Metaheuristics

An introduction, Chapter 1.0–1.4

Joen Dahlberg

Communications and Transport Systems
Department of Science and Technology
Linköping University

LINKÖPING UNIVERSITY

Outline

LINKÖPING UNIVERSITY

Outline

LINKÖPING UNIVERSITY

Etymology

Heuristic

"...the art of discovering new strategies (rules) to solve problems."

Etymology

Heuristic

"...the art of discovering new strategies (rules) to solve problems."

Meta

"...upper level methodology"

Etymology

Heuristic

"... the art of discovering new strategies (rules) to solve problems."

Meta

"... upper level methodology"

Metaheuristic search methods

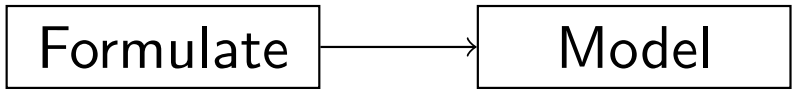
"... upper level general methodologies (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems."

Outline

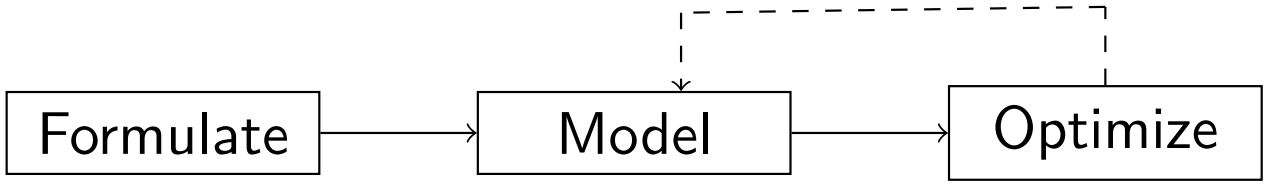
Classical process

Formulate

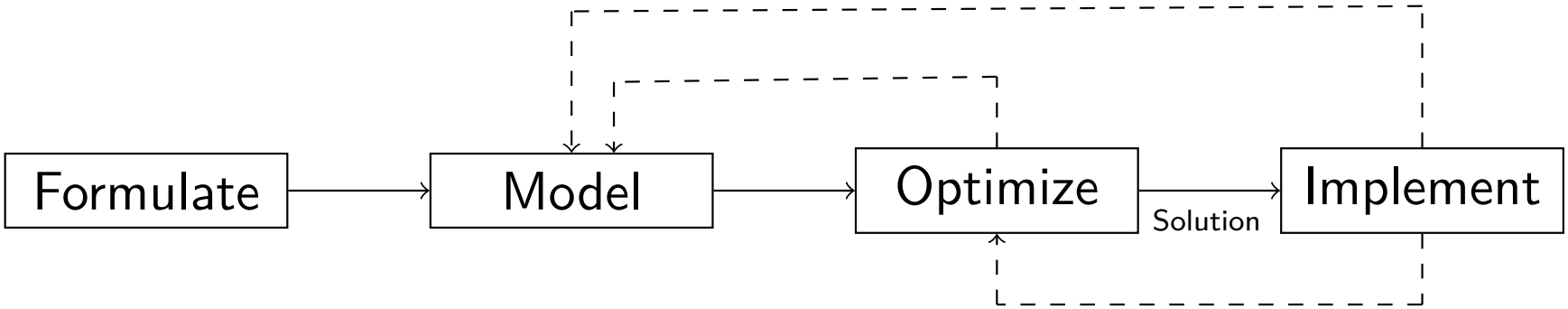
Classical process



Classical process



Classical process



Classical optimization models

"An optimization problem may be defined by the couple (S, f) where S represents the set of feasible solutions[1], and $f : S \to \mathbb{R}$ the objective function to optimize."

[1] S is also referred to as search space or feasible region.

Definition 1.1 Global optimum

A solution $s^* \in S$ is a global optimum if it has a better or equal objective function value than all solutions of the search space, that is, $\forall s \in S$, $f(s^*) \leqslant f(s)$.

The book is assuming minimization problems (maximizing an objective function f is equivalent to minimizing $-f$.

Linear programming

$$
\begin{array}{lrcl}
\text{min} & cx & & \\
\text{subject to} & Ax & \geqslant & b \\
& x & \geqslant & 0
\end{array}
$$

Linear programming

$$
\begin{array}{lrcl}
\text{min} & cx & & \\
\text{subject to} & Ax & \geqslant & b \\
& x & \geqslant & 0
\end{array}
$$

$$
\begin{array}{lrcrcl}
\text{max profit}= & 5x_1 & + & 4x_2 & & \\
\text{subject to} & 6x_1 & + & 4x_2 & \leqslant & 24 \\
& x_1 & + & 2x_2 & \leqslant & 6 \\
& & & x_1, x_2 & \geqslant & 0
\end{array}
$$

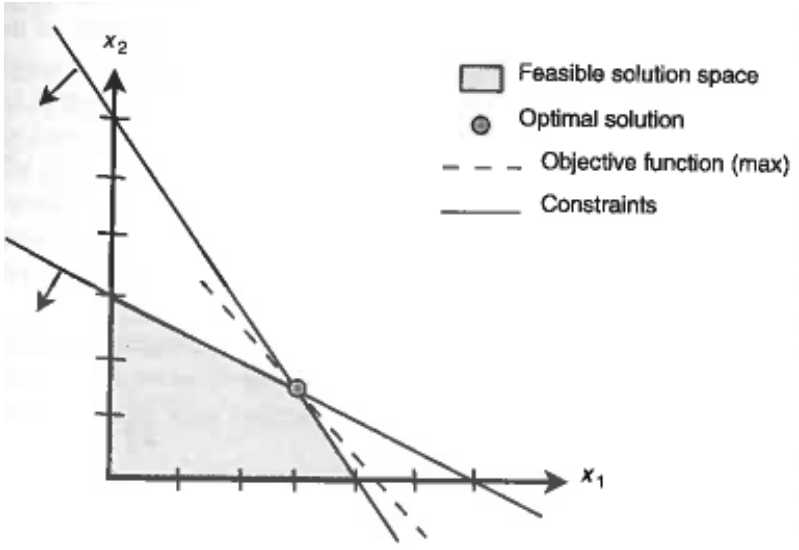
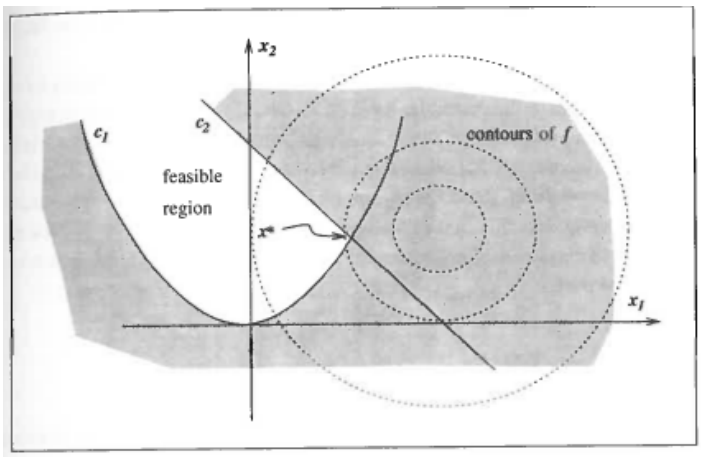


Figure 1.3 Graphical illustration of the LP model and its resolution.

Nonlinear programming

$$
\begin{array}{lrcrcl}
\text{min} & (x_1 - 2)^2 & + & (x_2 - 1)^2 & & \\
\text{subject to} & -x_1^2 & + & x_2 & \geqslant & 0 \\
& x_1 & + & x_2 & \leqslant & 2
\end{array}
$$

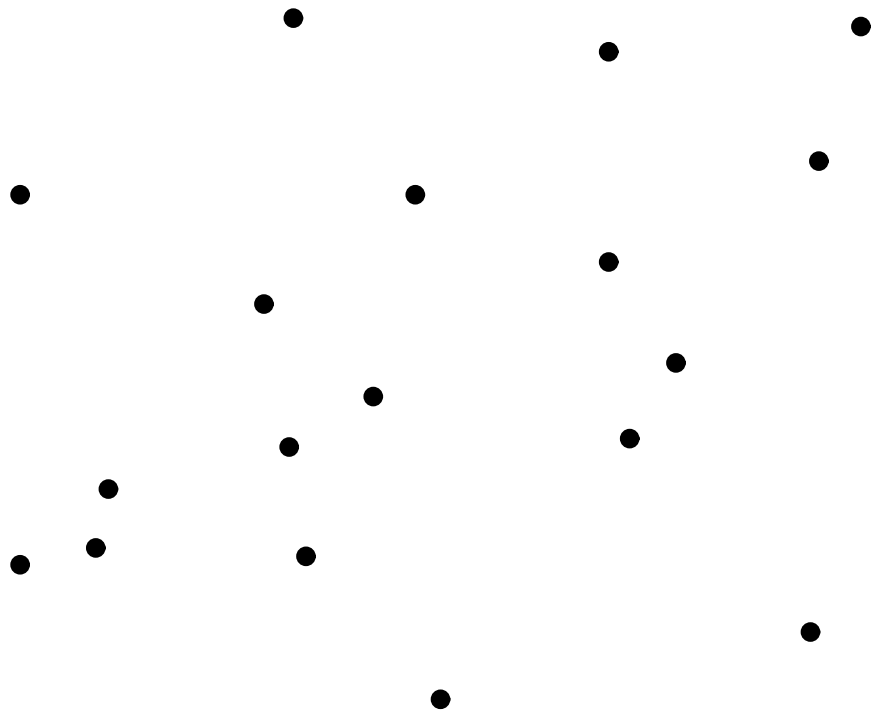


Example from Nocedal and Wright (2000).

Nocedal, J., & Wright, S. (2006). Numerical optimization. Springer Science & Business Media.

