GOPP

Heuristic algorithms for solving optimization problems

Aurélien Froger

aurelien.froger@u-bordeaux.fr

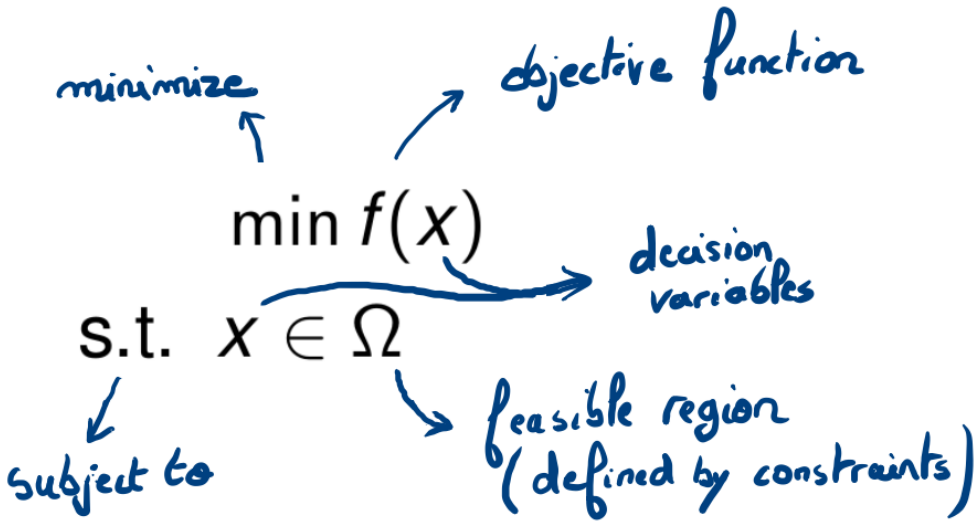
Université de Bordeaux
Institut de Mathématiques de Bordeaux
Inria Bordeaux Sud-Ouest - Equipe Edge

[Cours avancé]

Agenda

Modelization

Reminder: Solving an optimization problem can only be done after a phase of modeling (identify decision variables, constraints and objective function)



Solution

Solution

A solution is an assignment of values to decision variables.

A solution x can be :

- *feasible* if it respects all the constraints of the problem (i.e., $x \in \Omega$).
- *infeasible* in the contrary case.

The set of feasible solutions to an optimization problem defines the feasible region of this problem.

A feasible solution x^* is said to be optimal for an optimization problem if $f(x) \geq f(x^*)$ for all $x \in \Omega$

Introduction

How do we solve an optimization problem when

- we do not have much information about f (e.g., blackbox evaluation) and Ω?
- there does not exist efficient algorithms to solve a mathematical model (linear or non-linear, with or without integer variables) developed for the problem?
- the computational time must be reasonable (e.g., limited to several seconds or minutes)?

$\Rightarrow$ a possible answer: design a heuristic algorithm

Heuristics

Definition

Algorithms/methods that do not guarantee computing an optimal solution are called heuristic algorithms (or simply heuristics).

Aim: Efficiently generate one or several high-quality solutions in a reasonable computational time.

Heuristics characteristics

- Short computational times
- Simple
- Flexible

Heuristics

If you have modeled your problem as a linear (non-linear) program (with integer variables) and the model is of reasonable size, then the use of a **mathematical solver with a time limit** provides a heuristic.

In a large number of cases, one can compute better solutions with **dedicated heuristics**.

Decomposition-based heuristic

Idea: Decompose the problem into a sequence of subproblems that should be solved to finally arrive at a complete solution.

- Divide the problem into smaller subproblems easier to solve.
- Solve each subproblem (heuristically or exactly)
  Sub-problems can be solved *independently* or *sequentially* (keep constant all decisions related to previously solved subproblems)
- Build the solution of the original problem by combining all subproblem solutions.

Heuristics

We can classify heuristics into two types:

- **constructive heuristics**: a solution is built from scratch, step by step (by introducing new elements at each iteration), according to a set of rules defined before-hand.

- **improvement heuristics**: a solution is iteratively modified by successive small changes in order to improve its quality
(Remark: some of these heuristics work with multiple solutions)

Agenda

Constructive heuristics

Aim: Build a solution to a problem by expanding, at each iteration, a partial solution according to a simple and intuitive set of decision rules (no solution is obtained until the procedure is complete)

Key designing aspects:

- Try to avoid myopic choices (if possible)
- Try to find a trade-off between accuracy, speed, simplicity and flexibility

Constructive heuristics

Several existing frameworks:

- Greedy heuristics: at each iteration perform the expansion that gives the immediate minimum cost
- Regret procedures: at each iteration perform the expansion what would turn out to be the most expensive if not performed (look-ahead mechanism)
- List algorithms: sort a list of possible expansions according to a given criterion and perform each expansion in the order it appears in the list (ignoring those that are impossible because of past expansion decisions)

Remark: The frontier between different framework is not always clear.

