# Public Transportation in Rural Areas: The Clustered Dial-a-Ride Problem

Fabian Feitsch

November 16th, 2018

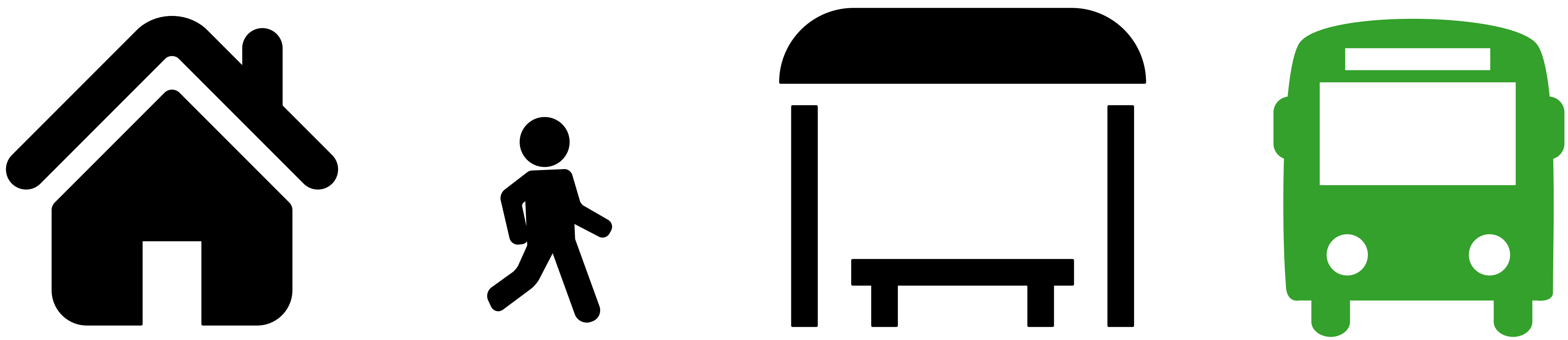
Attributions of third party images can be found on slide 12.

# Public Transportation

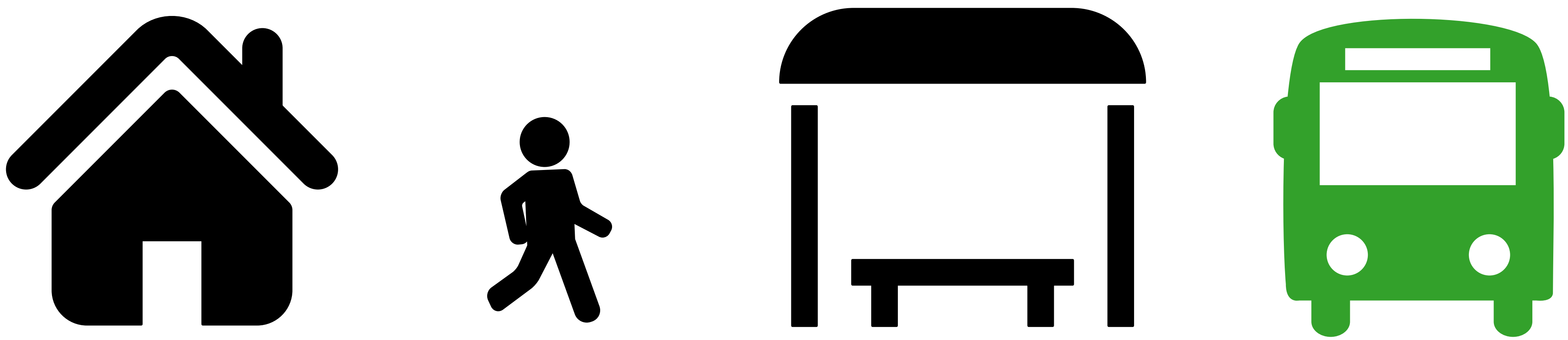
# Public Transportation

# Public Transportation

# Public Transportation



# Public Transportation

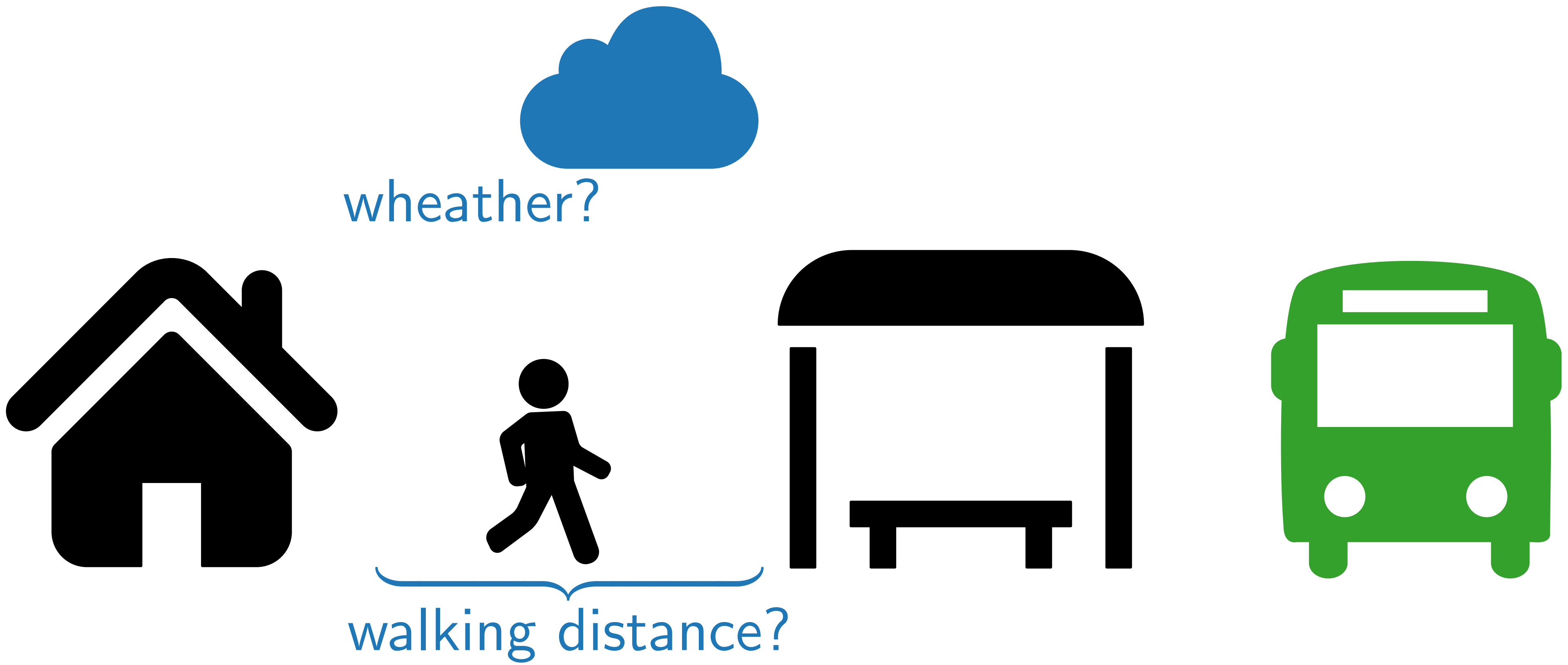


So far, so good?

# Public Transportation

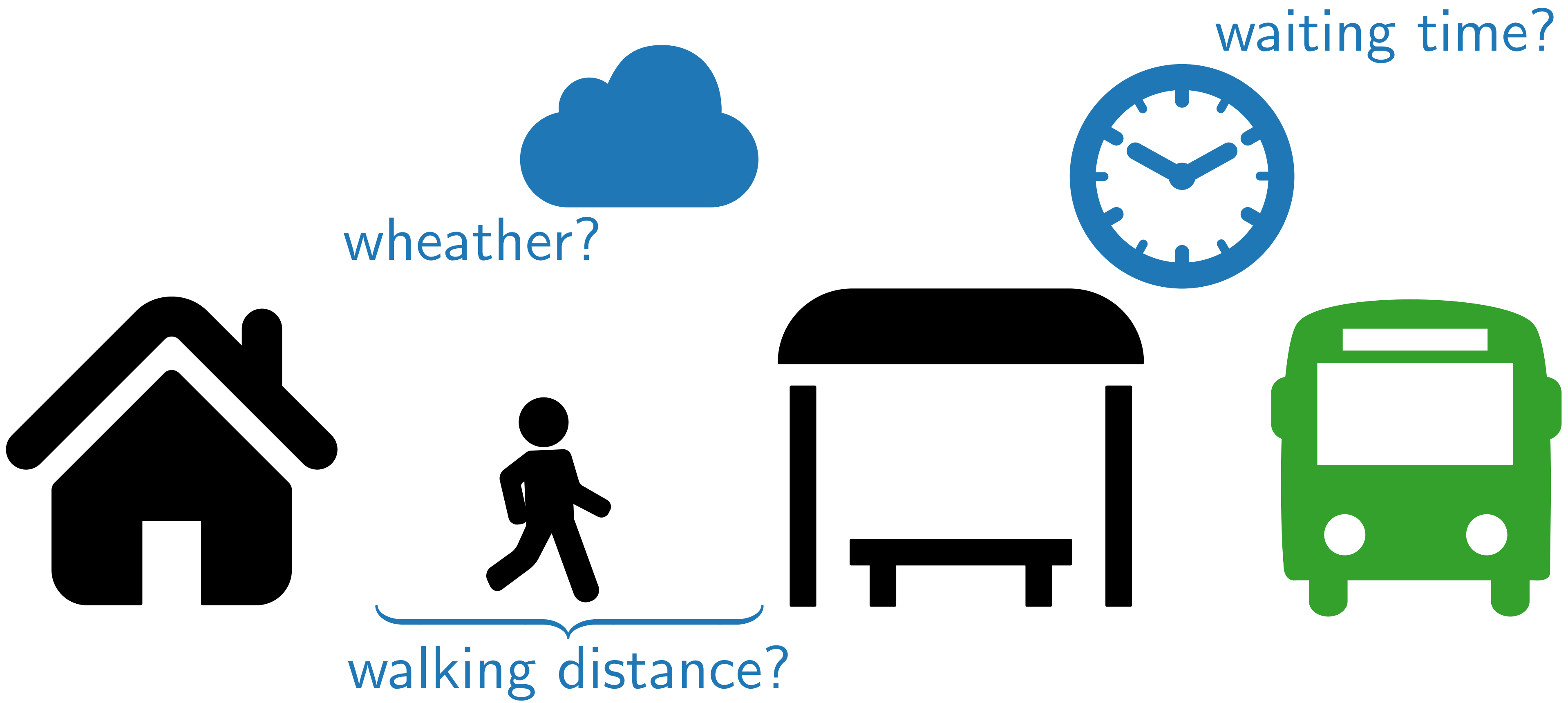

So far, so good?

