Introduction to Metaheuristics

Leandro Montero

UCO Angers

2025-2026

based on lectures of Fabien Tricoire

Outline

Outline

Objectives

Once this course is validated, you should:

- Know what is a metaheuristic
- Know why the world needs metaheuristics
- Be able to evaluate the quality of a metaheuristic
- Know how to design a metaheuristic for a given problem

During the course

- Several useful books, e.g.:
  - Handbook of Metaheuristics, Fred Glover & Gary A. Kochenberger, Kluwer's International Series, ISBN 1-4020-7263-5
  - Stochastic Local Search, Foundations and Applications, Holger H. Hoos & Thomas Stützle, Elsevier, ISBN 1-55860-872-9
  - Search Methodologies, Introductory Tutorials in Optimization and Decision Support Techniques, Edmund K. Burke & Graham Kendall, Springer, ISBN 0-387-23460-8
- Main source of information: these slides (you will get them)
- Interactive lectures
- Discussions
- Exercises

Outline

Hard Optimisation Problems

Imagine a problem with. . .

- Some operational constraints
  - Example: visit all customers
- At least one optimal solution
  - Optimise = Minimise or Maximise an Objective function (a.k.a. Fitness)
  - Example: Minimise the sum of operational costs

Hard Optimisation Problems

Imagine a problem with...

- Some operational constraints
  - Example: visit all customers
- At least one optimal solution
  - Optimise = Minimise or Maximise an Objective function (a.k.a. Fitness)
  - Example: Minimise the sum of operational costs

Such that finding this optimal solution is too hard...

- Even with the best programmer in the world
- Even with the best programming language
- Even with the best operating system
- Even with the fastest computer
- Even 20 years in the future

Such problems exist!

The Travelling Salesperson Problem (TSP)

A Travelling Salesperson wants to sell their goods to customers. . .

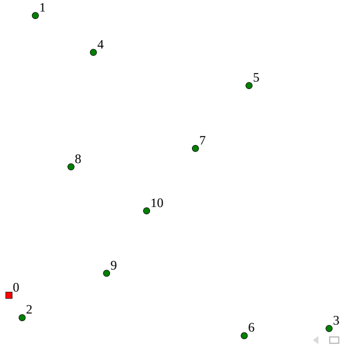
- Given data: distances using the road network
- Constraints: starting from a given depot, visit all n customers
- Decision: choose visiting order
- Objective: minimise total distance

The Travelling Salesperson Problem (TSP)

A Travelling Salesperson wants to sell their goods to customers. . .

- Given data: distances using the road network
- Constraints: starting from a given depot, visit all n customers
- Decision: choose visiting order
- Objective: minimise total distance

An example!

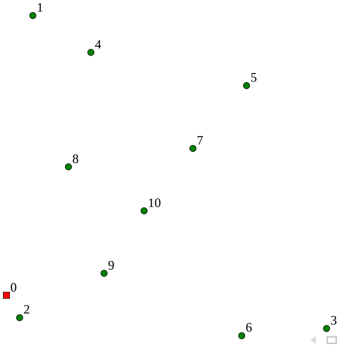


The Travelling Salesperson Problem (TSP)

A Travelling Salesperson wants to sell their goods to customers...

- Given data: distances using the road network
- Constraints: starting from a given depot, visit all n customers
- Decision: choose visiting order
- Objective: minimise total distance

An example! Basic approach: enumerate all solutions

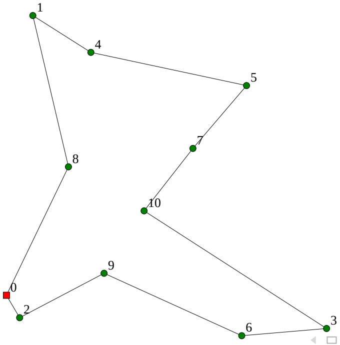


The Travelling Salesperson Problem (TSP)

A Travelling Salesperson wants to sell their goods to customers. . .

- Given data: distances using the road network
- Constraints: starting from a given depot, visit all *n* customers
- Decision: choose visiting order
- Objective: minimise total distance

An example! Basic approach: enumerate all solutions

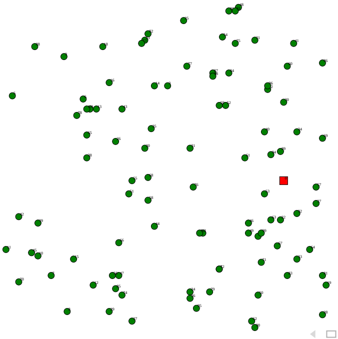


The Travelling Salesperson Problem (TSP)

A Travelling Salesperson wants to sell their goods to customers...

- Given data: distances using the road network
- Constraints: starting from a given depot, visit all *n* customers
- Decision: choose visiting order
- Objective: minimise total distance

<u>Until which size can a computer enumerate all solutions?</u>



The Travelling Salesperson Problem (TSP)

Exercise: how many possible ways to visit n customers?

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!

The Travelling Salesperson Problem (TSP)

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!
- Number of solutions $=$ number of permutations $= n!$
  ($\frac{n!}{2}$ if the distances are symmetric)

The Travelling Salesperson Problem (TSP)

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!
- Number of solutions = number of permutations = $n!$
  ($\frac{n!}{2}$ if the distances are symmetric)

Suppose our recent computer can process 2,000,000 solutions per second...

The Travelling Salesperson Problem (TSP)

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!
- Number of solutions = number of permutations = $n!$
  ($\frac{n!}{2}$ if the distances are symmetric)

Suppose our recent computer can process 2,000,000 solutions per second. . .

- 10 customers: 1.8 s

The Travelling Salesperson Problem (TSP)

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!
- Number of solutions = number of permutations = $n!$
  ($\frac{n!}{2}$ if the distances are symmetric)

Suppose our recent computer can process 2,000,000 solutions per second...

- 10 customers: 1.8 s
- 12 customers: 4 m

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!
- Number of solutions = number of permutations = $n!$
  ($\frac{n!}{2}$ if the distances are symmetric)

Suppose our recent computer can process 2,000,000 solutions per second. . .

- 10 customers: 1.8 s
- 12 customers: 4 m
- 15 customers: 7.5 d

The Travelling Salesperson Problem (TSP)

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!
- Number of solutions = number of permutations = $n!$
  ($\frac{n!}{2}$ if the distances are symmetric)

Suppose our recent computer can process 2,000,000 solutions per second. . .

- 10 customers: 1.8 s
- 12 customers: 4 m
- 15 customers: 7.5 d
- 20 customers: 38,547 years

The Travelling Salesperson Problem (TSP)

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!
- Number of solutions = number of permutations = $n!$
  ($\frac{n!}{2}$ if the distances are symmetric)

Suppose our recent computer can process 2,000,000 solutions per second...

- 10 customers: 1.8 s
- 12 customers: 4 m
- 15 customers: 7.5 d
- 20 customers: 38,547 years
- 99 customers: $1.48 \cdot 10^{142}$ years! (Universe is $1.38 \cdot 10^{10}$ y.o.)

The Travelling Salesperson Problem (TSP)

Exercise: how many possible ways to visit n customers?

- A solution is a permutation!
- Number of solutions $=$ number of permutations $= n!$
  ($\frac{n!}{2}$ if the distances are symmetric)

Suppose our recent computer can process 2,000,000 solutions per second...

- 10 customers: 1.8 s
- 12 customers: 4 m
- 15 customers: 7.5 d
- 20 customers: 38,547 years
- 99 customers: $1.48 \cdot 10^{142}$ years! (Universe is $1.38 \cdot 10^{10}$ y.o.)