Constructive heuristics

Traveling salesman problem (TSP)

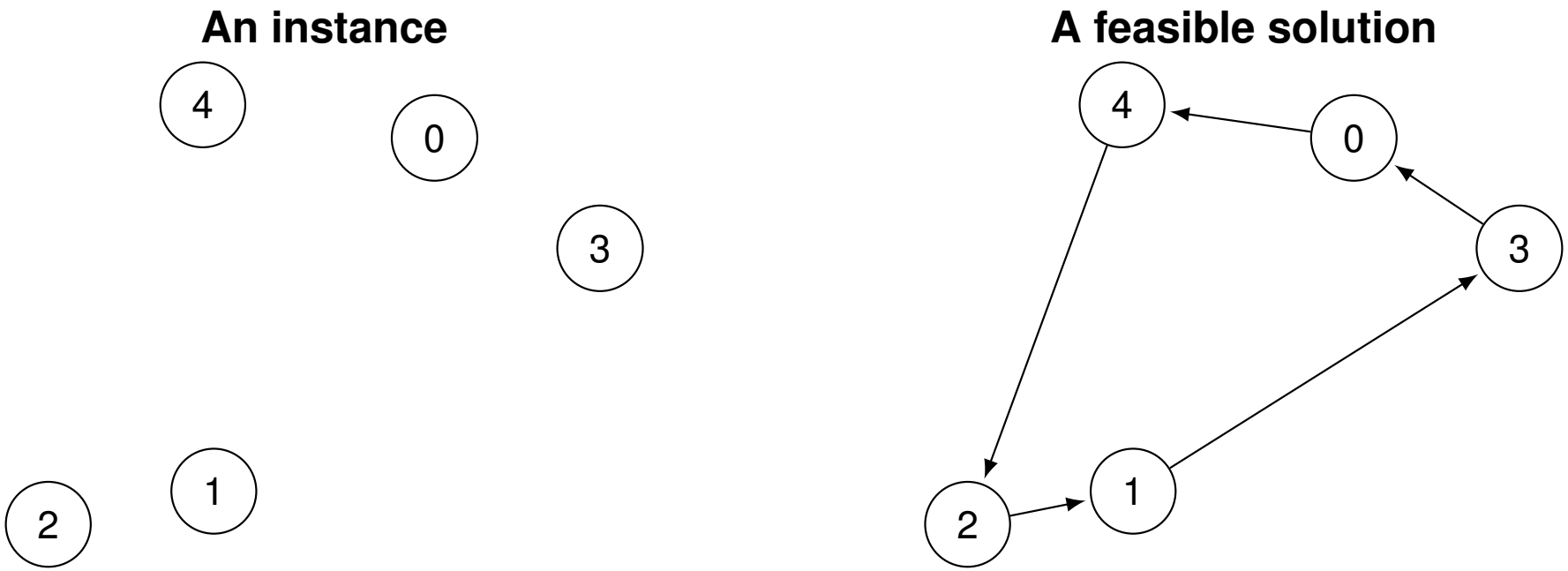
Data:

- $\mathcal{I} = \{1, \ldots, n\}$: set of cities to visit
- $d_{ij} \in \mathbb{R}^+$, $i, j \in \mathcal{I}, i \neq j$: distance from i to j

Problem: Find the minimum distance circuit visiting all cities exactly once and returning to the starting city

Constructive heuristics

Traveling salesman problem



Solution representation: sequence of cities to visit

Heuristic: Nearest-neighbor heuristic, Insertion heuristic (with regret), Clark-and-Wright heuristic

Constructive heuristics

Coloring problem

Data:

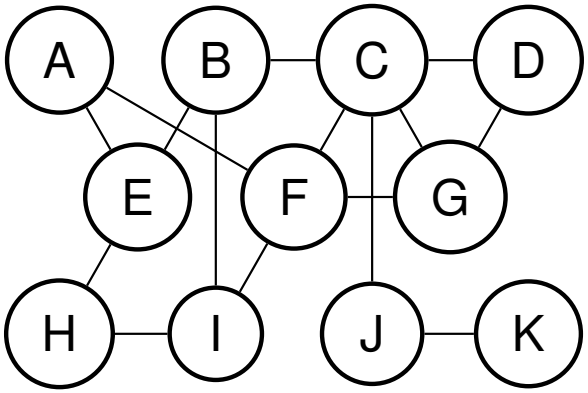
- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ a graph
  - $\mathcal{V}$: set of vertices
  - $\mathcal{E}$: set of edges (each edge connects 2 vertices of the graph)
- $\mathcal{K}$: an ordered set of colors ($\mathcal{K} = \{1, ..., K\}$ with $K \geq |\mathcal{V}|$)

Problem: Assign a color to each vertex so that two vertices connected by an edge do not have the same color while minimizing the total number of colors used.

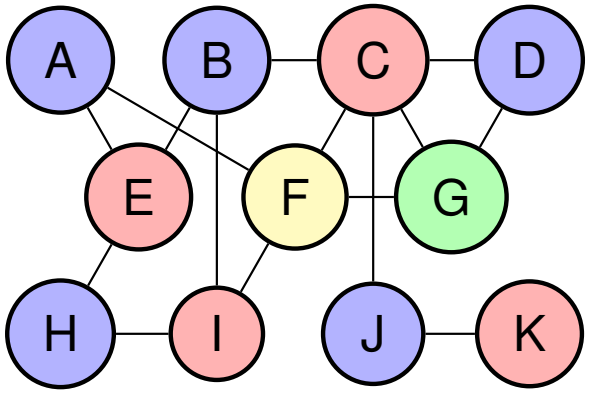
Constructive heuristics

Graph coloring problem

An instance



A feasible solution



Solution representation: list of vertices (implicit)

Heuristic: takes a list of vertices as input and selects the vertices in order and assign to each vertex the first possible color according to previous choices

