Introduction to Optimization

Valentina Cacchiani

Department of Electrical, Electronic and Information Engineering
“Guglielmo Marconi”,
University of Bologna,
Italy

July 2023

Table of contents

- Introduction to Mixed Integer Linear Programming
- MILP models: ideal and improved formulations
- Dynamic Programming and Branch-and-Bound
- Heuristic, metaheuristic and matheuristic algorithms
- Robust optimization
- Seminar: models and heuristic algorithms for real-life applications

Introduction to Mixed Integer Linear Programming

Optimization Problems

- Optimization problem: find the *best* solution out of all feasible solutions of a decision-making problem
- Three main elements:
  - variables: decisions to be made in the problem
  - an objective: the goal to be achieved
  - constraints: given restrictions on the decisions.

Classification

- Deterministic versus with-uncertainty (stochastic, robust)
- Continuous versus Discrete
- Single versus Multiple objectives
- Non-Linear, Convex, Linear functions to define the objective and the constraints

$$\min f(x)$$
$$x \in \mathcal{F}$$

- x: decision variables
- $f(x)$: objective function
- $\mathcal{F}$: feasible set

Mixed Integer Linear Programming

$$\min c^T x$$
$$Ax \geq b$$
$$x \in \mathcal{F}$$

where $\mathcal{F} = \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$

Mixed Integer Linear Programming

$$\begin{aligned} \min c^T x \\ Ax \geq b \\ x \in \mathcal{F} \end{aligned}$$

$$\begin{aligned} \min c^T x = \min c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \\ a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \geq b_1 \\ a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \geq b_2 \\ \ldots \\ a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \geq b_m \\ x_1, x_2, \ldots, x_p \geq 0, \text{integer} \\ x_{p+1}, x_{p+2}, \ldots x_n \geq 0. \end{aligned}$$

Linear Programming

$$\min c^T x$$
$$Ax \geq b$$
$$x \in \mathcal{F}$$

$$\begin{aligned} \min c^T x = \min c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \\ a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \geq b_2 \\ \ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \geq b_m \\ x_1, x_2, \ldots, x_n \geq 0 \end{aligned}$$

Computational complexity

- Computational Complexity of problem P: complexity of the best algorithm for solving any instance of P in the worst case
- Complexity of an Algorithm for P: measure of the time, as a function of the instance size, it takes, in the worst case, to solve any instance of P
- Size of a Problem P: number of input values that define an instance of P

Computational complexity

- Class $\mathcal{P}$ (Polynomial problems): problem P is polynomial if there exists at least one algorithm that can solve P, in the worst case, within a computing time that grows according to a polynomial function of the size of P
- Class $\mathcal{NP}$ (Nondeterministic Polynomial): problems that can be solved in polynomial time through a *non deterministic Turing Machine* (an ideal "lucky" algorithm that always makes the correct choice in a decision tree).

Computational complexity

$\mathcal{NP}$-Complete Problems: problem P is $\mathcal{NP}$-Complete if both conditions hold:

- P belongs to class $\mathcal{NP}$
- There exists an $\mathcal{NP}$-Complete Problem T which is reducible to P ($T \propto P$), i.e.:
  - for any instance t of T, it is possible to define, in polynomial time in the size of t, an instance p of P such that, if the solution of p has been determined, then the solution of t can be obtained in polynomial time in the size of t.

Computational complexity

$\mathcal{NP}$-Hard problems

- These problems are not necessarily in $\mathcal{NP}$ (are as hard as $\mathcal{NP}$-Complete problems)
- These problems P_* have the characteristic that $P_1 \propto P_*$ for all $P_1 \in \mathcal{NP}$
- Hence the existence of a polynomial algorithm for P_* would imply that $\mathcal{P} = \mathcal{NP}$.
- The optimization version of $\mathcal{NP}$-complete problems belongs to the class of $\mathcal{NP}$-hard problems.

Computational complexity

LP and ILP

- LP: there exist algorithm that are polynomial
- ILP: it is $\mathcal{NP}$-complete

Exact versus Heuristic Algorithms

- Exact algorithms are designed to find an optimal solution to the problem and prove it is optimal
- If the problem is in class $\mathcal{P}$, we must derive a polynomial time algorithm to solve it
- When the problem is $\mathcal{NP}$-Hard and/or the size of the instance is large-scale, it might be not possible to compute an optimal solution in reasonable computing time
- In this case, it is appropriate to define a heuristic algorithm that *should* obtain good quality solutions in possibly short computing time.