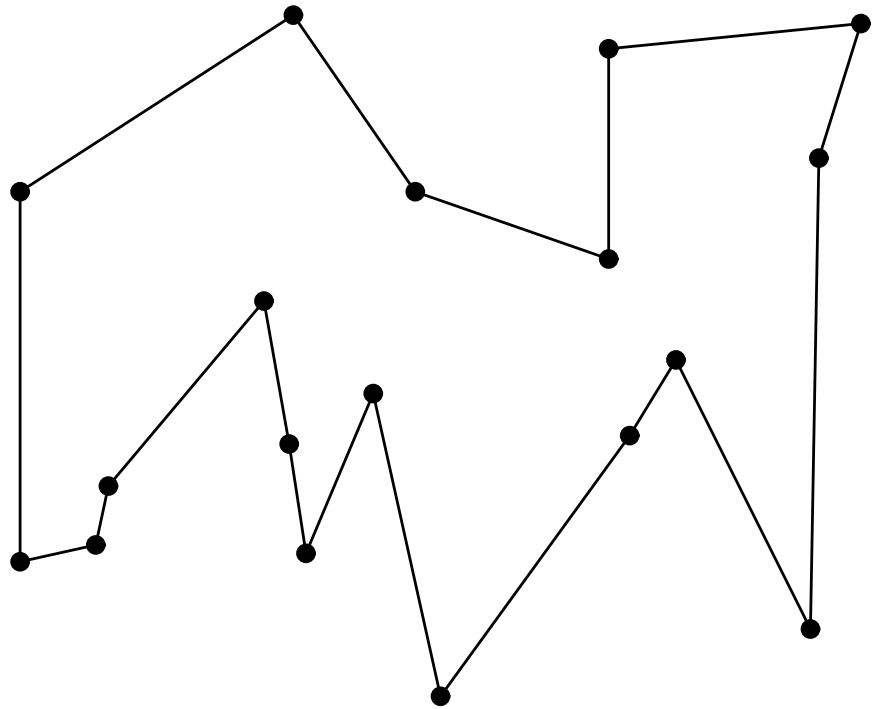
Combinatorial optimization

Traveling salesman problem (TSP)



Combinatorial optimization

Traveling salesman problem (TSP)



Combinatorial optimization

Traveling salesman problem (TSP)

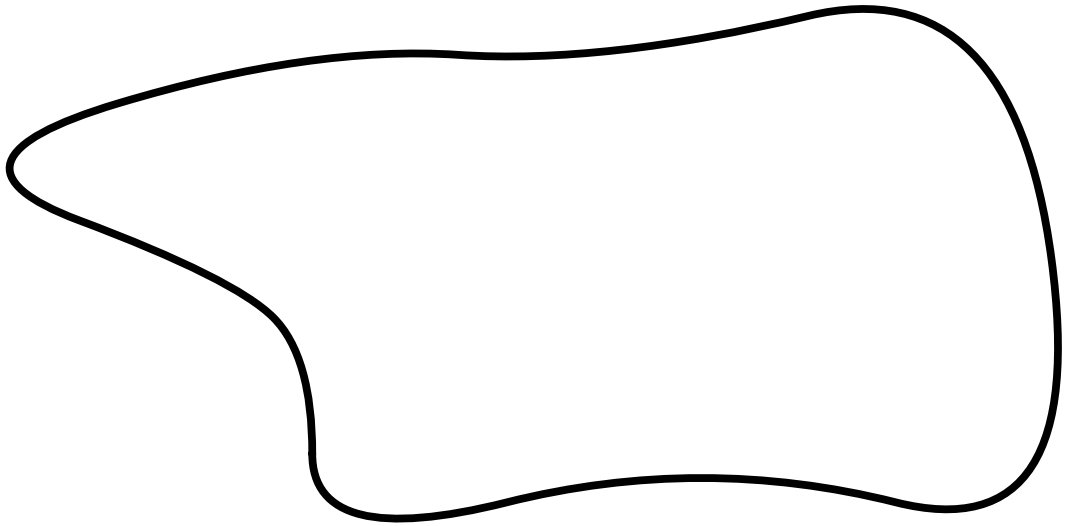
Figure 1.5 TSP instance with 24978 cities.

Outline

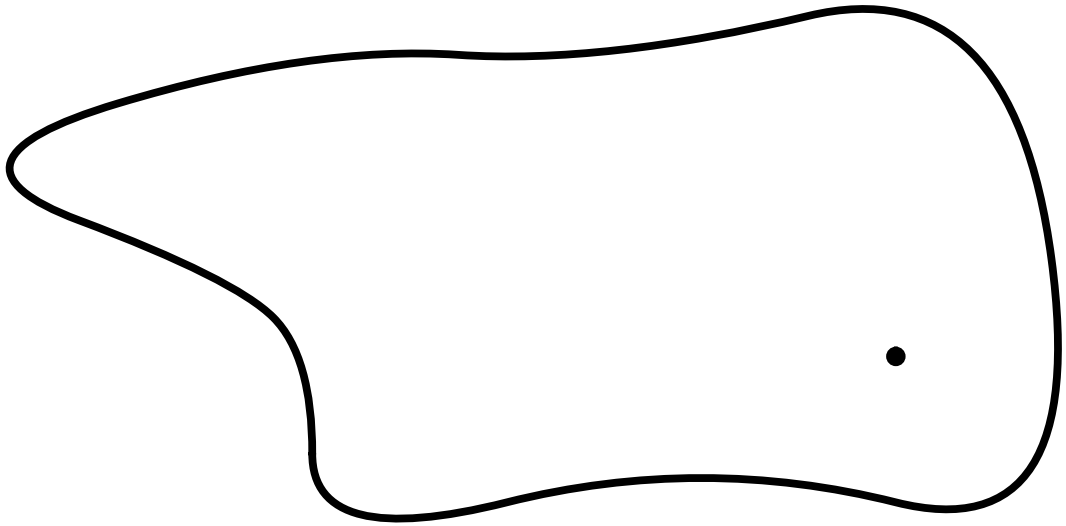
Local search – An example

Instead of looking in the whole feasible region for the best solution, we may search for the best solution in a local region of a given starting solution. And repeat for a number of steps, or until we find a local optimal solution.