Constructive heuristics

Bin packing problem

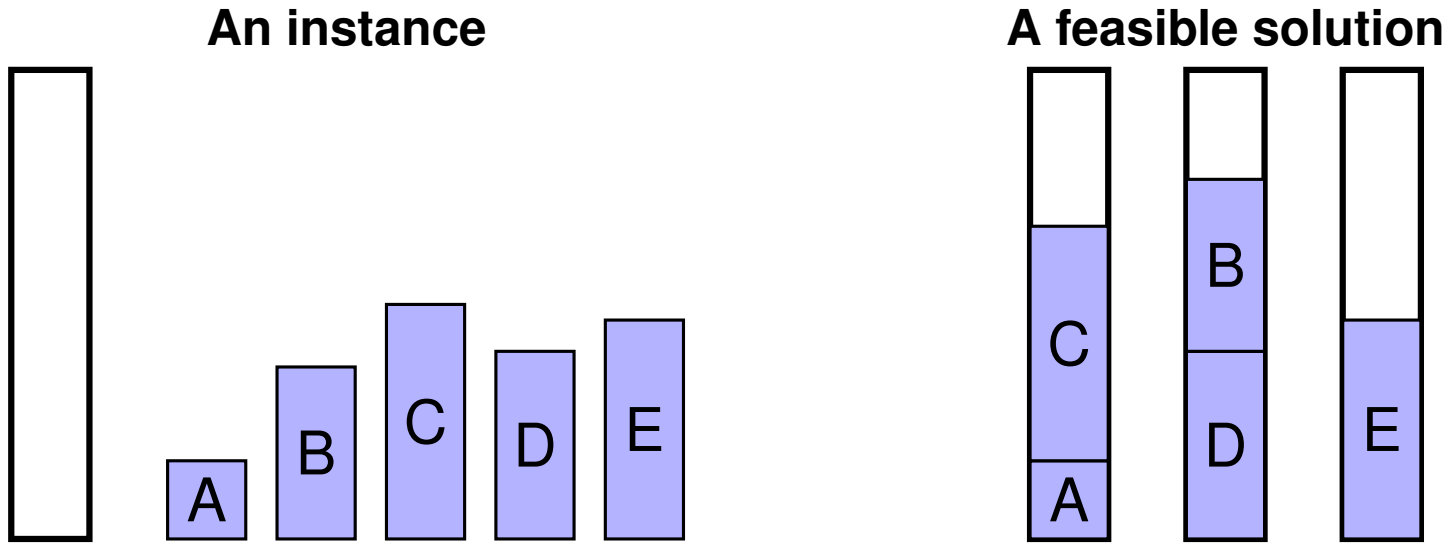
Data:

- $\mathcal{I}$: set of items
- w_i : weight (or size) of item $i \in \mathcal{I}$
- C : capacity of each bin

Problem: Find the smallest number of bins that will hold all the items

Constructive heuristics

Bin packing problem



Solution representation: list of items

Heuristic: takes a list of items as input and considers that the bins are indexed in increasing order of their creation, selects the items in order and places the item into a bin according to a decision rule: *first-fit* or *best-fit*.

Constructive heuristics

Tips and Tricks

- Keep it simple first and try to improve it only after having succeeded in building a solution
- Try slightly different versions of the same constructive heuristic
- Finding an initial solution is sometimes not easy due to some difficult constraints.
  $\rightarrow$ relax these constraints and penalize their violation into the objective function
- The quality of the solution built by a constructive heuristic is usually not a concern if an improvement heuristic is subsequently used.

Agenda

Improvement heuristics

Aim: Improves a *starting solution* to a problem by making modifications until a stopping criterion is verified

Local search algorithms are the most used improvement heuristics.

- They can be seen as *trial and error* methods performed in a *systematic way*.
- They require **defining the solution representation** and the search space Ω
- They explore the search space by iteratively making *elementary modifications* of the current solution.
- They rely on *neighborhood* structures (or neighborhoods).

Neighborhood

Neighborhood

A neighborhood structure (or simply a neighborhood) is a function N from Ω to $\mathcal{P}(\Omega)$ that associates a subset of Ω to any solution $s \in \Omega$.
A solution $s' \in N(s)$ is said to be a neighbor of s ($s \notin N(s)$).

- The transition from a solution s to a solution of $N(s)$ is referred to as a "move".
- At an iteration, the local search algorithm will (generally) *accept a move*.
- A move is said to be *better* than another one if the resulting solution has a smaller objective function.
- A move is said to be *improving* if the resulting solution has a smaller objective function than the current solution.

Neighborhood

For a given $s \in \Omega$, how to define $N(s)$?

- $N(s)$ should be a set of solutions that have similar elements with s (a neighbor of s should not be not too different from s)
  Generally, every solution in $N(s)$ can be obtained from s by a simple local transformation.
- $N(s)$ should be easy to build.
- The neighborhood should be large enough such that you do not get stuck in bad local optima *(the larger the neighborhood size, the better the solutions that can be reached in a single move)*
- The neighborhood should be small enough so that you can efficiently search the neighbors for the best move (the smaller the neighborhood size, the faster a new solution is computed).
  Ideally $N(s)$ should be explored in polynomial time.
- From any solution, it should be possible to reach an optimal solution by generating a sequence of neighboring solutions.

Neighborhood

Examples for the traveling salesman problem

- Extract a city from the tour and re-insert it at a different position
- Select two cities and exchange their positions in the tour
- Eliminate two non-adjacent arcs and reconnect the tour

Examples for the graph coloring problem

Context: We want to find a coloring of a graph with a given number K of colors. The starting solution is not feasible. Adjacent vertices have the same color. The objective function is to minimize the number of conflicts.

- Modify the color of 1 vertex.

Examples for the bin packing problem

- Exchange p items and q items between two arbitrary bins.