# Public Transportation



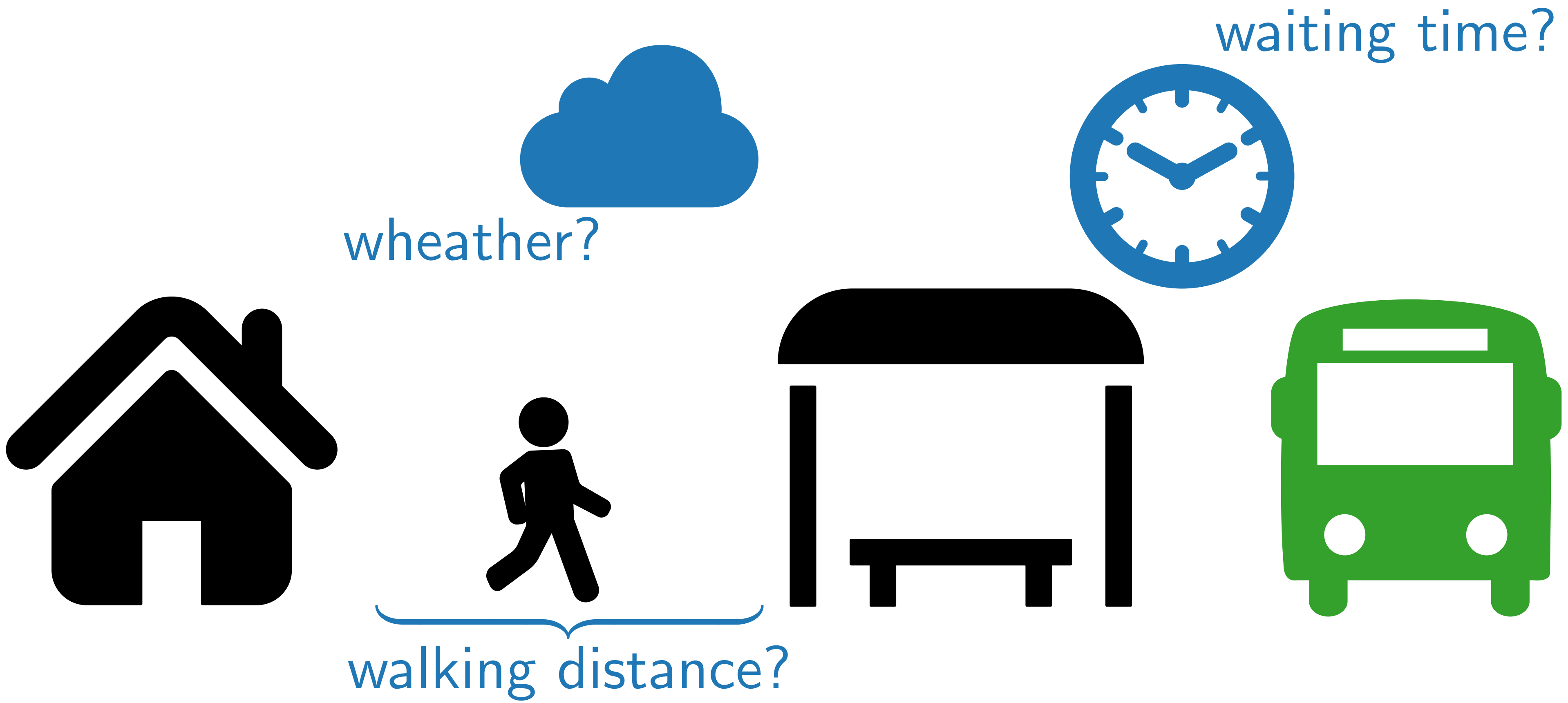

So far, so good?

# Public Transportation




So far, so good?

# Public Transportation




So far, so good? → Probably in the city, but not in villages!

# Public Transportation

So far, so good? → Probably in the city, but not in villages!

→ Doorstep Service in Rural Areas

# The Dial-a-Ride Problem

# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

# The Dial-a-Ride Problem

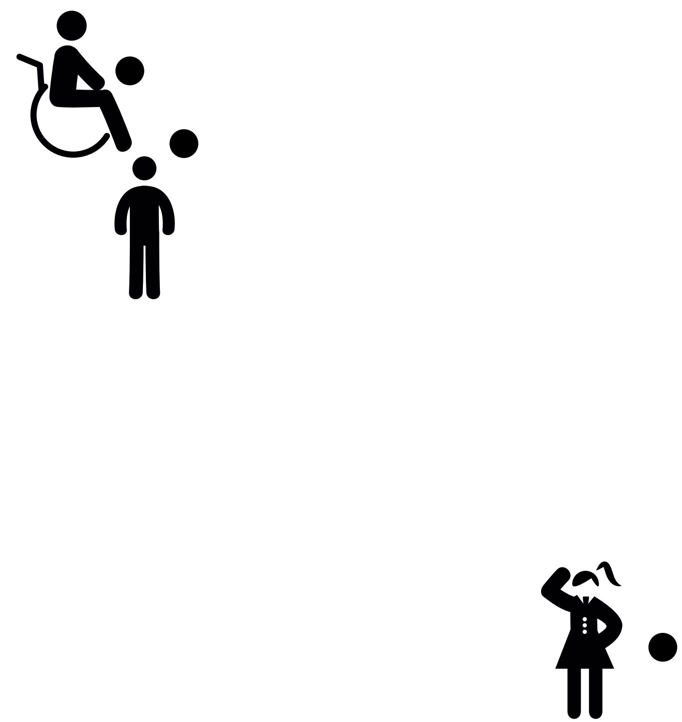
A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