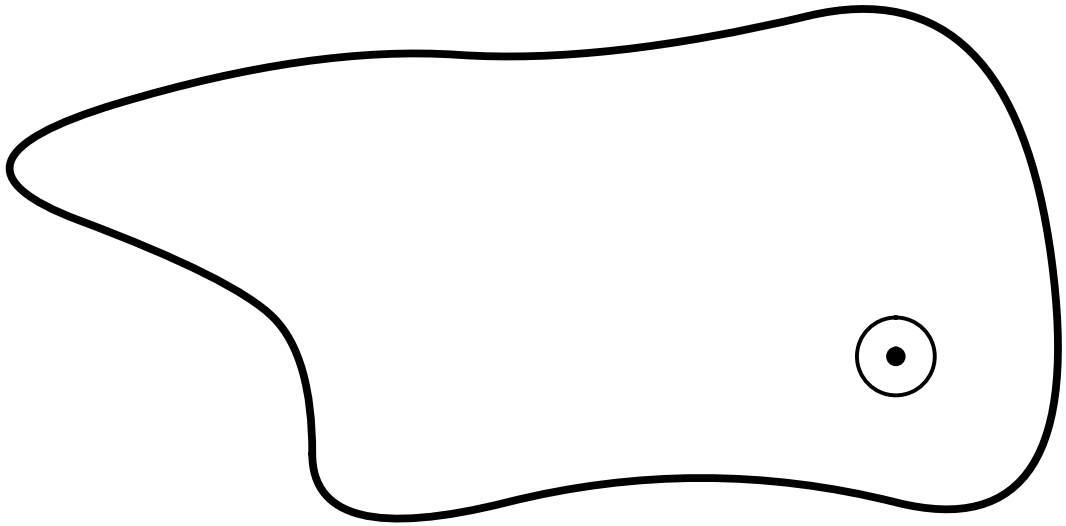
Local search – An example



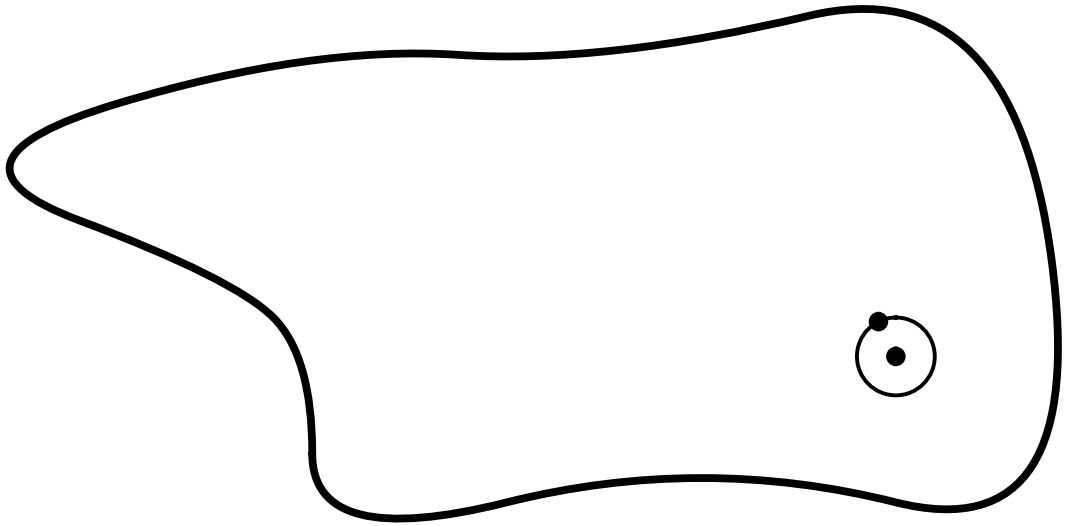
Local search – An example



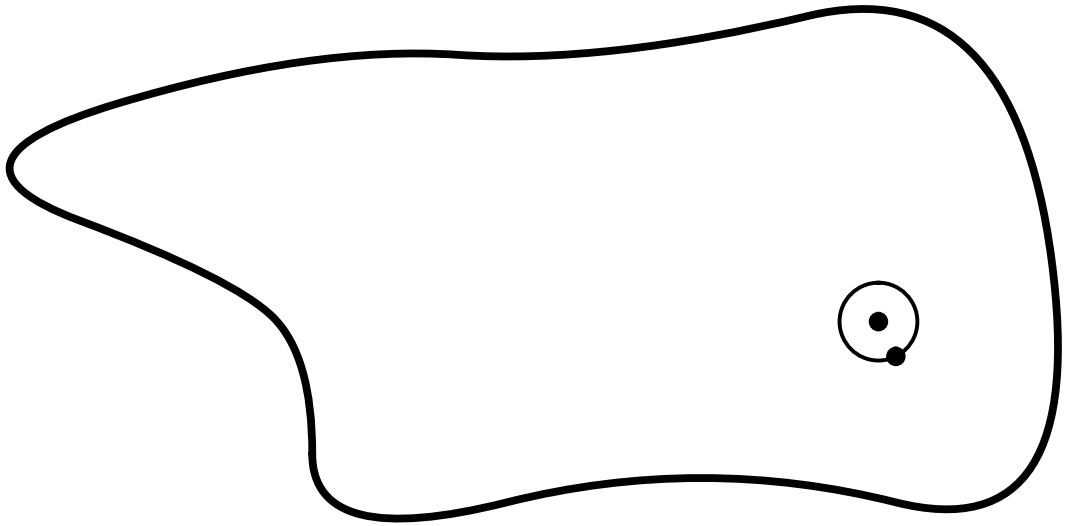
Local search – An example



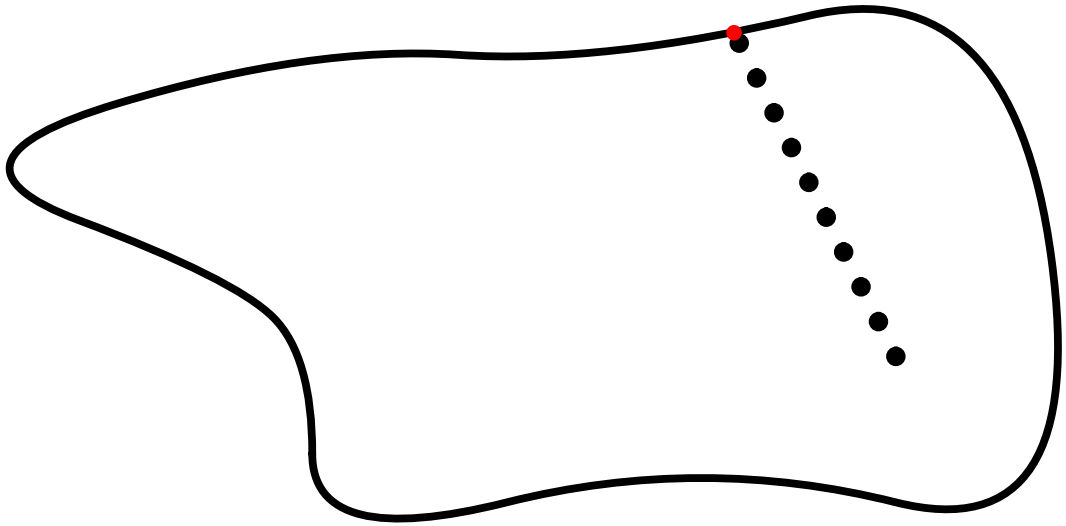
Local search – An example



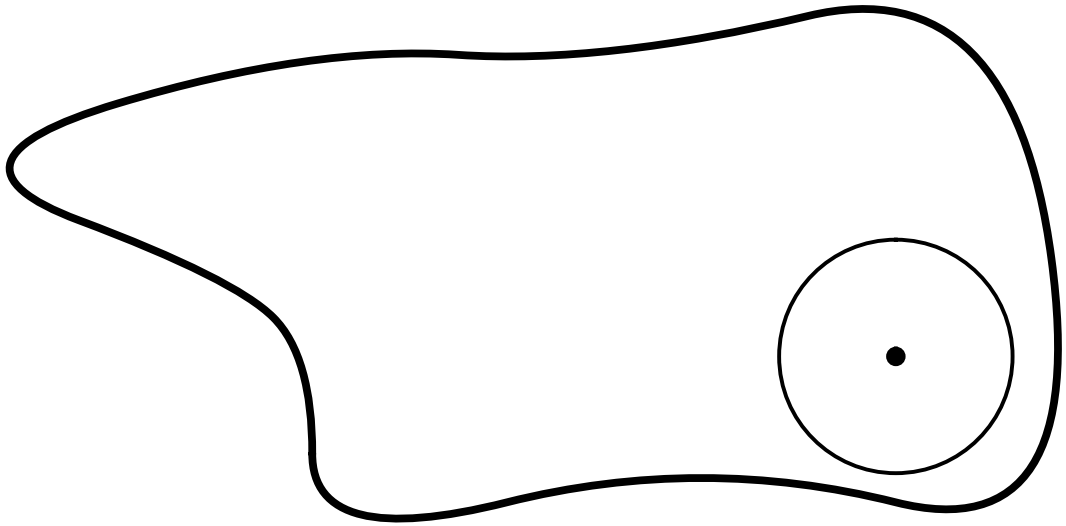
Local search – An example



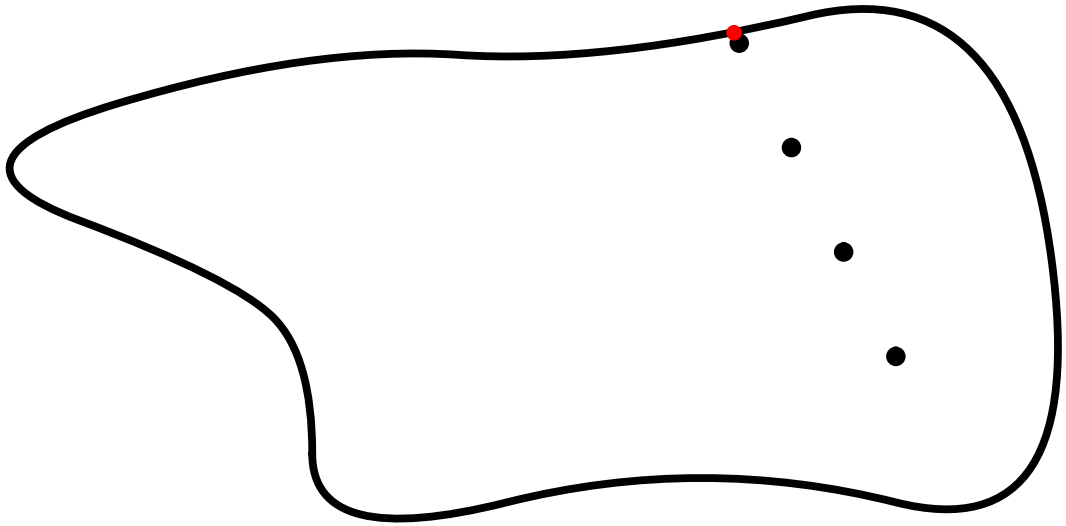
Local search – An example



Local search – An example



Local search – An example



Complexity of Algorithms

An algorithm needs two important resources to solve a problem: time and space. The time complexity is the number of steps to solve a problem of size n. (generally defined as a worst case analysis)

Complexity of Algorithms

An algorithm needs two important resources to solve a problem: time and space. The time complexity is the number of steps to solve a problem of size n. (generally defined as a worst case analysis)

Definition 1.2 Big$-\mathcal{O}$ notation.

An algorithm has a complexity $f(n) = \mathcal{O}(g(n))$ if there exits positive constants n_0 and c such that $\forall n > n_0$, $f(n) \leqslant c \cdot g(n)$.

Complexity of Algorithms

Definition 1.3 Polynomial time algorithm.

An algorithm is a polynomial time algorithm if its complexity is $\mathcal{O}(p(n))$, where $p(n)$ is a polynomial function of n.

Definition 1.4 Exponential time algorithm.

An algorithm is an exponential time algorithm if its complexity is $\mathcal{O}(c^n)$, where c is a real constant strictly superior to 1.

Complexity of Algorithms

Definition 1.5 Big$-\Omega$ notation.

An algorithm has a complexity $f(n) = \Omega(g(n))$ if there exits positive constants n_0 and c such that $\forall n > n_0$, $f(n) \geqslant c \cdot g(n)$. The complexity of the algorithm $f(n)$ is lower bounded by the function $g(n)$.

Definition 1.6 Big$-\Theta$ notation.

An algorithm has a complexity $f(n) = \Theta(g(n))$ if there exits positive constants n_0, c_1 and c_2 such that $\forall n > n_0$,
$c_1 \cdot g(n) \leqslant f(n) \leqslant c_2 \cdot g(n)$. The complexity of the algorithm $f(n)$ is lower (and upper) bounded by the function $g(n)$.

Complexity and Computation time

Complexity	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
$\mathcal{O}(x)$	0.00001 s	0.00002 s	0.00003 s	0.00004 s	0.00005 s
$\mathcal{O}(x^2)$	0.0001 s	0.0004 s	0.0009 s	0.0016 s	0.0025 s
$\mathcal{O}(x^5)$	0.1 s	0.32 s	24.3 s	1.7 min	5.2 min
$\mathcal{O}(2^x)$	0.001 s	1 s	17.9 min	12.7 days	35.7 years
$\mathcal{O}(3^x)$	0.059 s	58.0 min	6.5 years	3855 centuries	$2{\cdot}10^8$ centuries

Table 1.3 Search time of an algorithm as a function of the problem size and complexity.

Complexity of Problems

The complexity of a problem is equivalent to the complexity of the best algorithm solving that problem. A problem is tractable (or "easy") if there exists a polynomial time algorithm to solve it, or intractable otherwise.

Complexity Classes of Problems

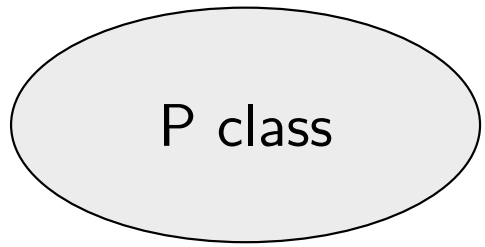


Figure 1.6 Complexity classes

A (decision) problem of class P can be solved by a deterministic machine in polynomial time.

