Cyclic Giant Tour Decoding for the EVRPTW

Christopher Bacher, Günther Raidl

bacher@ac.tuwien.ac.at

Algorithms and Complexity Group
TU Wien
July 6^{th}, 2016

Electric Vehicle Routing Problem

with Time Windows (EVRPTW)

- Introduced by Schneider et al. (2014)
- **Minimization of**
  - Number of vehicles (primary)
  - Travel distance (secondary)
- **Constraints**
  - Time windows
  - Vehicle capacity
  - Battery charge
- Additional **recharging stations**

Motivation

- Solution encoding as **cyclic** permutation of customers
- **Enhance** local search heuristics
- Use a decoder to **optimally**
  - **Insert** recharging stations
  - **Split** giant tour
- Use **Algebraic Dynamic Programming** for specification and solving
- **Reuse** information from decoding process

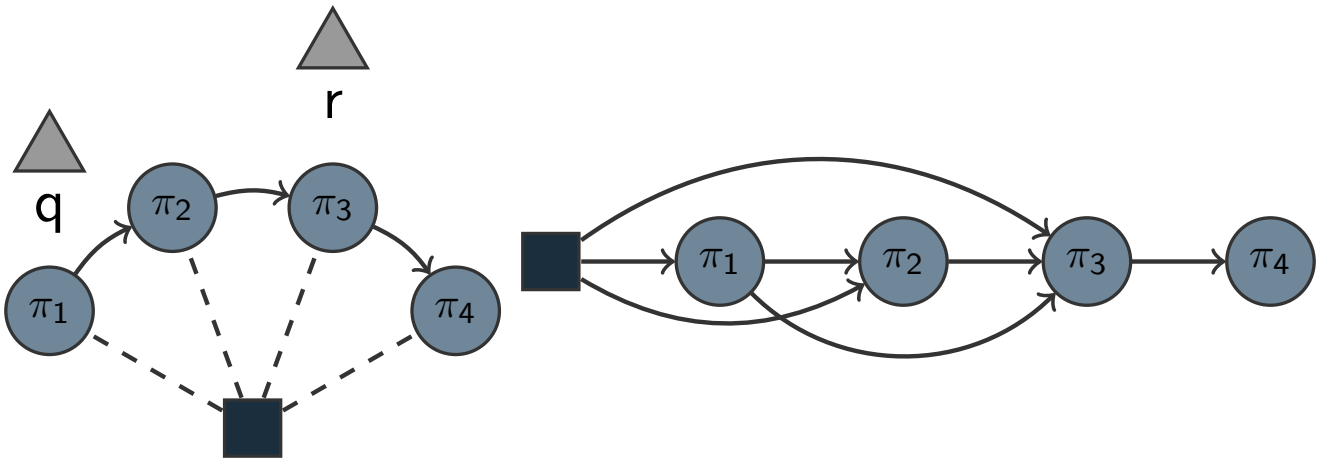
Related Work

- Prins et al. (2004)
  Split with shifts algorithm
- Montoya et al. (2015/6)
  Giant tour decoder for the G-VRP
  EVRPTW version seems to be in work
- Hiermann et al. (2016)
  EVRPTW tour decoder (work in progress)

Giant Tour Decoding for the EVRPTW

(aka Route-First-Cluster-Second)

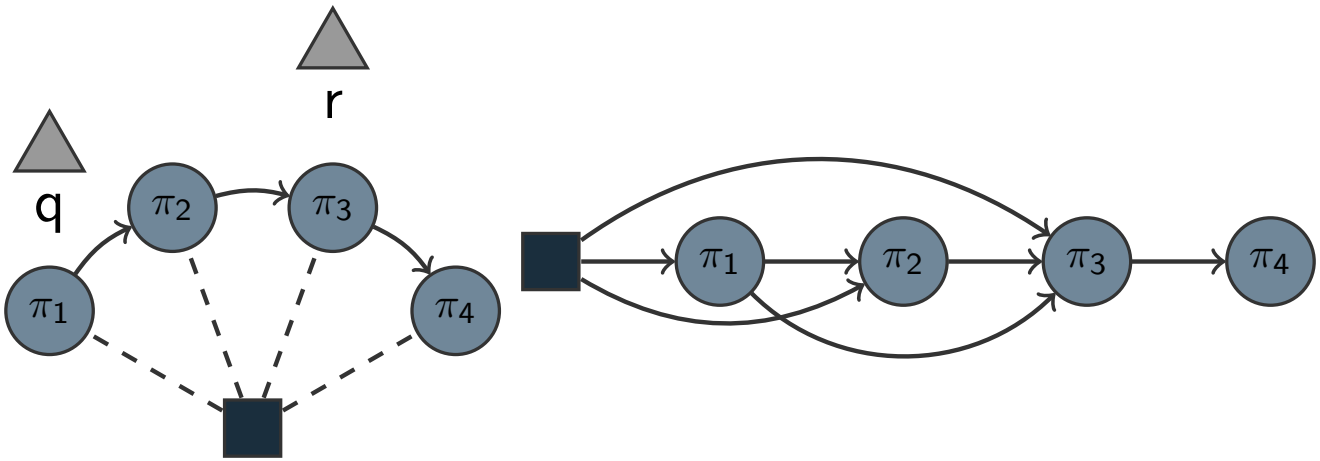
- Extension of *Split* algorithm for G-VRP by Montoya et al. (2015/6)
- Optimal decoding into tours and insertion of recharging stations
- Basically a Dynamic Programming algorithm