Local search algorithms

Tips and Tricks

- When f is very difficult to evaluate (or very expensive)
  $\rightarrow$ use an approximation of f to evaluate the neighborhood of s and compute the true value of a solution for a chosen move or a small subset of promising moves / use a cache memory
- When most of the neighbors of a solution have the same value of f
  $\rightarrow$ use an auxiliary objective function that will measure another desirable attribute of a solution (give relief to the solution space)
- When there are some constraints difficult to take into account (neighborhoods have few solutions)
  $\rightarrow$ relax some constraints and add to the function f a term that penalizes the violation of these constraints

Local search algorithms

Tips and Tricks (implementation)

- When evaluating every possible move:
  - avoid copying the solution
  - do not recompute the value of the whole solution ex nihilo calculate only the difference in value from the current solution
- Modify the current solution only when a move is accepted

Local search algorithms

Generalized procedure

1. Initialization
   - Build an initial solution s with a constructive heuristic (or randomly)
   - Set $s^* \leftarrow s$ (record the best solution found so far)
2. Move choice
   - Select $s' \in N(s)$
   - Set $s \leftarrow s'$ (i.e. replace s by s')
3. Update and termination
   - If $f(s) < f(s^*)$, then $s^* \leftarrow s$
   - If the stopping criterion is verified, then terminate and return s^*
     Otherwise go to step 2

Local search algorithms

Changing strategies in step 2 leads to different solution approaches:

- Pure descent
- Simulated annealing (SA)
- Tabu search (TS)

Changing strategies in step 1 and 3 leads to Variable Neighborhood Descent (VND).

Using *implicitly defined neighborhoods* leads to Large Neighborhood Search (LNS).

Local search algorithms
Pure descent

2. Move choice
   - Select $s' \in N(s)$ such that $f(s') < f(s)$
   - Set $s \leftarrow s'$ (i.e. replace s by s')

Stopping criterion: terminate if $\forall s' \in N(s)$, $f(s') \geq f(s)$

Decision: Select the first or best improving move

Major concern: Getting trapped into local optima

Local search algorithms
Simulated annealing

2. Move choice
   - Select randomly $s' \in N(s)$
   - If $f(s') < f(s)$, then accept s
     Otherwise, accept s with a probability p

Stopping criterion: maximum number of iterations / number of iterations without improvement of the best solution s^*

Major concern: How to ensure convergence?

$$\text{Set } p = \begin{cases} 1 & \text{if } f(s') < f(s) \\ e^{-(f(s')-f(s))/T} & \text{otherwise} \end{cases}$$

Start with a given value T and use a cooling schedule
- Geometric: $T = c \times T$ with $0 < c < 1$
- Boltzman cooling: $T = T/ln(k)$ with k the iteration number

Local search algorithms

Tabu search

Tabu list

- list the last m moves performed and forbid the reverse transformation (most frequent)
- list of the characteristics of the m last solutions or transformations (less frequent)

A neighbor (or a move) is *admissible* if it does not contain attributes from the tabu list or is improving (aspiration criterion).

2. Move choice
   - Select the best *admissible* neighbor $s' \in N(s)$
   - Set $s \leftarrow s'$ (even if $f(s') > f(s)$)
     Update the tabu list (a short-term memory to forbid the return to recently visited solutions)

Stopping criterion: maximum number of iterations / number of iterations without improvement of the best solution s^*

Local search algorithms
Tabu search

2. Move choice
   - Select the best *admissible* neighbor $s' \in N(s)$
   - Set $s \leftarrow s'$ (even if $f(s') > f(s)$)
     Update the tabu list (a short-term memory)

Stopping criterion: maximum number of iterations / number of iterations without improvement

Major concern: Diversification

- perform some infrequent transformations on the solution and restart the normal process from this new solution
- penalize frequent transformations or characteristics

Local search algorithms
Variable Neighborhood Descent

1. Initialization
   - Let $\mathcal{N} = \{N_1(), ..., N_K()\}$ be an ordered set of neighborhoods
   - Build an initial solution s and set $k \leftarrow 1$
   - Set $s^* \leftarrow s$ (record the best solution found so far)
2. Move choice
   - Select $s' \in N_k(s)$ such that $f(s') < f(s)$
   - Set $s \leftarrow s'$ *(if s' was found at the previous line)*
3. Update and termination
   - If $f(s) < f(s^*)$, then set $s^* \leftarrow s$
   - If a solution s' was found at step 2, then set $k \leftarrow 1$ and go to step 2
   - If $k < K$, then set $k \leftarrow k + 1$ and go to step 2
     Otherwise terminate and return $s*$

Adaptation: It is possible to randomly draw at every iteration the neighborhood to apply

Local search algorithms

Large Neighborhood Search

Neighborhoods are implicitly defined by *destroy* and *repair* operators.
The operators generally contain a *random* component.

1. Initialization
   - Build an initial solution s with a constructive heuristic (or randomly)
   - $s^* \leftarrow s$ (record the best solution found so far)
2. Move choice
   - Select (k_1, k_2) *(the destroy and repair operators to use at this iteration)*
   - $s' \leftarrow \texttt{repair}_{k_2}(\texttt{destroy}_{k_1}(s))$
   - If $f(s') < f(s)$, then $s \leftarrow s'$
3. Update and termination
   - If $f(s) < f(s^*)$, then $s^* \leftarrow s$
   - If the stopping criterion is verified, then terminate and return s^*
     Otherwise go to step 2

Stopping criterion: maximum number of iterations / number of iterations without improvement

Local search algorithms

A quick recap on what you need in a local search algorithm

- Define a constructive heuristic to build a starting solution
- Define the search space Ω
- Define the neighborhood(s)
- Define how you evaluate a solution
  Is the use of the objective function adequate?
- Define the strategy to explore the neighborhood(s) and the stopping criterion

Agenda

Metaheuristics

Definition

A metaheuristic is a search strategy that guides one or more subordinate heuristics to solve an optimization problem.