Complexity Classes of Problems

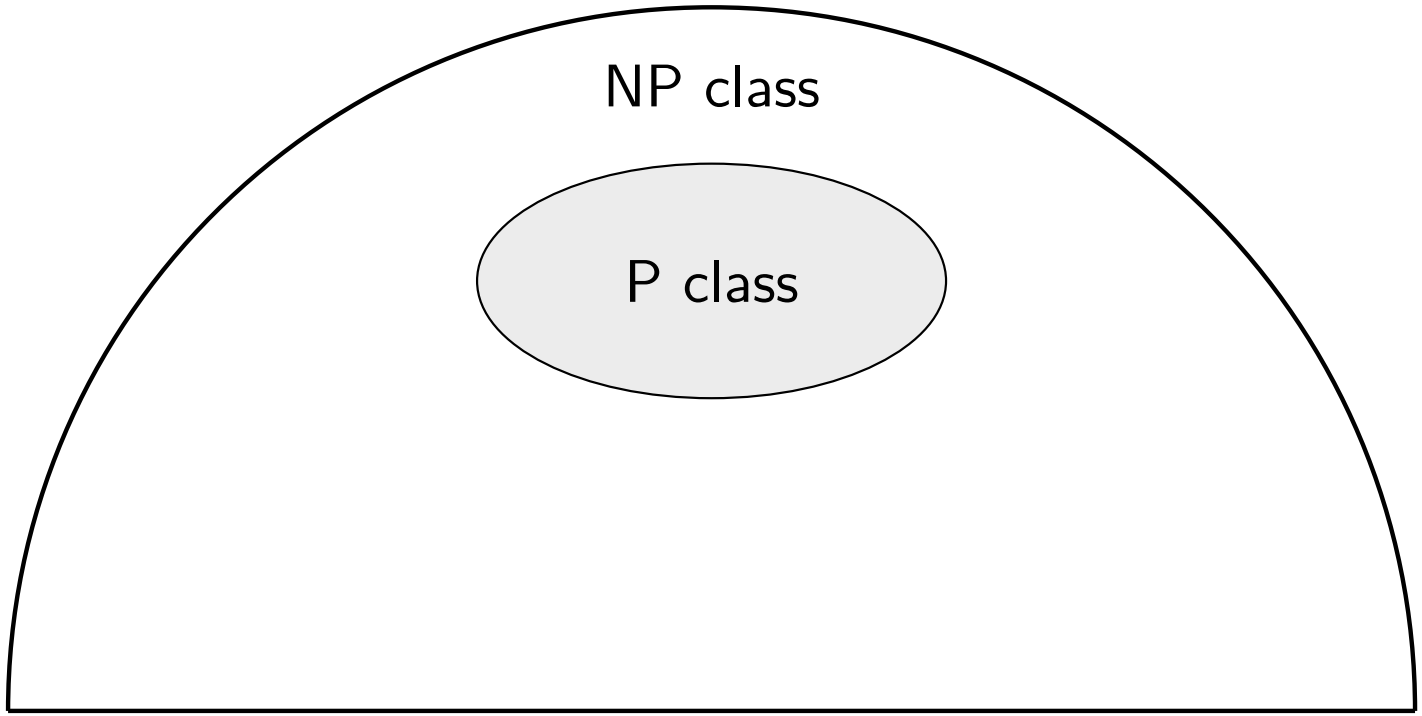


Figure 1.6 Complexity classes

A (decision) problem of class NP can be solved by a non deterministic machine in polynomial time.

Complexity Classes of Problems

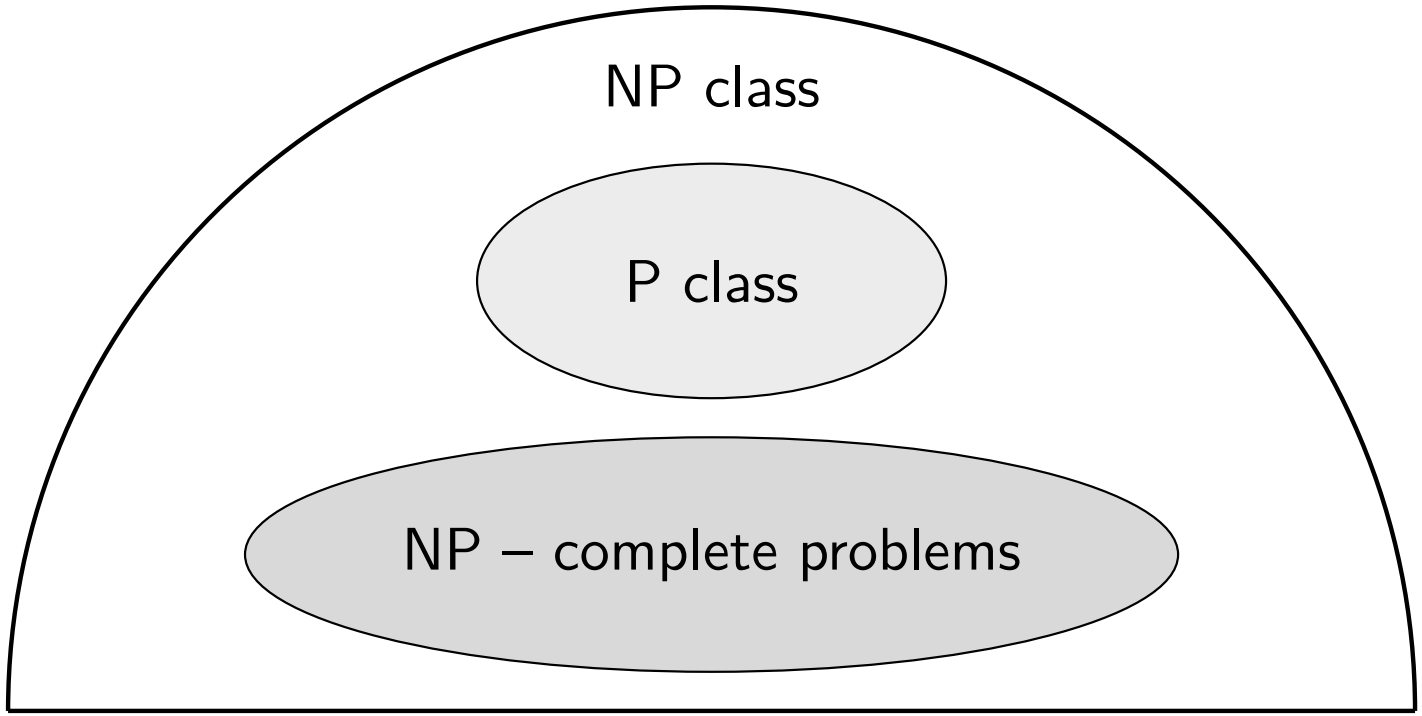


Figure 1.6 Complexity classes

A (decision) problem A of class NP–complete if *all* other problems of class NP are reduced polynomially to the problem A. An optimization problem associate with A is NP–hard.

Some examples

P–problems

NP–hard problems

Some examples

P–problems

- Minimum spanning tree
- Shortest path problems
- Maximum flow network
- Maximum bipartite matching
- Linear programming continuous models

NP–hard problems

Some examples

P–problems

- Minimum spanning tree
- Shortest path problems
- Maximum flow network
- Maximum bipartite matching
- Linear programming continuous models

NP–hard problems

- Sequencing and scheduling problems
- Assignment and location problems
- Grouping problems
- Routing and covering problems
- Knapsack and packing/cutting problems

Outline

LINKÖPING UNIVERSITY

Optimization under uncertainty

Characterized by the input data are subject to noise. A noisy objective function can be defined mathematically as follows:

$$f_{noisy}(x) = \int_{-\infty}^{+\infty} [f(x) + z] p(z) \mathrm{d}z$$

Optimization under uncertainty

Characterized by the input data are subject to noise. A noisy objective function can be defined mathematically as follows:

$$f_{noisy}(x) = \int_{-\infty}^{+\infty} [f(x) + z] p(z) \mathrm{d}z$$

Examples can be found in e.g. routing and scheduling problems. For instance, stochastic demand, stochastic transportation times or uncertainty in processing times.

Optimization under uncertainty

Characterized by the input data are subject to noise. A noisy objective function can be defined mathematically as follows:

$$f_{noisy}(x) = \int_{-\infty}^{+\infty} [f(x) + z]p(z)\mathrm{d}z$$

Examples can be found in e.g. routing and scheduling problems. For instance, stochastic demand, stochastic transportation times or uncertainty in processing times.
The simplest approach to handle uncertainty is to estimate the mean value of each parameter and solve a deterministic problem.

Dynamic optimization

Characterized by the input data are changing over time. The objective function is deterministic at a given time, but varies over the time:

$$f_{dynamic}(x) = f_t(x)$$

Dynamic optimization

Characterized by the input data are changing over time. The objective function is deterministic at a given time, but varies over the time:

$$f_{dynamic}(x) = f_t(x)$$

Examples can be found in e.g. dynamic routing problems. For instance, new entering customer orders or changes in the traffic situation (morning rush hour compared to the night).

Dynamic optimization

Characterized by the input data are changing over time. The objective function is deterministic at a given time, but varies over the time:

$$f_{dynamic}(x) = f_t(x)$$

Examples can be found in e.g. dynamic routing problems. For instance, new entering customer orders or changes in the traffic situation (morning rush hour compared to the night).
The key of solving dynamic problems is to detect the changes and respond to them. What information should be memorized and how to use that information.

Robust optimization

Characterized by how "well" a final solution may handle perturbations of the decision variables. The most used measure is to optimize the expected objective function given a probability distribution of the variation.

$$f_{robust}(x) = \int_{-\infty}^{+\infty} f(x+\delta)p(\delta)\mathrm{d}\delta$$

Robust optimization

Characterized by how "well" a final solution may handle perturbations of the decision variables. The most used measure is to optimize the expected objective function given a probability distribution of the variation.

$$f_{robust}(x) = \int_{-\infty}^{+\infty} f(x+\delta)p(\delta)\mathrm{d}\delta$$

Examples can be found in e.g. time table problem. For instance, how well will the time table handle delays.

