Exact and Heuristic Solution Methods for a VRP with Time Windows and Variable Service Start Time

Y. Arda, H. Küçükaydin, Y. Crama, S. Michelini

QuantOM - HEC - Université de Liège

ORBEL 29
February 5th, 2014

Table of Contents

Table of Contents

Starting scenario

- Our problem: a capacitated VRP with time windows, with additional key features.

Starting scenario

- Our problem: a capacitated VRP with time windows, with additional key features.
- Route cost depends on total route duration,

Starting scenario

- Our problem: a capacitated VRP with time windows, with additional key features.
- Route cost depends on total route duration,
- Variable starting time for each route,

Starting scenario

- Our problem: a capacitated VRP with time windows, with additional key features.
- Route cost depends on total route duration,
- Variable starting time for each route,
- Max allotted time for each route.

A classic solution method for the rich VRP: Branch-and-Price

- At each node of the branch-and-bound tree, the linear relaxation of the set-covering formulation is solved via column generation.

[1] Dror 1994.

A classic solution method for the rich VRP: Branch-and-Price

- At each node of the branch-and-bound tree, the linear relaxation of the set-covering formulation is solved via column generation.
- The pricing sub-problem is an elementary shortest path problem with resource constraints (ESPPRC).

[1] Dror 1994.

A classic solution method for the rich VRP: Branch-and-Price

- At each node of the branch-and-bound tree, the linear relaxation of the set-covering formulation is solved via column generation.
- The pricing sub-problem is an elementary shortest path problem with resource constraints (ESPPRC).
- If the underlying graph may have negative cost cycles, the ESPPRC is strongly NP-Hard[1].

[1] Dror 1994.

Exact Dynamic Programming for the ESPPRC [2]

- Each state associated to vertex i represents a path from the source s to i.

[2] Developed by Feillet et al. 2004, based on Desrochers and Soumis 1988; improvements are in Righini and Salani 2008.

Exact Dynamic Programming for the ESPPRC [2]

- Each state associated to vertex i represents a path from the source s to i.
- Each state includes a resource consumption vector R whose component R_r represents the quantity of resource r used along the path.

[2] Developed by Feillet et al. 2004, based on Desrochers and Soumis 1988; improvements are in Righini and Salani 2008.

Exact Dynamic Programming for the ESPPRC [2]

- Each state associated to vertex i represents a path from the source s to i.
- Each state includes a resource consumption vector R whose component R_r represents the quantity of resource r used along the path.
- Each state has an associated cost C and the optimal solution corresponds to a minimum cost state associated to the sink t.

[2] Developed by Feillet et al. 2004, based on Desrochers and Soumis 1988; improvements are in Righini and Salani 2008.

Exact Dynamic Programming for the ESPPRC [2]

- Each state associated to vertex i represents a path from the source s to i.
- Each state includes a resource consumption vector R whose component R_r represents the quantity of resource r used along the path.
- Each state has an associated cost C and the optimal solution corresponds to a minimum cost state associated to the sink t.
- Extension of a state from i to j corresponds to adding the arc (i, j) to a path from s to i.

[2] Developed by Feillet et al. 2004, based on Desrochers and Soumis 1988; improvements are in Righini and Salani 2008.

Exact Dynamic Programming for the ESPPRC [2]

- Each state associated to vertex i represents a path from the source s to i.
- Each state includes a resource consumption vector R whose component R_r represents the quantity of resource r used along the path.
- Each state has an associated cost C and the optimal solution corresponds to a minimum cost state associated to the sink t.
- Extension of a state from i to j corresponds to adding the arc (i, j) to a path from s to i.
- We terminate when all states have been extended in all feasible ways.

[2] Developed by Feillet et al. 2004, based on Desrochers and Soumis 1988; improvements are in Righini and Salani 2008.

Exact Dynamic Programming for the ESPPRC

- While extending states, we update the resource consumption values.