Now suppose we have a computer a million times faster:

- 20 customers: 2 weeks
- 22 customers: 18 years
- 25 customers: 245,760 years

The Parallel Machines Scheduling Problem

$P||C_{max}$: we want to perform n production jobs, we have m "parallel" identical machines.

The Parallel Machines Scheduling Problem

$P||C_{max}$: we want to perform n production jobs, we have m "parallel" identical machines.

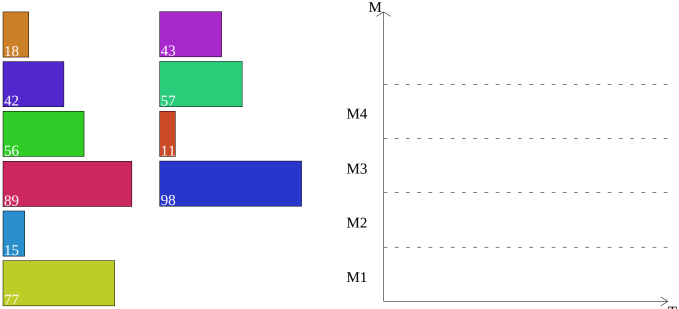
- Given data: for each job, its duration
- Constraint: perform all jobs
- Decision: assign jobs to machines
- Objective: end the last job as early as possible

The Parallel Machines Scheduling Problem

$P||C_{max}$: we want to perform n production jobs, we have m "parallel" identical machines.

- Given data: for each job, its duration
- Constraint: perform all jobs
- Decision: assign jobs to machines
- Objective: end the last job as early as possible

An example!

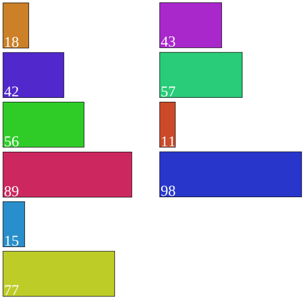


The Parallel Machines Scheduling Problem

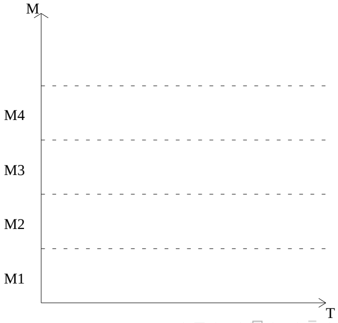
$P||C_{max}$: we want to perform n production jobs, we have m "parallel" identical machines.

- Given data: for each job, its duration
- Constraint: perform all jobs
- Decision: assign jobs to machines
- Objective: end the last job as early as possible

An example!



Basic approach: enumerate all solutions

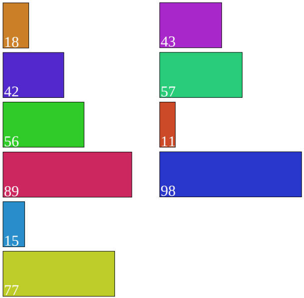


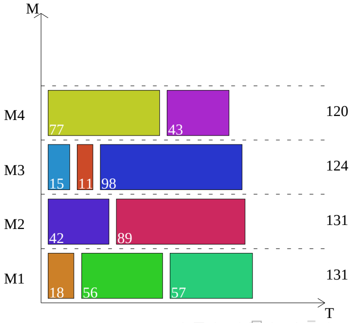
The Parallel Machines Scheduling Problem

$P||C_{max}$: we want to perform n production jobs, we have m "parallel" identical machines.

- Given data: for each job, its duration
- Constraint: perform all jobs
- Decision: assign jobs to machines
- Objective: end the last job as early as possible

An example!   Basic approach: enumerate all solutions

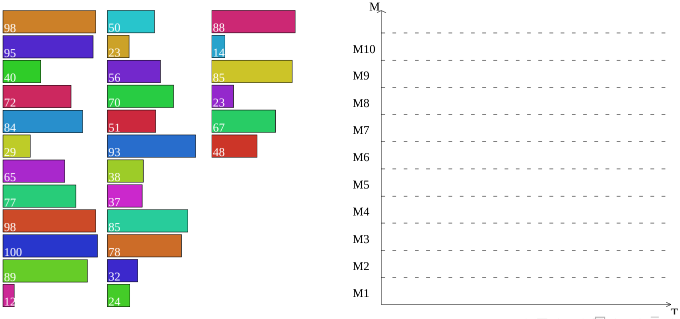




$P||C_{max}$: we want to perform n production jobs, we have m "parallel" identical machines.

- Given data: for each job, its duration
- Constraint: perform all jobs
- Decision: assign jobs to machines
- Objective: end the last job as early as possible

Until which size can a computer enumerate all solutions?



The Parallel Machines Scheduling Problem

Exercise: how many possible solutions? (m machines, n jobs)

Exercise: how many possible solutions? (m machines, n jobs)

- Let C_p be the number of solutions for p jobs

The Parallel Machines Scheduling Problem

Exercise: how many possible solutions? (m machines, n jobs)

- Let C_p be the number of solutions for p jobs
- Each job can be assigned to any of the m machines

The Parallel Machines Scheduling Problem

Exercise: how many possible solutions? (m machines, n jobs)

- Let C_p be the number of solutions for p jobs
- Each job can be assigned to any of the m machines
- Therefore $C_1 = m$

The Parallel Machines Scheduling Problem

Exercise: how many possible solutions? (m machines, n jobs)

- Let C_p be the number of solutions for p jobs
- Each job can be assigned to any of the m machines
- Therefore $C_1 = m$
- Now for all of these m solutions, we can assign a second job to any of the m machines, so $C_2 = m \cdot C_1 = m^2$

The Parallel Machines Scheduling Problem

Exercise: how many possible solutions? (m machines, n jobs)

- Let C_p be the number of solutions for p jobs
- Each job can be assigned to any of the m machines
- Therefore $C_1 = m$
- Now for all of these m solutions, we can assign a second job to any of the m machines, so $C_2 = m \cdot C_1 = m^2$
- In a similar way, $C_n = m \cdot C_{n-1} = m \cdot m \cdot C_{n-2}$

The Parallel Machines Scheduling Problem

Exercise: how many possible solutions? (m machines, n jobs)

- Let C_p be the number of solutions for p jobs
- Each job can be assigned to any of the m machines
- Therefore $C_1 = m$
- Now for all of these m solutions, we can assign a second job to any of the m machines, so $C_2 = m \cdot C_1 = m^2$
- In a similar way, $C_n = m \cdot C_{n-1} = m \cdot m \cdot C_{n-2}$
- Total: $\underbrace{m \cdot m \cdot m \cdot \ldots \cdot m}_{n \text{ times}}$

The Parallel Machines Scheduling Problem

Exercise: how many possible solutions? (m machines, n jobs)

- Let C_p be the number of solutions for p jobs
- Each job can be assigned to any of the m machines
- Therefore $C_1 = m$
- Now for all of these m solutions, we can assign a second job to any of the m machines, so $C_2 = m \cdot C_1 = m^2$
- In a similar way, $C_n = m \cdot C_{n-1} = m \cdot m \cdot C_{n-2}$
- Total: $\underbrace{m \cdot m \cdot m \cdot \ldots \cdot m}_{n \text{ times}} = m^n$

The Parallel Machines Scheduling Problem

Exercise: how many possible solutions? (m machines, n jobs)

- Let C_p be the number of solutions for p jobs
- Each job can be assigned to any of the m machines
- Therefore $C_1 = m$
- Now for all of these m solutions, we can assign a second job to any of the m machines, so $C_2 = m \cdot C_1 = m^2$
- In a similar way, $C_n = m \cdot C_{n-1} = m \cdot m \cdot C_{n-2}$
- Total: $\underbrace{m \cdot m \cdot m \cdot \ldots \cdot m}_{n \text{ times}} = m^n$

Suppose our computer can process 500,000 solutions per second...

m	n	Time
4	10	2 s
5	15	17 h
6	20	232 years
8	25	$2.39 \cdot 10^9$ years
10	30	$6.34 \cdot 10^{16}$ years

Warehouse Location (a.k.a. p-median Problem)

We are a company with m warehouses and n customers. . .