Robust optimization

Characterized by how "well" a final solution may handle perturbations of the decision variables. The most used measure is to optimize the expected objective function given a probability distribution of the variation.

$$f_{robust}(x) = \int_{-\infty}^{+\infty} f(x+\delta)p(\delta)\mathrm{d}\delta$$

Examples can be found in e.g. time table problem. For instance, how well will the time table handle delays.
Robust optimization has to find a trade off between the quality of solutions and their robustness in terms of decision variable disturbance.

Outline

LINKÖPING UNIVERSITY

Classical Optimization Methods

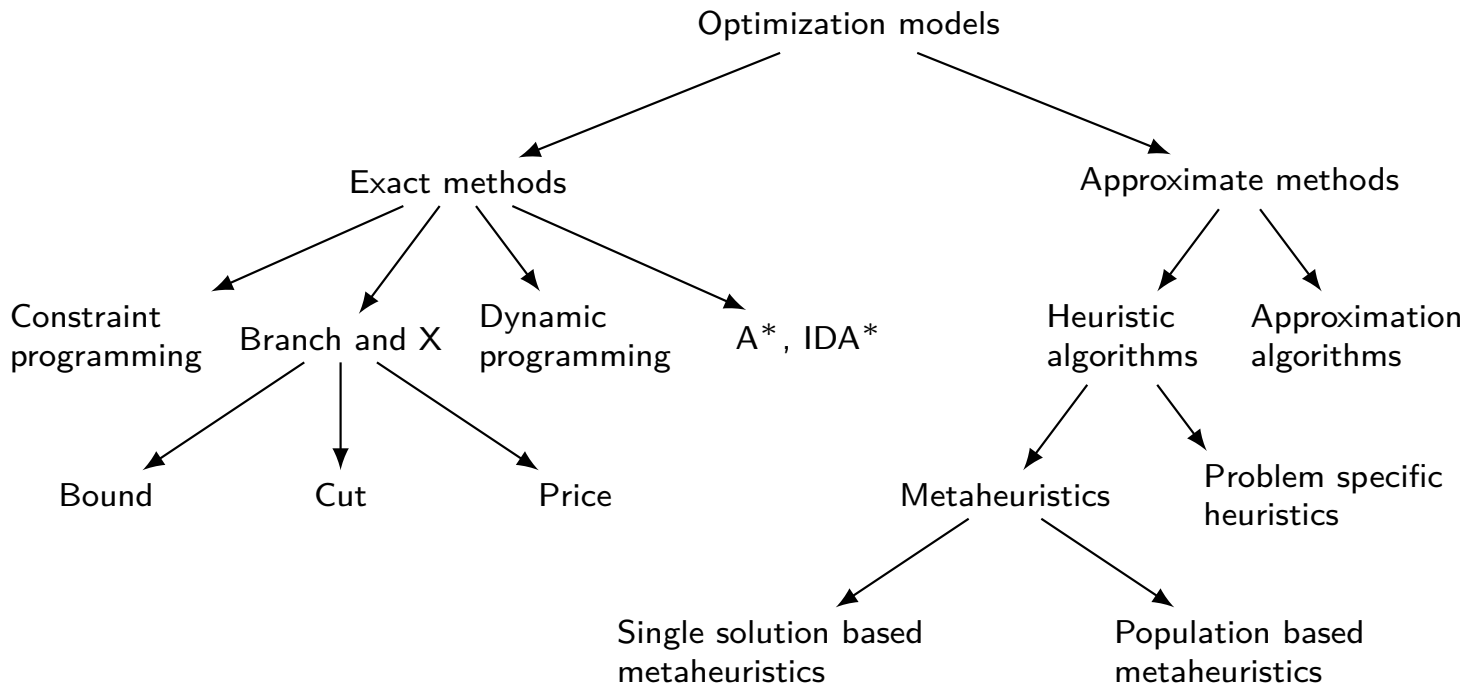


Figure 1.7 Classical optimization methods.

Exact Methods

Exact methods ensures global optimum and is good for small instances of difficult (NP–hard) problems.

Optimization Problems	Quadratic Assignment	Flow Shop Scheduling	Graph Coloring	Capacitated Vehicle Routing
Size of the instance	30 objects	100 jobs 20machines	100 nodes	60 clients

Table 1.4 Order of magnitude of maximal size of instances that state of the art exact methods solve to optimality.

Approximation algorithms

Approximation algorithms provides provable solution quality and run time bounds.

Definiton 1.7 $\varepsilon-$Approximation algorithms

An algorithm has an approximation factor ε if its time complexity is polynomial and for any input instance it produces a solution a such that

$$a \leqslant \varepsilon \cdot s \quad \text{if } \varepsilon > 1$$
$$a \geqslant \varepsilon \cdot s \quad \text{if } \varepsilon < 1$$

where s is the global optimal solution, and the factor ε defines the relative performace guarantee.

Metaheuristics

Unlike exact methods, metaheuristics allow to tackle large size problem instances by delivering satisfactory solutions in reasonable time. There is no guarantee to find global optimal solutions. In designing a metaheuristic, two contradictory criteria must be taken into account: exploration of the search space (diversification) and exploitation of the best solutions found (intensification).

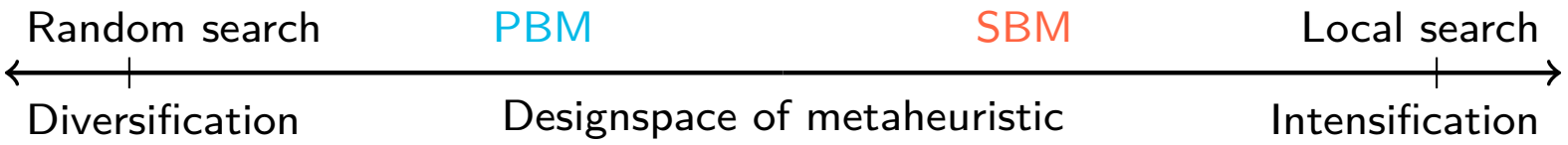


Figure 1.9 Two conflicting criteria
PBM – Population based metaheuristics
SBM – Single solution based metaheuristics

Types of metaheuristics

- Nature inspired: Evolutionary algorithms, Simulated annealing.

Types of metaheuristics

- Nature inspired: Evolutionary algorithms, Simulated annealing.
- No memory usage: Local search, GRASP, Simulated annealing.
- Memory usage: Tabu search.

Types of metaheuristics

- Nature inspired: Evolutionary algorithms, Simulated annealing.
- No memory usage: Local search, GRASP, Simulated annealing.
- Memory usage: Tabu search.
- Deterministic: Local search, Tabu search.
- Stochastic: Simulated annealing, evolutionary algorithms.

Types of metaheuristics

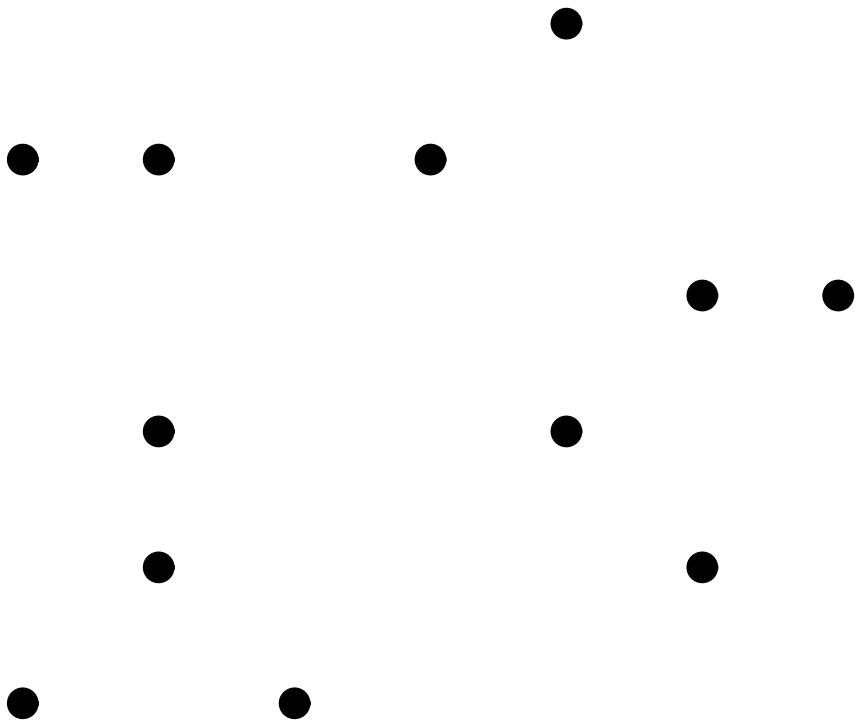
- Nature inspired: Evolutionary algorithms, Simulated annealing.
- No memory usage: Local search, GRASP, Simulated annealing.
- Memory usage: Tabu search.
- Deterministic: Local search, Tabu search.
- Stochastic: Simulated annealing, evolutionary algorithms.
- Population: Evolutionary algorithms.
- Single solution: Local search, Simulated annealing.

Types of metaheuristics

- Nature inspired: Evolutionary algorithms, Simulated annealing.
- No memory usage: Local search, GRASP, Simulated annealing.
- Memory usage: Tabu search.
- Deterministic: Local search, Tabu search.
- Stochastic: Simulated annealing, evolutionary algorithms.
- Population: Evolutionary algorithms.
- Single solution: Local search, Simulated annealing.
- Iterative: Starts with a (or a set of) complete solutions and transform it over time.
- Greedy: Starts with an empty solution and a complete solution is built over time.