Exact Dynamic Programming for the ESPPRC

- While extending states, we update the resource consumption values.
- E.g., if we extend to j we update the amount q_i relative to capacity

$$q_j = q_i + d_j,$$

where d_j is the demand at j.

Exact Dynamic Programming for the ESPPRC

- While extending states, we update the resource consumption values.
- E.g., if we extend to j we update the amount q_i relative to capacity

$$q_j = q_i + d_j,$$

where d_j is the demand at j.

- To enforce feasibility with regards to capacity, we need to check if $q_j \leq Q$.

Exact Dynamic Programming for the ESPPRC

- While extending states, we update the resource consumption values.
- E.g., if we extend to j we update the amount q_i relative to capacity

$$q_j = q_i + d_j,$$

  where d_j is the demand at j.
- To enforce feasibility with regards to capacity, we need to check if $q_j \leq Q$.
- To enforce elementarity, we introduce a dummy unitary resource El_k, which is consumed when vertex k is visited.

Exact Dynamic Programming for the ESPPRC

- To accelerate the algorithm, we eliminate the states that are dominated:

Exact Dynamic Programming for the ESPPRC

- To accelerate the algorithm, we eliminate the states that are dominated:

Dominance rules

State $(C', R', (\mathrm{EI}_k)'_{k\in V}, i)$ dominates $(C'', R'', (\mathrm{EI}_k)''_{k\in V}, i)$ iff

$$C' \leq C''$$
$$R' \leq R''$$
$$(\mathrm{EI}_k)'_{k\in V} \leq (\mathrm{EI}_k)''_{k\in V}$$

and at least one of the equalities is strict.

Table of Contents

Problem description

- For each vertex we have:

Problem description

- For each vertex we have:
  - a time window $[a_i, b_i]$,

Problem description

- For each vertex we have:
  - a time window $[a_i, b_i]$,
  - service time s_i,

Problem description

- For each vertex we have:
  - a time window $[a_i, b_i]$,
  - service time s_i,
  - delivery demand d_i,

Problem description

- For each vertex we have:
  - a time window $[a_i, b_i]$,
  - service time s_i,
  - delivery demand d_i,
  - a revenue (dual price) η_i.

Problem description

- For each vertex we have:
  - a time window $[a_i, b_i]$,
  - service time s_i,
  - delivery demand d_i,
  - a revenue (dual price) η_i.
- There is a single vehicle available at any time for a duration $\mathcal{S}$.

Problem description

- For each vertex we have:
  - a time window $[a_i, b_i]$,
  - service time s_i,
  - delivery demand d_i,
  - a revenue (dual price) η_i.
- There is a single vehicle available at any time for a duration $\mathcal{S}$.
- The total cost of a path P depends on total distance D_P and total travel time T_P.

Problem description

- For each vertex we have:
  - a time window $[a_i, b_i]$,
  - service time s_i,
  - delivery demand d_i,
  - a revenue (dual price) η_i.
- There is a single vehicle available at any time for a duration $\mathcal{S}$.
- The total cost of a path P depends on total distance D_P and total travel time T_P.
- We aim to find the service start time T_s and path P that minimize the total cost:

$$C_P(T_s) = \alpha D_P + \beta T_P(T_s) - \sum_{i \in P} \eta_i.$$

Resources for DP

- We need a service start time resource T_i, so that the state at i is feasible iff $T_i \in [a_i, b_i]$;

Resources for DP

- We need a service start time resource T_i, so that the state at i is feasible iff $T_i \in [a_i, b_i]$;
- a delivery demand resource Del_i, requiring $\mathrm{Del}_i \in [0, Q]$;

Resources for DP

- We need a service start time resource T_i, so that the state at i is feasible iff $T_i \in [a_i, b_i]$;
- a delivery demand resource Del_i, requiring $\mathrm{Del}_i \in [0, Q]$;
- a total spent time resource S_i, requiring $S_i \in [0, S]$;

Resources for DP

- We need a service start time resource T_i, so that the state at i is feasible iff $T_i \in [a_i, b_i]$;
- a delivery demand resource Del_i, requiring $\mathrm{Del}_i \in [0, Q]$;
- a total spent time resource S_i, requiring $S_i \in [0, S]$;
- a dummy resource $(\mathrm{El}_k)^i_{k \in V}$, requiring $(\mathrm{El}_k)^i \in [0, 1], \ \forall k \in V$.

