

# CIVIL-557

## Decision-Aid Methodologies in Transportation

### Lecture V

### Metaheuristics II

Fabian Torres

Transport and Mobility Laboratory TRANSP-OR  
École Polytechnique Fédérale de Lausanne EPFL

- 1 Large Neighborhood Search
- 2 ALNS
- 3 Matheuristics
  - Column Generation (CG)

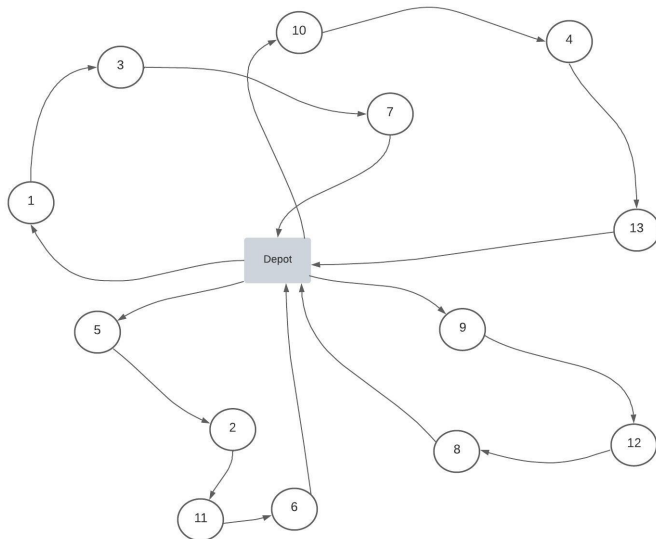
# Outline

- 1 Large Neighborhood Search
- 2 ALNS
- 3 Metaheuristics
  - Column Generation (CG)

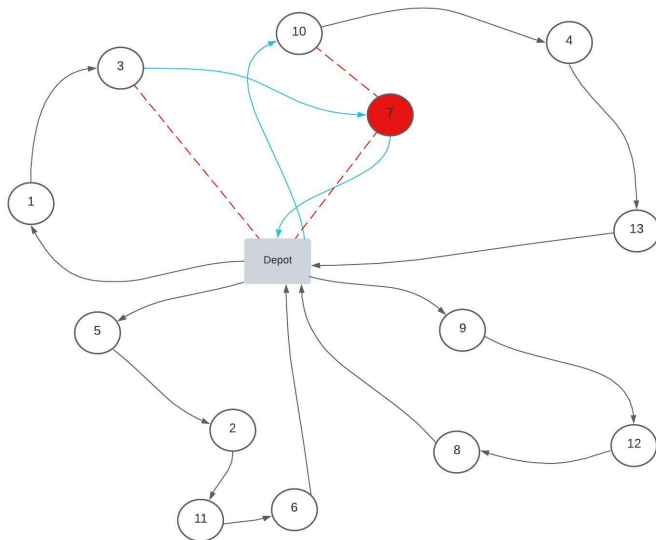
# Neighborhood Search/Local Search

- The neighborhood ( $N_k$ ) of a solution 'w' in the 2-opt neighborhood is the set of solutions that can be reached from 'w' by deleting two edges in 'w' and adding two other edges in order to reconnect the tour.
- Simple example of a neighborhood for the CVRP is the **relocate** neighborhood.
- In this neighborhood,  $N(w)$  is defined as the set of solutions that can be created from w by relocating a single customer. The customer can be moved to another position in its current route or to another route.