Giant Tour Decoding for the EVRPTW

(aka Route-First-Cluster-Second)

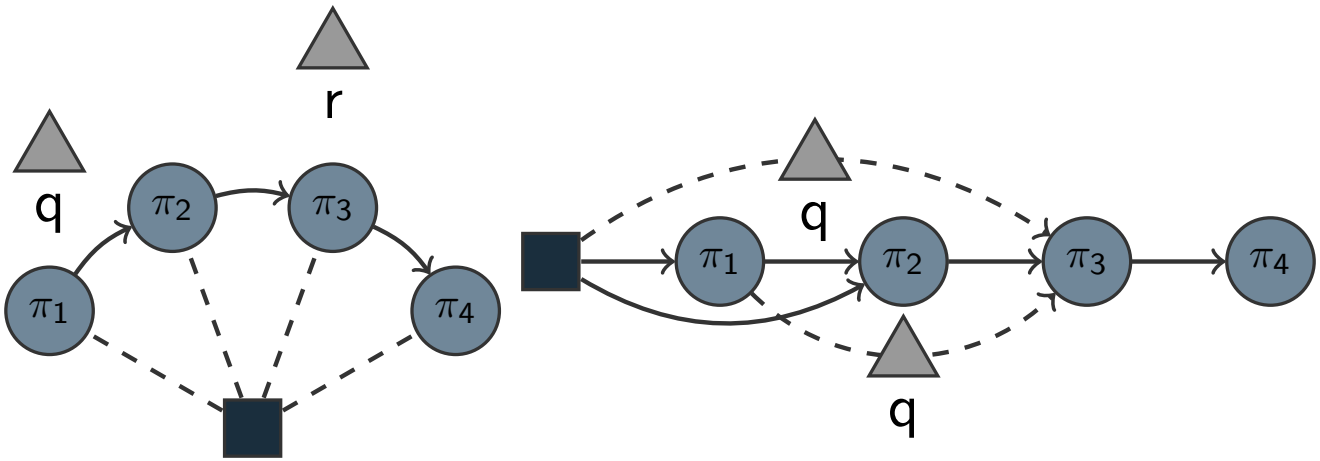
- Extension of *Split* algorithm for G-VRP by Montoya et al. (2015/6)
- Optimal decoding into tours and insertion of recharging stations
- Basically a Dynamic Programming algorithm



Giant Tour Decoding for the EVRPTW

(aka Route-First-Cluster-Second)

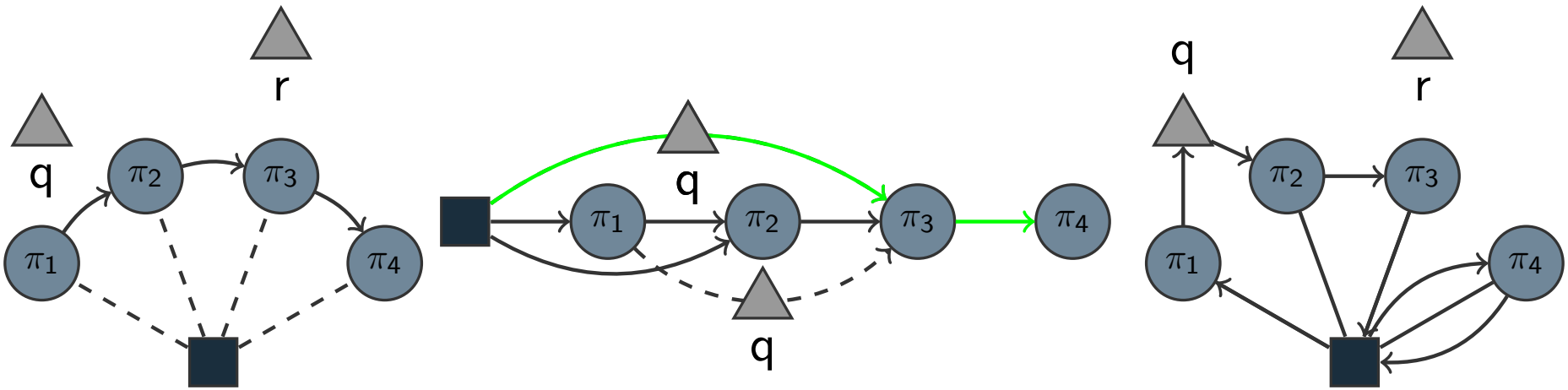
- Extension of *Split* algorithm for G-VRP by Montoya et al. (2015/6)
- Optimal decoding into tours and insertion of recharging stations
- Basically a Dynamic Programming algorithm