Resources for DP

- We need a service start time resource T_i, so that the state at i is feasible iff $T_i \in [a_i, b_i]$;
- a delivery demand resource Del_i, requiring $\mathrm{Del}_i \in [0, Q]$;
- a total spent time resource S_i, requiring $S_i \in [0, S]$;
- a dummy resource $(\mathrm{El}_k)^i_{k \in V}$, requiring $(\mathrm{El}_k)^i \in [0, 1]$, $\forall k \in V$.
- A DP state for vertex i in our scenario is therefore

$$(C_i, T_i, S_i, \mathrm{Del}_i, (\mathrm{El}_k)^i_{k \in V}).$$

Dominance rules: issue with time dependency

- T_i, S_i, and the total cost of the subpath s-i C_i clearly depend on the starting time T_s.

Dominance rules: issue with time dependency

- T_i, S_i, and the total cost of the subpath s-i C_i clearly depend on the starting time T_s.
- The DP state for i in our scenario then becomes

$$(C_i(T_s), T_i(T_s), S_i(T_s), \mathrm{Del}_i, (\mathrm{El}_k)^i_{k \in V}).$$

Dominance rules: issue with time dependency

- T_i, S_i, and the total cost of the subpath s-i C_i clearly depend on the starting time T_s.
- The DP state for i in our scenario then becomes

$$(C_i(T_s), T_i(T_s), S_i(T_s), \mathrm{Del}_i, (\mathrm{El}_k)^i_{k \in V}).$$

- We must therefore take into account an infinite number of Pareto-optimal states.

Dominance rules: issue with time dependency

- T_i, S_i, and the total cost of the subpath s-i C_i clearly depend on the starting time T_s.
- The DP state for i in our scenario then becomes

$$(C_i(T_s), T_i(T_s), S_i(T_s), \mathrm{Del}_i, (\mathrm{El}_k)^i_{k \in V}).$$

- We must therefore take into account an infinite number of Pareto-optimal states.
- We can't apply directly normal dominance rules.

Time functions

- We will consider a path on a network with n vertices:

$$P : s = 0 \to \cdots \to i - 1 \to i \to \cdots \to n + 1 = t$$

.

- Let us define the adjusted travel time: $\bar{t}_{i-1,i} := t_{i-1,i} + s_{i-1}$

Time functions

- We will consider a path on a network with n vertices:

$$P : s = 0 \to \cdots \to i-1 \to i \to \cdots \to n+1 = t$$

.

- Let us define the adjusted travel time: $\bar{t}_{i-1,i} := t_{i-1,i} + s_{i-1}$
- The minimum travel time from $s = 0$ to i: $\theta_i := \sum_{k=0}^{i-1} \bar{t}_{k,k+1}$

Time functions

- We will consider a path on a network with n vertices:

$$P : s = 0 \to \cdots \to i-1 \to i \to \cdots \to n+1 = t$$

.

- Let us define the adjusted travel time: $\bar{t}_{i-1,i} := t_{i-1,i} + s_{i-1}$
- The minimum travel time from $s = 0$ to i: $\theta_i := \sum_{k=0}^{i-1} \bar{t}_{k,k+1}$
- The latest feasible start time from the source: $l_i := \min_{1 \le j \le i}\{b_j - \theta_j\}$

Time functions

- We will consider a path on a network with n vertices:

$$P : s = 0 \rightarrow \cdots \rightarrow i-1 \rightarrow i \rightarrow \cdots \rightarrow n+1 = t$$

.