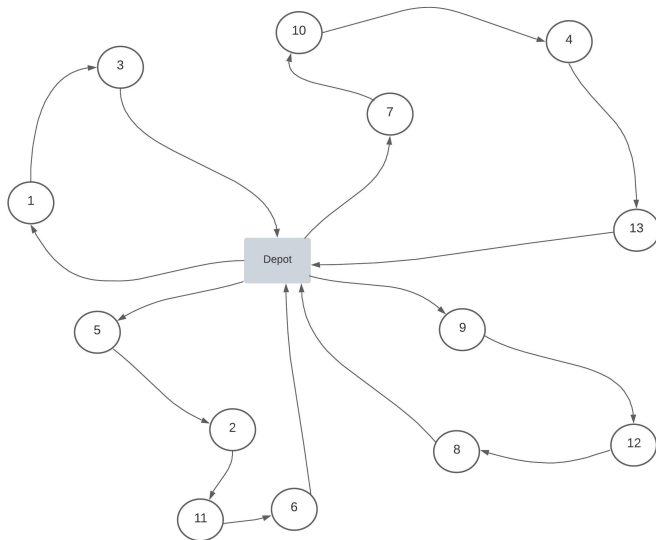
# Relocate heuristic



# Relocate heuristic



# Relocate heuristic



# Small Neighborhood Search

## Neighborhood Size:

- If we take the 2-opt algorithm for the TSP as an example, the total number of solutions that are possible from a given solution 'w' are:

$$\mathcal{O}(n^2)$$

- For the **relocate** algorithm for the VRP, the total number of different solutions that are possible from a given solution 'w' are also:

$$\mathcal{O}(n^2)$$

- These “possible” solutions are the Neighborhood of a given solution 'w'.

**Definition:** A small neighborhood is a neighborhood that has a size of  $\mathcal{O}(n^k)$ , where  $k \leq 3$ .



# Large Neighborhood Search (LNS)

- LNS is based on the **destroy and recreate** idea;
- Some authors use different terms, e.g., ruin and rebuild, destroy and repair, etc. However, the main idea is the same;
- A **destroy heuristic** destroys a part of the current solution;
- A **rebuild heuristic** repairs the destroyed solution.

## Destroy heuristic:

- Can remove a number of customers from a current solution.

## Rebuild heuristic:

- Inserts the removed customers to form a new solution.

# Large Neighborhood Search (LNS)

- Local Search heuristics get stuck in local optima;
- By destroying a solution, LNS can escape local optima and go to a different neighborhood.

**Neighborhood size** In a VRP with 100 customers, if LNS removes 15% of the customers, the number of possible solutions is the following:

$$\binom{100}{15} = \frac{100!}{15! \times 85!} = 2.5 \times 10^{17}$$

That is the reason why this is called **Large Neighborhood Search**

- $x^*$  Initial solution;
- $x_b$  Current solution;
- $f(x)$  The objective function;
- $accept(x_1, x_2)$  Criteria to accept a new solution as a current solution;

---

**Algorithm 1:** Large Neighborhood Search Algorithm

---

**Input:** Initial solution  $x^*$

$x_b \leftarrow x^*$ ; Incumbent solution ;

$x \leftarrow x^*$ ; Current solution ;

**repeat**

$x_k \leftarrow \text{rebuild}(\text{destroy}(x));$

**if**  $\text{accept}(x_k, x)$  **then**

$x \leftarrow x_k;$

**if**  $f(x_k) < f(x_b)$  **then**

$x_b \leftarrow x_k;$

**until** *stopping criterion is met*;

**return**  $x_b$ ;

---

**The acceptance criteria**  $accept(x_k, w)$ : There are different strategies to select an acceptance criteria.

- Hill-climber, if LNS accepts only improving solutions.
- Threshold accepting. Similar to Simulated annealing, select a threshold  $\mathcal{T}_0$ , if a the difference  $f(x_k) - f(w)$  is smaller than  $\mathcal{T}_0$  then we accept the solution.
- Simulated annealing criteria. Variations of SA have been considered where the temperature ' $\mathcal{T}$ ' decreases linearly.

# LNS: Destroy method

The “**degree of destruction**” is the most important choice when implementing the destroy method.

- **Destroy too little:** If LNS removes only one customer from the solution, then LNS would be the same as the relocate heuristic.
  - The LNS algorithm is reduced to a small neighborhood algorithm.
  - LNS will not be able to escape local optima.
- **Destroy too much:** If LNS destroys all the solution, then LNS is a constructive heuristic, e.g., random insertion.
  - We are rebuilding the solution from scratch.
- Balance is required so that the degree of destruction is not too small or too large. Here are some strategies:
  - Increase the degree of destruction gradually.
  - Chose the degree of destruction randomly, from a range that is not too small or too large.