Giant Tour Decoding for the EVRPTW

(aka Route-First-Cluster-Second)

- Extension of *Split* algorithm for G-VRP by Montoya et al. (2015/6)
- Optimal decoding into tours and insertion of recharging stations
- Basically a Dynamic Programming algorithm



Enhancing Giant Tour Decoding

Instead of decoding a **permutation** . . .

Enhancing Giant Tour Decoding

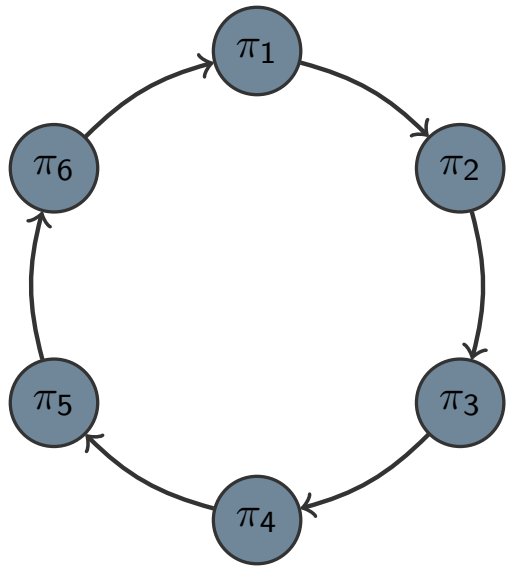
Instead of decoding a **permutation** . . .

. . . decode a **cyclic permutation**

- At least as good as a permutation

But . . .

- Naive implementation: $O(n * O(\text{Split}))$
- Repair of full infeasible tour
- Split recalculation on each change



How to deal with . . .

- Capacity constraints
- Time window constraints
- Battery constraints
- Recharging

. . . in a Dynamic Program **more easily and systematically**?

Algebraic Dynamic Programming (ADP)

Alternative view on Dynamic Programmning (Giegerich et al., 2002)

- Formal grammar defines the search space by **decomposition**
- Separate **evaluation algebra**
- Separate search space **traversal**
- Works for sequence data—originally intended for bioinformatics

Algebraic Dynamic Programming (ADP)

Alternative view on Dynamic Programmning (Giegerich et al., 2002)

- Formal grammar defines the search space by **decomposition**
- Separate **evaluation algebra**
- Separate search space **traversal**
- Works for sequence data—originally intended for bioinformatics

Why should we use this?

Why Algebraic Dynamic Programming (ADP)?

- Grammar naturally captures **recursive** structure
  - Terminals capture input
  - Non-Terminals are compound objects, states, and tables
- **No clutter**—evaluation defined afterwards
- No index handling (needs to be sacrificied later)

Why Algebraic Dynamic Programming (ADP)?

- Grammar naturally captures **recursive** structure
  - Terminals capture input
  - Non-Terminals are compound objects, states, and tables
- **No clutter**—evaluation defined afterwards
- No index handling (needs to be sacrificied later)

Moreover it provides . . .

- . . . **clean methodology**
- . . . **common language**

Why Algebraic Dynamic Programming (ADP)?

- Grammar naturally captures **recursive** structure
  - Terminals capture input
  - Non-Terminals are compound objects, states, and tables
- **No clutter**—evaluation defined afterwards
- No index handling (needs to be sacrificied later)

Moreover it provides . . .