- Local search procedures (pure descent, SA, TS, VND, etc.) can be used as subordinate heuristics within a metaheuristic framework
- TS and SA can also be considered as metaheuristics

Metaheuristics

Metaheuristics can be divided into two categories:

- single-solution or trajectory metaheuristics
- population metaheuristics

Metaheuristics

Metaheuristics are based on two ingredients whose use must be *balanced*:

- *Diversification* implies a broad exploration of the search space
- *Intensification* emphasizes the search of local optima

Exploitation?

- *No exploitation* of the information accumulated during the search
  Greedy Randomized Adaptive Search (GRASP)
- *Exploitation* of the information accumulated during the search
  Variable Neighborhood Search (VNS)
  Iterated Local Search (ILS)
  Genetic algorithm (GA)

Trajectory metaheuristics

Greedy Randomized Adaptive Search

1. Initialization
   - Set $f^* \leftarrow \infty$
2. Diversification and intensification
   - Build a random solution s (use of a constructive heuristic which contains *randomness*)$s' \leftarrow$ `localsearch`(s)
3. Update and termination
   - If $f(s') < f^*$ set $s^* \leftarrow s'$ and $f^* \leftarrow f(s^*)$
   - If the stopping criterion is verified, then terminate and return s^* Otherwise go to step 2

Stopping criterion: maximum number of iterations / number of iterations without improvement of the best solution s^*

Trajectory metaheuristics
Variable Neighborhood Search

1. Initialization
   - Let $\mathcal{N} = \{N_1(), ..., N_K()\}$ be an ordered set of neighborhoods
   - Build an initial solution s with a constructive heuristic (or randomly)$s^* \leftarrow s$ (record the best solution found so far)$k \leftarrow 1$
2. Diversification and intensification
   - Select $s' \in N_k(s)$
   - $s'' \leftarrow \texttt{localsearch}(s')$
3. Update and termination
   - If $f(s'') < f(s^*)$ set $s^* \leftarrow s''$, $s \leftarrow s''$, $k \leftarrow 1$, and go to step 2
   - If $k < K$ set $k \leftarrow k + 1$ and go to step 2
     Otherwise terminate and return $s*$

Trajectory metaheuristics
Iterated Local Search

1. Initialization
   - Build an initial solution s with a constructive heuristic (or randomly)
   - $s^* \leftarrow s$ (record the best solution found so far)
2. Diversification and intensification
   - $s' \leftarrow \texttt{perturbate}(s)$ *(create a new solution from s)*
   - $s'' \leftarrow \texttt{localsearch}(s')$
3. Update and termination
   - If $f(s'') < f(s^*)$ set $s^* \leftarrow s''$
   - If $f(s'') < f(s)$ set $s \leftarrow s''$
   - If the stopping criterion is verified, then terminate and return s^* Otherwise go to step 2

Stopping criterion: maximum number of iterations / number of iterations without improvement of the best solution s^*

Population metaheuristics

Genetic algorithm

Most known population metaheuristic inspired by *evolution*.

Evolution

- Populations are composed by individuals
- The best fitted individuals have higher probabilities of surviving to compose a new generation
- The offspring of two good fitted individuals tends to be a better individual
- A mutation is an alteration of the DNA of an individual that makes him *different*

Optimization

- Generations are composed by solutions to the problem
- The best solutions have higher probabilities of being recombined with other solutions to form new solutions
- The solution formed by parts of two good solutions tends to be a better solution
- Solutions are perturbed to foster diversification

Population metaheuristics

Genetic algorithm

1. Initialization
   - $k \leftarrow 1$ (number of generations)
   - Build an initial population of solutions $\mathcal{P}$
2. Diversification and intensification
   - $\mathcal{P} \leftarrow \texttt{mutate}(\mathcal{P})$ *(apply a small perturbation to some solutions)*
   - $\mathcal{P}' \leftarrow \texttt{crossover}(\mathcal{P})$ *(build new solutions from the population)*
   - $\mathcal{P} \leftarrow \texttt{select}(\mathcal{P}, \mathcal{P}')$ *(select the solutions for the next generation)*
3. Update and termination
   - Update s^* as the best solution calculated so far
   - Set $k \leftarrow k + 1$
   - If the stopping criterion is verified, then terminate and return s^*
     Otherwise go to step 2

Stopping criterion: maximum number of generations

Solution representation

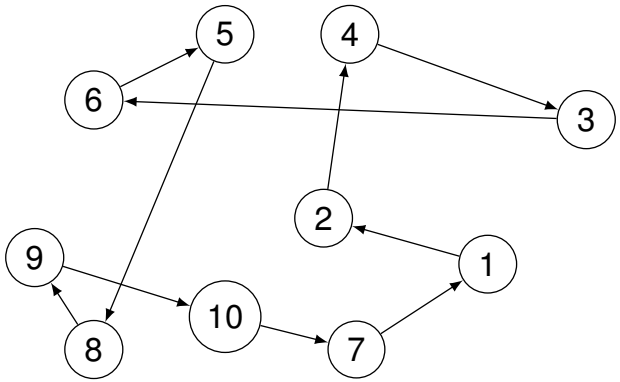
Every solution is represented as a **chromosome**.
A chromosome should map a solution easily and the classical genetic operators should be easily adaptable