Warehouse Location (a.k.a. p-median Problem)

We are a company with m warehouses and n customers...

- Given data: cost induced by servicing customers through warehouses (e.g. distance)

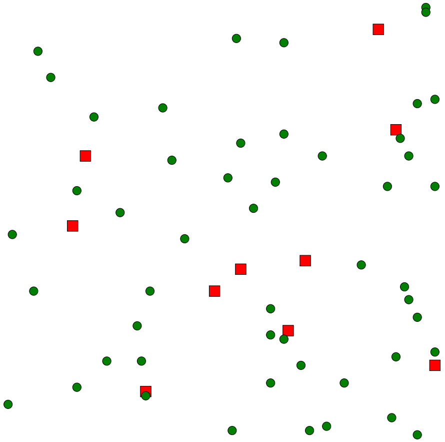
Warehouse Location (a.k.a. p-median Problem)

We are a company with m warehouses and n customers...

- Given data: cost induced by servicing customers through warehouses (e.g. distance)
- Constraint 1: No more than p warehouses may be open
- Constraint 2: Each customer must be serviced by a warehouse
- Decision 1: which warehouses should we open?
- Decision 2: which open warehouse should service which customer?
- Objective: minimise total service cost

Warehouse Location (a.k.a. p-median Problem)

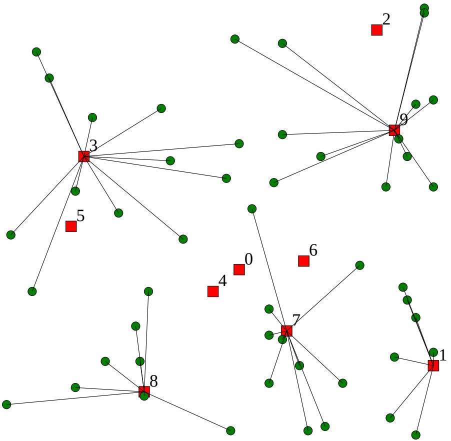
An example! $(m = 10, p = 5)$



Warehouse Location (a.k.a. p-median Problem)

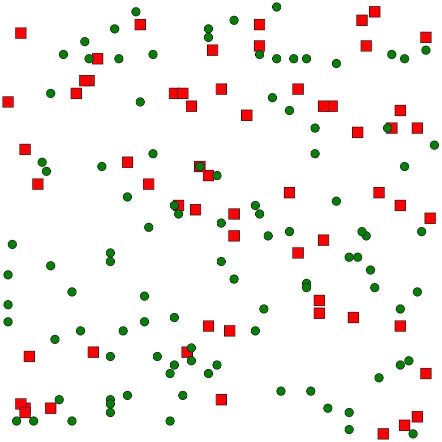
An example! $(m = 10, p = 5)$

Basic approach: enumerate



Warehouse Location (a.k.a. p-median Problem)

Until which size can a computer enumerate all solutions?



Exercise: how many possible solutions?

Warehouse Location

<u>Exercise: how many possible solutions?</u>

- A solution = which depots should be open
- (then we can assign to each customer its closest depot)

Warehouse Location

Exercise: how many possible solutions?

- A solution = which depots should be open
- (then we can assign to each customer its closest depot)
- Number of solutions using exactly p warehouses: $\binom{m}{p}$
- There are solutions using p warehouses,

Warehouse Location

Exercise: how many possible solutions?

- A solution = which depots should be open
- (then we can assign to each customer its closest depot)
- Number of solutions using exactly p warehouses: $\binom{m}{p}$
- There are solutions using p warehouses,
- Some other using $p-1$ warehouses,

Warehouse Location

Exercise: how many possible solutions?

- A solution = which depots should be open
- (then we can assign to each customer its closest depot)
- Number of solutions using exactly p warehouses: $\binom{m}{p}$
- There are solutions using p warehouses,
- Some other using $p - 1$ warehouses, and so on down to 1.

Warehouse Location

Exercise: how many possible solutions?

- A solution = which depots should be open
- (then we can assign to each customer its closest depot)
- Number of solutions using exactly p warehouses: $\binom{m}{p}$
- There are solutions using p warehouses,
- Some other using $p-1$ warehouses, and so on down to 1.
- Total quantity of solutions: $\binom{m}{p} + \binom{m}{p-1} + \ldots + \binom{m}{1}$

Exercise: how many possible solutions?

- A solution = which depots should be open
- (then we can assign to each customer its closest depot)
- Number of solutions using exactly p warehouses: $\binom{m}{p}$
- There are solutions using p warehouses,
- Some other using $p-1$ warehouses, and so on down to 1.
- Total quantity of solutions: $\binom{m}{p} + \binom{m}{p-1} + \ldots + \binom{m}{1}$
- Which is equal to: $\sum_{k=1}^{p} \binom{m}{k}$

Warehouse Location

Suppose our recent computer can process 4,000 solutions per second...

m	p	Time
10	5	0.15 s
20	10	2.5 m
30	20	3 days
40	20	5 years
50	20	904 years
60	30	$5 \cdot 10^6$ years

Warehouse Location

Suppose our recent computer can process 4,000 solutions per second...

m	p	Time
10	5	0.15 s
20	10	2.5 m
30	20	3 days
40	20	5 years
50	20	904 years
60	30	$5 \cdot 10^6$ years

Some insight

- Question: Evaluation is quite time-consuming. Why?

Warehouse Location

Suppose our recent computer can process 4,000 solutions per second. . .

m	p	Time
10	5	0.15 s
20	10	2.5 m
30	20	3 days
40	20	5 years
50	20	904 years
60	30	$5 \cdot 10^6$ years

Some insight

- Question: Evaluation is quite time-consuming. Why?
- Answer: given a solution, finding the best open depot for each customer takes time.

Outline

Algorithms

(personal) Definition

Algorithm: a succession of operations performing data manipulation.

Algorithms

(personal) Definition

Algorithm: a succession of operations performing data manipulation.

These data can be:

- Constants
  - Cannot be modified
  - Typically: problem input

Algorithms

(personal) Definition

Algorithm: a succession of operations performing data manipulation.

These data can be:

- Constants
  - Cannot be modified
  - Typically: problem input
- Variables
  - Can be modified
  - Useful to store computation results

Algorithms

(personal) Definition

Algorithm: a succession of operations performing data manipulation.