- Let us define the adjusted travel time: $\bar{t}_{i-1,i} := t_{i-1,i} + s_{i-1}$
- The minimum travel time from $s = 0$ to i: $\theta_i := \sum_{k=0}^{i-1} \bar{t}_{k,k+1}$
- The latest feasible start time from the source: $l_i := \min_{1 \leq j \leq i}\{b_j - \theta_j\}$
- The earliest feasible service start time at vertex i: $\tilde{a}_i := \max\{a_i, \tilde{a}_{i-1} + \bar{t}_{i-1,i}\}$.

Time functions

- From the recursion $T_i(T_s) = \max\{a_i, T_{i-1}(T_s) + \bar{t}_{i-1,i}\}$ we can derive:

Time functions

- From the recursion $T_i(T_s) = \max\{a_i, T_{i-1}(T_s) + \bar{t}_{i-1,i}\}$ we can derive:

Description of the service start time function

For all i, if $\tilde{a}_i < l_i + \theta_i$,

$$T_i(T_s) = \begin{cases} \tilde{a}_i, & \text{if } T_s \leq \tilde{a}_i - \theta_i, \\ T_s + \theta_i, & \text{if } \tilde{a}_i - \theta_i \leq T_s \leq l_i; \end{cases}$$

otherwise

$$T_i(T_s) = \tilde{a}_i, \text{ for } T_s \leq l_i$$

Time functions

- From the recursion $T_i(T_s) = \max\{a_i, T_{i-1}(T_s) + \bar{t}_{i-1,i}\}$ we can derive:

Description of the service start time function

For all i, if $\tilde{a}_i < l_i + \theta_i$,

$$T_i(T_s) = \begin{cases} \tilde{a}_i, & \text{if } T_s \leq \tilde{a}_i - \theta_i, \\ T_s + \theta_i, & \text{if } \tilde{a}_i - \theta_i \leq T_s \leq l_i; \end{cases}$$

otherwise

$$T_i(T_s) = \tilde{a}_i, \text{ for } T_s \leq l_i$$

- They are piecewise linear functions from which the other time-dependent functions derive directly.

New Dominance Rules and Resource Extension

- We can now define new labels and their resource extension functions:

New Dominance Rules and Resource Extension

- We can now define new labels and their resource extension functions:

$$
\begin{aligned}
-l_i &= -\min\{l_{i-1}, b_i - \theta_i\} \\
\tilde{a}_i &= \max\{a_i, \tilde{a}_{i-1} + \overline{t}_{i-1,i}\} \\
A_i &= \max\{A_{i-1} + \beta\overline{t}_{i-1,i}, \beta(\tilde{a}_i - l_i)\} \\
\delta_i &= \delta_{i-1} + \alpha c_{i-1,i} - \eta_{i-1} \\
\mathrm{Del}_i &= \mathrm{Del}_{i-1} + d_i \\
\mathrm{El}_k^i &= \begin{cases} \mathrm{El}_k^{i-1} + 1 & \text{if} k = i \\ \mathrm{El}_k^{i-1} & \text{otherwise} \end{cases} \qquad \forall k \in V
\end{aligned}
$$

where $\theta_i = \theta_{i-1} + \overline{t}_{i-1,i}$ and A_i is the minimum value of the function $T_i(T_s)$.

Bidirectional DP

- To accelerate the procedure, we start it simultaneously from the sink, extending states *backwards*.

Bidirectional DP

- To accelerate the procedure, we start it simultaneously from the sink, extending states *backwards*.
- It suffices to invert the time windows with a constant M and change direction of the arcs, then use monodirectional DP:

$$[a_i, b_i] \Rightarrow [M - b_i, M - a_i], (i, j) \Rightarrow (j, i)$$

Bidirectional DP

- To accelerate the procedure, we start it simultaneously from the sink, extending states *backwards*.
- It suffices to invert the time windows with a constant M and change direction of the arcs, then use monodirectional DP:

$$[a_i, b_i] \Rightarrow [M - b_i, M - a_i], (i, j) \Rightarrow (j, i)$$

- States are extended until the total amount of time spent is smaller than $S/2$, i.e. we consider total travel time as a *critical resource*.