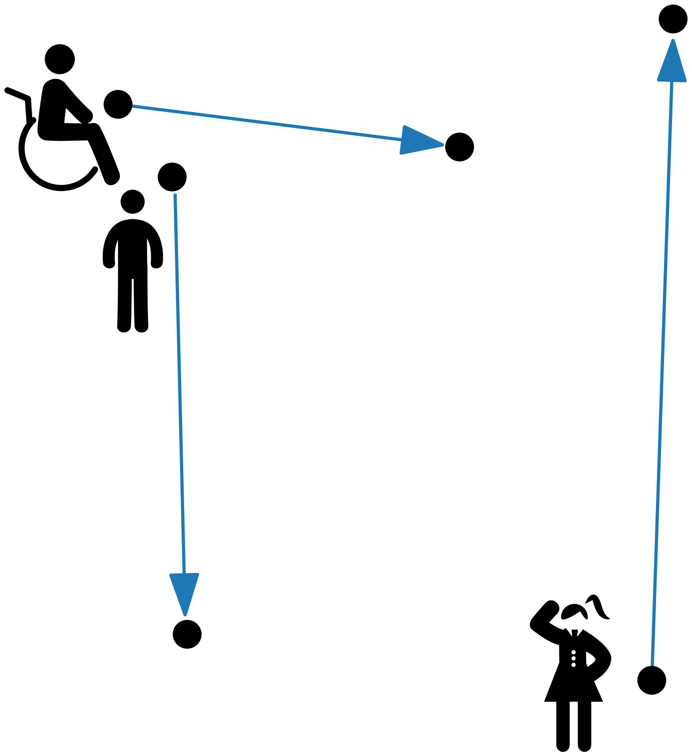
$n :=$ number of riders



# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

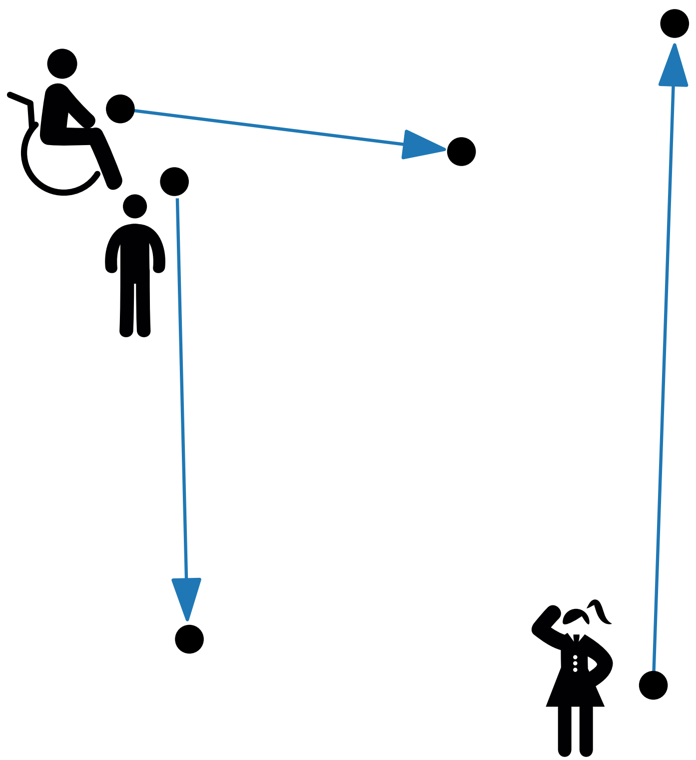


# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

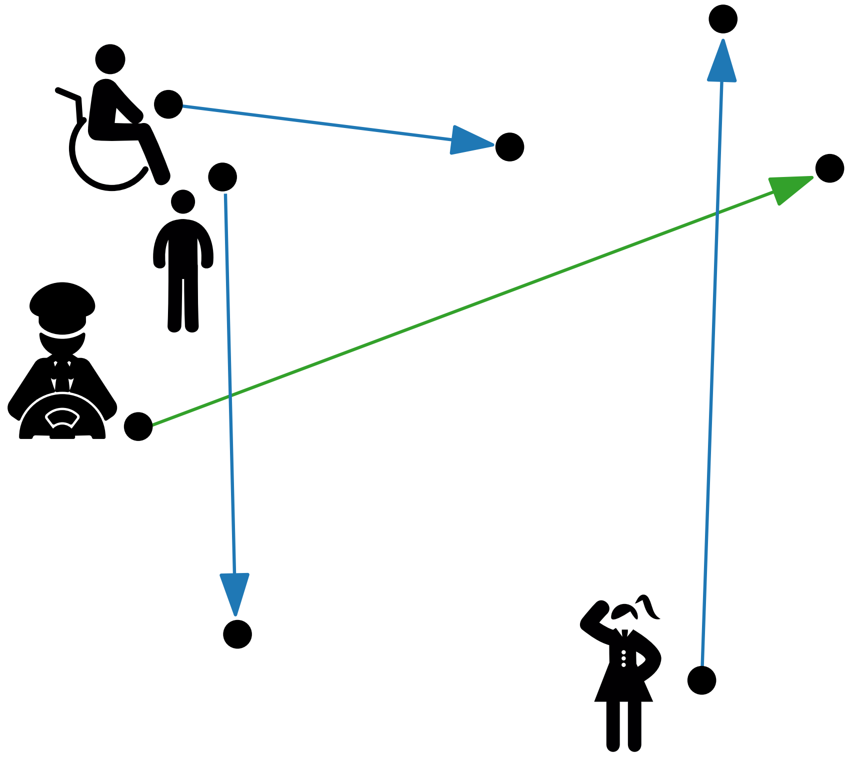


# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$



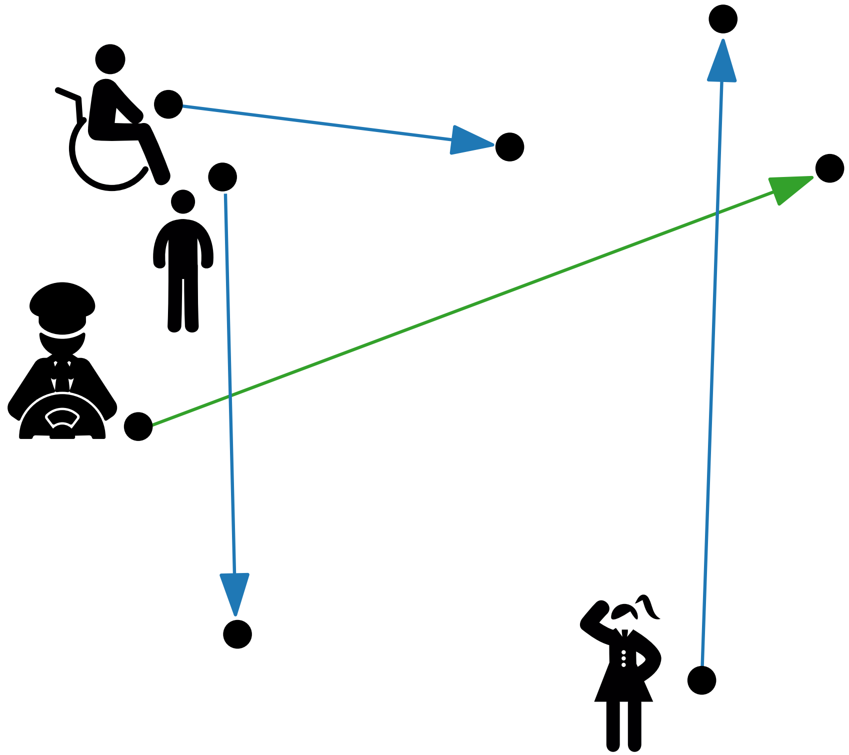
# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix



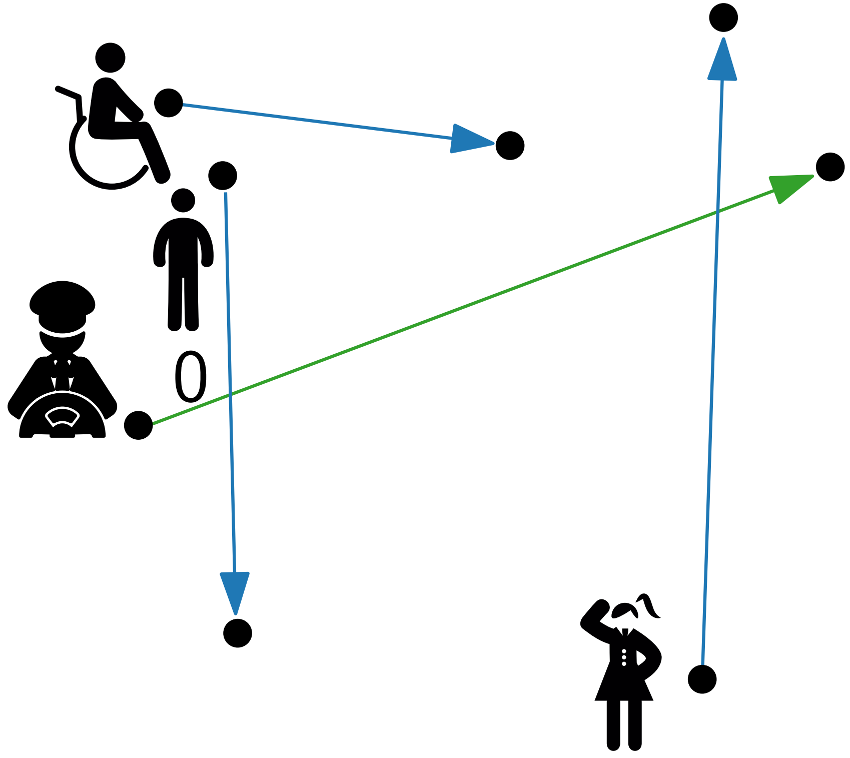
# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)



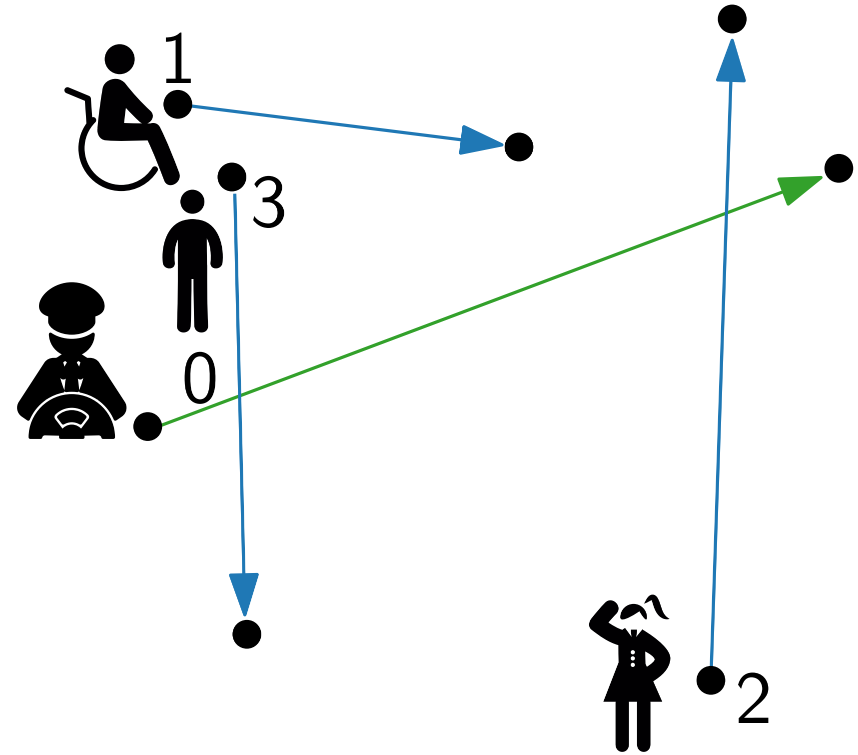
# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)
enumerate pickups



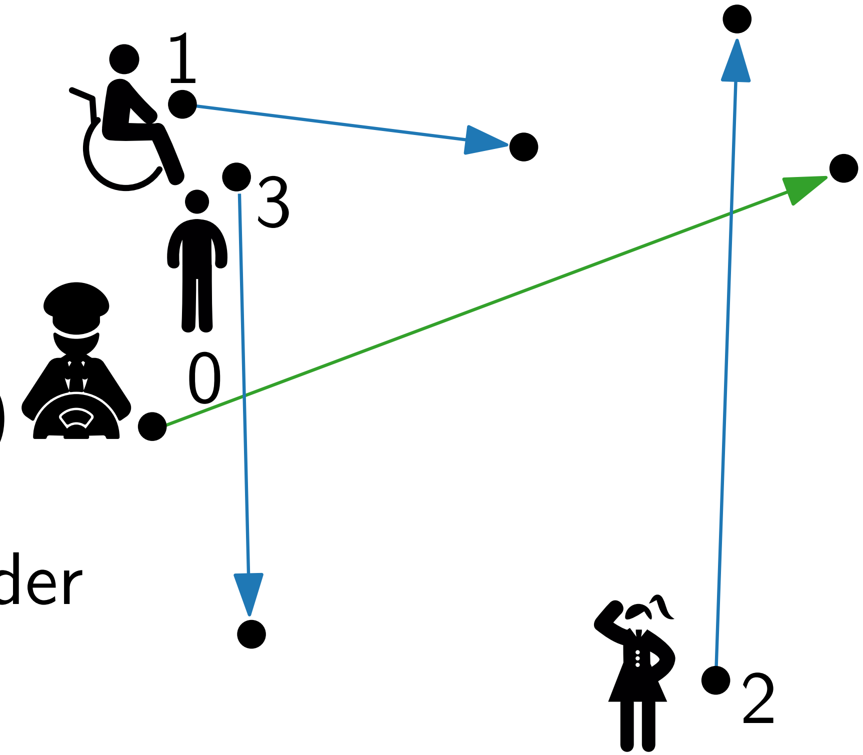
# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)
enumerate pickups
enumerate dest's in same order