These data can be:

- Constants
  - Cannot be modified
  - Typically: problem input
- Variables
  - Can be modified
  - Useful to store computation results
- Both kinds can be arranged into data structures
  - Example 1: Lists
  - Example 2: Arrays

Algorithms: basic operations

Assignment

Definition

Assignment: setting the value of a variable. In this course, the assignment is represented with the following arrow operator: $\leftarrow$

Definition

Assignment: setting the value of a variable. In this course, the assignment is represented with the following arrow operator: $\leftarrow$

Exercise: what is the value of variable z at the end of this algorithm?

$$x \leftarrow 3$$
$$y \leftarrow 4$$
$$z \leftarrow \sqrt{x^2 + y^2}$$

Definition

Assignment: setting the value of a variable. In this course, the assignment is represented with the following arrow operator: $\leftarrow$

Exercise: what is the value of variable z at the end of this algorithm?

$$x \leftarrow 3$$
$$y \leftarrow 4$$
$$z \leftarrow \sqrt{x^2 + y^2}$$

Answer: 5

Performing different operations depending on the context

Conditionals allow to perform parts of an algorithm only if certain conditions are met. In this course, this is achieved with the keywords if, then and else.

Performing different operations depending on the context

Conditionals allow to perform parts of an algorithm only if certain conditions are met. In this course, this is achieved with the keywords if, then and else.

Exercise: what is the value of variable z at the end of this algorithm?

$x \leftarrow 3$
$y \leftarrow 4$
if $x < y$ **then**
  $z \leftarrow 1$
else
  $z \leftarrow 2$
end if

Performing different operations depending on the context

Conditionals allow to perform parts of an algorithm only if certain conditions are met. In this course, this is achieved with the keywords if, then and else.

Exercise: what is the value of variable z at the end of this algorithm?

$x \leftarrow 3$
$y \leftarrow 4$
if $x < y$ **then**
  $z \leftarrow 1$
else
  $z \leftarrow 2$
end if

Answer: 1

Repeatedly perform similar operations

Loops allow to perform the same part of an algorithm several times until a given condition is met. In this course, this is achieved with the keywords while, for, do, repeat and until.

Repeatedly perform similar operations

Loops allow to perform the same part of an algorithm several times until a given condition is met. In this course, this is achieved with the keywords while, for, do, repeat and until.

Exercise: what is the value of variable z at the end of this algorithm?

$x \leftarrow 0$
$z \leftarrow 0$
while $x < 10$ **do**
  $z \leftarrow z + 2$
  $x \leftarrow x + 1$
end while

Repeatedly perform similar operations

Loops allow to perform the same part of an algorithm several times until a given condition is met. In this course, this is achieved with the keywords while, for, do, repeat and until.

Exercise: what is the value of variable z at the end of this algorithm?

$x \leftarrow 0$
$z \leftarrow 0$
while $x < 10$ **do**
  $z \leftarrow z + 2$
  $x \leftarrow x + 1$
end while

Answer: 20

Note: the following algorithm performs an equivalent task on z

$z \leftarrow 0$
for $x \leftarrow 1$ to 10 **do**
  $z \leftarrow z + 2$
end for

Exercise: write an equivalent algorithm using repeat (...) until

Note: the following algorithm performs an equivalent task on z

$z \leftarrow 0$
for $x \leftarrow 1$ to 10 **do**
    $z \leftarrow z + 2$
end for

Exercise: write an equivalent algorithm using repeat (...) until

$x \leftarrow 0$
$z \leftarrow 0$
repeat
    $z \leftarrow z + 2$
    $x \leftarrow x + 1$
until $x = 10$

"(..) allow a function to specify a return value to be passed back to the code that called the function" (Wikipedia)

Example function: $min(a, b)$

if $a < b$ **then**
    return a
else
    return b
end if

"(..) allow a function to specify a return value to be passed back to the code that called the function" (Wikipedia)

Example function: $min(a, b)$

if $a < b$ **then**
  return a
else
  return b
end if

Exercise: write $hypot(a, b)$, returning the square root of the sum of the squares of two numbers

$hypot(a, b)$

"(..) allow a function to specify a return value to be passed back to the code that called the function" (Wikipedia)

Example function: $min(a, b)$

if $a < b$ **then**
  return a
else
  return b
end if

Exercise: write $hypot(a, b)$, returning the square root of the sum of the squares of two numbers

$hypot(a, b)$

return $\sqrt{a^2 + b^2}$

Introducing time complexity

Definition

Time complexity of an algorithm: the number of steps performed by this algorithm.

Introducing time complexity

Definition

Time complexity of an algorithm: the number of steps performed by this algorithm.

- Example: counting from 1 to n has a time complexity of n.

Introducing time complexity

Definition

Time complexity of an algorithm: the number of steps performed by this algorithm.

- Example: counting from 1 to n has a time complexity of n.
- Problem: the exact number of elementary steps also depends on the computer, programming language, programmer, etc

Introducing time complexity

Definition

Time complexity of an algorithm: the number of steps performed by this algorithm.

- Example: counting from 1 to n has a time complexity of n.
- Problem: the exact number of elementary steps also depends on the computer, programming language, programmer, etc

Notation

- The Big $\mathcal{O}$ notation is used to represent complexity, and means "in the order of".
- For instance, our complete enumeration algorithm for the $P||C_{max}$ has time complexity $\mathcal{O}(m^n)$, since it evaluates m^n solutions.

(N.B.: the Big $\mathcal{O}$ notation is also used for other measures)

Introducing time complexity (2)

Exercise: what is the time complexity of the following algorithm?
(n and m are given constants, and drinking coffee is in $\mathcal{O}(1)$, a.k.a. constant time)

$x \leftarrow 0$
while $x < n$ **do**
  for $i \leftarrow 1$ to m **do**
    drink coffee
  end for
  $x \leftarrow x + 1$
end while

Introducing time complexity (2)

Exercise: what is the time complexity of the following algorithm?
(n and m are given constants, and drinking coffee is in $\mathcal{O}(1)$, a.k.a. constant time)

$x \leftarrow 0$
while $x < n$ **do**
  for $i \leftarrow 1$ to m **do**
    drink coffee
  end for
  $x \leftarrow x + 1$
end while

Answer: $\mathcal{O}(m \cdot n)$

Complexity of a problem

Definition

Time complexity of a problem: the time complexity of the fastest known algorithm to solve this problem.

Complexity of a problem

Definition

Time complexity of a problem: the time complexity of the fastest known algorithm to solve this problem.

Problems can be sorted into classes:

- P is the class of problems of polynomial time complexity
- NP is the class of problems that can be solved by a **N**on-deterministic **P**olynomial time algorithm (P $\subseteq$ NP)
- (this means it's actually hard to solve problems in NP $\setminus$ P)

Complexity of a problem

Definition

Time complexity of a problem: the time complexity of the fastest known algorithm to solve this problem.

Problems can be sorted into classes:

- P is the class of problems of polynomial time complexity
- NP is the class of problems that can be solved by a **N**on-deterministic **P**olynomial time algorithm ($P \subseteq NP$)
- (this means it's actually hard to solve problems in $NP \setminus P$)
- NP-Complete problems are "the hardest problems in NP"; they are all equivalent!