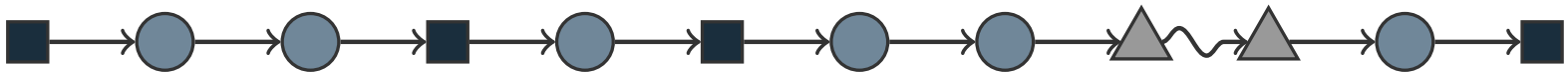
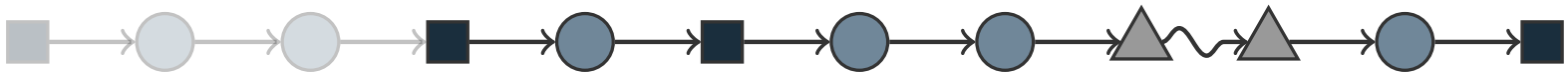
- . . . **clean methodology**
- . . . **common language**
- . . . General Purpose Solver Software for Dynamic Programming

ADP for Giant Tour Decoding needs . . .

. . . some **non-standard extensions**

- **Real**-valued indices: battery levels & time windows
- **Explicit** indices & calculations: cyclicity, recharge time, . . .
- **Constraints** on indices: easy exclusion of infeasible solutions
- **Multiple objectives**: vehicles & distance
- **Incremental evaluation** for efficiency

A Cyclic Decoder for EVRPTW: Grammar (I)

Symbol	Name	Represents
S	Start symbol	Represent all feasible tour decompositions 
R	Routes	At least one further route/path 
P	Path/Route	Path to the depot via next customer 
$F^{(q,r)}$	Recharging chain	Path to the depot via recharging chain 
C	Vehicle capacity	Required vehicle capacity for next *Path*

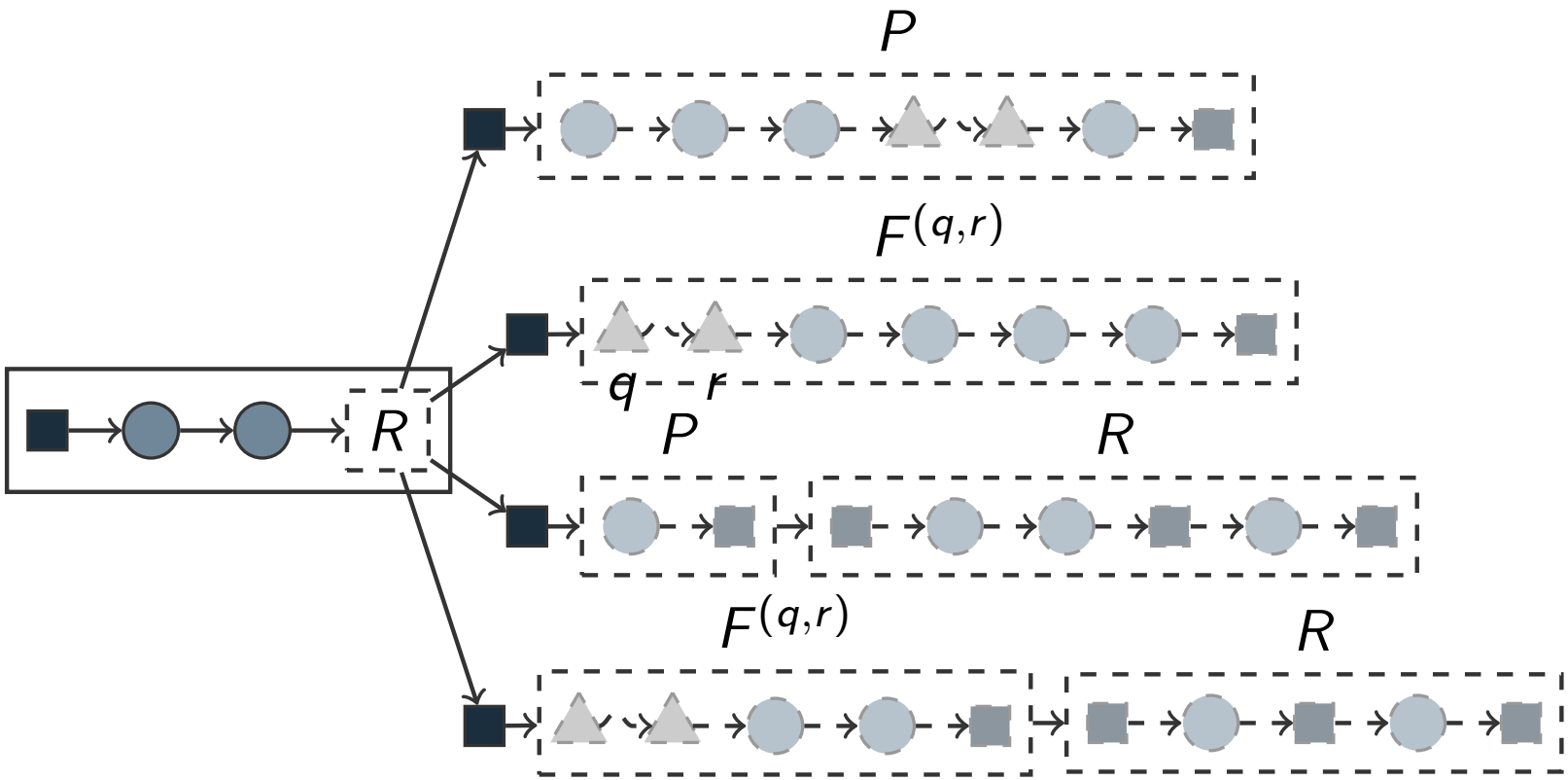
A Cyclic Decoder for EVRPTW: Grammar (II)