## • Exact Method

- An exact method can reconstruct the partially destroyed solution in the best possible way;
- It could lead to higher quality solutions;
- In some applications it can take too long;

## • Heuristic

- An exact rebuild method can be bad for the diversification of the current solution.
- Heuristic can escape local optima more often since it does not always find the same optima solution.
- Heuristics are fast!

# Outline

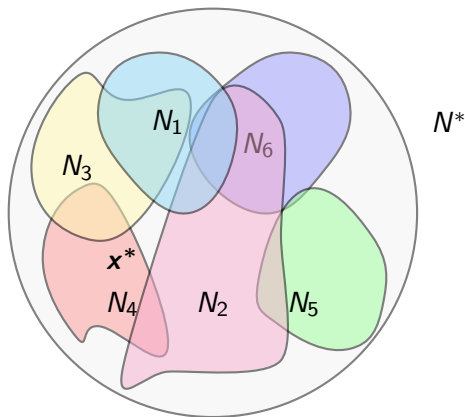
- 1 Large Neighborhood Search
- 2 **ALNS**
- 3 Metaheuristics
  - Column Generation (CG)



# Adaptive Large Neighborhood Search (ALNS)

- ALNS is an extends the LNS heuristic, by allowing multiple **destroy** and multiple **rebuild** methods.
- Each destroy/repair method is assigned a weight that controls how often that particular method is attempted during the search.
- The weights are adjusted dynamically as the search progresses so that the heuristic adapts to the instance at hand and to the state of the search.

# Adaptive Large Neighborhood Search (ALNS)



---

**Algorithm 2:** Adaptive Large Neighborhood Search

---

**Input:** Initial solution  $x^*$

$\rho^- \leftarrow (1, \dots, 1)$ ;  $\rho^+ \leftarrow (1, \dots, 1)$  Initial weights set to 1;

$x_b \leftarrow x^*$ ; Incumbent solution ;

$x \leftarrow x^*$ ; Current solution ;

**repeat**

    select destroy and repair methods  $d \in \Omega^-$  and  $r \in \Omega^+$  using  $\rho^-$   
    and  $\rho^+$ ;

$x_k \leftarrow r(d(x))$ ; Destroy and rebuild ;

**if**  $\text{accept}(x_k, x)$  **then**

$x \leftarrow x_k$ ;

**if**  $f(x_k) < f(x_b)$  **then**

$x_b \leftarrow x_k$ ;

    update  $\rho^-$  and  $\rho^+$ ;

**until** *stopping criterion is met*;

**return**  $x_b$ ;

---

# ALNS: Probability of selection

- $\Omega^-$  Set of all destroy methods;
- $\Omega^+$  Set of all rebuild methods;
- $\phi_j^-$  Probability of choosing destroy method “j”;
- $\rho_j^-$  Weight of destroy method “j”;
- $\phi_j^+$  Probability of choosing rebuild method “j”;
- $\rho_j^+$  Weight of rebuild method “j”;

The probability to choose the “j” destroy method is the following:

$$\phi_j^- = \frac{\rho_j^-}{\sum_{k=1}^{|\Omega^-|} \rho_k^-}$$

# ALNS: Reward and Update

**Calculate the reward of each heuristic used:**

$$\psi = \max \begin{cases} \omega_1 & \text{if the new solution is a new global best,} \\ \omega_2 & \text{if the new solution is better than the current one,} \\ \omega_3 & \text{if the new solution is accepted,} \\ \omega_4 & \text{if the new solution is rejected,} \end{cases}$$

$$\omega_1 \geq \omega_2 \geq \omega_3 \geq \omega_4 \geq 0$$

**Update weights:**

$$\rho_j^- = \lambda \rho_j^- + (1 - \lambda) \psi, \quad \rho_i^+ = \lambda \rho_i^+ + (1 - \lambda) \psi$$

Where  $\lambda$  is the decay parameter that controls how sensitive the weights are to changes to the performance of the heuristics.

- The weights of the heuristics that were not used remain unchanged.
- The goal is to select heuristics that work well for the problem being solved.
- Coupled neighborhoods: Some remove heuristics and rebuild heuristics go well together, or might be incompatible together. In such cases, one may define a set of rebuild heuristics that can be used with a specific destroy heuristic.