Complexity of a problem

Definition

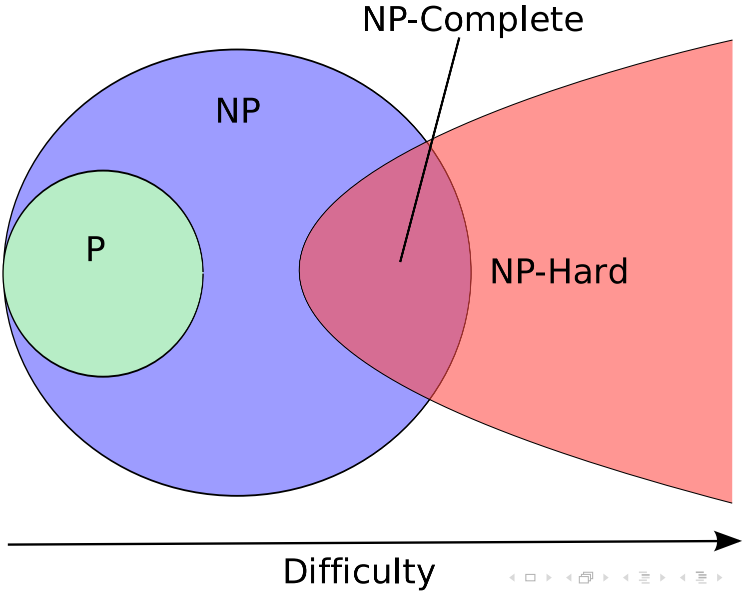
Time complexity of a problem: the time complexity of the fastest known algorithm to solve this problem.

Problems can be sorted into classes:

- P is the class of problems of polynomial time complexity
- NP is the class of problems that can be solved by a **N**on-deterministic **P**olynomial time algorithm (P $\subseteq$ NP)
- (this means it's actually hard to solve problems in NP $\setminus$ P)
- NP-Complete problems are "the hardest problems in NP"; they are all equivalent!
- NP-hard problems are "at least as hard as NP-complete problems"

Most of the interesting combinatorial problems are NP-Hard, including TSP, $P||C_{max}$ and p-Median.

Complexity classes: an overview



Solving interesting problems

Bottom line

Those NP-hard problems are hard to solve!
But complete enumeration is not the most clever approach. . .

Solving interesting problems

Bottom line

Those NP-hard problems are hard to solve!
But complete enumeration is not the most clever approach...

There are alternatives:

- Exact methods
  - Example 1: Constraint Programming
  - Example 2: Linear Programming

Solving interesting problems

Bottom line

Those NP-hard problems are hard to solve!
But complete enumeration is not the most clever approach. . .

There are alternatives:

- Exact methods
  - Example 1: Constraint Programming
  - Example 2: Linear Programming
  - Improved enumeration: disregard certain solutions
  - Allows to solve larger problems than complete enumeration
  - May also be extremely long: a TSP with 15,112 nodes in 22.6 years on a computer from 2001
  - Size limitations occur sooner or later

Solving interesting problems

Bottom line

Those NP-hard problems are hard to solve!
But complete enumeration is not the most clever approach...

There are alternatives:

- Exact methods
  - Example 1: Constraint Programming
  - Example 2: Linear Programming
  - Improved enumeration: disregard certain solutions
  - Allows to solve larger problems than complete enumeration
  - May also be extremely long: a TSP with 15,112 nodes in 22.6 years on a computer from 2001
  - Size limitations occur sooner or later
- Heuristics
- Metaheuristics (general-purpose heuristic-based methods)

Outline

Heuristic approaches

A metaheuristic is a special kind of heuristic method. . .

(personal) Definition

Heuristic: a method based on common sense, empirical knowledge, and intuition.

Heuristic approaches

A metaheuristic is a special kind of heuristic method...

(personal) Definition

Heuristic: a method based on common sense, empirical knowledge, and intuition.

- This is neither a lecture on common sense nor on intuition.
- We will work on method and build knowledge.

Construction Vs. Improvement

Two critical questions

1. How do we construct a solution?
2. How do we improve a constructed solution?

Construction Vs. Improvement

Two critical questions

1. How do we construct a solution?
2. How do we improve a constructed solution?

Different questions call for different answers. . .

Two possible answers

1. With a construction heuristic
2. With an improvement heuristic

Construction heuristics

Idea: from a problem input, construct a solution to this problem.

Example: greedy algorithms

- Iteratively satisfy problem constraints
- Use a <u>greedy criterion</u> to achieve this
- Stop when all constraints are satisfied
- Remark: The greedy criterion should take the objective into account

Construction heuristics

Idea: from a problem input, construct a solution to this problem.

Example: greedy algorithms

- Iteratively satisfy problem constraints
- Use a greedy criterion to achieve this
- Stop when all constraints are satisfied
- Remark: The greedy criterion should take the objective into account

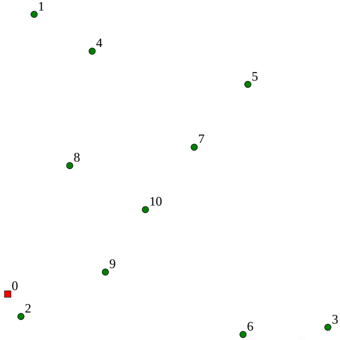
Exercise: design a greedy heuristic for the TSP

1. Which constraints do we iteratively satisfy?
2. What is the greedy criterion?

Greedy heuristic for the TSP

Nearest neighbour construction

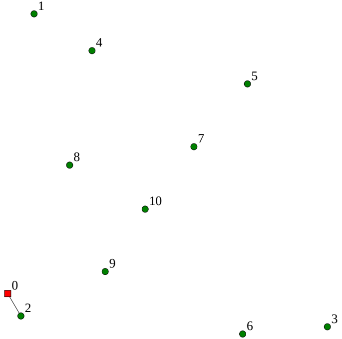
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

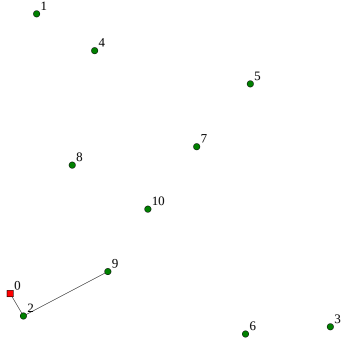
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

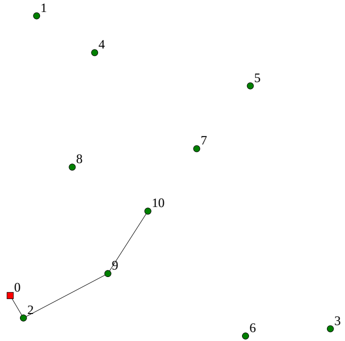
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

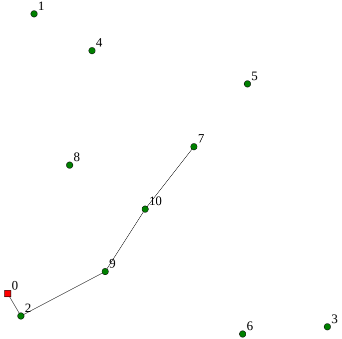
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

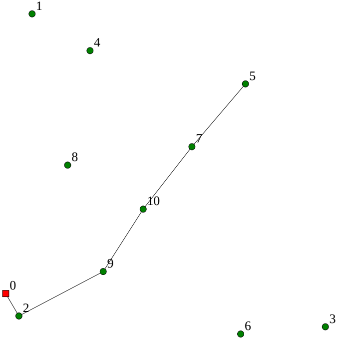
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

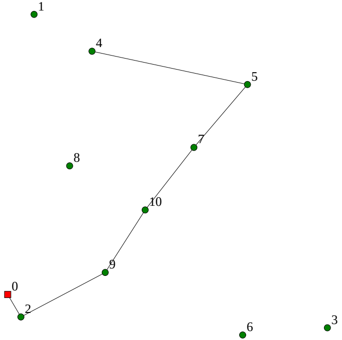
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

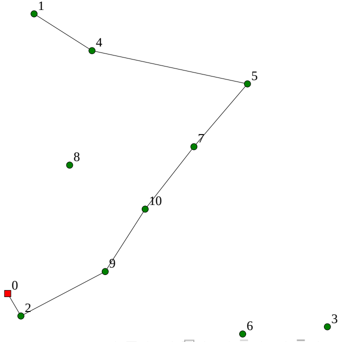
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