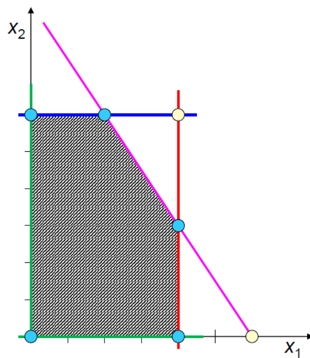
LP: Simplex Algorithm

- LP problems can be solved efficiently by the Simplex Algorithm (although it is not polynomial)
- Theorem: Given an LP problem (F, ϕ) with non empty and bounded (from below for minimization problems) feasible set F and objective function ϕ, there always exists an optimal vertex of F

LP: Feasible region

$$\max z = 3x_1 + 5x_2$$
$$\begin{aligned} s.t. \quad x_1 \quad\quad &\leq 4 \\ 2x_2 &\leq 12 \\ 3x_1 + 2x_2 &\leq 18 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Vertices of F are indicated in light blue

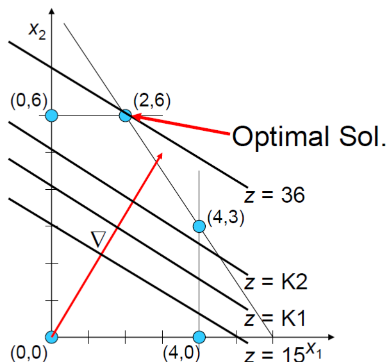


LP: Graphical solution

- Draw lines
  $z = c^T x = constant$
  (perpendicular to the gradient)
- Find the intersection between F and the line associated with max z value

max $z = 3x_1 + 5x_2$

$\nabla = (3, \quad 5)$



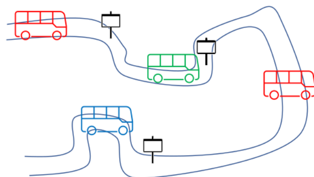
Modelling

- Given a real-world system, determine the system variables, system constraints, and objective(s) for operating the system
- Translate variables and constraints into the form of a mathematical optimization problem: the formulation
- Simplifications: the mathematical optimization problem should be tractable

A real-life example: Bus Scheduling Optimization

- The problem integrates Timetabling and Vehicle Scheduling for a bus company operating in a transport network with electric vehicles

BUS TIMETABLE					
07:45	09:25	11:55	14:50	17:20	20:00
08:00	09:45	12:30	15:10	17:35	20:30
08:10	10:05	12:45	15:30	18:00	21:00
08:20	10:30	13:10	15:55	18:15	21:30
08:45	11:10	13:55	16:45	19:00	22:30



- The problem was proposed by M.A.I.O.R. Company as a challenge of the EU MINOA Project (Mixed Integer Non-linear Optimization: Algorithms and Applications)

Timetabling

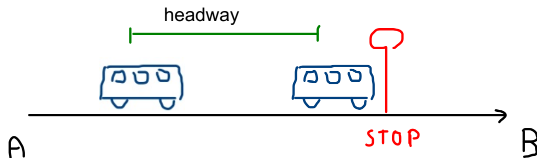
- Determine the bus timetable at each terminal for each line and direction of the transport network.
- This corresponds to selecting a set of trips for each line and direction (each trip corresponds to a line and a direction and has fixed departure and arrival times)
- Desired bus frequency varies along the day.
- Day partitioned into K time windows.