We consider a cyclic encoding for the EVRPTW ...

$$
\begin{aligned}
S &\rightarrow R && \text{(Start)} \\
R &\rightarrow C0P && \text{(Customer)} \\
&\mid C0F^{q,r} && \forall q, r \in F \text{ (Charge)} \\
&\mid C0PR && \text{(Split)} \\
&\mid C0F^{q,r}R && \forall q, r \in F \text{ (Split)} \\
P &\rightarrow \pi P && \text{(Customer)} \\
&\mid \pi F^{q,r} && \forall q, r \in F \text{ (Charge)} \\
&\mid 0 && \text{(Depot)} \\
F^{q,r} &\rightarrow qrP && q, r \in F \text{ (Customer)} \\
C &\rightarrow \pi C \\
&\mid \epsilon
\end{aligned}
$$

A Cyclic Decoder for EVRPTW: Grammar (II)

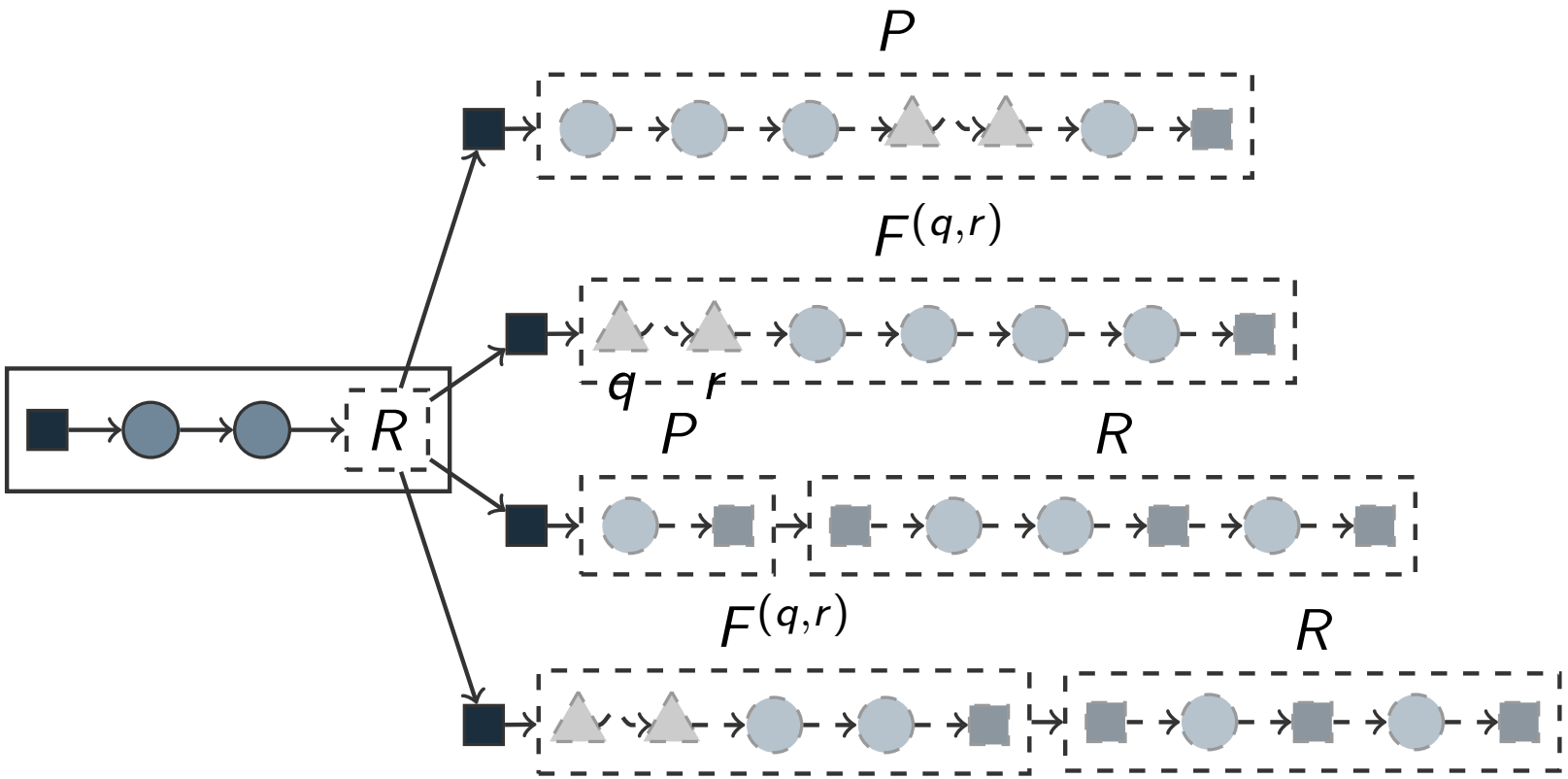
Have a look at the decomposition of **Routes** . . .