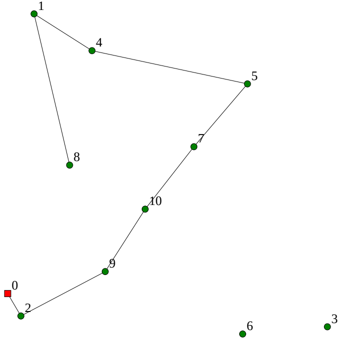
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

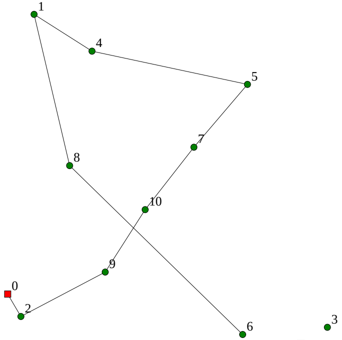
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

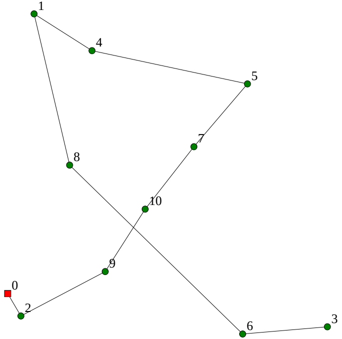
	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-



Greedy heuristic for the TSP

Nearest neighbour construction

	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-

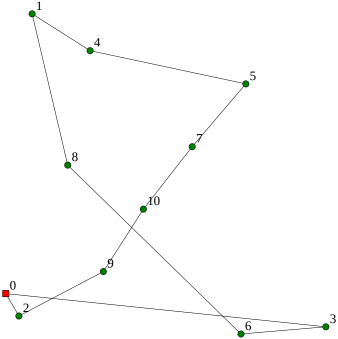


Greedy heuristic for the TSP

Nearest neighbour construction

	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-

- Total cost = 335

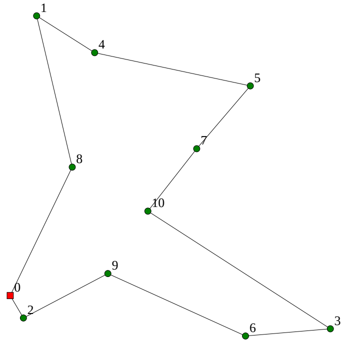


Greedy heuristic for the TSP

Nearest neighbour construction

	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-

- Total cost = 335
- But there exists a better solution, with cost = 308 (9% better)

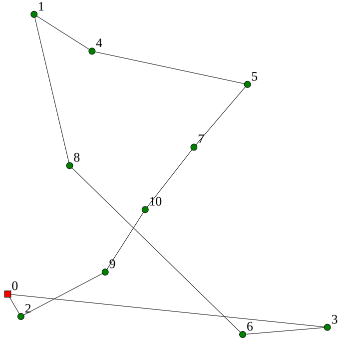


Greedy heuristic for the TSP

Nearest neighbour construction

	1	2	3	4	5	6	7	8	9	10
0	76	7	73	69	79	54	58	38	23	39
1	-	82	108	16	52	99	51	42	72	59
2	82	-	69	74	81	50	60	42	22	40
3	108	69	-	92	68	19	57	73	52	52
4	16	74	92	-	36	84	35	31	60	45
5	52	81	68	36	-	68	21	46	60	41
6	99	50	19	84	68	-	52	60	35	40
7	51	60	57	35	21	52	-	28	39	20
8	42	42	73	31	46	60	28	-	30	21
9	72	22	52	60	60	35	39	30	-	19
10	59	40	52	45	41	40	20	21	19	-

- Total cost = 335
- But there exists a better solution, with cost = 308 (9% better)
- So it would be nice to improve this tour...



Definition

Neighbourhood: the set of solutions that can be obtained by applying a given operator to a given solution.

Improvement heuristics

Definition

Neighbourhood: the set of solutions that can be obtained by applying a given operator to a given solution.

Solutions in the neighbourhood of x are its neighbours.

Definition

Neighbourhood: the set of solutions that can be obtained by applying a given operator to a given solution.

Solutions in the neighbourhood of x are its neighbours.

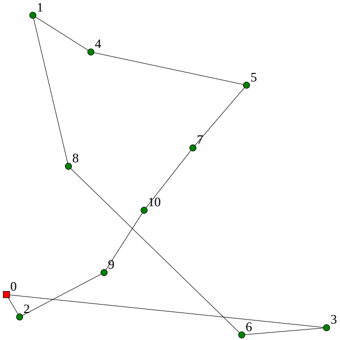
- Operator example for the TSP: move one node to another position in the tour.
- Exercise: find other neighbourhoods for the TSP.

Definition

Local Search: a heuristic aiming at improving a solution by performing neighbourhood exploration.

Improvement heuristics for the TSP

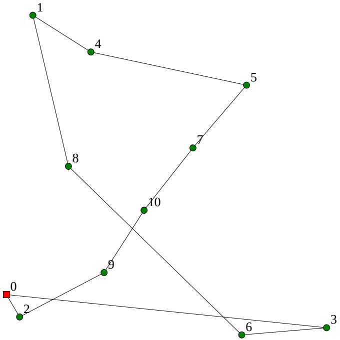
An example: the "node move" neighbourhood



Suppose that visiting order starts with node 2...
We can:

Improvement heuristics for the TSP

An example: the "node move" neighbourhood



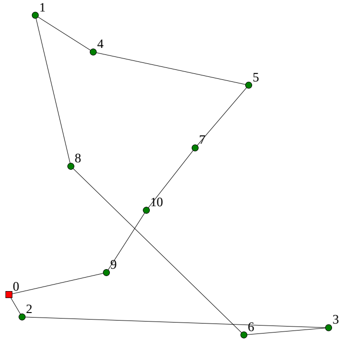
Suppose that visiting order starts with node 2...

We can:

1. Move 2 at the end of the tour

Improvement heuristics for the TSP

An example: the "node move" neighbourhood



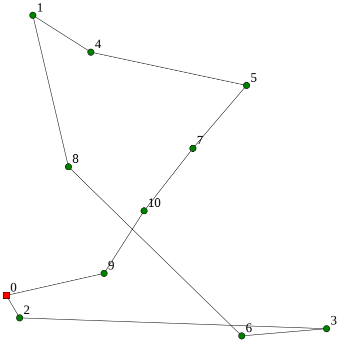
Suppose that visiting order starts with node 2...

We can:

1. Move 2 at the end of the tour

Improvement heuristics for the TSP

An example: the "node move" neighbourhood



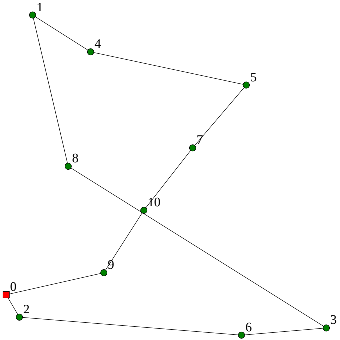
Suppose that visiting order starts with node 2...