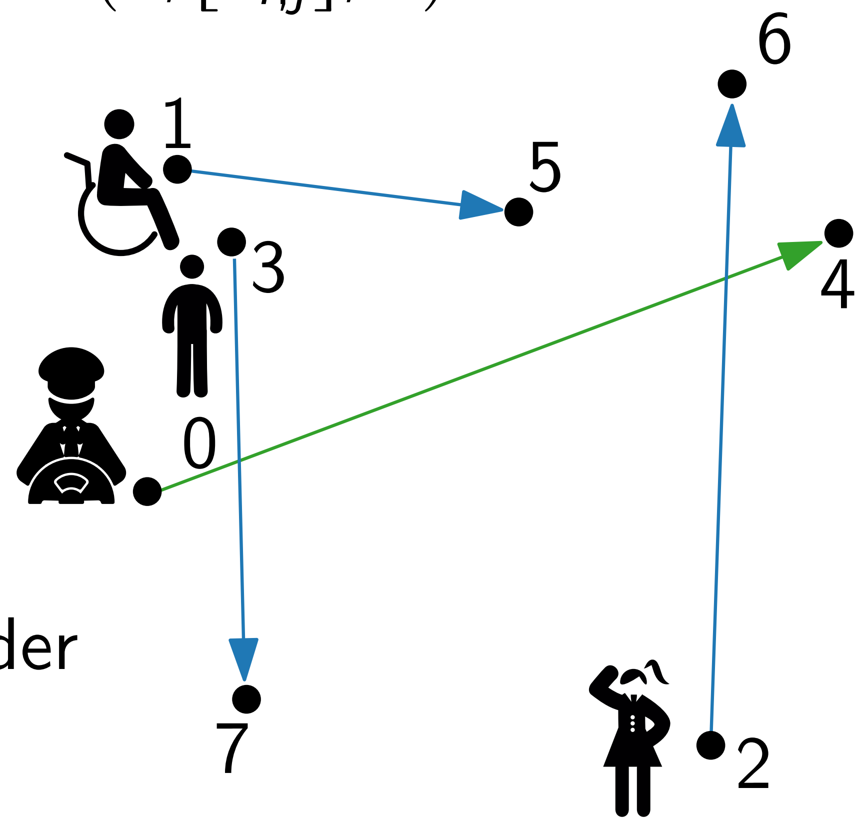
# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)
enumerate pickups
enumerate dest's in same order



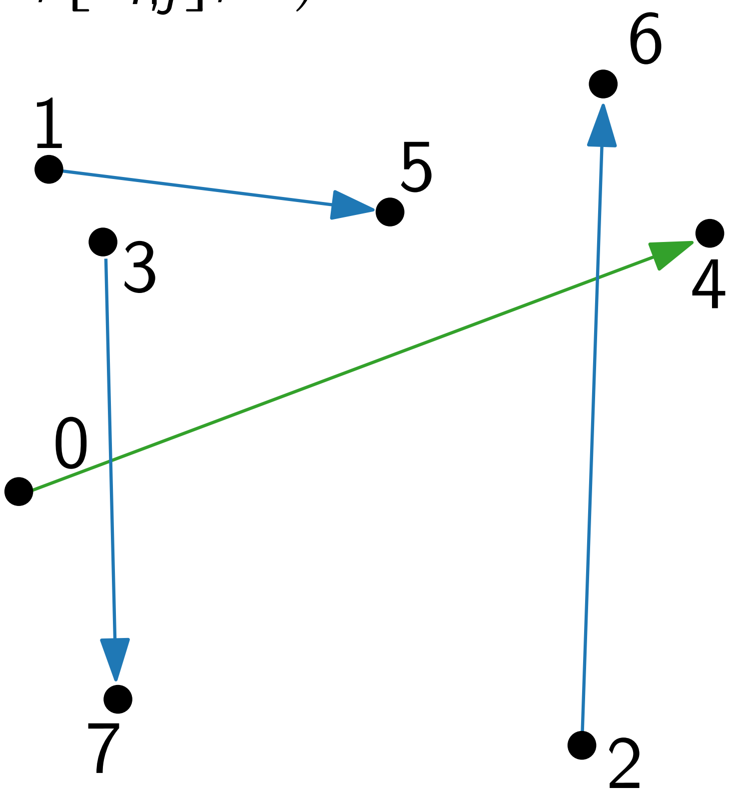
# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)
enumerate pickups
enumerate dest's in same order



# The Dial-a-Ride Problem

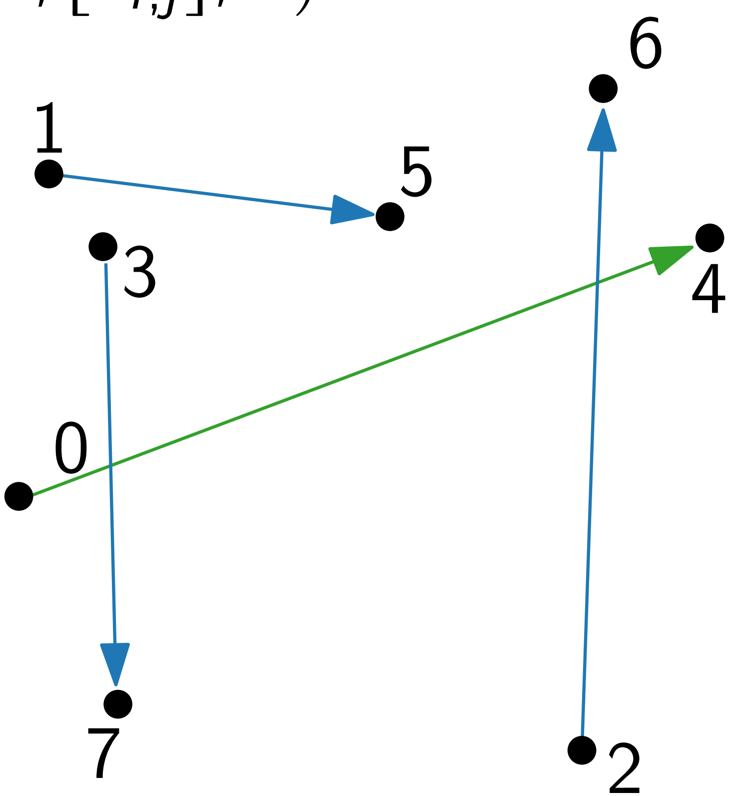
A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)
enumerate pickups
enumerate dest's in same order

$S :=$number of seats in the vehicle



# The Dial-a-Ride Problem

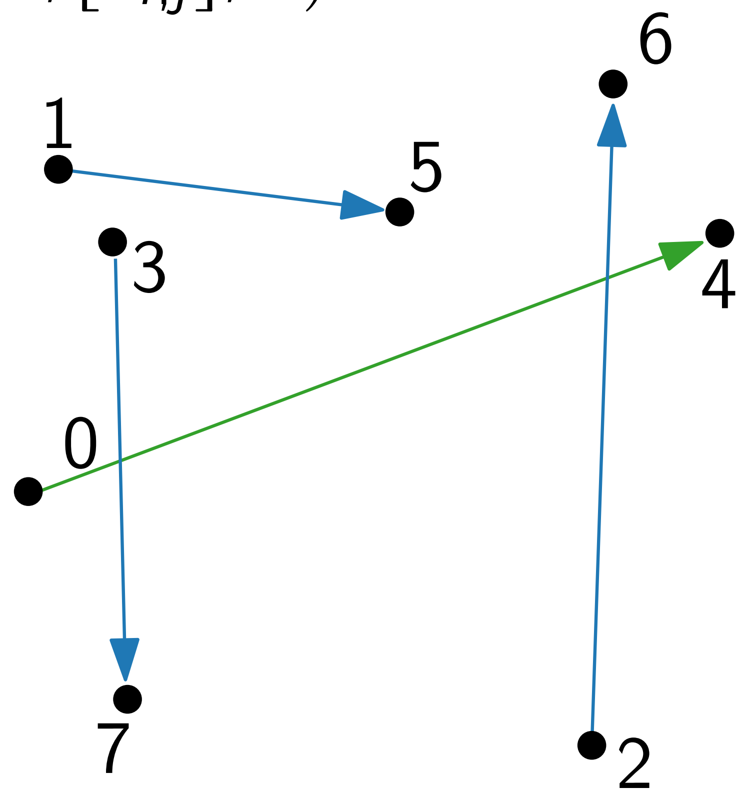
A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)
enumerate pickups
enumerate dest's in same order

$S :=$number of seats in the vehicle
for this presentation: $S = \infty$



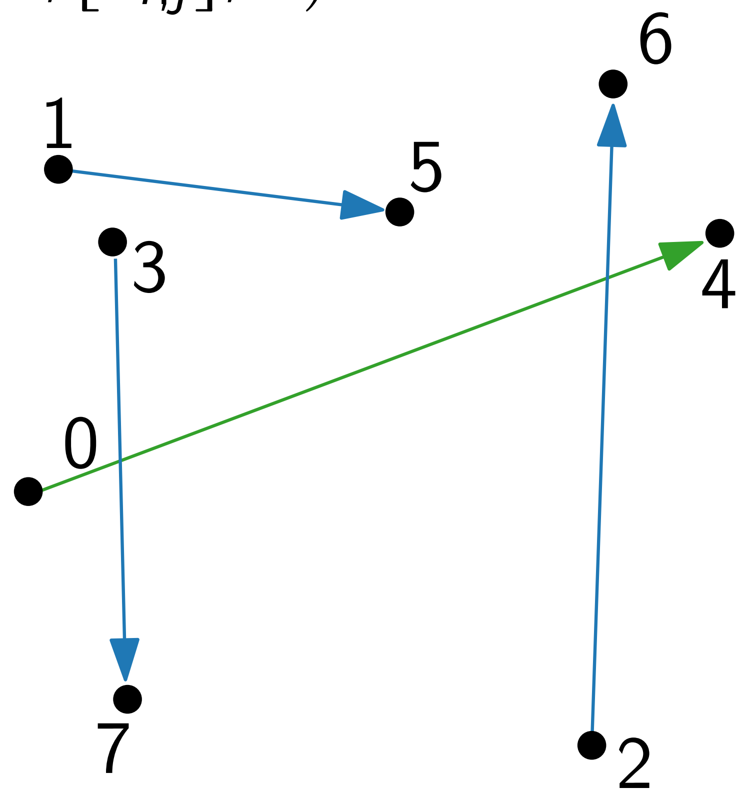
# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)
enumerate pickups
enumerate dest's in same order




$S :=$number of seats in the vehicle
for this presentation: $S = \infty$

Objective: Feasible tour minimizing the sum of total distances.