Bidirectional DP

- To accelerate the procedure, we start it simultaneously from the sink, extending states *backwards*.
- It suffices to invert the time windows with a constant M and change direction of the arcs, then use monodirectional DP:

$$[a_i, b_i] \Rightarrow [M - b_i, M - a_i], (i, j) \Rightarrow (j, i)$$

- States are extended until the total amount of time spent is smaller than $S/2$, i.e. we consider total travel time as a *critical resource*.
- The earliest feasible service start time becomes the latest feasible service end time: $M - \tilde{a}_i^b = \tilde{b}_i$.

Bidirectional DP

- To accelerate the procedure, we start it simultaneously from the sink, extending states *backwards*.
- It suffices to invert the time windows with a constant M and change direction of the arcs, then use monodirectional DP:

$$[a_i, b_i] \Rightarrow [M - b_i, M - a_i], (i, j) \Rightarrow (j, i)$$

- States are extended until the total amount of time spent is smaller than $S/2$, i.e. we consider total travel time as a *critical resource*.
- The earliest feasible service start time becomes the latest feasible service end time: $M - \tilde{a}_i^b = \tilde{b}_i$.
- The latest feasible start time from the depot becomes the earliest feasible arrival time at the depot: $M - l_i^b = e_i$.

Path concatenation

- To see if we obtain a feasible path this needs to be true:

[3] Savelsbergh 1992.

Path concatenation

- To see if we obtain a feasible path this needs to be true:

$$\begin{aligned} \tilde{a}_i &\leq \tilde{b}_i \\ \mathrm{Del}_i + \mathrm{Del}_i^b - d_i &\leq Q \\ \mathrm{El}_k^i + \mathrm{El}_k^{ib} &\leq 1, \forall k \in V \setminus \{i\} \\ T_P &\leq S, \end{aligned}$$

  where T_P is the total travel time of path P obtained by concatenation.

- We need a *concatenation theorem*[3] to compute the actual total travel time T_P - we can't sum the partial times directly.

[3] Savelsbergh 1992.

Table of Contents

DP improvements[4]: Duplicate Elimination

- During the phase of concatenation of forward and backward labels, the same path can be generated multiple times.

[4] Described in Righini and Salani 2008.

DP improvements[4]: Duplicate Elimination

- During the phase of concatenation of forward and backward labels, the same path can be generated multiple times.
- The path $P = s \rightarrow \cdots \rightarrow j \rightarrow i \rightarrow k \rightarrow \cdots \rightarrow t$ can be obtained by concatenating different pairs of labels, e.g. (l_i^{fw}, l_i^{bw}) or (l_j^{fw}, l_j^{bw}).

[4]Described in Righini and Salani 2008.

DP improvements[4]: Duplicate Elimination

- During the phase of concatenation of forward and backward labels, the same path can be generated multiple times.
- The path $P = s \rightarrow \cdots \rightarrow j \rightarrow i \rightarrow k \rightarrow \cdots \rightarrow t$ can be obtained by concatenating different pairs of labels, e.g. (l_i^{fw}, l_i^{bw}) or (l_j^{fw}, l_j^{bw}).
- Before each concatenation at i we check the forward and backward consumption of the critical resource, $R_{r,i}^{\text{fw}}$ and $R_{r,i}^{\text{bw}}$.

[4] Described in Righini and Salani 2008.

DP improvements[4]: Duplicate Elimination

- During the phase of concatenation of forward and backward labels, the same path can be generated multiple times.
- The path $P = s \rightarrow \cdots \rightarrow j \rightarrow i \rightarrow k \rightarrow \cdots \rightarrow t$ can be obtained by concatenating different pairs of labels, e.g. (l_i^{fw}, l_i^{bw}) or (l_j^{fw}, l_j^{bw}).
- Before each concatenation at i we check the forward and backward consumption of the critical resource, $R_{r,i}^{\text{fw}}$ and $R_{r,i}^{\text{bw}}$.
- We accept it only if they are as close as possible to half of the overall consumption of the resource along the path, i.e. iff $\Phi_i := |R_{r,i}^{\text{fw}} - R_{r,i}^{\text{bw}}|$ is minimum.