# ALNS: Time vs quality

- If a heuristic  $h_1$  is faster than a heuristic  $h_2$ , it can be used for hundreds of iterations for a single iteration of the slower heuristic  $h_2$ .
- The rewards in ALNS favor heuristics that find higher quality solutions, even if they require a large number of iterations in comparison to faster heuristics that require less iterations.
- The quality of the solution quality has to be evaluated with time. For example, if 100 iterations of  $h_1$ , on average, finds better solutions than a single iteration of  $h_2$ , and 100 iterations takes the same time as 1 iteration of  $h_2$ , then,  $h_1$  is better.
- If all heuristics take the same time then it is not a problem.
- Otherwise, one can adjust the score  $\psi$  with a measure of time consumption.

# Outline

- 1 Large Neighborhood Search
- 2 ALNS
- 3 Matheuristics
  - Column Generation (CG)



**Matheuristic** are a family of metaheuristics that combine powerful exact methods like MIP and metaheuristics.

Examples:

- LNS, can use an exact method to rebuild the partially destroyed solution.
- Variable fixing: In a Branch-and-Bound algorithm, find a variable that is fractional and set to one, resolve without branching and fix another fractional value to one, until a feasible solution is found.
- Column generation: Solve the pricing problem and convert the relaxation problem to an MIP and solve.

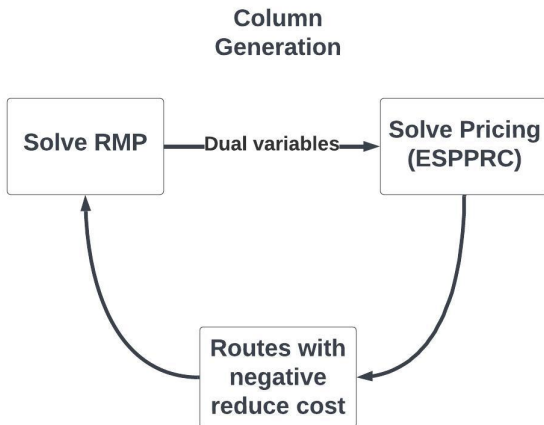
# Master Problem: The set-covering formulation

- The **Set-covering formulation** is solved efficiently with commercial solvers, e.g., Gurobi or CPLEX.

## Restricted Master problem (RMP)

$$\begin{aligned} & \text{Minimize } \sum_{r \in \bar{\Omega}} c_r \lambda_r \\ & \text{s.t. } \sum_{r \in \bar{\Omega}} a_{ir} \lambda_r \geq 1 && \forall i \in N, \\ & \lambda_r \in \{0, 1\} && \forall r \in \bar{\Omega}. \end{aligned}$$

# Column Generation

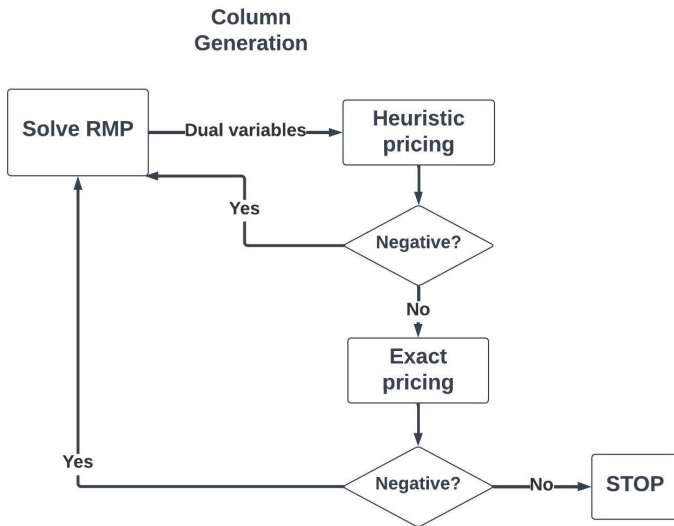


# Heuristic Pricing

It is not necessary to produce the shortest path in every iteration of CG. All that is needed is a path with a negative reduced cost.

- The exact method to find the shortest path can be time consuming;
- Dual values can be unstable at the start of the algorithm taking many iterations to stabilize;
- Fast heuristics can be used to find a negative reduced cost route;
- The exact method for the pricing problem only needs to be used to prove optimality of the master problem.