Greedy Algorithms or Constructive Algorithms

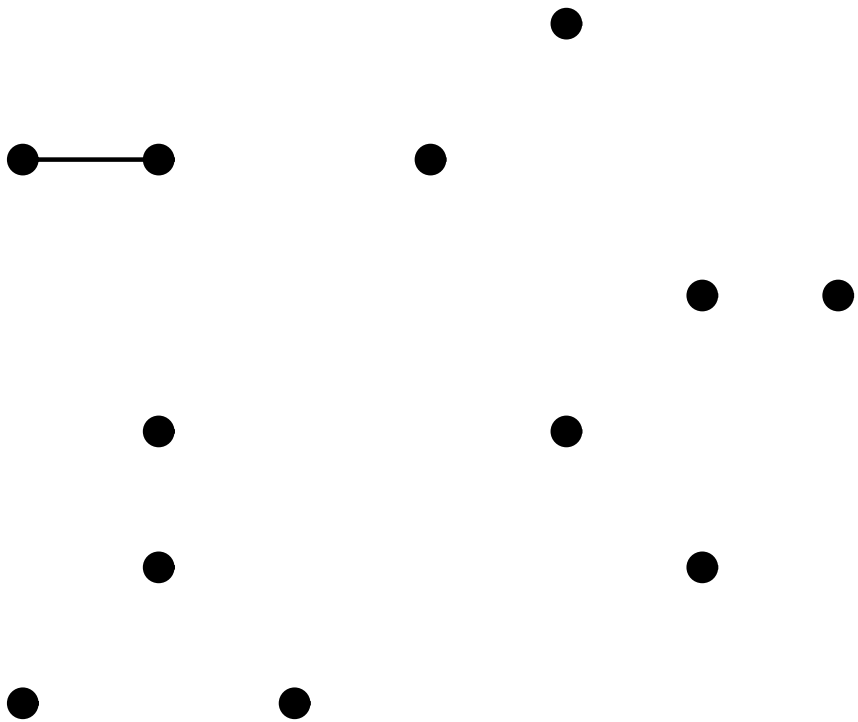
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

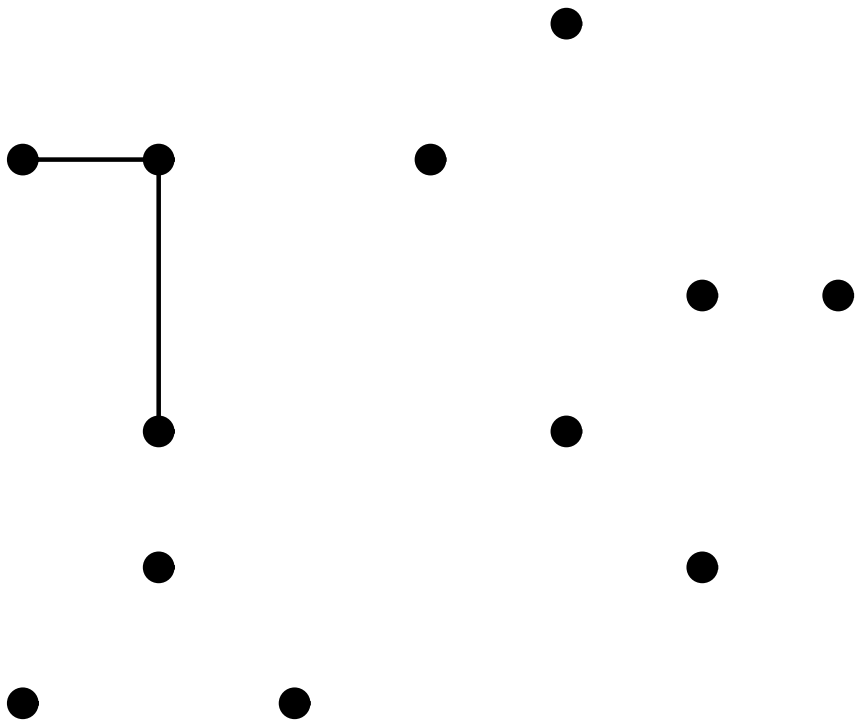
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

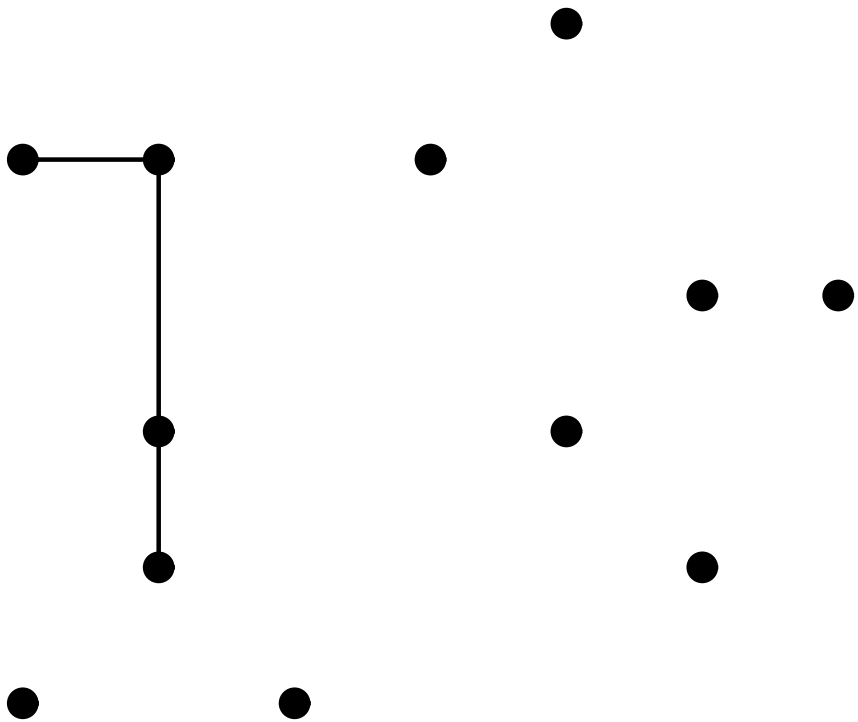
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

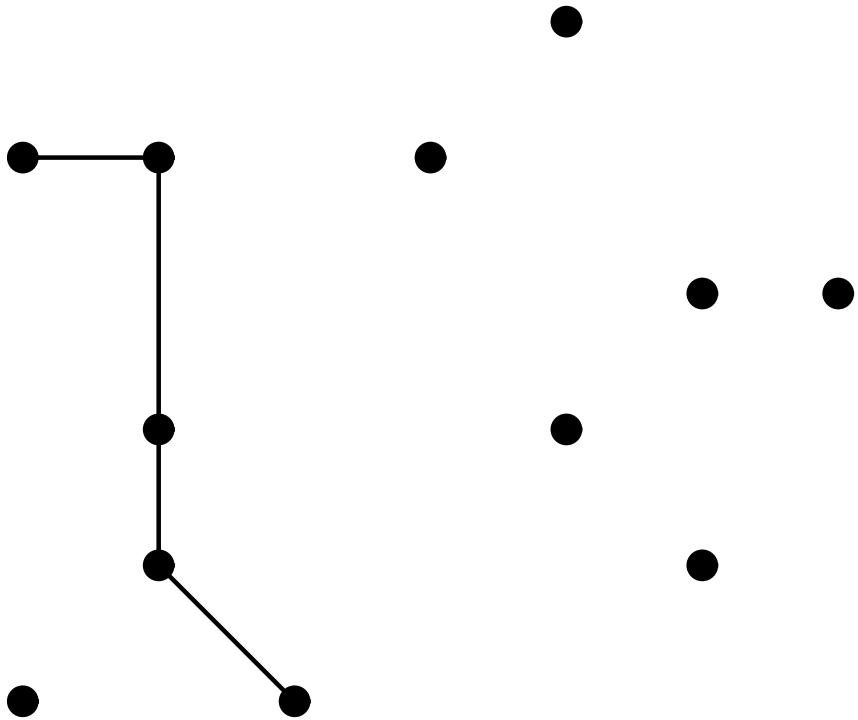
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

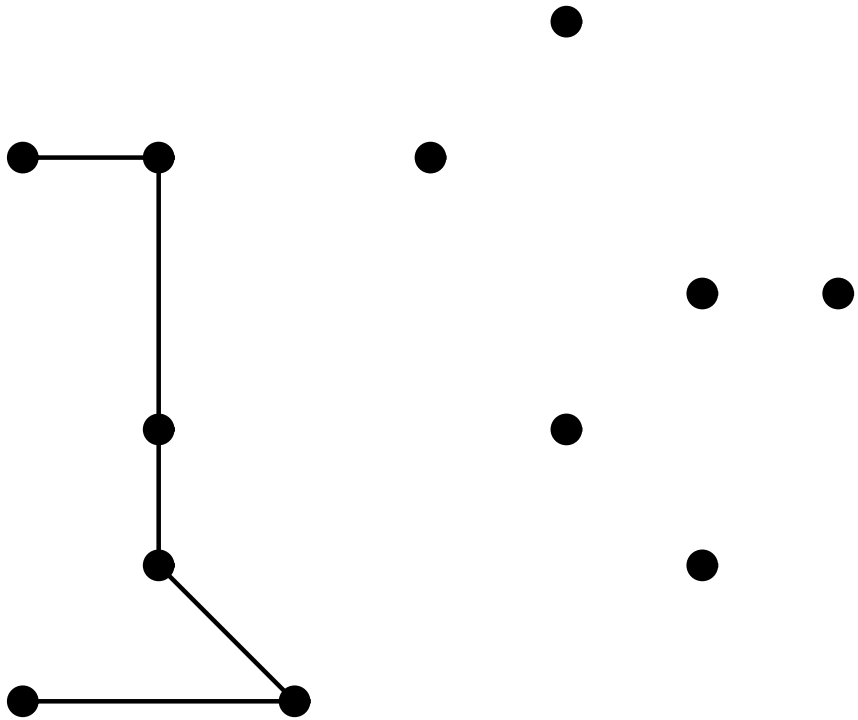
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