[4] Described in Righini and Salani 2008.

DP improvements[4]: Duplicate Elimination

- During the phase of concatenation of forward and backward labels, the same path can be generated multiple times.
- The path $P = s \to \cdots \to j \to i \to k \to \cdots \to t$ can be obtained by concatenating different pairs of labels, e.g. (l_i^{fw}, l_i^{bw}) or (l_j^{fw}, l_j^{bw}).
- Before each concatenation at i we check the forward and backward consumption of the critical resource, $R_{r,i}^{\text{fw}}$ and $R_{r,i}^{\text{bw}}$.
- We accept it only if they are as close as possible to half of the overall consumption of the resource along the path, i.e. iff $\Phi_i := |R_{r,i}^{\text{fw}} - R_{r,i}^{\text{bw}}|$ is minimum.
- The test is performed in constant time since we need only to check Φ_k if $R_{r,i}^{\text{fw}} < R_{r,i}^{\text{bw}}$ or Φ_j otherwise.

[4] Described in Righini and Salani 2008.

DP improvements: Decremental State Space Relaxation

- In **State Space Relaxation**[5] we project the state-space $\mathcal{S}$ used in DP to a lower dimensional space $\mathcal{T}$, so that the new states retain the cost.

[5] Developed by Christofides, Mingozzi, and Toth 1981.

DP improvements: Decremental State Space Relaxation

- In **State Space Relaxation**[5] we project the state-space $\mathcal{S}$ used in DP to a lower dimensional space $\mathcal{T}$, so that the new states retain the cost.
- When applying this to the elementarity constraints, the number of states to explore is reduced, at the cost of feasibility.

[5] Developed by Christofides, Mingozzi, and Toth 1981.

DP improvements: Decremental State Space Relaxation

- In **State Space Relaxation**[5] we project the state-space $\mathcal{S}$ used in DP to a lower dimensional space $\mathcal{T}$, so that the new states retain the cost.
- When applying this to the elementarity constraints, the number of states to explore is reduced, at the cost of feasibility.
- **Decremental** State Space Relaxation (DSSR) is a generalization of both this method and DP with elementarity constraints.

[5] Developed by Christofides, Mingozzi, and Toth 1981.

DP improvements: Decremental State Space Relaxation

- In **State Space Relaxation**[5] we project the state-space $\mathcal{S}$ used in DP to a lower dimensional space $\mathcal{T}$, so that the new states retain the cost.
- When applying this to the elementarity constraints, the number of states to explore is reduced, at the cost of feasibility.
- **Decremental** State Space Relaxation (DSSR) is a generalization of both this method and DP with elementarity constraints.
- We maintain a set Θ of **critical** nodes on which the elementarity constraints are enforced at each iteration of DP.

[5] Developed by Christofides, Mingozzi, and Toth 1981.

DP improvements: Decremental State Space Relaxation

- In **State Space Relaxation**[5] we project the state-space $\mathcal{S}$ used in DP to a lower dimensional space $\mathcal{T}$, so that the new states retain the cost.
- When applying this to the elementarity constraints, the number of states to explore is reduced, at the cost of feasibility.
- **Decremental** State Space Relaxation (DSSR) is a generalization of both this method and DP with elementarity constraints.
- We maintain a set Θ of **critical** nodes on which the elementarity constraints are enforced at each iteration of DP.
- If at the end of DP the optimal path is not feasible, we update Θ with the nodes that are visited multiple times.

[5] Developed by Christofides, Mingozzi, and Toth 1981.