- Direct encoding: list of values for each variable
- Indirect encoding: only the mains decisions are encoded (problem dependent)

Population metaheuristics

Genetic algorithm

Solution representations for the traveling salesman problem:



the indices in the array indicate the visiting order of the cities that are represented by the integers inside it

1	2	3	4	5	6	7	8	9	10
1	2	4	3	6	5	8	9	10	7

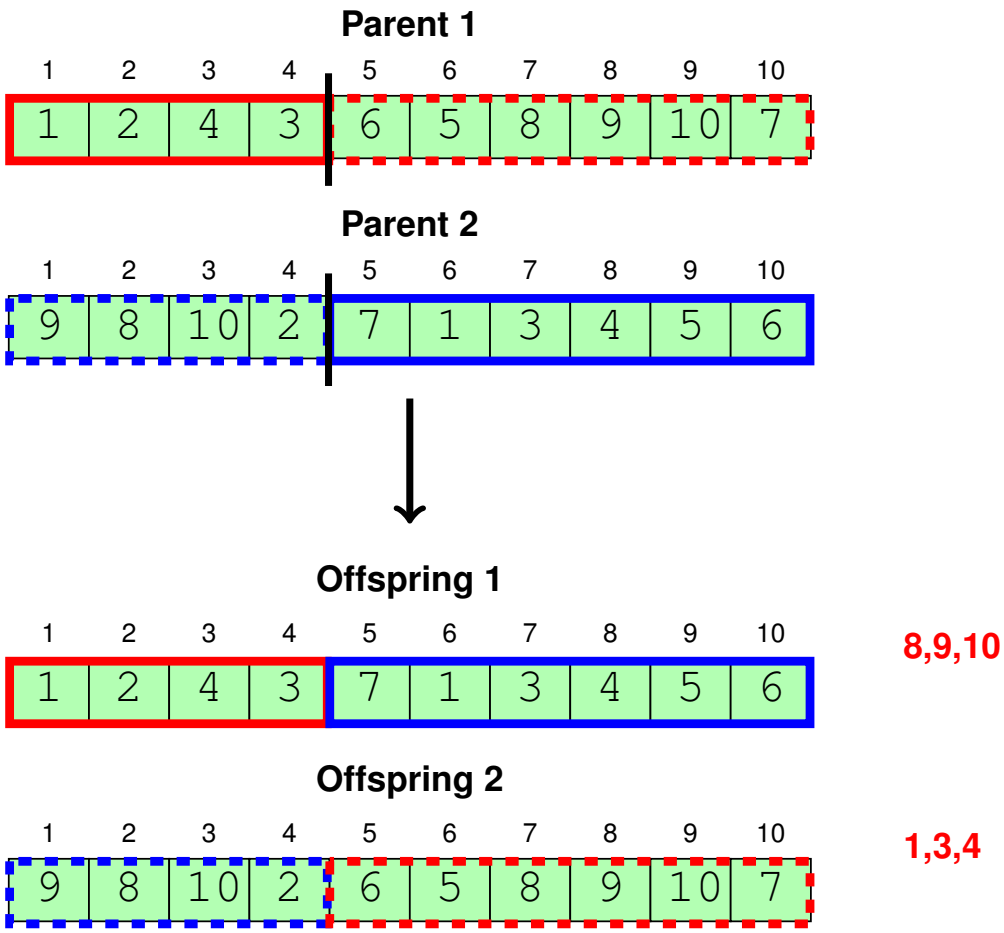
the indices in the array represent the cities and the integer at each index indicates what city comes next in the tour