# Column Generation



# CG: Pricing Heuristic

**Recall** the dominance rules **and** Feasibility check for the exact method of the ESPPRC

**Dominance rules:** Label  $\mathcal{L}_1 = (i_1, c_1, T_1, Q_1, V_1, L_1)$  dominates label  $\mathcal{L}_2 = (i_2, c_2, T_2, Q_2, V_2, L_2)$  if the following is true:

- ①  $i_1 = i_2$
- ②  $c_1 \leq c_2$
- ③  $T_1 \leq T_2$
- ④  $Q_1 \leq Q_2$
- ⑤  $V_1 \subseteq V_2 \leftarrow$  complicating rules!

## Feasible extension

- if  $j^n \in V^e$ , **Then** the extension is not feasible.  
or
- if  $T^e + t_{(i^e, j^n)} > b_{j^n}$ , **Then** the extension is not feasible.  
or
- if  $Q^e + q_{j^n} > Q$ , **Then** the extension is not feasible.

## Change dominance rules and **keep** feasibility checks

- 1  $i_1 = i_2$
  - 2  $c_1 \leq c_2$
  - 3  $T_1 \leq T_2$
  - 4  $Q_1 \leq Q_2$
- if  $j^n \in V^e$ , **Then** the extension is not feasible.  
**or**
  - if  $T^e + t_{(i^e, j^n)} > b_{j^n}$ , **Then** the extension is not feasible.  
**or**
  - if  $Q^e + q_{j^n} > Q$ , **Then** the extension is not feasible.

- Eliminating complicating dominance rules makes the algorithm terminate in pseudo-polynomial time. However, there is no guarantee that the solution is optimal.
- Keeping all feasibility checks guarantee that the solution (if found) will be feasible.
- If more dominance rules are eliminated the algorithm will terminate faster.

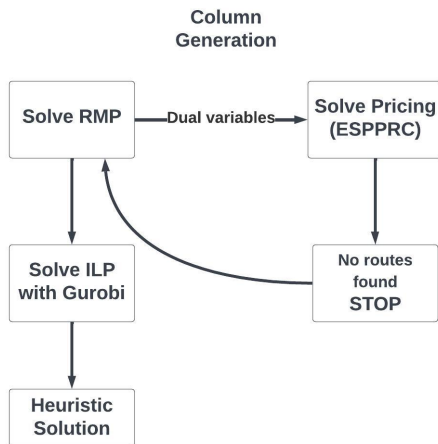
**Strategy:** Develop multiple heuristics and order them from fastest to slowest. The fast heuristics produce solutions of smaller quality while the slower ones will be able to find a negative column more easily.

- Start by using the fastest heuristic to produce columns.
- If the fastest heuristic found a negative column, add to and resolve the LRMP.
- If the fastest heuristic fails to find a negative column, move to the second fastest, etc.
- If all heuristics have failed to produce a negative column then use the exact method.



# Column Generation heuristic

Commercial solvers, e.g., Gurobi, can solve set partitioning models quickly.



# Column Generation heuristic

The solution is **not necessarily** optimal. To find the optimal solution we must use Branch-and-Price, i.e., every time we branch we must solve the pricing problem again.

- The RMP with a restricted number of routes will most likely not have the optimal routes in the subset of columns.
- Commercial solvers do not have the feature of adding columns in the middle of branching.
- If the exact method was used in the last iteration of CG then we at least get a lower bound for the problem.

# Column Generation: Variable fixing

Another strategy widely used is to fix fractional variables to an integer, instead of branching.

- 1 Solve the MP, find a fractional variable close to 1, e.g., 0.95, and set equal to 1;
- 2 Solve pricing problem until no negative column is found.
- 3 Select another fractional variable that is close to 1 and set equal to 1.
- 4 Repeat until you find a feasible solution.

At any point we can stop the algorithm and turn the RMP into an IP and solve the problem with a commercial solver. The set of routes contained in the restricted set of routes will be larger, and increase the probability of finding better solutions.

- Wolsey, L. A. (1998). *Integer programming*. Wiley.
- Gendreau, M., & Potvin, J.-Y. (Eds.). (2010). *Handbook of metaheuristics*. Springer.