# The Dial-a-Ride Problem

A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

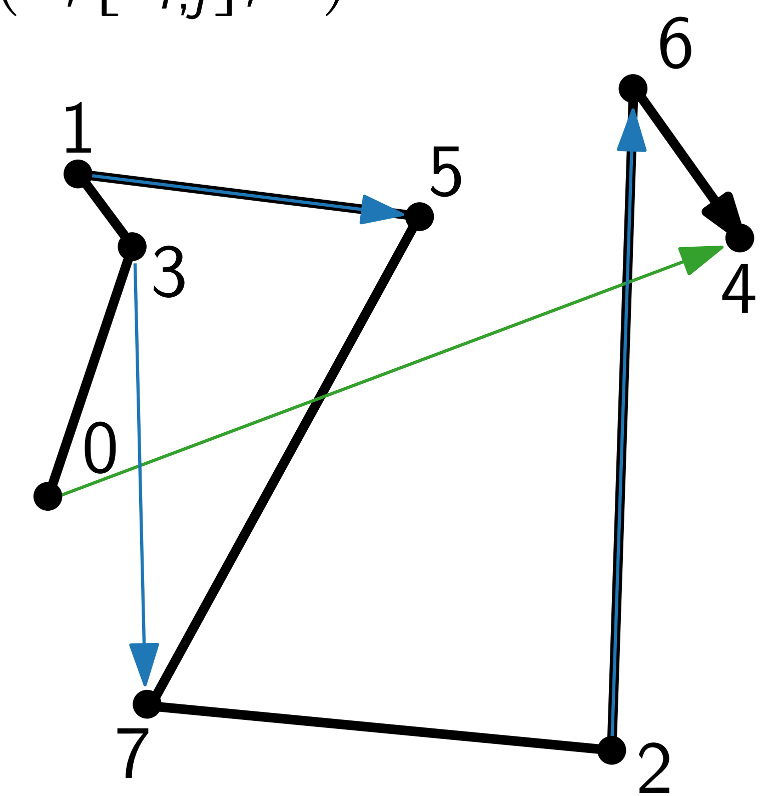
Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix

start with 0 (driver's pickup)

enumerate pickups

enumerate dest's in same order




$S :=$number of seats in the vehicle

for this presentation: $S = \infty$

Objective: Feasible tour minimizing the sum of total distances.

# The Dial-a-Ride Problem

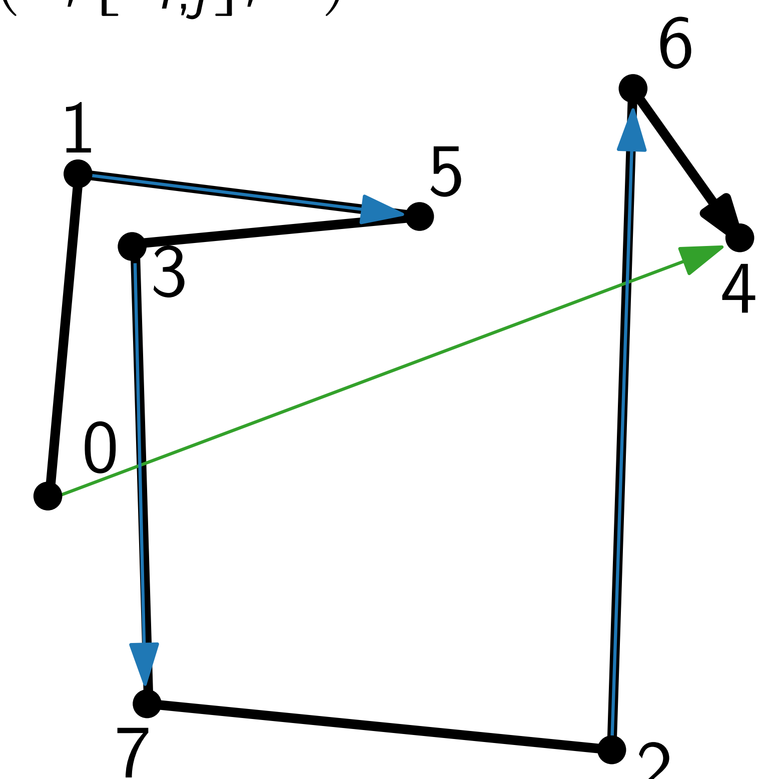
A *Dial-a-Ride instance* is a triple $I = (n, [d_{i,j}], S)$.

$n :=$ number of riders

Number of persons $m = n + 1$

$[d_{i,j}] :=$ distance matrix
start with 0 (driver's pickup)
enumerate pickups
enumerate dest‘s in same order

$S :=$number of seats in the vehicle
for this presentation: $S = \infty$




Objective: Feasible tour minimizing the sum of total distances.

# An Exact Algorithm

# An Exact Algorithm

There is an exact algorithm by Psaraftis, 1980.

# An Exact Algorithm

There is an exact algorithm by Psaraftis, 1980.

It works similar to the Held-Karp-algorithm.

# An Exact Algorithm

There is an exact algorithm by Psaraftis, 1980.

It works similar to the Held-Karp-algorithm.

Running Time: $O^*(3^{n-1})$.

# An Exact Algorithm

There is an exact algorithm by Psaraftis, 1980.

It works similar to the Held-Karp-algorithm.

Running Time: $O^*(3^{n-1})$.

Can be generalized to solve partial instances:

# An Exact Algorithm

There is an exact algorithm by Psaraftis, 1980.

It works similar to the Held-Karp-algorithm.

Running Time: $O^*(3^{n-1})$.

Can be generalized to solve partial instances:

# An Exact Algorithm

There is an exact algorithm by Psaraftis, 1980.

It works similar to the Held-Karp-algorithm.

Running Time: $O^*(3^{n-1})$.

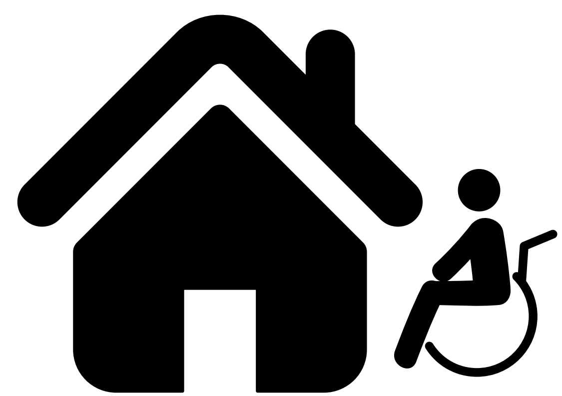
Can be generalized to solve partial instances:

Find best tour such that

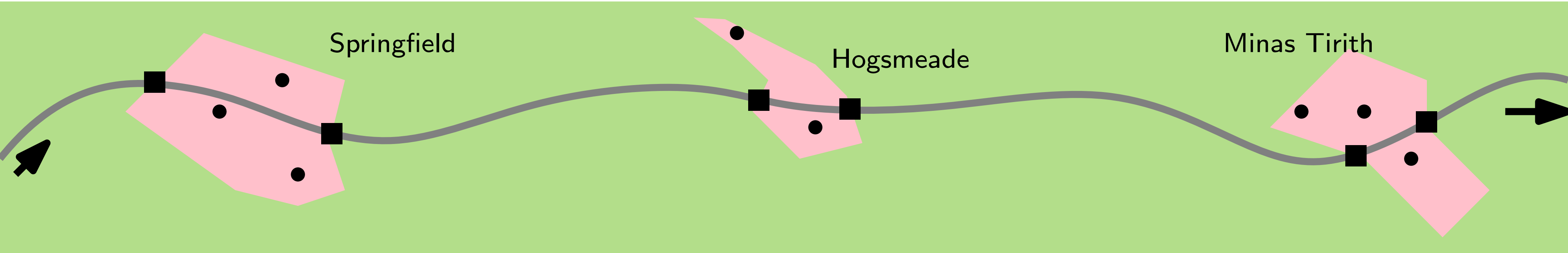
a) girl is delivered

b) waiting customer is fetched

c) boy is still on board.

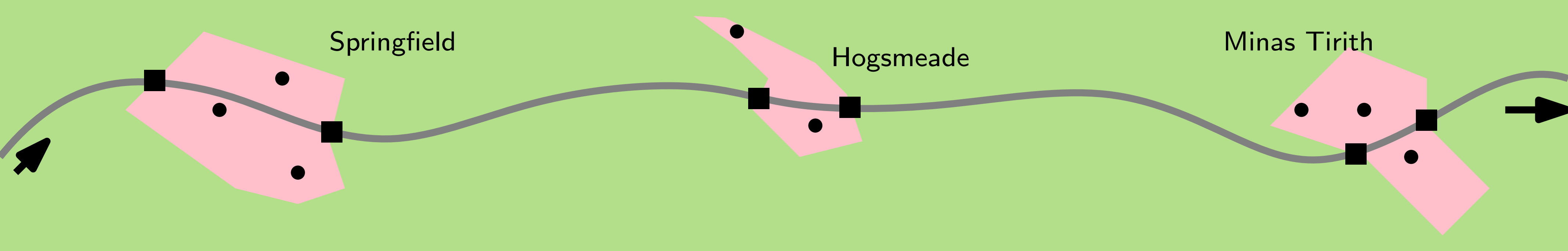


# Back to Rural Areas . . .



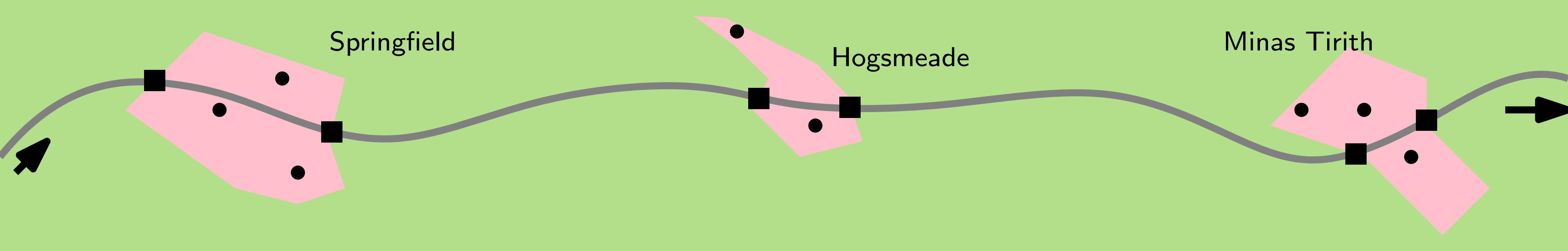
# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:



# Back to Rural Areas . . .

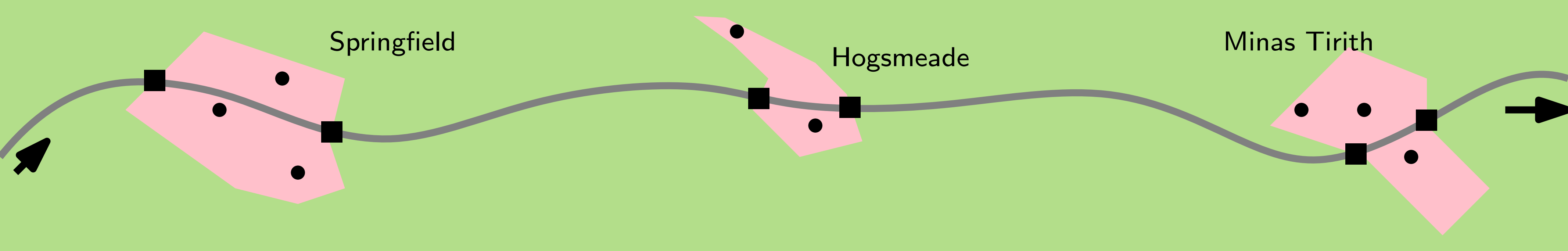
A rural Dial-a-Ride instance typically looks like this:




Assumptions:

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:

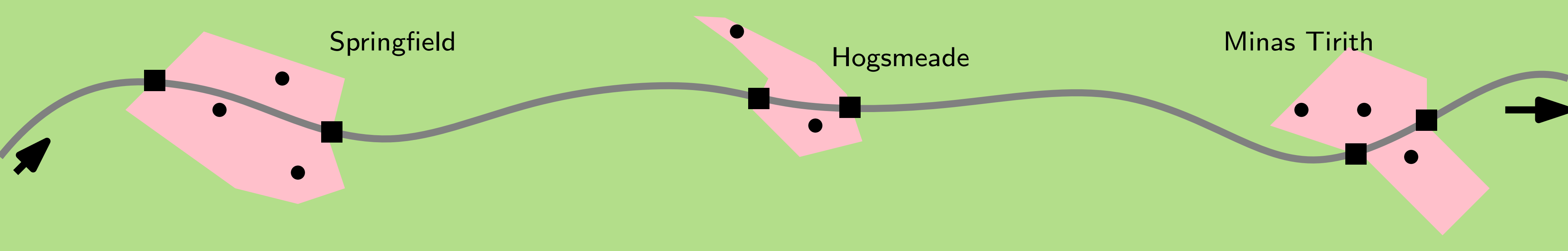



Assumptions:

→ Locations are inside clusters.

# Back to Rural Areas ...

A rural Dial-a-Ride instance typically looks like this:



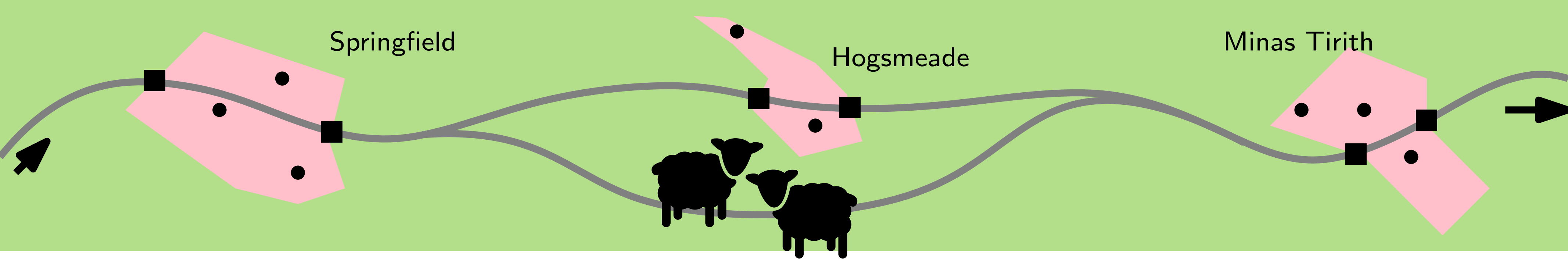

Assumptions:

→ Locations are inside clusters.

→ Bypasses do not exist.

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:



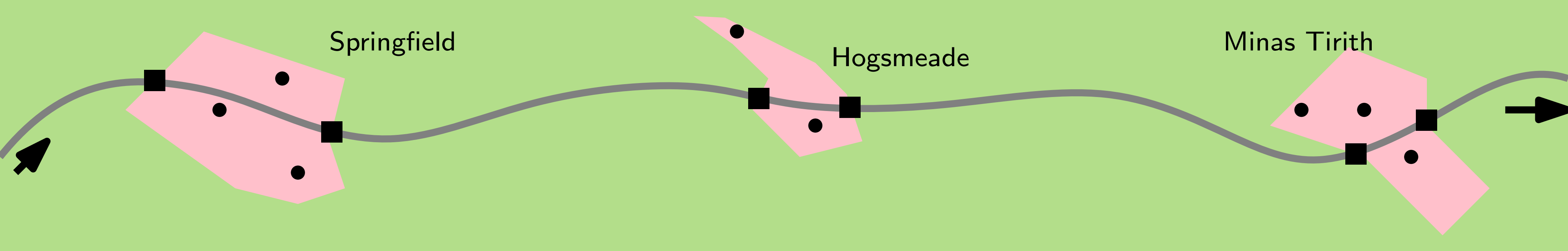

Assumptions:

→ Locations are inside clusters.

→ Bypasses do not exist.

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:

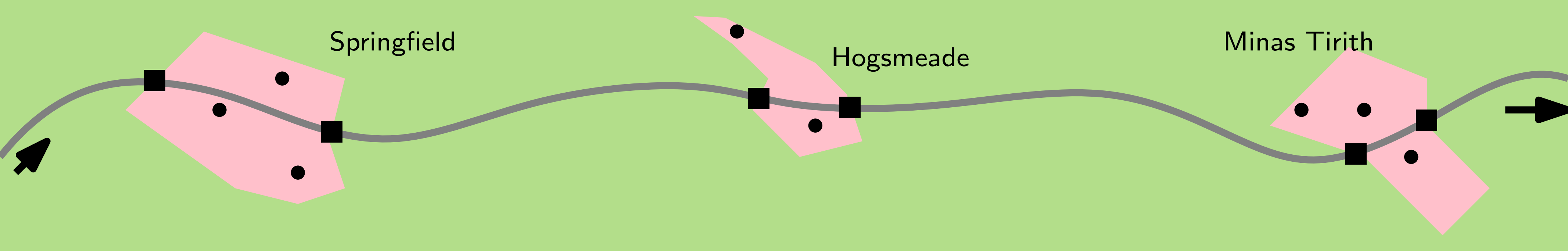



Assumptions:

→ Locations are inside clusters.
→ Bypasses do not exist.
→ All riders head in the same direction.

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:




Assumptions:

→ Locations are inside clusters.

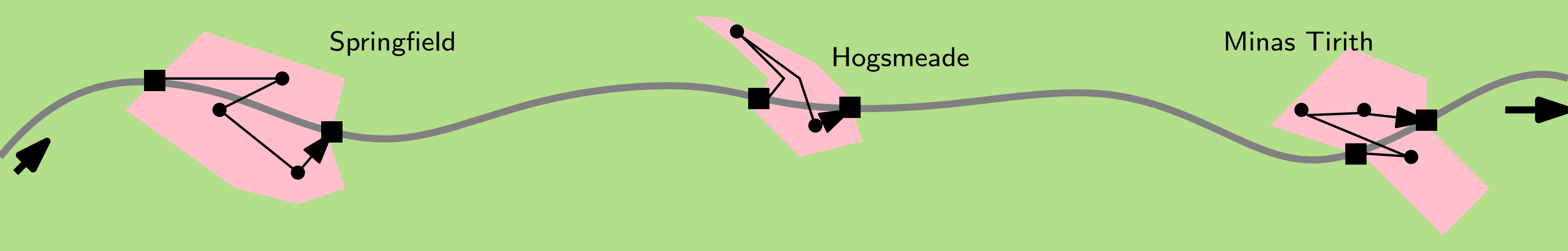
→ Bypasses do not exist.

→ All riders head in the same direction.

*Seems* to be simpler than the Dial-a-Ride Problem . . .

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:




Assumptions:

→ Locations are inside clusters.

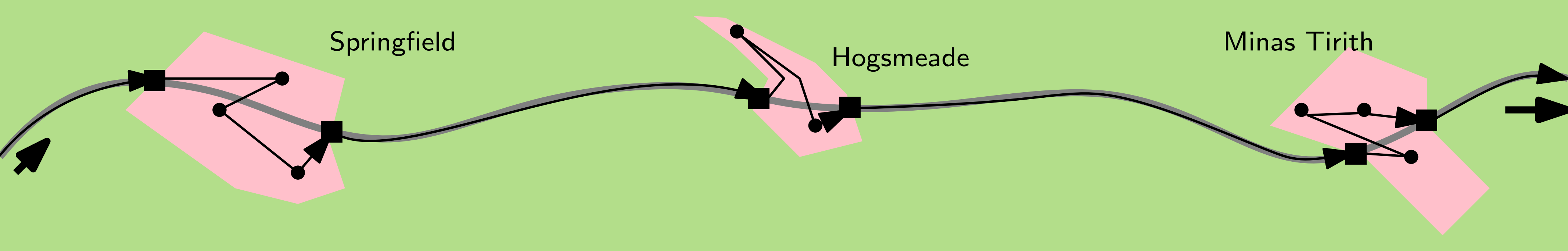
→ Bypasses do not exist.

→ All riders head in the same direction.

*Seems* to be simpler than the Dial-a-Ride Problem . . .

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:



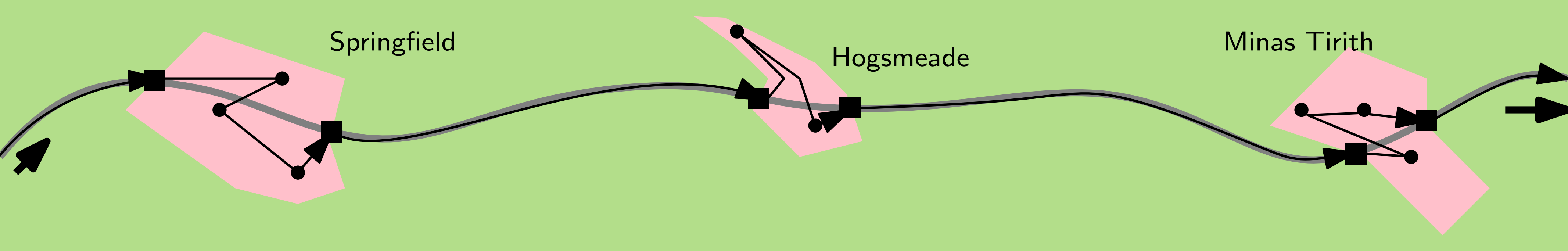

Assumptions:

→ Locations are inside clusters.
→ Bypasses do not exist.
→ All riders head in the same direction.