We can:

1. Move 2 at the end of the tour
2. Move 6 after 3

Improvement heuristics for the TSP

An example: the "node move" neighbourhood



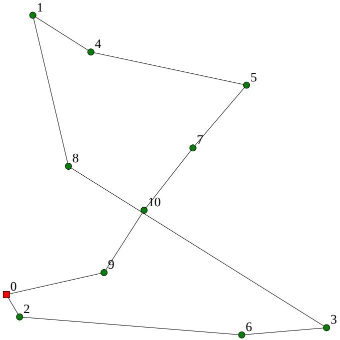
Suppose that visiting order starts with node 2...

We can:

1. Move 2 at the end of the tour
2. Move 6 after 3

Improvement heuristics for the TSP

An example: the "node move" neighbourhood

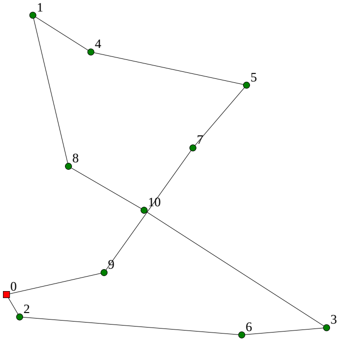


Suppose that visiting order starts with node 2. . .
We can:

1. Move 2 at the end of the tour
2. Move 6 after 3
3. Move 10 after 8

Improvement heuristics for the TSP

An example: the "node move" neighbourhood



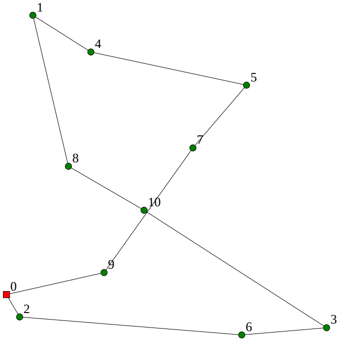
Suppose that visiting order starts with node 2...

We can:

1. Move 2 at the end of the tour
2. Move 6 after 3
3. Move 10 after 8

Improvement heuristics for the TSP

An example: the "node move" neighbourhood

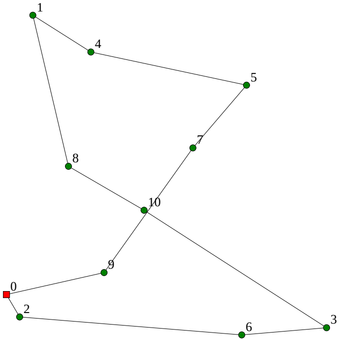


Suppose that visiting order starts with node 2...

We can:

1. Move 2 at the end of the tour
2. Move 6 after 3
3. Move 10 after 8

Then there is no more improvement!



Suppose that visiting order starts with node 2...

We can:

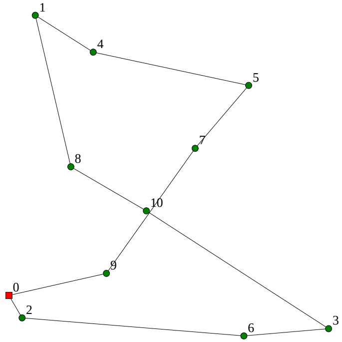
1. Move 2 at the end of the tour
2. Move 6 after 3
3. Move 10 after 8

Then there is no more improvement!

- For this solution, cost = 326

Improvement heuristics for the TSP

An example: the "node move" neighbourhood



Suppose that visiting order starts with node 2...

We can:

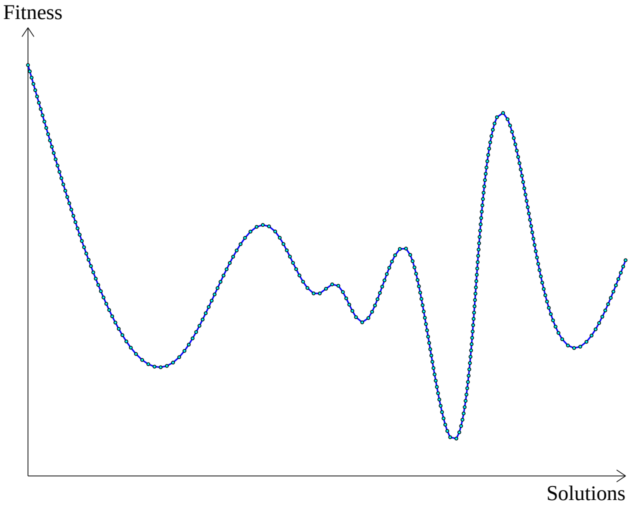
1. Move 2 at the end of the tour
2. Move 6 after 3
3. Move 10 after 8

Then there is no more improvement!

- For this solution, cost = 326
- We are still far from 308...

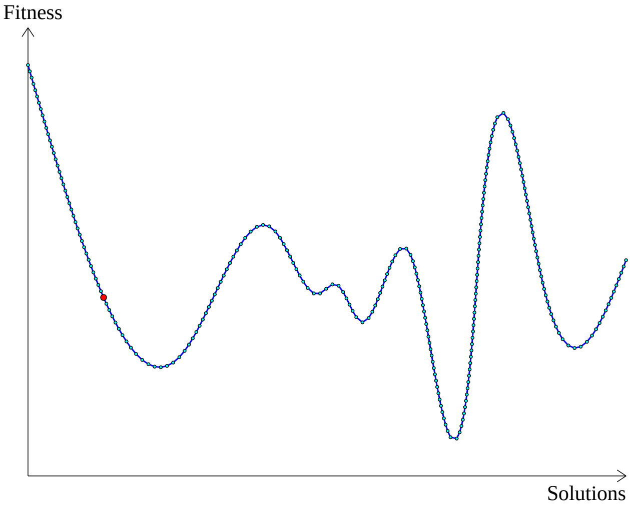
The local optimum issue

Assume neighbour solutions are adjacent points in this Solution Space. . .



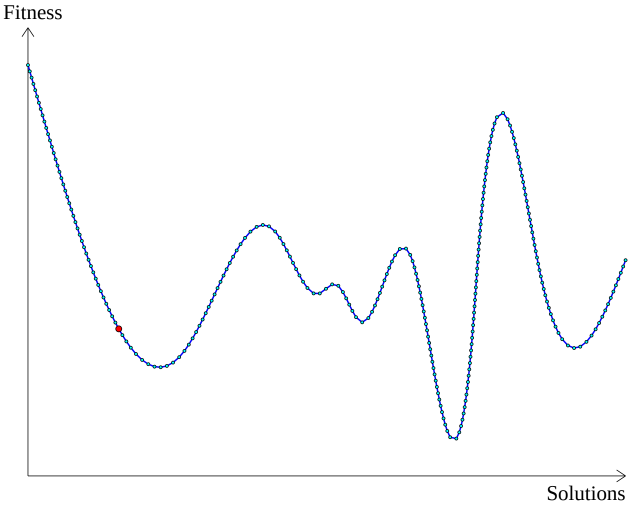
The local optimum issue

Assume neighbour solutions are adjacent points in this Solution Space. . .



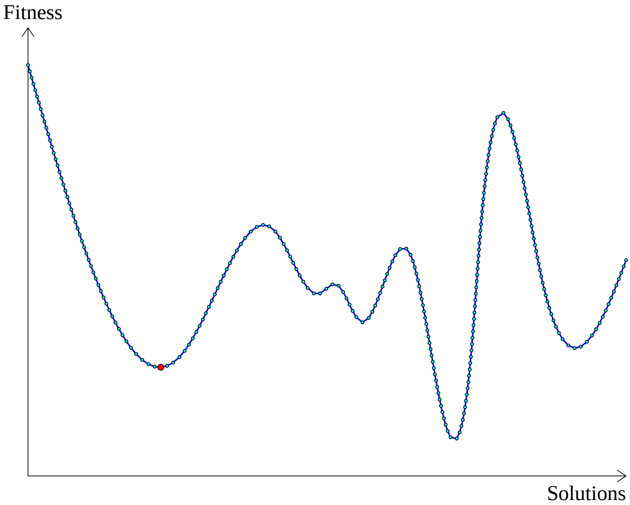
The local optimum issue

Assume neighbour solutions are adjacent points in this Solution Space. . .