Timetabling Constraints

- For each time window: respect minimum headway and maximum headway

BUS TIMETABLE

07:45	09:25	11:55	14:50	17:20	20:00
08:00	09:45	12:30	15:10	17:35	20:30
08:10	10:05	12:45	15:30	18:00	21:00
08:20	10:30	13:10	15:55	18:15	21:30
08:45	11:10	13:55	16:45	19:00	22:30

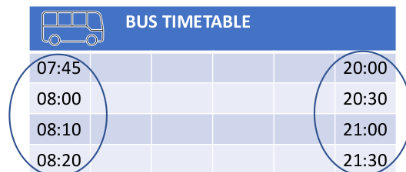


Timetabling Constraints

- For each time window: respect minimum headway and maximum headway

BUS TIMETABLE					
07:45	09:25	11:55	14:50	17:20	20:00
08:00	09:45	12:30	15:10	17:35	20:30
08:10	10:05	12:45	15:30	18:00	21:00
08:20	10:30	13:10	15:55	18:15	21:30
08:45	11:10	13:55	16:45	19:00	22:30

- For each line and direction: select at least one initial and one final trip in given sets



BUS TIMETABLE					
07:45					20:00
08:00					20:30
08:10					21:00
08:20					21:30

Timetabling Objective

Quality of service to the passengers

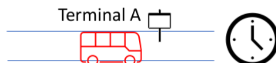
- For each time window: the ideal headway is given
- Minimize the sum, over all consecutive trip pairs, of the relative headway deviation from the ideal one, penalized by a quadratic function.
- The objective function can be linearized!

Vehicle Scheduling

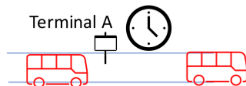
- Determine the sequence of trips (vehicle block) that have to be executed by each vehicle starting and ending at the depot.
- Each vehicle always performs:
  - a pull-in trip (from the depot to the start-terminal of a trip)
  - a pull-out trip (from the end-terminal of a trip to the depot)
- The fleet includes traditional Internal Combustion Engine (ICE) vehicles and electric vehicles.

Vehicle Scheduling Constraints

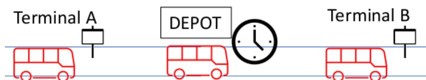
- Minimum and maximum stop time at each terminal



- Trips in sequence for a bus:
  - In-line: enough time (min/max stop) between the two trips

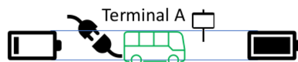


  - Out-line: change terminal only through the depot

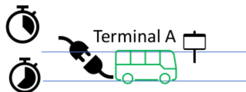


Vehicle Scheduling Constraints

- **Electric vehicles**: maximum number of available buses
- Partial recharges are allowed but with duration at least a given minimum recharging time and respecting the maximum battery level



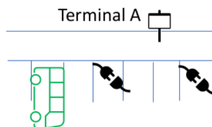
- Fast and slow recharge requiring different recharge time



Vehicle Scheduling Constraints

- Parking/charging constraints: maximum number of parking and charging slots available

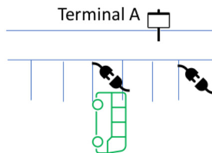
- ICE vehicles can park in a charging slot.
- Electric vehicles can park without charging.
- Moving from a charging to a parking slot and vice versa is allowed but no split of charge



Vehicle Scheduling Constraints

- Parking/charging constraints: maximum number of parking and charging slots available

- ICE vehicles can park in a charging slot.
- Electric vehicles can park without charging.
- Moving from a charging to a parking slot and vice versa is allowed but no split of charge



Vehicle Scheduling Objective

Bus Company costs

- Minimize the total cost given by the sum of:
  - fixed cost (higher for ICE vehicles)
  - cost proportional to the additional stopping time of the vehicle at a terminal but no cost is paid during a recharge
  - cost proportional to the sum of deadhead trip times (pull-in and pull-out trips)
  - a cost for CO_2 emissions proportional to the total driving time (zero for electric vehicles)

MILP models: ideal and improved formulations

Modelling

- Let $\mathcal{F}$ be the set of feasible solutions to the problem
- $\mathcal{F} \subseteq \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$
- A formulation may have auxiliary variables that are not directly required in the problem
- A formulation
  $\mathcal{S} = \{(x, y) \in$
  $(\mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}) \times (\mathbb{Z}_+^t \times \mathbb{R}_+^{n-t}) \quad \text{such that} \quad Ax + Gy \geq b\}$
  is valid if $\mathcal{F} = \text{proj}_x(\mathcal{S})$
- There may be more than one valid formulation
- Different formulations yield different performance of the solution methods
- Finding a good formulation is crucial