*Seems* to be simpler than the Dial-a-Ride Problem . . .

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:




Assumptions:

→ Locations are inside clusters.

→ Bypasses do not exist.

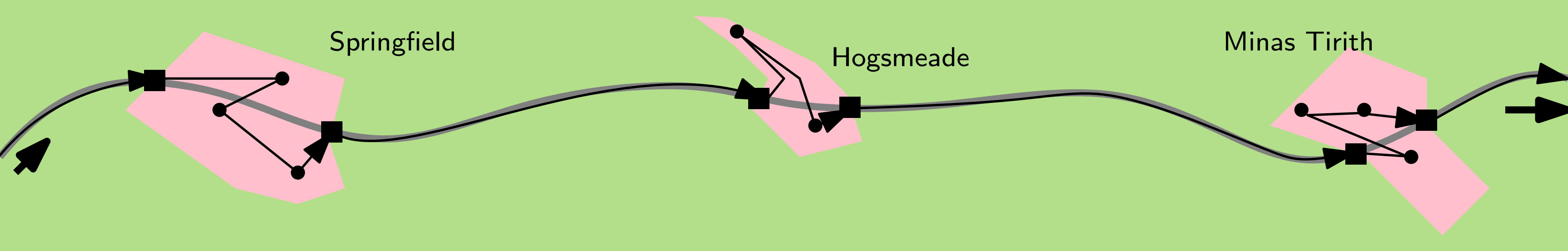
→ All riders head in the same direction.

*Seems* to be simpler than the Dial-a-Ride Problem . . .

→ $\overrightarrow{T^*}$-algorithm

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:




Assumptions:

→ Locations are inside clusters.
→ Bypasses do not exist.
→ All riders head in the same direction.

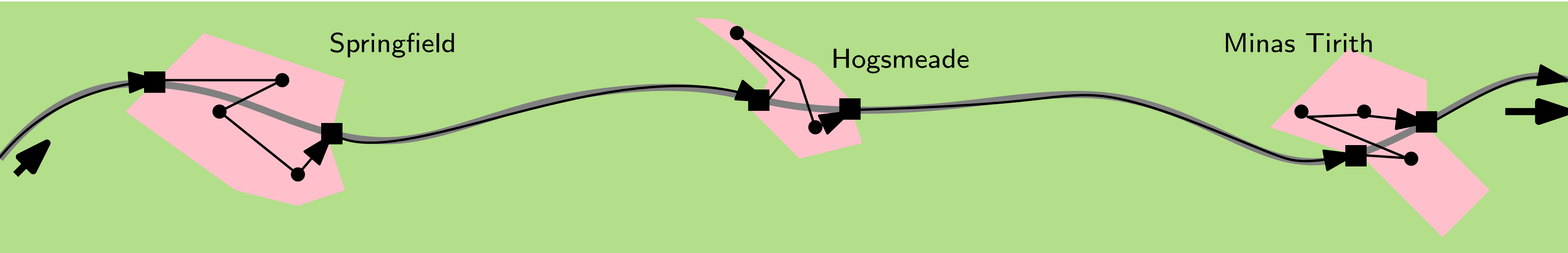
*Seems* to be simpler than the Dial-a-Ride Problem . . .

→ $\overrightarrow{T^*}$-algorithm

Goal:

# Back to Rural Areas . . .

A rural Dial-a-Ride instance typically looks like this:




Assumptions:

→ Locations are inside clusters.

→ Bypasses do not exist.

→ All riders head in the same direction.

*Seems* to be simpler than the Dial-a-Ride Problem . . .

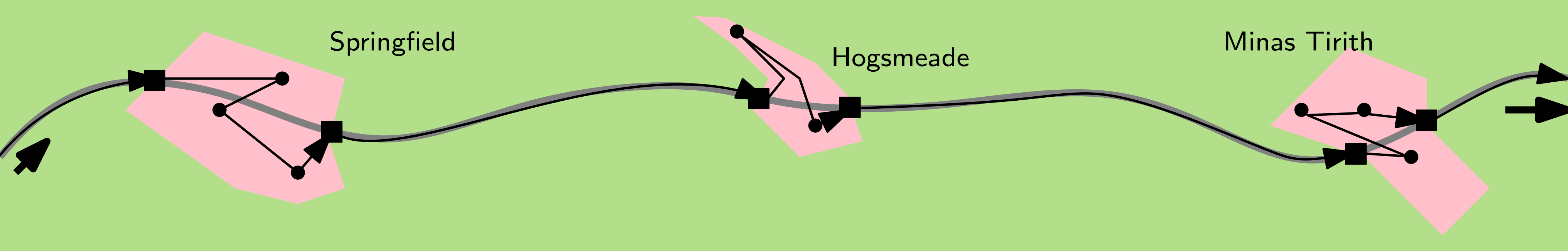
→ $\overrightarrow{T^*}$-algorithm

Goal:

Classify instances whose optimal tour is unidirectional.

# Back to Rural Areas ...

A rural Dial-a-Ride instance typically looks like this:




Assumptions:

→ Locations are inside clusters.

→ Bypasses do not exist.

→ All riders head in the same direction.

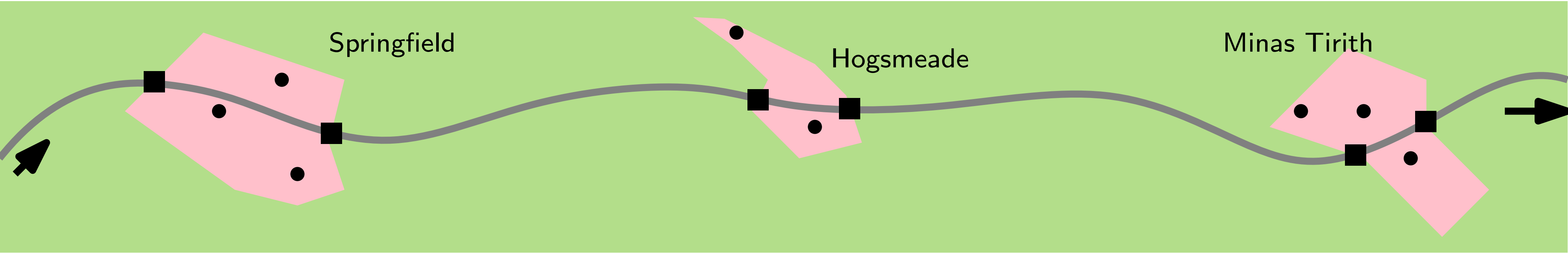
*Seems* to be simpler than the Dial-a-Ride Problem ...

→ $\overrightarrow{T^*}$-algorithm

Goal:

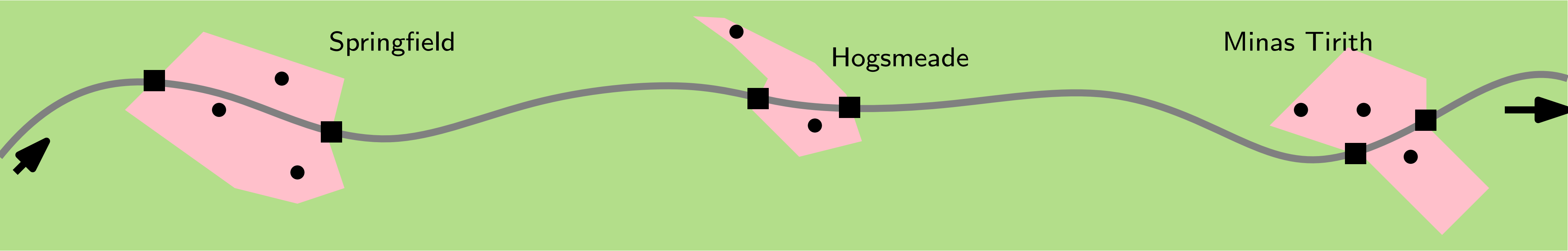
Classify instances whose optimal tour is unidirectional. (without computing it)

# A Classifier



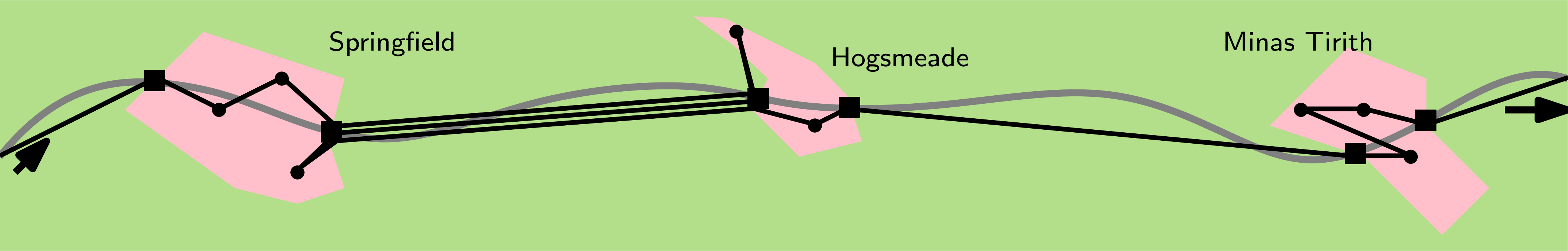
# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.



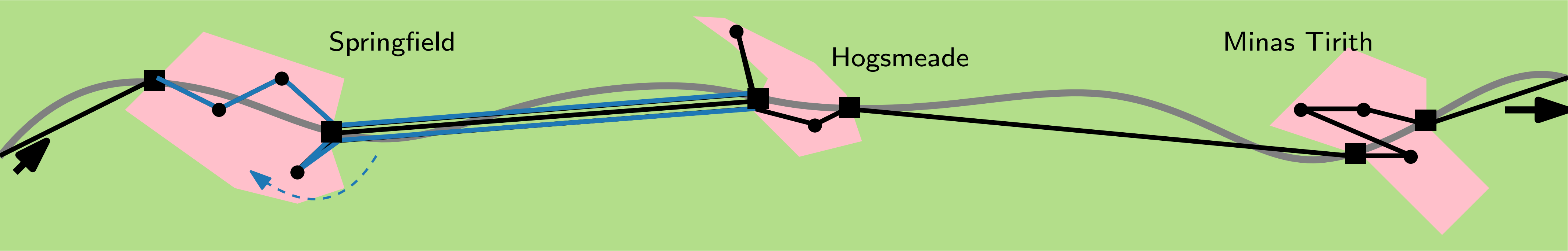
# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.



