness (x_{best}) then x_{best} = x
Update B
Stopping condition true?
no
yes
Output x_{best}

GD accepts non-improving as determined by a changing water level B

Simulated Annealing


Generate initial solution x
Set best known solution x_{best} = x
Set current 'temperature' T = T_0
Explore N(x) and select one neighbour solution x' $\in$ N(x)
Improving candidate solution? fitness(x') $\geq$ fitness (x) ?
yes
x = x'
If fitness(x) > fitness(x_{best}) then x_{best} = x
no
Calculate acceptance probability P_A
r = random [0,1]
If $P_A \geq$ r then x = x'
Update temperature T according to the 'Cooling Schedule'
Stopping condition true?
no
yes
Output x_{best}

SA accepts non-improving in a probabilistic manner

Cooling Schedule in SA

In general, the current temperature T_i is determined by:

- Initial temperature T_0 (translates to initial probability)
- Final temperature T_F (translates to final probability)
- Decrement step t_{step}, i.e. number of iterations between temperature decrements.
- Cooling factor ΔT, i.e. the proportion of the temperature reduction.
- Re-heating step t_{reheat}, i.e. number of iterations after which the temperature is increased to the initial temperature or to another value.

The arithmetic cooling schedule modifies the temperature as follows.

$$\text{every } t_{step} \text{ iterations}: \quad T_i = T_{i-1} - \Delta T$$

$$\text{after } t_{reheat} \text{ iterations}: \quad T_i = T_0$$

the initial temperature is set to a high value in the standard SA algorithm, but it can also be set to a low value or to zero.

The geometric cooling schedule modifies the temperature as follows.

$$\text{every } t_{step} \text{ iterations}: \quad T_i = \alpha T_{i-1} \quad \text{where } 0 < \alpha < 1$$

alternatively

$$\text{every } t_{step} \text{ iterations}: \quad T_i = \frac{\alpha T_{i-1}}{1 + \alpha T_{i-1}} \quad \text{where } 0 < \alpha < 1$$

Note: re-heating can also be performed

The quadratic cooling schedule modifies the temperature as follows.

$$\text{every } t_{step} \text{ iterations}: \quad T_i = ai2 + bi + c \quad \text{where } a = \frac{T_0 T_F}{I_{total}}, b = \frac{T_F - T_0}{I_{total}}, c = T_0$$

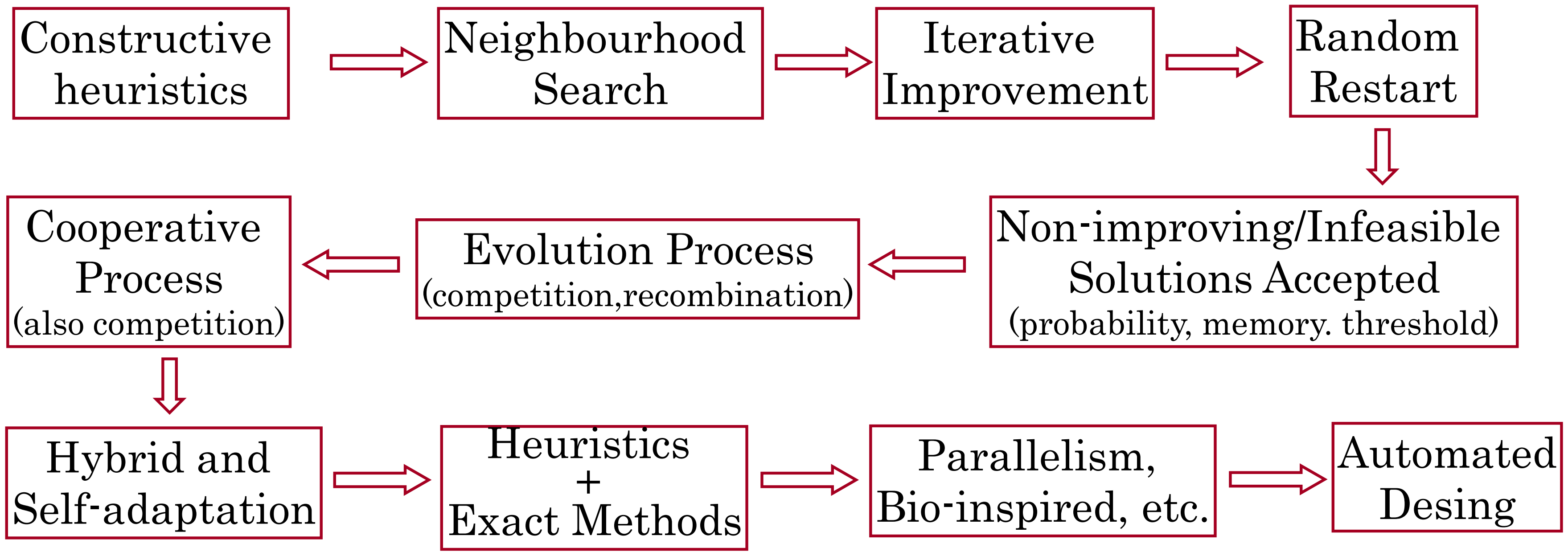
Typically, the acceptance probability is calculated as follows.

$$P_A = \exp^{-\left(\Delta fitness / T_i\right)}$$

Progress on Heuristic Search

A heuristic is a 'rule of thumb' based on domain knowledge from a particular application, that gives guidance in the solution of a problem (Oxford Dictionary of Computing).