Formulation Strength and Ideal Formulations

- Consider two formulations A and B for the same ILP.
- Denote the feasible regions corresponding to their LP relaxations as $\mathcal{P}_A$ and $\mathcal{P}_\mathcal{B}$.
- Formulation A is said to be at least as strong as formulation B if $\mathcal{P}_A \subseteq \mathcal{P}_\mathcal{B}$.
- If the inclusion is strict, then A is stronger than B.
- If $\mathcal{S}$ is the set of all feasible integer solutions for the ILP, then $conv(\mathcal{S}) \subseteq \mathcal{P}_A$.
- A is ideal if $conv(\mathcal{S}) = \mathcal{P}_A$

Different formulations

Example: Knapsack Problem

- Given N items, each with profit p_j and weight w_j ($j \in N$), and the knapsack capacity W, find the maximum profit subset of items whose total weight does not exceed the capacity.
- A formulation:

$$\max \sum_{j \in N} p_j x_j$$

$$\sum_{j \in N} w_j x_j \leq W$$

$$x_j \in \{0, 1\} \qquad j \in N$$

Different formulations

Example: Knapsack Problem - an alternative formulation

- Consider a subset of items $C \subseteq N$ such that $\sum_{j \in C} w_j > W$. This set C is called a cover.
- A cover C is minimal if $\sum_{j \in C \setminus \{i\}} w_j \leq W$ for all $i \in C$.
- An alternative formulation:

$$\max \sum_{j \in N} p_j x_j$$

$$\sum_{j \in C} x_j \leq |C| - 1 \quad \forall \quad \text{minimal covers} \quad C$$

$$x_j \in \{0, 1\} \qquad\qquad j \in N$$

- This formulation has an exponential number of inequalities
- They can be added on the fly: constraint separation

Different formulations

- The previous formulation can be further improved by considering the extension of a minimal cover C: $E(C) = C \cup \{k \in N \setminus C : w_k \geq w_j \; \forall j \in C\}$
- Then, the following inequalities are valid: $\sum_{j \in E(C)} x_j \leq |C| - 1 \quad \forall$ minimal covers $\quad C$

Dynamic Programming and Branch-and-Bound

Dynamic programming

- Is an approach that transforms a complex problem into a sequence of easier problems
- It has the essential feature of dividing an optimization problem into multiple stages, which are solved sequentially (one stage at a time)
- Each stage has an associated state: the state contains the information needed to make the future decisions
- It can be used to solve an optimization problem or to solve a problem appearing as a subproblem of a more complex one

Dynamic programming

- Introduced by R. Bellman. Quoting from his report at RAND Corporation (RAND from "research and development"):
- The theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes
- We have a physical system whose state at any time is determined by a set of quantities called state variables
- At certain times, we are called upon to make decisions which will affect the state of the system
- These decisions are equivalent to transformations of the state variables
- The outcome of the preceding decisions is to be used to guide the choice of the future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state
- A sequence of decisions will be called a policy

Dynamic programming

- A classical approach to these mathematical problems would be to consider the set of all possible sequences of decisions (all feasible policies), compute the return for each such policy, and then maximize the return over the set of all feasible policies
- It is often not practical
- How much information is actually required to carry-out a multi-stage decision process?
- It is sufficient to furnish a general prescription which determines at any stage the decision to be made in terms of the current state of the system

Dynamic programming

- Principle of Optimality: An optimal policy has the property that, whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions.
- System described at any time by vector $p = (p_1, p_2, \ldots, p_M)$ with p constrained to lie in finite region D.
- Consider an N-stage process to be carried out to maximize the N-stage return $R(p)$ of the final state.
- Let $T = \{T_k\}$ be a set of transformations such that $p \in D \to T_k(p) \in D \; \forall k$.

Dynamic programming

- A policy consists of a selection of N transformations $TR = (T_1, T_2, \dots, T_N)$ yielding successively states
  $p_1 = T_1(p)$
  $p_2 = T_2(p_1)$
  $\dots$
  $p_N = T_N(p_{N-1})$.
- A optimal policy leads to the maximum value of the N-stage return $R(p_N)$ determined as a function of the initial vector and the number of stages: $f_N(p) = \max_{TR} R(p_N)$.