All alternatives produce C first . . .

A Cyclic Decoder for EVRPTW: Grammar (II)

Have a look at the decomposition of **Routes** . . .



All alternatives produce C first . . .

. . . fails if capacity limit is exceeded.

A Cyclic Decoder for EVRPTW: Grammar (II)

We consider a cyclic encoding for the EVRPTW ...

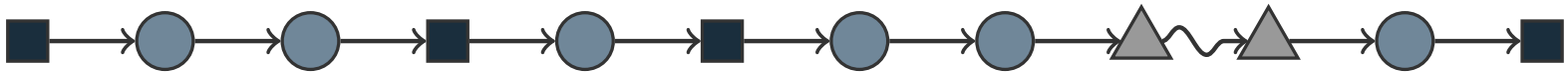
$$
\begin{aligned}
S &\rightarrow R && \text{(Start)} \\
R &\rightarrow C0P && \text{(Customer)} \\
&\mid C0F^{q,r} && \forall q, r \in F \text{ (Charge)} \\
&\mid C0PR && \text{(Split)} \\
&\mid C0F^{q,r}R && \forall q, r \in F \text{ (Split)} \\
P &\rightarrow \pi P && \text{(Customer)} \\
&\mid \pi F^{q,r} && \forall q, r \in F \text{ (Charge)} \\
&\mid 0 && \text{(Depot)} \\
F^{q,r} &\rightarrow qrP && q, r \in F \text{ (Customer)} \\
C &\rightarrow \pi C \\
&\mid \epsilon
\end{aligned}
$$

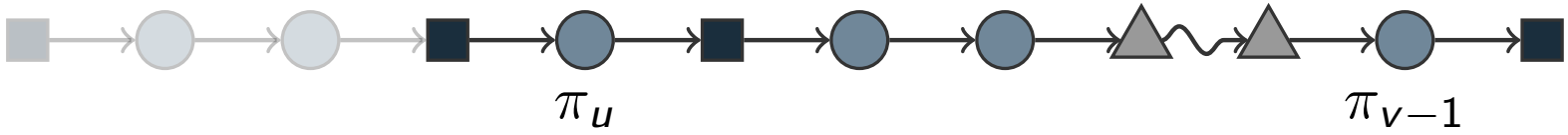
Cyclicity is not obvious from the Unindexed Grammar ...

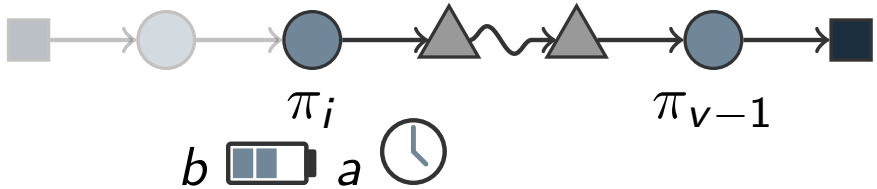
... and cannot be deduced by standard ADP approaches!

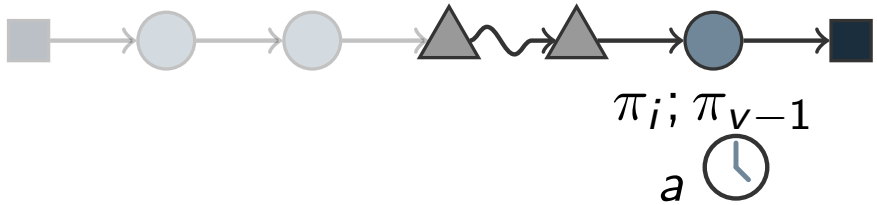
A Cyclic Decoder for EVRPTW: Indexed Grammar (I)