A meta-heuristic is a iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions (Voss et al. 1999).


Constructive heuristics
Neighbourhood Search
Iterative Improvement
Random Restart
Non-improving/Infeasible Solutions Accepted (probability, memory. threshold)
Evolution Process (competition,recombination)
Cooperative Process (also competition)
Hybrid and Self-adaptation
Heuristics + Exact Methods
Parallelism, Bio-inspired, etc.
Automated Desing

Some Design Issues for Heuristics

Delta Fitness Evaluation

Refers to making only incremental updates to the objective function by calculating only the effect of the differences between the candidate solution x' and the current solution x.

The goal of delta fitness evaluation is to make local search more efficient particularly in problems with computationally expensive objective functions.

Example. Given solution s for a symmetric TSP with 5 cities and its corresponding fitness value:

$$s = c_1 c_3 c_4 c_2 c_5 \quad \text{then} \quad f(s) = d(c_1,c_3) + d(c_3,c_4) + d(c_4,c_2) + d(c_2,c_5) + d(c_5,c_1)$$

after a 2-edge-exchange move which gives solution s':

$$s = c_1 c_3 c_4 c_2 c_5 \quad \text{and} \quad s' = c_1 c_4 c_2 c_5 c_3 \quad \text{then delta fitness evaluation is :}$$

$$f(s') = d(c_1,c_4) + d(c_4,c_2) + d(c_2,c_5) + d(c_5,c_3) + d(c_3,c_1)$$

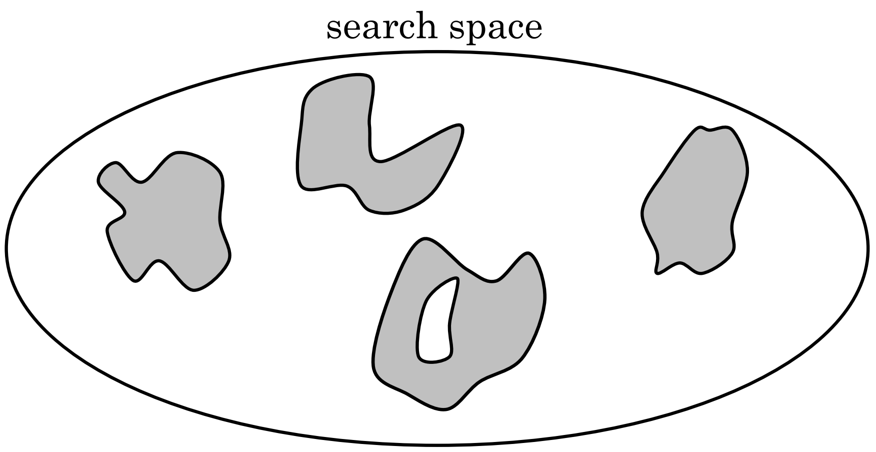
$$f(s') = f(s) - d(c_3,c_4) - d(c_5,c_1) + d(c_1,c_4) + d(c_5,c_3)$$

Handling Constraints

In heuristic optimization one of the key issues is how to deal with constraints.

Should infeasible solutions be considered during the search?
or
Should infeasible solutions be rejected straight away?

The objective function plays a crucial role in ranking solutions both feasible and infeasible.



The search space usually consists of disjoint subsets of feasible and infeasible solutions, although there is no guarantee that these subsets are connected.

Issues When Dealing with Infeasible Solutions

- Evaluate quality of feasible and infeasible solutions
- Discriminate between feasible and infeasible solutions
- Decide if infeasibility should be eliminated
- Decide if infeasibility should be repaired
- Decide if infeasibility should be penalised
- Initiate the search with feasible solutions, infeasible ones or both
- Maintain feasibility with specialised representations, moves and operators
- Design a decoder to translate an encoding into a feasible solution
- Focus the search on specific areas, e.g. in the boundaries

A constraint handling technique is required to help the effective and efficient exploration of the search space.

Rejecting Infeasible Solutions

This is a simple strategy that has serious limitations particularly on those problems where generating feasible solutions is considerably more difficult that generating infeasible ones.

A heuristic search algorithm might need to cross between the feasible and infeasible regions of the search space.

In local search, assessing the feasibility of neighbour solutions might be helpful as it could help to obtain an idea of the fitness landscape.