Dynamic programming

- We apply the principle of optimality.
- Let p be the initial state and suppose we apply transformation T_k thus obtaining state $T_k(p)$.
- The maximum return from the **following** $N-1$ **stages** is $f_{N-1}(T_k(p))$
- Hence transformation T_k must be chosen such that $f_N(p) = \max_k f_{N-1}(T_k(p))$ for $N = 2, 3, \ldots$

Dynamic programming - Knapsack Problem

- The solution of a problem can be broken down into a sequence of decisions, each associated with a subproblem.
- The original problem solution is divided into stages, and a recurrence relation is found which leads from one stage to the previous one.
- Example: Knapsack Problem: J set of n items, π profit vector, w weight vector, C capacity

$$\max \sum_{j \in J} \pi_j x_j$$

$$\sum_{j \in J} w_j x_j \leq C$$

$$x_j \in \{0, 1\} \qquad j \in J$$

Dynamic programming - Knapsack Problem

- Let z be the residual capacity ($0 \leq z \leq C$) and m an integer representing the number of items ($1 \leq m \leq n$).
- $f_m(z) = \max\{\sum_{i=1}^{m} \pi_i x_i : \sum_{i=1}^{m} w_i x_i \leq z, x_i = 0 \quad \text{or} \quad 1, \forall i = 1, \ldots, m\}$
- We start by solving for one item:
  - $f_1(z) = 0$ if $0 \leq z < w_1$
  - $f_1(z) = \pi_1$ if $w_1 \leq z \leq C$.
- The recursive equations for the mth stage ($m = 2, \ldots, n$) are:

$$f_m(z) = \begin{cases} f_{m-1}(z) & \text{if } 0 \leq z < w_m \\ \max\{\pi_m + f_{m-1}(z - w_m), f_{m-1}(z)\} & \text{if } w_m \leq z \leq C \end{cases}$$

Branch and Bound

Branch and Bound

- It is a recursive and divide-and-conquer approach proposed by Land and Doig.
- Let $\mathcal{S}$ be the feasible set for the MILP $\max_{x \in \mathcal{S}} c^T x$
- Consider a partition of $\mathcal{S}$ into subsets $\mathcal{S}_1, \ldots, \mathcal{S}_k$
- Then: $\max_{x \in \mathcal{S}} c^T x = \max_{\{1 \leq i \leq k\}} \{\max_{x \in \mathcal{S}_i} c^T x\}$ (solve the smaller subproblems recursively)
- Branching: divide the original problem into smaller subproblems.
- Bounding: obtain a bound on the optimal solution value of a subproblem $\max_{x \in \mathcal{S}_i} c^T x$

Branch and Bound

- Solving the bounding problem $\rightarrow x^*$ can achieve two goals:
  - If the solution x^* of the bounding problem is feasible ($x^* \in \mathcal{S}$), then it is a valid lower bound
  - If the upper bound $c^T x^*$ is worse than the best lower bound, we can prune the subproblem
- The easiest way to construct a bounding problem is by considering the Linear Programming relaxation of $\mathcal{S}_i$
- The most frequent branching method is by variable disjunction: select a fractional variable with value x_h^* in the LP solution and create two subproblems by imposing constraints $x_h \leq \lfloor x_h^* \rfloor$ and $x_h \geq \lceil x_h^* \rceil$

Branch and Bound

- The efficiency of the algorithm strongly depends on some algorithmic choices
- The bounding methods: which relaxation to solve? how much effort to compute a good bound on the subproblem?
- The method of branching
- The method of selecting the next candidate subproblem to process (e.g., best first): how much effort to choose a good candidate?
- Two of the most crucial decisions are:
  - variable selection (which fractional variable to branch on)
  - node selection (which of the currently open nodes to process next).
- Preprocessing and heuristics to compute lower bounds

Bounding

- By computing stronger bounds, we can achieve a reduction in tree size but more computing time is required in processing each subproblem.
- To compute a bound, we can apply a relaxation: relax some of the constraints and solve the resulting problem.
- MILP: $\max\{c^T x \quad \text{such that} \quad x \in \mathcal{S}\}$
- $\mathcal{S} = \mathcal{P} \cap (\mathbb{Z}_+^p \times \mathbb{R}_+^{n-p})$ with
  $\mathcal{P} = \{x \in \mathbb{R}^n \quad \text{such that} \quad Ax \leq b\}$
- A relaxation $z_R = \max\{z_R(x) \quad \text{such that} \quad x \in \mathcal{S}_R\}$ has the properties:
  - $\mathcal{S} \subseteq \mathcal{S}_R$
  - $z_R(x) \geq c^T x$, for all $x \in \mathcal{S}$

Bounding

- A relaxation has to be much easier to solve than the original problem and has to provide a good bound
- The easiest way to obtain relaxations is to drop some of the constraints defining the feasible set $\mathcal{S}$.
- A common relaxation is the LP-relaxation.