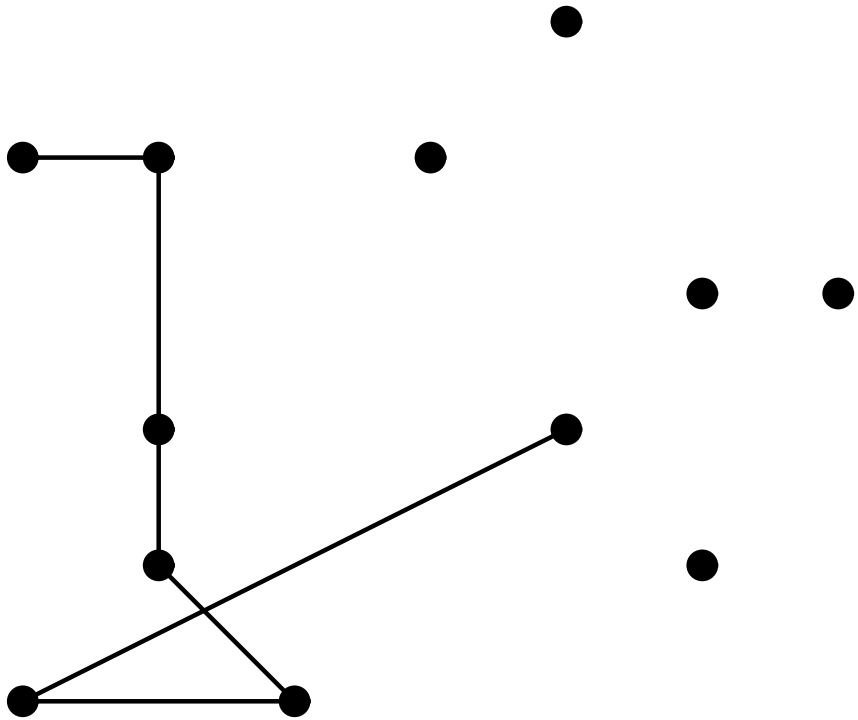
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

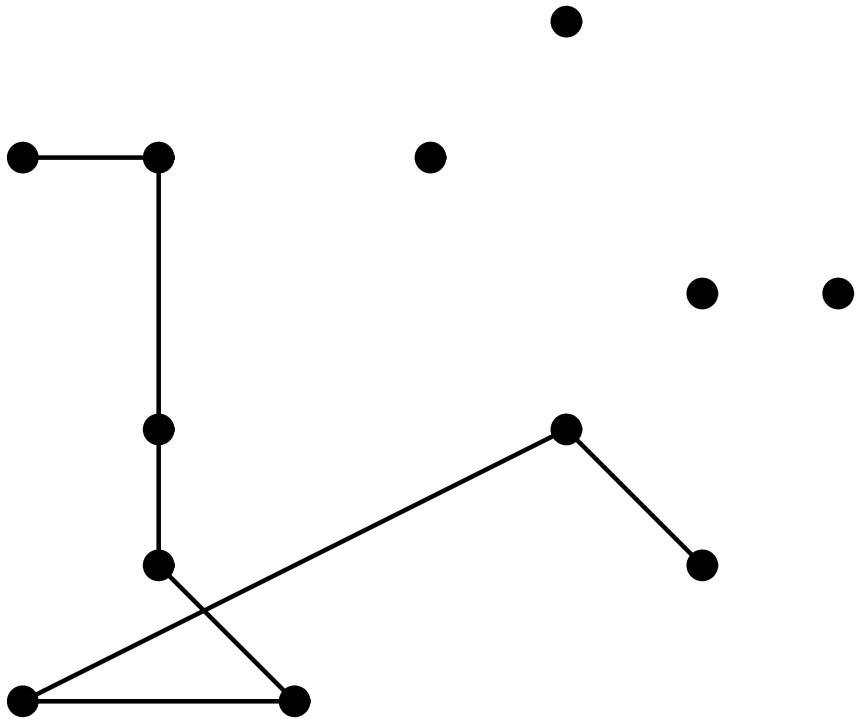
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

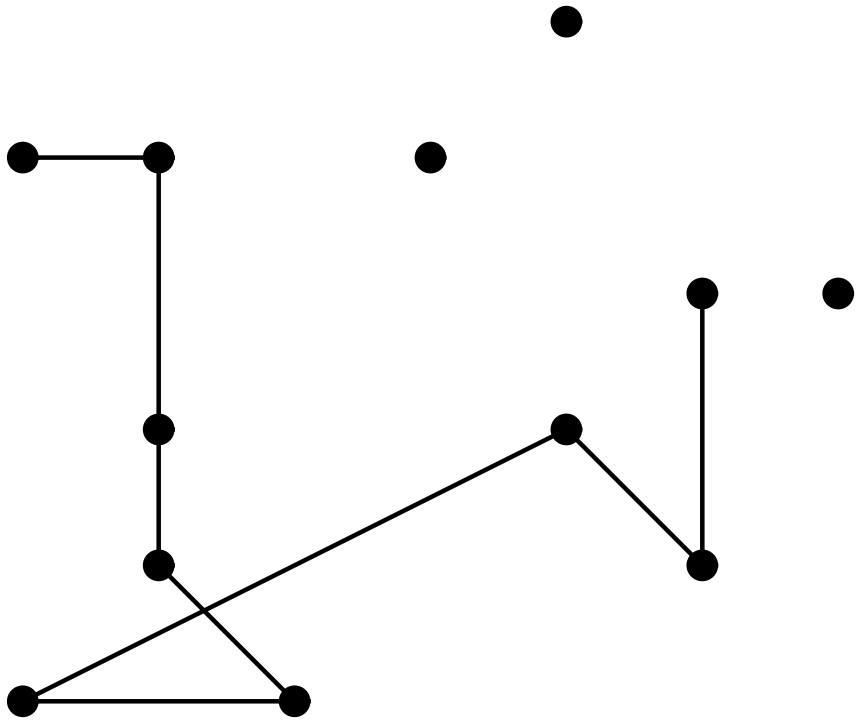
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

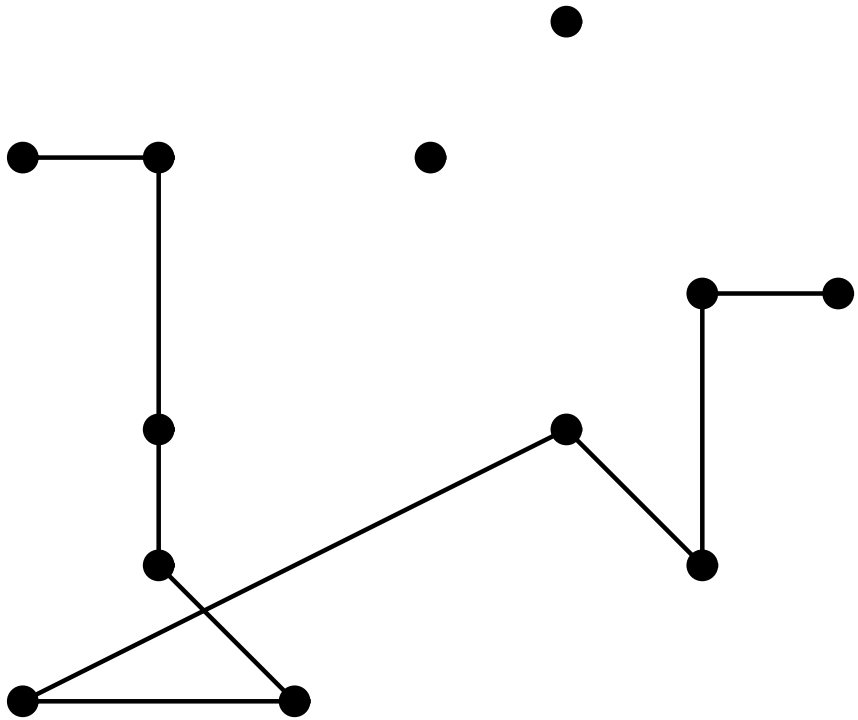
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

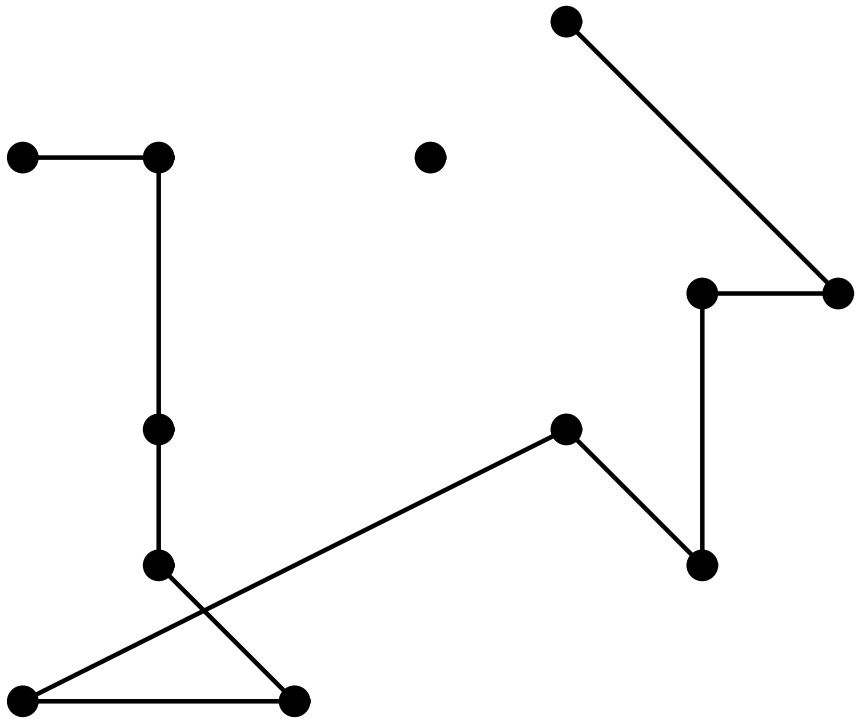
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