1	2	3	4	5	6	7	8	9	10
2	4	6	3	8	5	1	9	10	7

Population metaheuristics

Genetic algorithm

Single point crossover operator

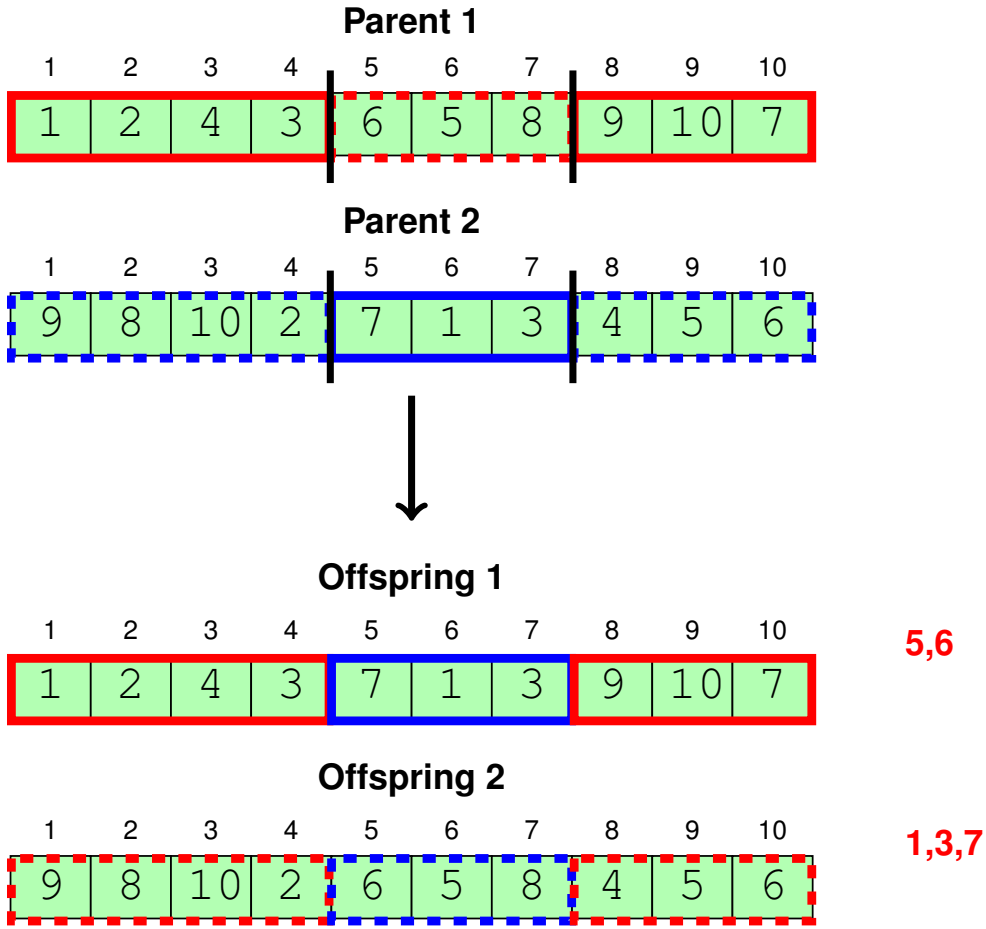


→ Need of **repair operators**

Population metaheuristics

Genetic algorithm

Two-points crossover operator

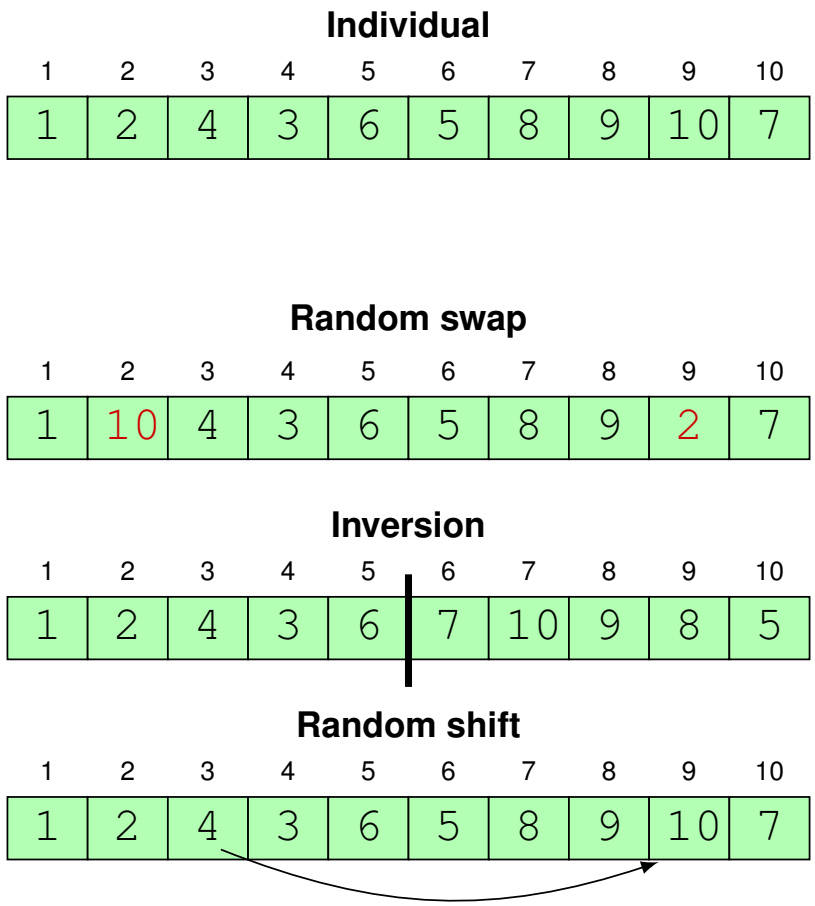


→ Need of **repair operators**

Population metaheuristics

Genetic algorithm

Mutation operator



Population metaheuristics
Genetic algorithm

Capacitated Vehicle routing problem (CVRP)

Data:

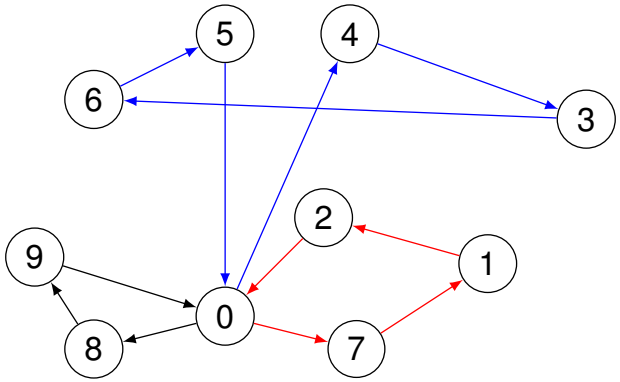
- an unlimited number of homogenous vehicles with cargo capacity Q
- a depot (denoted 0)
- a set $\mathcal{I} = \{1, \ldots, n\}$ of customers to visit with a demand of q_i for each customer i
- c_{ij} : cost of going from i to j (carbon emissions, ...)