LP-relaxation

$$(P) \quad \begin{aligned} \min\ & c^T x \\ & Ax \le b \\ & x \in \mathbb{Z}^n_+ \end{aligned} \qquad\qquad (LP) \quad \begin{aligned} \min\ & c^T x \\ & Ax \le b \\ & x \in \mathbb{R}^n_+ \end{aligned}$$

Branch and Bound - Searching the branching tree

In the branch and bound process, we want to:

- quickly find a good integer feasible solution to also prune useless subproblems
- prove that the current solution is optimal

Many heuristic rules for searching the branching tree have been proposed, but no single heuristic dominates the others, and their performance very likely depends on the class of considered problems.

Branch and Bound - Node selection

It is important to decide how to explore the branching tree:

- best-first strategy: in which one always considers the most promising node
- depth-first strategy: in which one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed
- hybrid methods combining the two strategies above (e.g., two-phase methods that alternate depth-first and best-first)
- estimate-based methods that exploit the notion of estimating the value of the best feasible solution that can be derived in a certain subtree
- Machine Learning based branching: for example, learning an efficient approximation of strong branching by supervised learning techniques.

An example

Solve by branch and bound problem P^0:

$$\begin{aligned} \max z = x_1 + 4x_2 \\ 2x_1 - x_2 \geq 0 \\ x_1 + 3x_2 \leq 9 \\ x_1, x_2 \geq 0, \ \textit{integer} \end{aligned}$$

- Select the fractional variable with minimum index
- Explore first the node generated by $\leq$ condition in depth-first strategy
- Solve the LP-relaxation in a graphical way

Heuristic, metaheuristic and matheuristic algorithms

Heuristic Algorithms

- In many practical cases, having a "good" solution is enough
- Even when we aim at finding an optimal solution, a heuristic algorithm can help reduce the computing time
- To evaluate the performance of a heuristic algorithm we should consider:
  - time required to terminate (running time, computational complexity, worst-case performance)
  - required memory
  - quality of the solution found (optimality gap)

Heuristic Algorithms

- Heuristics can be used as a stand-alone method or in combination with an exact method (e.g., branch and bound)
- They can be divided into solution-based or model-based
- Heuristics can also be classified as follows:
  - greedy, constructive
  - local search
  - metaheuristic
  - based on relaxations
  - matheuristic

Solution-based Algorithms

Heuristic Algorithms - Greedy, Constructive

- The solution is constructed step by step starting from an empty solution
- Iteratively, the best element that is feasible is chosen and added to the partial solution
- The choices made cannot be changed
- The best element can be chosen based on scores associated to all elements
- The relevant features are the structure of the solution, its feasibility and the scores

Heuristic Algorithms - Local Search

- It consists of starting from a feasible solution and applying some changes (moves) to try to improve it
- It constructs a set of feasible solutions
- The changes are applied based on the definition of some neighborhood: a set of feasible solutions close to the current one
- A new solution is selected in the neighborhood of the current one if the former improves the latter: for example, one can choose the first improving solution or the best improving one
- If no solution improves the current one, the procedure is terminated
- The local search can terminate in a local optimum
- A relevant feature is the size of the neighborhood

Heuristic Algorithms - Randomized algorithms

- Randomization is a way to achieve diversification
- Multi-start greedy: start from different initial elements and apply a greedy algorithm
- Multi-start combined with local search
- Greedy Randomized Adaptive Search Procedure (GRASP): combines greedy and local search. At each step in the greedy algorithm, instead of selecting the best element, consider the set of k best candidates, and randomly select one of them. Local search is then applied to the computed feasible solution. The process can be iterated to obtain several feasible solutions.
  The greedy is adaptive since the scores of the elements are updated based on the previous iterations.

Heuristic Algorithms - Metaheuristic algorithms

The main drawback of local search algorithms is that they obtain a local minimum.