DSSR strategies

- In the implementation of DSSR we can make decisions with regards to:

DSSR strategies

- In the implementation of DSSR we can make decisions with regards to:
  - initialization of the critical vertex set;

DSSR strategies

- In the implementation of DSSR we can make decisions with regards to:
  - initialization of the critical vertex set;
  - which vertices we insert in the set at the end of an iteration;

DSSR strategies

- In the implementation of DSSR we can make decisions with regards to:
  - initialization of the critical vertex set;
  - which vertices we insert in the set at the end of an iteration;
  - how many elementary paths we want to obtain for the CG procedure.

DSSR strategies

- In the implementation of DSSR we can make decisions with regards to:
  - initialization of the critical vertex set;
  - which vertices we insert in the set at the end of an iteration;
  - how many elementary paths we want to obtain for the CG procedure.
- These decisions involve trade-offs (e.g. cost of an iteration vs number of iterations).

DSSR strategies

- In the implementation of DSSR we can make decisions with regards to:
    - initialization of the critical vertex set;
    - which vertices we insert in the set at the end of an iteration;
    - how many elementary paths we want to obtain for the CG procedure.
- These decisions involve trade-offs (e.g. cost of an iteration vs number of iterations).
- We can associate parameters to these decisions, which we can then tune. parameters.

Table of Contents

Matheuristics

- **Matheuristics** are 'heuristics algorithms made by the interoperation of metaheuristics and mathematical programming techniques'.[6]

[6] Boschetti et al. 2009.

[7] Archetti and Speranza 2014.

Matheuristics

- **Matheuristics** are ‘heuristics algorithms made by the interoperation of metaheuristics and mathematical programming techniques’.[6]
- For routing problems, we can classify them in three classes[7].

[6] Boschetti et al. 2009.

[7] Archetti and Speranza 2014.

Matheuristics

- **Matheuristics** are ‘heuristics algorithms made by the interoperation of metaheuristics and mathematical programming techniques’.[6]
- For routing problems, we can classify them in three classes[7].
  - **Decomposition approaches**: we identify subproblems that are solved independently, then combine their solutions. E.g. *Cluster first-route second* approaches.

[6] Boschetti et al. 2009.

[7] Archetti and Speranza 2014.

Matheuristics

- **Matheuristics** are 'heuristics algorithms made by the interoperation of metaheuristics and mathematical programming techniques'.[6]
- For routing problems, we can classify them in three classes[7].
    - **Decomposition approaches**: we identify subproblems that are solved independently, then combine their solutions. E.g. *Cluster first-route second* approaches.
    - **Improvement heuristics**: by solving a MILP, we improve an heuristic solution.

[6] Boschetti et al. 2009.

[7] Archetti and Speranza 2014.

Matheuristics

- **Matheuristics** are 'heuristics algorithms made by the interoperation of metaheuristics and mathematical programming techniques'.[6]
- For routing problems, we can classify them in three classes[7].
  - **Decomposition approaches**: we identify subproblems that are solved independently, then combine their solutions. E.g. *Cluster first-route second* approaches.
  - **Improvement heuristics**: by solving a MILP, we improve an heuristic solution.
  - **Branch-and-Price based approaches**, classified in *restricted master heuristics*, *heuristic branching* approaches, and *relaxation based* approaches.

[6] Boschetti et al. 2009.

[7] Archetti and Speranza 2014.

Restricted Master Heuristics

- The optimal solution of the master problem restricted to any subset of generated columns provides an heuristic solution.

[8] Joncour et al. 2010.

[9] Danna and Le Pape 2005.

Restricted Master Heuristics

- The optimal solution of the master problem restricted to any subset of generated columns provides an heuristic solution.
- The columns can either be generated heuristically or by solving exactly the pricing problem.

[8] Joncour et al. 2010.

[9] Danna and Le Pape 2005.

Restricted Master Heuristics

- The optimal solution of the master problem restricted to any subset of generated columns provides an heuristic solution.
- The columns can either be generated heuristically or by solving exactly the pricing problem.
- However, the master problem defined over a subset of columns is often infeasible[8], so we have to adopt techniques to recover feasibility or devise ways to obtain a suitable set of columns.