The local optimum issue

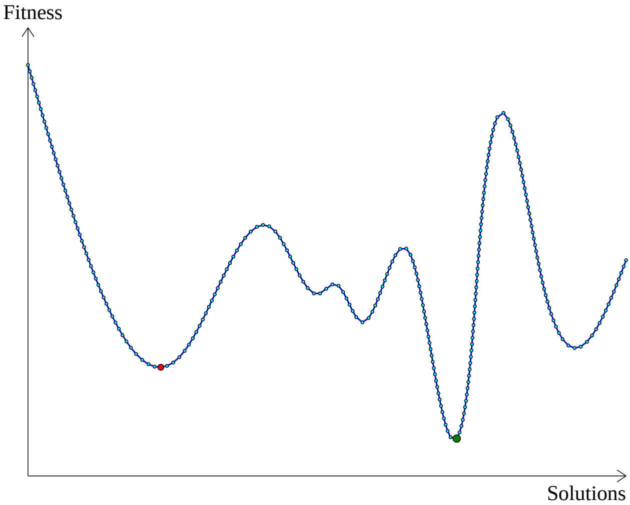
Assume neighbour solutions are adjacent points in this Solution Space. . .



No more improvement in the neighbourhood!

The local optimum issue

Assume neighbour solutions are adjacent points in this Solution Space. . .



No more improvement in the neighbourhood!
Note that the global optimum is also a local optimum

Simple heuristics: limitations

A heuristic typically exploits problem structure

- Good news: using information from the problem really helps

Simple heuristics: limitations

A heuristic typically exploits problem structure

- Good news: using information from the problem really helps
- Bad news: heuristic designed for problem X cannot be applied to problem Y
- Bad news: sometimes a heuristic is “just not enough”
  - Local optima
  - How do we explore a neighbourhood?

Simple heuristics: limitations

A heuristic typically exploits problem structure

- Good news: using information from the problem really helps
- Bad news: heuristic designed for problem X cannot be applied to problem Y
- Bad news: sometimes a heuristic is "just not enough"
    - Local optima
    - How do we explore a neighbourhood?

Metaheuristics aim at dealing with these bad news!

Outline

Etymology

There is no commonly agreed definition!

(personal) Definition

Metaheuristic: a general-purpose heuristic method manipulating other, often problem-specific, heuristic methods.

Etymology

There is no commonly agreed definition!

(personal) Definition

Metaheuristic: a general-purpose heuristic method manipulating other, often problem-specific, heuristic methods.

Other definitions:

- A top-level general strategy which guides other heuristics to search for feasible solutions in domains where the task is hard. (various sources on the web)
- A set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. (Metaheuristics Network)

Etymology

There is no commonly agreed definition!

(personal) Definition

Metaheuristic: a general-purpose heuristic method manipulating other, often problem-specific, heuristic methods.

Other definitions:

- A top-level general strategy which guides other heuristics to search for feasible solutions in domains where the task is hard. (various sources on the web)
- A set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. (Metaheuristics Network)

Most metaheuristics include stochastic components (a.k.a. non-deterministic; e.g. randomness is involved in the process)

A few milestones

Term introduction

- 1986, Fred Glover: "Future Paths for Integer Programming and Links to Artificial Intelligence"
- This paper is about Tabu Search, a well-known metaheuristic.
- (but there were metaheuristics before that...)

A few milestones

Term introduction

- 1986, Fred Glover: "Future Paths for Integer Programming and Links to Artificial Intelligence"
- This paper is about Tabu Search, a well-known metaheuristic.
- (but there were metaheuristics before that...)

A few dates

- 1965: first Evolution Strategy
- 1975: first Genetic Algorithms
- 1983: Simulated Annealing
- 1986: Tabu Search
- 1991: Ant Colony Optimisation
- 1997: Variable Neighbourhood Search

Evaluating Metaheuristics

- There is no such thing as a “Best Metaheuristic”
- Some theoretical results back this claim
- But there are still relevant questions!

Evaluating Metaheuristics

- There is no such thing as a “Best Metaheuristic”
- Some theoretical results back this claim
- But there are still relevant questions!

Relevant questions

1. Is the metaheuristic well-designed?
   $\rightarrow$ Qualitative analysis
2. Once applied to a given problem, does it behave well?
   $\rightarrow$ Quantitative analysis

Qualitative analysis

Again, no common agreement; personal suggestions:

Suggestion 1: Clarity, Modularity, Simplicity

A metaheuristic should be easy to apply to any given hard problem!

- Clarity: is it easy to understand what it does?
- Modularity: can it easily be applied to different problems?
- Simplicity: is it easy to implement?

Qualitative analysis (2)

Suggestion 2: Intensification

- The global optimum is also a local optimum
- A good metaheuristic should be good at finding local optima!

Qualitative analysis (2)

Suggestion 2: Intensification

- The global optimum is also a local optimum
- A good metaheuristic should be good at finding local optima!

Suggestion 3: Diversification

- Interesting solutions might reside in different "regions"
- A good metaheuristic should provide a broad exploration of the solution space!

Intensification and Diversification can be seen as opposite goals...

Quantitative analysis

Goal: once the metaheuristic is implemented, evaluate it.

Interesting criteria include:

- Ability to find good/optimal solutions
  - Comparison with optimal solutions (if available)
  - Comparison with Lower/Upper Bounds (LB/UB, if available)
  - Comparison with other metaheuristics

Quantitative analysis

Goal: once the metaheuristic is implemented, evaluate it.

Interesting criteria include:

- Ability to find good/optimal solutions
  - Comparison with optimal solutions (if available)
  - Comparison with Lower/Upper Bounds (LB/UB, if available)
  - Comparison with other metaheuristics
- Reliability and stability over several runs
  - Homemade indicators (e.g. Average gap to the optimum)
  - Traditional statistical tools (standard deviation etc)

Quantitative analysis

Goal: once the metaheuristic is implemented, evaluate it.

Interesting criteria include:

- Ability to find good/optimal solutions
  - Comparison with optimal solutions (if available)
  - Comparison with Lower/Upper Bounds (LB/UB, if available)
  - Comparison with other metaheuristics
- Reliability and stability over several runs
  - Homemade indicators (e.g. Average gap to the optimum)
  - Traditional statistical tools (standard deviation etc)
- Reasonable computational time
  - "Reasonable" $\rightarrow$ highly subjective
  - Context-dependent
  - Usually: between a few seconds and 1 hour

Quantitative analysis

Goal: once the metaheuristic is implemented, evaluate it.

Interesting criteria include:

- Ability to find good/optimal solutions
  - Comparison with optimal solutions (if available)
  - Comparison with Lower/Upper Bounds (LB/UB, if available)
  - Comparison with other metaheuristics
- Reliability and stability over several runs
  - Homemade indicators (e.g. Average gap to the optimum)
  - Traditional statistical tools (standard deviation etc)
- Reasonable computational time
  - "Reasonable" $\rightarrow$ highly subjective
  - Context-dependent
  - Usually: between a few seconds and 1 hour

Keep this in mind!