Metaheuristic algorithms try to overcome this issue:

- Very Large Neighborhood Search (VLNS)
- Simulated Annealing
- Iterated Local Search
- Tabu Search
- Population based heuristic

Heuristic Algorithms - Very Large Neighborhood Search

- A critical issue in Local Search is the choice of the neighborhood
- As a rule of thumb, if we consider a larger neighborhood we can find a better solution
- However, exploring a larger neighborhood requires more computing time.
- Thus it is not guaranteed to find a better solution by exploring a larger neighborhood: it is important to explore it in an efficient way
- VLNS considers a neighborhood whose size is very large (exponential) with respect to the size of the problem
- The neighborhood can be (partially) explored in a heuristic way (e.g., *k*-exchange neighborhood)
- In this category, we can also have the search of the neighborhood performed by network-flow algorithms or dynamic programming algorithms (e.g., neighborhoods defined

Heuristic Algorithms - Simulated Annealing

- The main idea is to allow for occasional moves that produce feasible solutions of higher cost
- It is inspired by the annealing process of metals
- Each solution has a neighborhood: suppose we are in solution x, then we select at random a neighbor solution y
- We compute the difference of the costs $c(y) - c(x)$. If $c(y) \leq c(x)$, we have an improved solution and we move to y

Heuristic Algorithms - Simulated Annealing

- If $c(y) > c(x)$ then we move to y with probability given by: $e^{-\frac{(c(y)-c(x))}{T}}$, where T is a positive constant called temperature.
- If we do not move to y, another random neighbor solution is selected.
- When T is small, cost increase is unlikely. When the difference $c(y) - c(x)$ is large, the probability of accepting a worsening is small.
- The temperature is reduced during the iterations: $T_k = \alpha T_{k1}$, with $\alpha \in [0, 1]$
- The final temperature is selected to that it is very unlikely to select a worsening solution

Heuristic Algorithms - Iterated Local Search

- Consists of three main steps: perturbation, local search, and acceptance criterion
- Start from an initial solution and apply Local Search until reaching a local optimum
- Then, apply a "kick" by perturbation: this perturbation usually consists in a move that was not considered in the Local Search algorithm and allows moving farther from the current solution (e.g., destroy and repair operators). It can also be based on random changes
- Acceptance criterion can be deterministic (e.g., number of not-improving iterations) or based on a probability
- ILS can also accept infeasible solutions in the perturbation step, that can then be made feasible by local search

Heuristic Algorithms - Tabu Search

- The main idea is to accept worsening moves but store a tabu list of moves to avoid cycling on the same solutions
- Given a solution x, it moves to the best neighbor solution y even if it is not improving, but without applying moves of the tabu list
- Then the tabu list is updated
- To avoid that good solutions are discarded by tabu moves, if the new solution is better than the best solution found, then it is accepted (aspiration criteria)
- There can also be softer aspiration criteria which consider how far a non-improving solution is from the best solution found
- A relevant feature is the size of the tabu list and the way of storing the moves/attributes in the tabu list to efficiently check it

Heuristic Algorithms - Population based heuristic

- The most common ones are the Genetic Algorithms
- They consider a population of solutions (instead of a single one)
- The population evolves and part of the population is replaced by new individuals
- A fitness function is used to measure the quality of each individual

Heuristic Algorithms - Population based heuristic

- GA starts by creating an initial population
- Then, three main steps are applied:
  - selection: a subset of the individuals are selected and used to generate new solutions (random selection, higher probability for better individuals)
  - crossover: this is the recombination step, in which subsets of two or more individuals are combined to generate new ones
  - mutation: is applied to a single individual to generate a similar one
- The population usually keeps the same size, hence often the new population includes the best individual of the previous one

Model-based Algorithms

Heuristic Algorithms - Matheuristic algorithms

- Heuristic algorithms based on Mathematical Programming
- Exploit the mathematical formulation of the problem
- Combine an exact solution approach with the definition of neighborhoods
- Relaxation-based algorithms

Heuristic Algorithms based on Relaxations

- LP-based heuristic:
  - iterative rounding of the variables based on the LP-solution values (also called diving): choice of which variables to fix, how many variables
  - rounding of all variables at once
  - rounding with option of backtracking
  - randomized rounding for 0-1 problems (interpret the value of a variable as the probability to chose it in the solution)