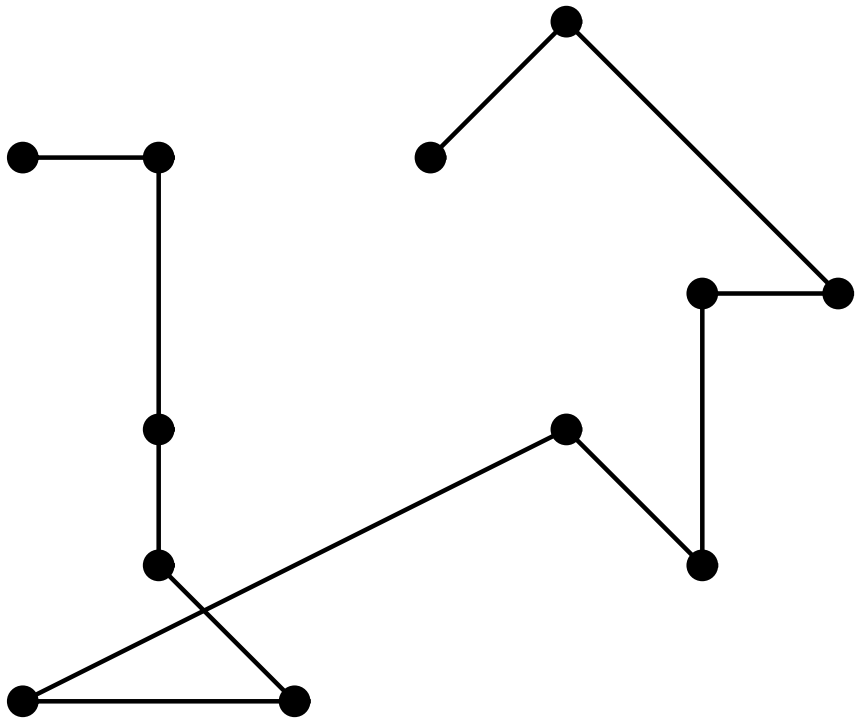
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

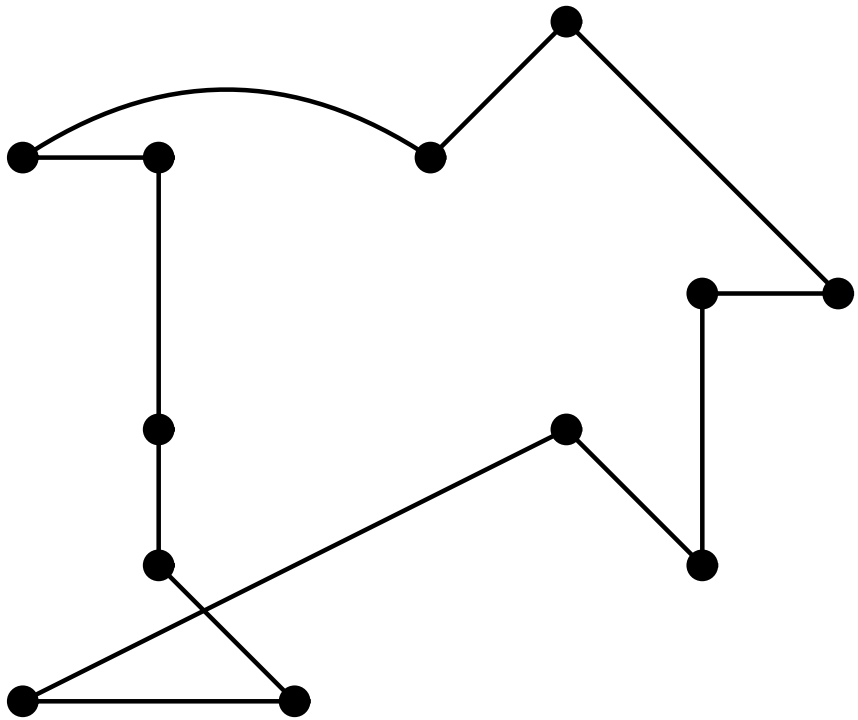
Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

Greedy Algorithms or Constructive Algorithms

Example of nearest neighbour search solving a TSP, Lundgren et al. (2003)



Lundgren, J., Rönnqvist, M., & Värbrand, P. (2003). Optimeringslära, Studentlitteratur.

When using Metaheuristics?

In general

Three important aspects are:

When using Metaheuristics?

In general

Three important aspects are:

- The complexity of the problem.

When using Metaheuristics?

In general

Three important aspects are:

- The complexity of the problem.
- The size of the input instance.

When using Metaheuristics?

In general

Three important aspects are:

- The complexity of the problem.
- The size of the input instance.
- The structure of the instance.

When using Metaheuristics?

In general

Three important aspects are:

- The complexity of the problem.
- The size of the input instance.
- The structure of the instance.

So, analyze the complexity of the problem and see if the problem can be reduced to a classical problem already solved in the literature, then look at the state of the art best known optimization algorithms solving that problem.

When using Metaheuristics?

A summation

- An easy problem (P class) with very large instances.
- An easy problem (P class) with hard real time constraints.
- A difficult problem (NP–hard class) with moderate size and/or difficult structures.
- Optimization problems with time consuming objective function(s) and/or constraints.
- Non analytic models.
- Non deterministic models: uncertainty, robust or noisy problems.

Outline

LINKÖPING UNIVERSITY

All iterative metaheuristics needs a representation of solutions handled by the algorithm and a definition of the objective function.

Representation

All iterative metaheuristics needs a representation of solutions handled by the algorithm and a definition of the objective function.

The representation must have the following characteristics:

- **Completeness:** All solutions of the problem must be represented.
- **Connexity:** A search path must exist between any two solutions.
- **Efficiency:** The representation must be easy to manipulate by the algorithm.

Representation

All iterative metaheuristics needs a representation of solutions handled by the algorithm and a definition of the objective function.

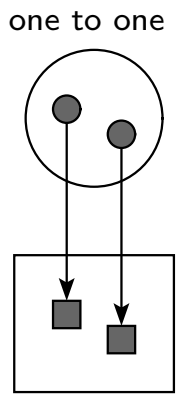
Some common representations of classical problems are:

- **Binary encoding:** 1001001101111, Knapsack problems
- **Vector of discrete values:** $(3, 5, 1, 7, 3)$, Assignment problems
- **Vector of real values:** $(2.5, 4.5, 1.0, 1.7, 3.3)$, Cont. opt.
- **Permutation:** A–D–B–C–G–F–I–H–E, TSP

Mapping between Solutions and Representations

- **one to one:** Traditional, a vector of variable values.

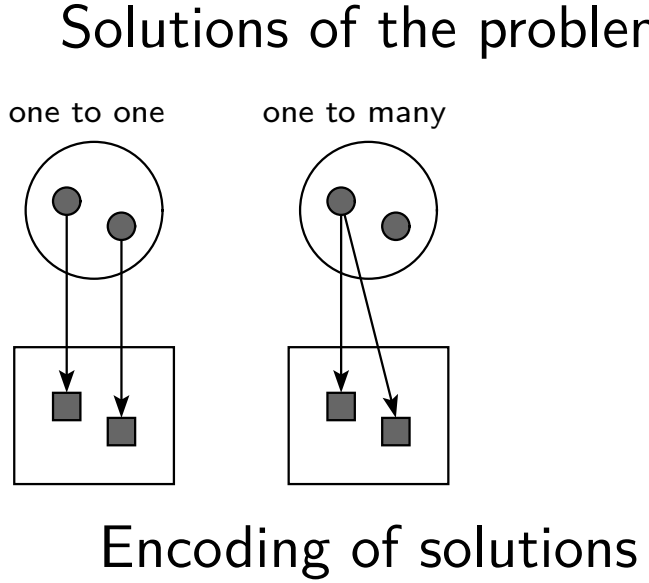
Solutions of the problem



Encoding of solutions

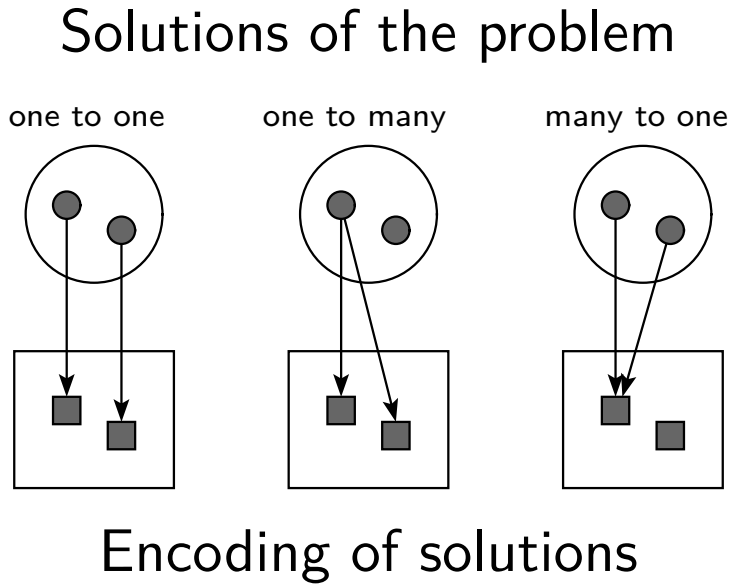
Mapping between Solutions and Representations

- **one to one:** Traditional, a vector of variable values.
- **one to many:** Permutations, TSP.



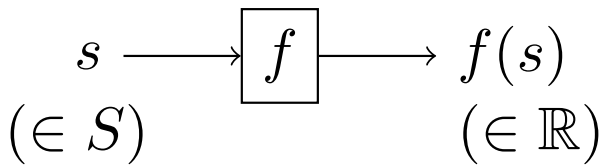
Mapping between Solutions and Representations

- **one to one:** Traditional, a vector of variable values.
- **one to many:** Permutations, TSP.
- **many to one:** Indirect encoding, can be used due to symmetry?



objective function

The objective function formulates the goal to achieve by evaluating solutions, $f : S \to \mathbb{R}$.



The objective function can be analytic, e.g. $f(x) = cx$. But it might be more difficult to evaluate, e.g. x is input data to a simulation and the simulation output is the objective function, or even subjective. Most of the time, in meta heuristics, the intensive part is the evaluation of the objective function.

Thank you for listening!

www.liu.se

LINKÖPING UNIVERSITY