Symbol	Name	Indices
S	No indices	Represent all feasible tour decompositions
R	u, v	At least one further route/path
P	i, v, b, a	Path to the depot via next customer
$F^{(q,r)}$	i, v, a	Path to the depot via recharging chain
C	u, v	Required vehicle capacity for next *Path*









A Cyclic Decoder for EVRPTW: Indexed Grammar (II)

We consider a cyclic encoding for the EVRPTW ...

$$
\begin{aligned}
S &\to R_{s,t} && \forall s \in V, t = s + |V| \\
R_{u,v} &\to C_{u,v}\ 0 P_{u \neq v, v} && \\
&\mid C_{u,v}\ 0 F^{q,r}_{u \neq v, v} && \forall q, r \in F \\
&\mid C_{u,v'}\ 0 P_{u \neq v', v'} R_{v',v} && \forall v' \in V : u < v' < v \\
&\mid C_{u,v'}\ 0 F^{q,r}_{u < v', v} R_{v',v} && \forall v' \in V : u < v' < v \\
P_{i, v \neq i} &\to \pi_i P_{i+1,v} && \\
&\mid \pi_i F^{q,r}_{i+1,v} && \forall q, r \in F \\
P_{v,v} &\to 0 && \\
F^{q,r}_{i, v \neq i} &\to qr P_{i,u,v} && q, r \in F \\
C_{u,v} &\to \pi_u C_{u+1,v} && q_u + \sigma_q(C_{u,v}) \leq Q \\
C_{v,v} &\to \epsilon &&
\end{aligned}
$$

A Cyclic Decoder for EVRPTW: Indexed Grammar (II)

We consider a cyclic encoding for the EVRPTW ...

A closer look at **cyclicity** ...

$$S \to R_{s,t} \qquad \forall s \in V, t = s + |V|$$

$$C_{u,v} \to \pi_u C_{u+1,v} \qquad q_u + \sigma_q(C_{u,v}) \leq Q$$

$$C_{v,v} \to \epsilon$$

A Cyclic Decoder for EVRPTW: Indexed Grammar (II)

We consider a cyclic encoding for the EVRPTW . . .

A closer look at **cyclicity** . . .

$$S \to R_{s,t} \qquad \forall s \in V, t = s + |V|$$

. . . and the **indexed Path**

$$\begin{aligned} P_{i,v \neq i} &\to \pi_i P_{i+1,v} \\ &\mid \pi_i F^{q,r}_{i+1,v} \qquad \forall q, r \in F \\ P_{v,v} &\to 0 \end{aligned}$$

$$\begin{aligned} C_{u,v} &\to \pi_u C_{u+1,v} \qquad q_u + \sigma_q(C_{u,v}) \leq Q \\ C_{v,v} &\to \epsilon \end{aligned}$$

A Cyclic Decoder for EVRPTW: Indexed Grammar (II)

We consider a cyclic encoding for the EVRPTW . . .

A closer look at **cyclicity** . . .

$$S \to R_{s,t} \qquad \forall s \in V, t = s + |V|$$

. . . and the **indexed Path**

$$\begin{aligned} P_{i,v \neq i} &\to \pi_i P_{i+1,v} \\ &\mid \pi_i F^{q,r}_{i+1,v} \qquad \forall q, r \in F \\ P_{v,v} &\to 0 \end{aligned}$$

Note: All customer indices i, u, v are modulo $|V|$ (omitted for clarity)

$$\begin{aligned} C_{u,v} &\to \pi_u C_{u+1,v} \qquad q_u + \sigma_q(C_{u,v}) \leq Q \\ C_{v,v} &\to \epsilon \end{aligned}$$

A Cyclic Decoder for EVRPTW: Constraint Grammar

$$S \rightarrow R_{s,t} \qquad \forall s \in V,\ t = s + |V|$$

$$R_{u,v} \rightarrow C_{u,v}\ 0P_{\ldots,(B-b_{0,u}\geq 0),(t_{0,u}\leq l_u)}$$

$$\mid C_{u,v}\ 0F^{q,r}_{\ldots,(t_{0,q}+\frac{B-b_{0,q}}{\phi}\leq l_u)} \qquad \forall q,r \in F,\ 0 \leq B - b_{0,q}$$

$$\mid C_{u,v}\ 0P_{\ldots,(B-b_{0,u}\geq 0),(t_{0,u}\leq l_u)}R_{v',v} \qquad \forall v' \in V : u < v' < v$$

$$P_{i,v\neq i,b,a} \rightarrow \pi_i P_{\ldots,(b-b_{i,i+1}\geq 0),(a+t_{i,i+1}\leq l_{i+1})}$$