Problem: Find the set of tours to be performed by the vehicles in order to minimize the total cost

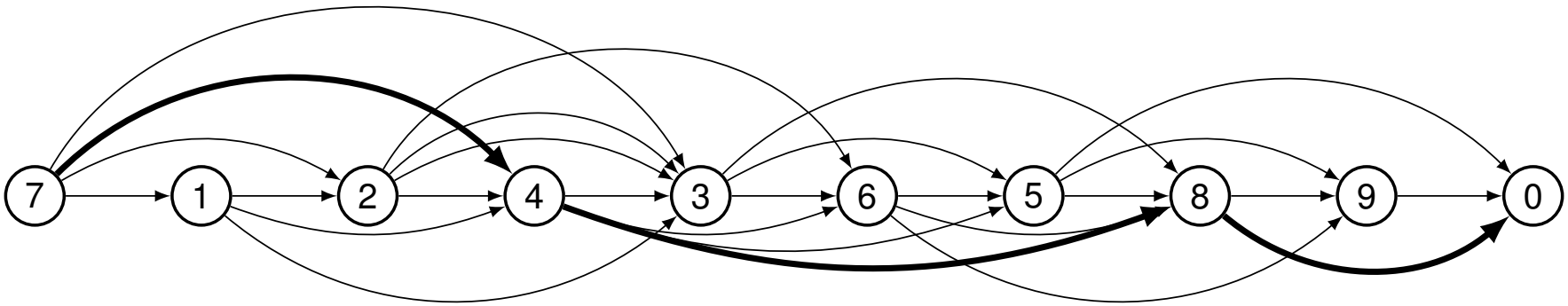
Population metaheuristics

Genetic algorithm

Solution representation for the CVRP:



1	2	3	4	5	6	7	8	9
7	1	2	4	3	6	5	8	9



Tips and Tricks

- Choose a low probability of mutation (mutation = a small perturbation of a solution)
- Select the solutions for crossover (to create new solutions) according to their objective value
- Define measures of "resemblance" between individuals and do not select individuals who "resemble" individuals already in the population
- With a given probability p, apply a local search procedure to every individual of the population (memetic algorithm)

Agenda

Software/Tools

No existing libraries/software for local search algorithms and metaheuristics (to my knowledge)

Some Python modules for genetic algorithms:

- DEAP
- PyGAD
- geneticalgorithm

Other tools that can be used:

- Or-Tools (free)
- LocalSolver (commercial)

Agenda

Matheuristics

Definition

A matheuristic is a method that combines the use of mathematical programming solver with heuristics/metaheuristics.

Aim: Make use of the increased efficiency of mixed integer linear programming solvers.

Matheuristics can be classified according to:

- How heuristics and mathematical programming solvers are combined
  - Collaboration
  - Integration
- What they are used for
  - Constructing a solution
  - Improving a solution

Matheuristics

Construction matheuristics

Examples of matheuristic based on solving mathematical models:

- *Truncated search*: truncate the search space of a mathematical model or use a restrictive stopping criterion
- *LP and fix*: iteratively fix the value of the variable according to their LP values
- *Relax and fix*: impose integrality by decreasing importance and solve sequentially a sequence of MILP models to find a heuristic solution to the original MILP model
- *Relax and repair*: solve a relaxation of the original problem to optimality and repair the obtained solution

Matheuristics

Decomposition-based matheuristics

Idea: Decompose the problem into a sequence of subproblems that should be solved to finally arrive at a complete solution.

- Divide the problem into smaller subproblems easier to solve.
- Solve each subproblem (heuristically or exactly using MILP) Sub-problems can be solved *independently* or *sequentially* (keep constant all decisions related to previously solved subproblems)
- Build the solution of the original problem by combining all subproblem solutions.

Examples for the capacitated vehicle routing problem :
Cluster first-route second approaches
Partial Optimization Metaheuristic Under Special Intensification Conditions (POPMUSIC)

Matheuristics

Improvement matheuristics

Collaboration approach: A MILP model is used to once improve the heuristic solution or to build a part of the solution.

- Improve the solution by revising some heuristic decisions
- Generate parts of the solution from solutions computed by a local search algorithm and try to assemble them to build a complete solution

Improvement matheuristics

Integrated approach: MILP models are used as an intensification tool and solved only at certain steps of the algorithm

- Improving the solution after each local search move by optimizing only a part of the solution
- Recombining solutions

Improvement matheuristics

Integrated approach (continued):

- Exactly Searching Large Neighborhoods
  - Fix and unfix decision variables (LNS)
  - *Local Branching*: find an improving solution in the neighborhood of a given reference solution by adding to the MILP model a constraint that allows at most k binary variables to flip their values
    use the constraint as a branching criterion
  - *Proximity search*: given a reference solution, impose solution improvement as a constraint of the MILP model
    change the objective function to stay close to the reference solution (Hamming distance)
  - *Kernel search*: identify a subset (a *kernel*) of promising decision variables (e.g., using heuristic solutions, LP-relaxation) and then partition the remaining ones into buckets
    concatenate the buckets one at a time to the kernel to check whether improving solutions can be found

THANK YOU

aurelien.froger@u-bordeaux.fr

References

- Puchinger, J., & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In International work-conference on the interplay between natural and artificial computation (pp. 41-53). Springer, Berlin, Heidelberg.
- Glover, F. W., & Kochenberger, G. A. (Eds.). (2006). Handbook of metaheuristics (Vol. 57). Springer Science & Business Media.
- Ball, M. O. (2011). Heuristics based on mathematical programming. Surveys in Operations Research and Management Science, 16(1), 21-38.
- Archetti, C., & Speranza, M. G. (2014). A survey on matheuristics for routing problems. EURO Journal on Computational Optimization, 2(4), 223-246.
- Fischetti, M., & Fischetti, M. (2018). Matheuristics. In Handbook of heuristics (pp. 121-153). Springer.
- Maniezzo, V., Boschetti, M. A., Stützle, T. (2021). Matheuristics, algorithms and implementations. EURO Advanced Tutorials on Operational Research. Springer