[8] Joncour et al. 2010.

[9] Danna and Le Pape 2005.

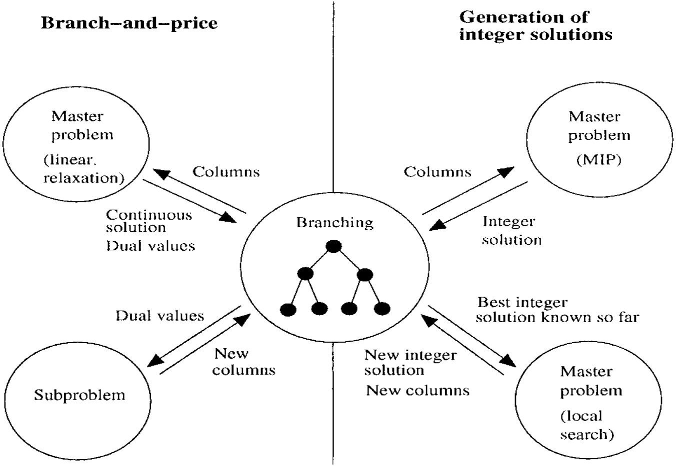
Restricted Master Heuristics

- The optimal solution of the master problem restricted to any subset of generated columns provides an heuristic solution.
- The columns can either be generated heuristically or by solving exactly the pricing problem.
- However, the master problem defined over a subset of columns is often infeasible[8], so we have to adopt techniques to recover feasibility or devise ways to obtain a suitable set of columns.
- Within the BP framework, we can use the RMH in a collaboration scheme with a metaheuristic[9], in order to obtain good solutions early in the procedure.

[8] Joncour et al. 2010.

[9] Danna and Le Pape 2005.

Collaboration scheme[10]



[10] Image from Danna and Le Pape 2005.

Thanks for your attention.

References I

Claudia Archetti and M Grazia Speranza. "A survey on matheuristics for routing problems". In: *EURO Journal on Computational Optimization* 2.4 (2014), pp. 223–246.

Andrea Bettinelli, Alberto Ceselli, and Giovanni Righini. "A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows". In: *Transportation Research Part C: Emerging Technologies* 19.5 (2011), pp. 723–740.

Marco A Boschetti et al. "Matheuristics: Optimization, simulation and control". In: *Hybrid Metaheuristics*. Springer, 2009, pp. 171–177.

Nicos Christofides, Aristide Mingozzi, and Paolo Toth. "State-space relaxation procedures for the computation of bounds to routing problems". In: *Networks* 11.2 (1981), pp. 145–164.

Emilie Danna and Claude Le Pape. "Branch-and-price heuristics: A case study on the vehicle routing problem with time windows". In: *Column Generation*. Springer, 2005, pp. 99–129.

Guy Desaulniers and Daniel Villeneuve. "The shortest path problem with time windows and linear waiting costs". In: *Transportation Science* 34.3 (2000), pp. 312–319.

References II

Martin Desrochers and François Soumis. "A generalized permanent labeling algorithm for the shortest path problem with time windows". In: *INFOR Information Systems and Operational Research* (1988).

Moshe Dror. "Note on the complexity of the shortest path models for column generation in VRPTW". In: *Operations Research* 42.5 (1994), pp. 977–978.

Dominique Feillet et al. "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems". In: *Networks* 44.3 (2004), pp. 216–229.

Cédric Joncour et al. "Column generation based primal heuristics". In: *Electronic Notes in Discrete Mathematics* 36 (2010), pp. 695–702.

Giovanni Righini and Matteo Salani. "New dynamic programming algorithms for the resource constrained elementary shortest path problem". In: *Networks* 51.3 (2008), pp. 155–170.

Martin WP Savelsbergh. "The vehicle routing problem with time windows: Minimizing route duration". In: *ORSA journal on computing* 4.2 (1992), pp. 146–154.