Heuristic Algorithms - LP-based Local Search

Local Branching:

- Given a reference solution $\bar{x}$ with $\bar{S} = j \in \mathbb{B} : \bar{x}_j = 1$:
- Explore the k-OPT neighborhood as the set of the feasible solutions satisfying $\Delta(x, \bar{x}) = \sum_{j \in \bar{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus S} x_j \leq k$

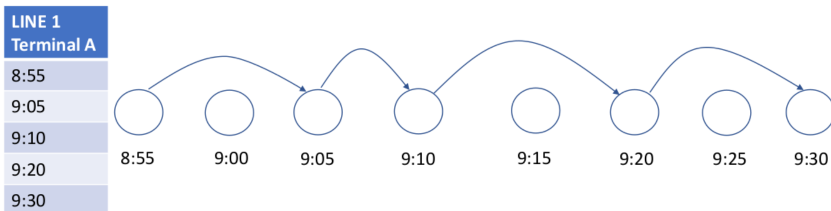
LP-based Heuristic for Bus Scheduling

Heuristic Solution Method

- It is based on an Integer Linear Programming (ILP) model formulated on two graphs:
  - timetabling graph
  - vehicle scheduling graph
- It is not an exact model for the integrated problem:
  - parking/charging constraints are handled heuristically
  - For the electric vehicles:
    - moving from a parking slot to a charging slot and vice versa is now allowed
    - the vehicle recharges always for the maximum available time (up to the maximum battery level) starting its recharge at the first available time instant

Timetabling Graph

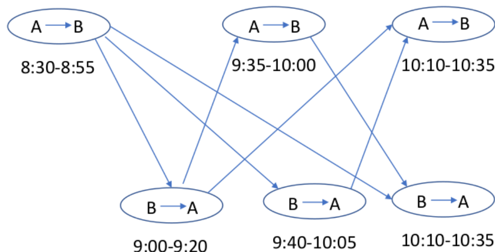
- We adopt the same graph as in Carosi et al. (2019) for the timetabling problem.
- One graph for each line and direction.
- In each graph:
  - one node for each trip (departure time from the terminal)
  - arcs correspond to consecutive trips (departure times) that respect minimum and maximum headways



- Cost linearization: cost of an arc corresponds to the difference between the actual and the ideal headways.

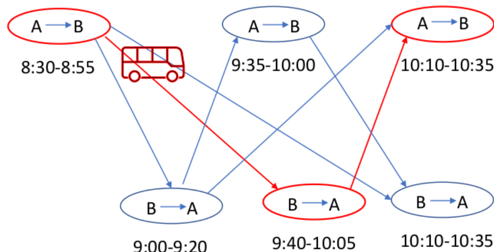
Vehicle Scheduling Graph

- Nodes correspond to trips.
- Arcs correspond to in-line or out-line compatibilities.
- A path in the graph is a vehicle block.



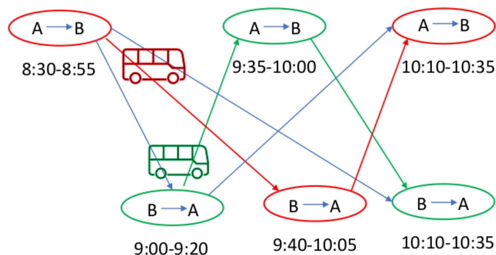
Vehicle Scheduling Graph

- Nodes correspond to trips.
- Arcs correspond to in-line or out-line compatibilities.
- A path in the graph is a vehicle block.



Vehicle Scheduling Graph

- Nodes correspond to trips.
- Arcs correspond to in-line or out-line compatibilities.
- A path in the graph is a vehicle block.



Heuristic Solution Method

- LP-relaxation of the ILP model.
- Generation of vehicle blocks by Dynamic Programming
- It is executed for a given time limit (that depends on the instance size)
- One vehicle block at a time is fixed (variable with the highest fractional value in the LP-solution)
- The corresponding parking/charging slots are occupied.
- The trips incompatible with the selected ones are removed.
- New vehicle blocks are generated but taking into account only the available slots and trips.
- The fixing process is iterated until all timetabling headways are respected and hence we have a feasible solution.