Important issue: assess $\frac{|S_F|}{|S|}$ where,

S is the search space under consideration

S_F is the feasible portion of S

Enforcing Feasible Solutions

The use of specialised representations, decoders, neighbourhood moves and reproduction operators is a reasonable way to avoid generating infeasible solutions.

The characteristics of a good decoder are as follows:

- For each feasible solution there is an encoded one
- Each encoded solution corresponds to a feasible solution
- All feasible solutions should be represented by the same number of encoded ones
- The decoding procedure is computationally fast
- Small changes to the encoded solution represent small changes to the feasible solution

For example, a decoder can be used to convert a binary string into a solution to the MULTIPLE KNAPSACK problem.

Repairing Infeasible Solutions

This is a popular strategy to handle constraints in heuristic search where an infeasible solution x_{inf} is transformed into a repaired feasible solution x_{rep}.

Then, x_{rep} can replace x_{inf} or it might be that x_{inf} remains but it is evaluated as x_{rep}.

It is possible that in a population-based heuristic only a number of infeasible solutions are repaired in order to maintain diversity.

The major weakness of the repairing strategy is that is very problem dependant.

For example, in the MULTIPLE KNAPSACK problem a common repair strategy would be simply to remove items from the overloaded knapsack.

Penalising Infeasible Solutions

The key issue is to establish the relationship between the infeasible solution and the feasible part of the search space in order to assess the value of the components in the infeasible solution.

$$x_{inf} \Leftrightarrow S_{feasible} \left\{ \begin{array}{l} \bullet \text{ fixed penalty for infeasibility} \\ \bullet \text{ penalty proportional to degree of infeasibility} \\ \bullet \text{ penalty proportional to cost of repairing} \end{array} \right.$$

Penalties applied can be fixed or varied as the search progresses.

When tuning penalty functions: discrimination vs. exploration

For example, for the MULTIPLE KNAPSACK problem a way to penalise infeasible solutions can be to add a weighted penalty due to the amount of exceeded capacity.

Treating Constraints As Objectives

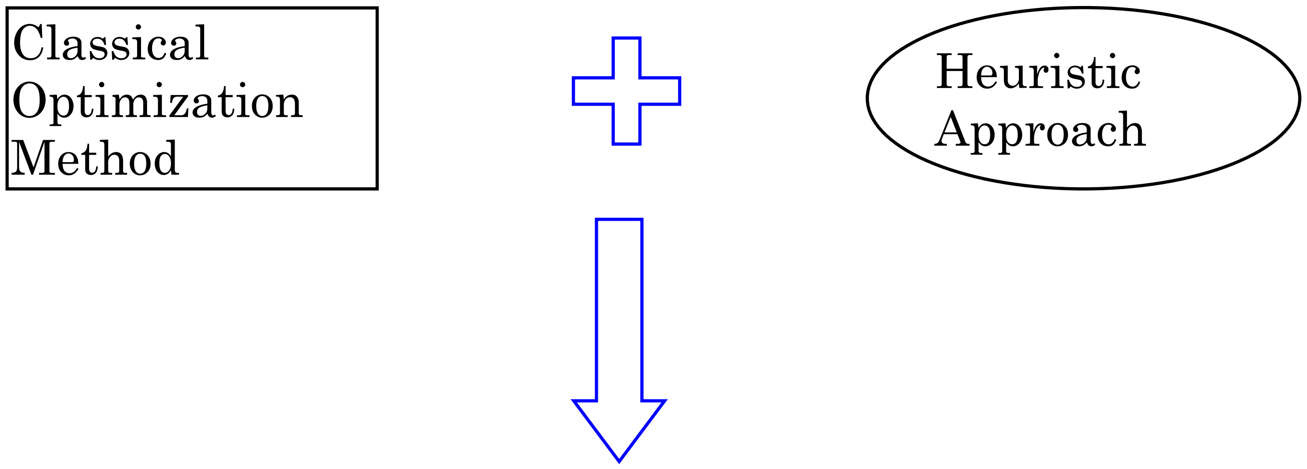
Another strategy is to set an evaluation function for each constraint (or group of constraints) in addition to the main objective function and then to tackle with a multi-objective approach.

This approach is particularly useful when just finding feasible solutions is very difficult as this provides a smoother fitness landscape for the heuristic search to operate.

It is important to choose appropriate scales for the different objectives (maybe normalise them) so that the search is not biased.

Having several objectives then allows the application of MOO techniques like: weighted aggregating functions, Pareto optimisation, goal programming, etc.

Integrating Classical Methods and Heuristics



- Use a classical method as part of a multi-stage approach in order to solve a part of the problem.
- Embed a classical method within a heuristic approach.
- Embed a heuristic approach within a classical method.
- Cross-fertilization by incorporating an ingredient of a classical method into a heuristic approach.