$$\mid \pi_i F^{q,r}_{\ldots,(a+t_{i,q}+\frac{b-b_{i,q}}{\phi}\leq l_{i+1})} \qquad \forall q,r \in F,\ 0 \leq b - b_{i,q}$$

$$P_{v,v,*} \rightarrow 0$$

$$F^{q,r}_{i,v\neq i,a} \rightarrow qrP_{\ldots,(B-b_{r,i}\geq 0),(a+t^{q,r}+t_{r,i}\leq l_i)} \qquad q,r \in F$$

$$\ldots$$

A Cyclic Decoder for EVRPTW: Constraint Grammar

$$S \rightarrow R_{s,t} \qquad \forall s \in V,\ t = s + |V|$$

$$R_{u,v} \rightarrow C_{u,v}\ 0P_{\ldots,(B-b_{0,u}\geq 0),(t_{0,u}\leq l_u)}$$

Finally the **constrained Path** ...

$$P_{i,v\neq i,b,a} \rightarrow \pi_i P_{\ldots,(b-b_{i,i+1}\geq 0),(a+t_{i,i+1}\leq l_{i+1})}$$

$$\mid \pi_i F^{q,r}_{\ldots,(a+t_{i,q}+\frac{b-b_{i,q}}{\phi}\leq l_{i+1})} \qquad \forall q, r \in F, 0 \leq b - b_{i,i+1}$$

$$P_{v,v,*} \rightarrow 0$$

$$P_{v,v,*} \rightarrow 0$$

$$F^{q,r}_{i,v\neq i,a} \rightarrow qrP_{\ldots,(B-b_{r,i}\geq 0),(a+t^{q,r}+t_{r,i}\leq l_i)} \qquad q, r \in F$$

$$\ldots$$

Evaluation and Dominance

Evaluation algebra σ_d for travel distance (among #vehicles, vehicle load)

$$\sigma_d(R^{(1)}) = d_{0,i} + \sigma_d(P)$$
$$\sigma_d(R^{(2)}) = d_{0,q} + \sigma_d(F)$$
$$\sigma_d(R^{(3)}) = d_{0,i} + \sigma_d(P) + \sigma_d(R)$$
$$\dots$$

Evaluation and Dominance

Evaluation algebra σ_d for travel distance (among #vehicles, vehicle load)

$$\sigma_d(R^{(1)}) = d_{0,i} + \sigma_d(P)$$
$$\sigma_d(R^{(2)}) = d_{0,q} + \sigma_d(F)$$
$$\sigma_d(R^{(3)}) = d_{0,i} + \sigma_d(P) + \sigma_d(R)$$
$$\dots$$

In EVRPTW a state A **dominates** a **compatible** state B if it has:

- **Fewer vehicles** $\sigma_k(A) \leq \sigma_k(B)$
- or **Less distance** and equal #vehicles
  $\sigma_k(A) = \sigma_k(B) \wedge \sigma_d(A) \leq \sigma_d(B)$
- and **Higher battery** $A.b \geq B.b$
- and **Earlier arrival** $A.t \leq B.t$

Whistle—ADP for combinatorial optimization

. . . no need to implement all this by yourself!

- Tailored for **combinatorial optimization**—not bioinformatics
- Generates modifiable **C++ code** (or Rust for C ABI compatibility)
- Supports **integer and float indices**
- Uses a new **compatibility and dominance** mechanism instead of objective functions

Whistle—ADP for combinatorial optimization

. . . no need to implement all this by yourself!

- Tailored for **combinatorial optimization**—not bioinformatics
- Generates modifiable **C++ code** (or Rust for C ABI compatibility)
- Supports **integer and float indices**
- Uses a new **compatibility and dominance** mechanism instead of objective functions
- *(Will)* Support **Index Propagators** for special cases supporting index change deduction
- *(Will)* Support **set indices**
  with different evaluation algorithms:
  Direct Method, **Successive Approximation Method** (new)
- (Will) Support **different traversal algorithms**:
  DFS (current), Greedy, A*, Beam-Search
- (Will) Support **Partial Invalidation** and **Shadowing**

Conclusion & Future Work

Conclusion

- An algorithm formalized as ADP can be solved automatically
- Provided an ADP formulation for EVRPTW solution decoding
- Whistle ADP framework provides enhanced DP functionality for combinatoric optimization

Conclusion & Future Work

Conclusion

- An algorithm formalized as ADP can be solved automatically
- Provided an ADP formulation for EVRPTW solution decoding
- Whistle ADP framework provides enhanced DP functionality for combinatoric optimization

Future Work

- Finish & release *Whistle* framework
- Finish open proofs on the expressiveness of Extended ADP
- Extend the approach to more complex charging schemes

Thank you for your attention, and not dozing off!