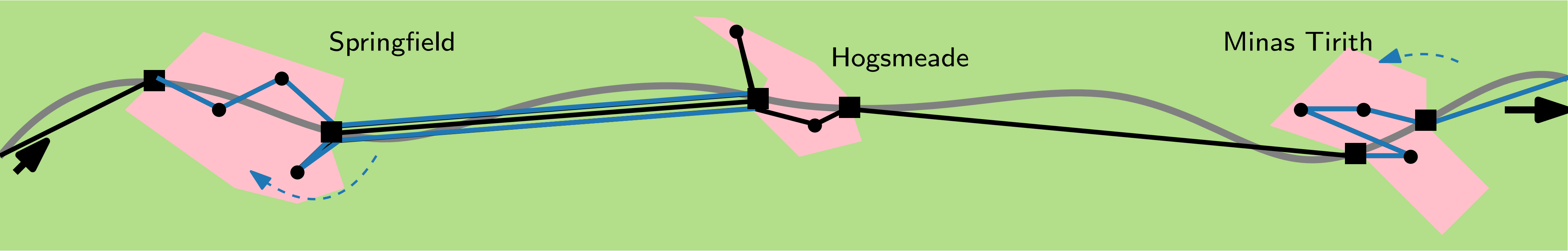
# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.



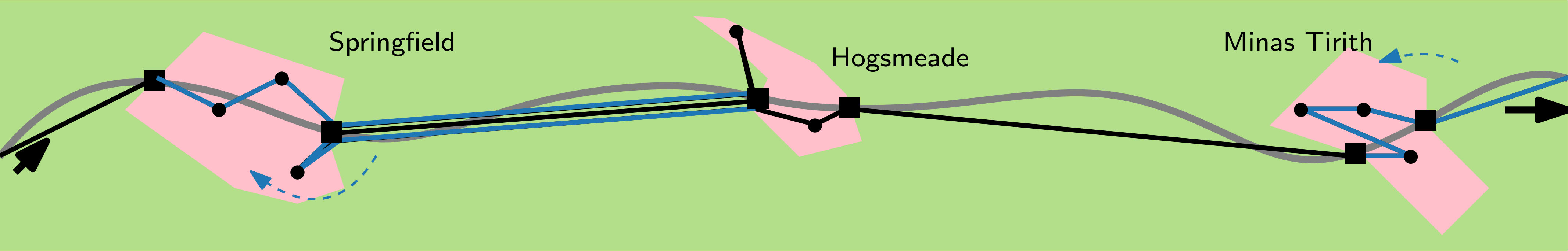
# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.



# A Classifier

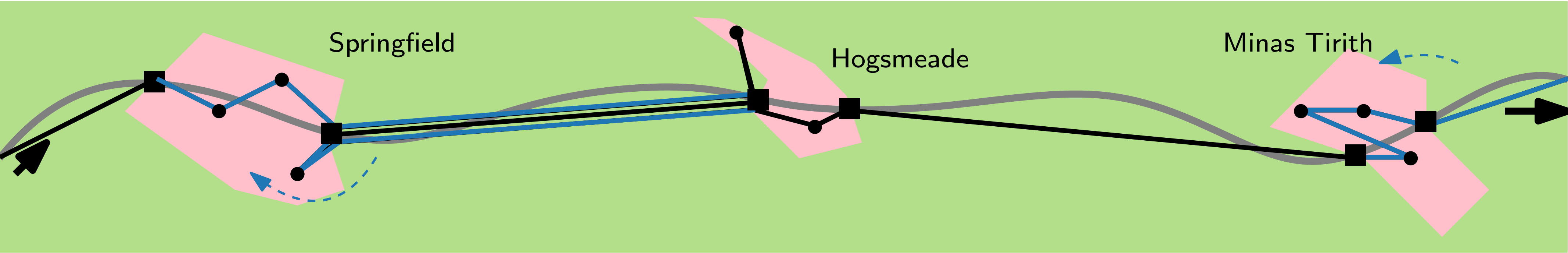
**Idea:** Distribute the costs of a tour to the clusters.




Let $C_1, \ldots C_q$ be the clusters.

# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.

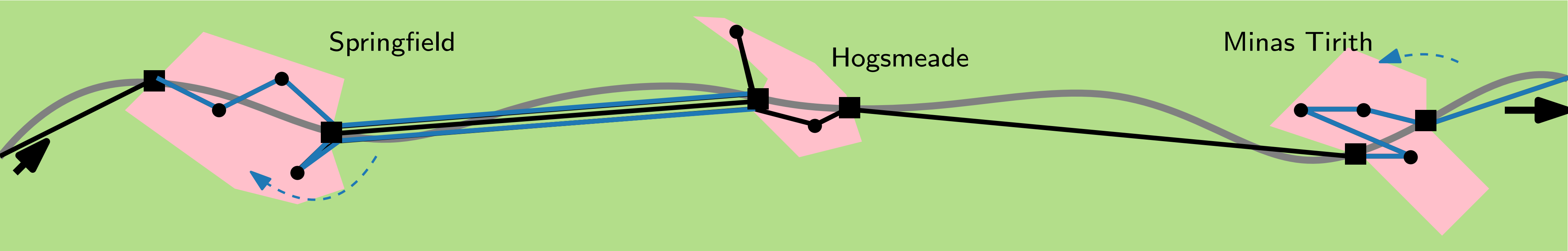



Let $C_1, \ldots C_q$ be the clusters.

Let $\Upsilon(T, C_i) \in \mathbb{R}^+$

# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.

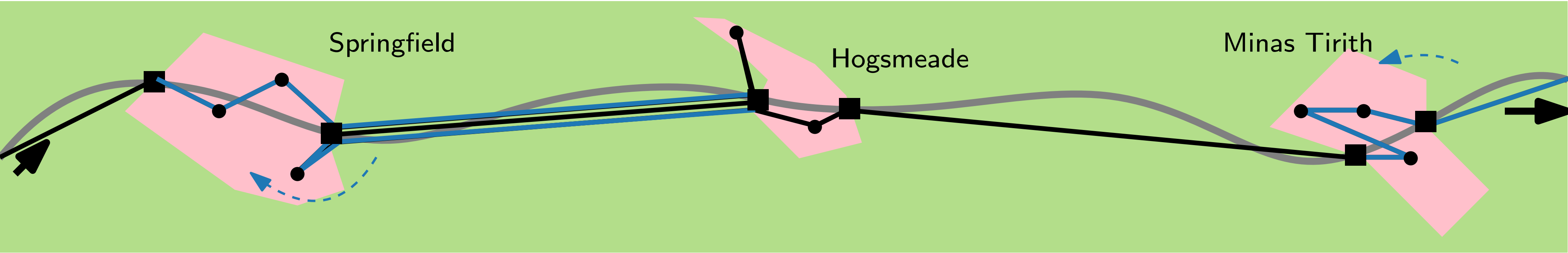



Let $C_1, \ldots C_q$ be the clusters.

Let $\Upsilon(T, C_i) \in \mathbb{R}^+$ such that $\forall T\colon \sum_{i=1}^{q} \Upsilon(T, C_i) = c(T)$

# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.



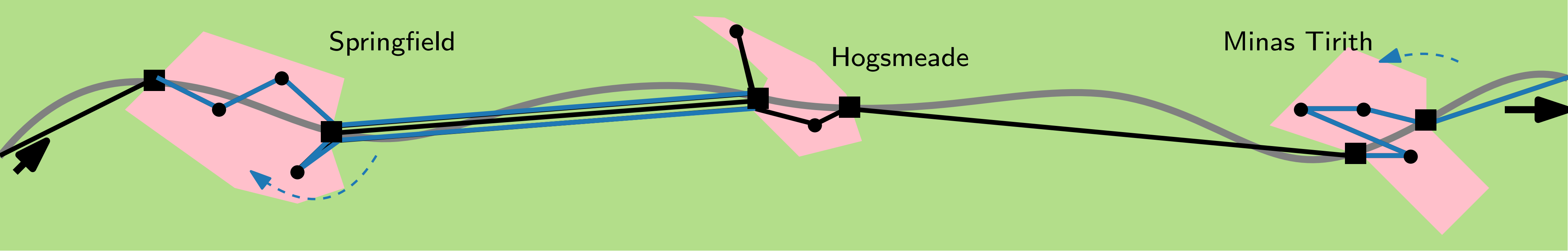

Let $C_1, \ldots C_q$ be the clusters.

Let $\Upsilon(T, C_i) \in \mathbb{R}^+$ such that $\forall T\colon \sum_{i=1}^{q} \Upsilon(T, C_i) = c(T)$

Let $\Phi(C_i)$ be a lower bound on $\Upsilon(T^*, C_i)$.

# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.




Let $C_1, \ldots C_q$ be the clusters.

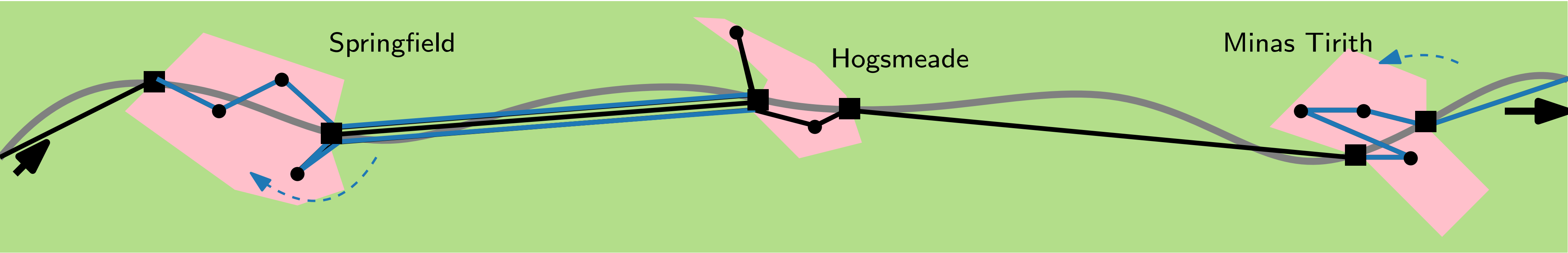
Let $\Upsilon(T, C_i) \in \mathbb{R}^+$ such that $\forall T\colon \sum_{i=1}^{q} \Upsilon(T, C_i) = c(T)$

Let $\Phi(C_i)$ be a lower bound on $\Upsilon(T^*, C_i)$.

**Theorem** (= Classifier):

# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.




Let $C_1, \dots C_q$ be the clusters.

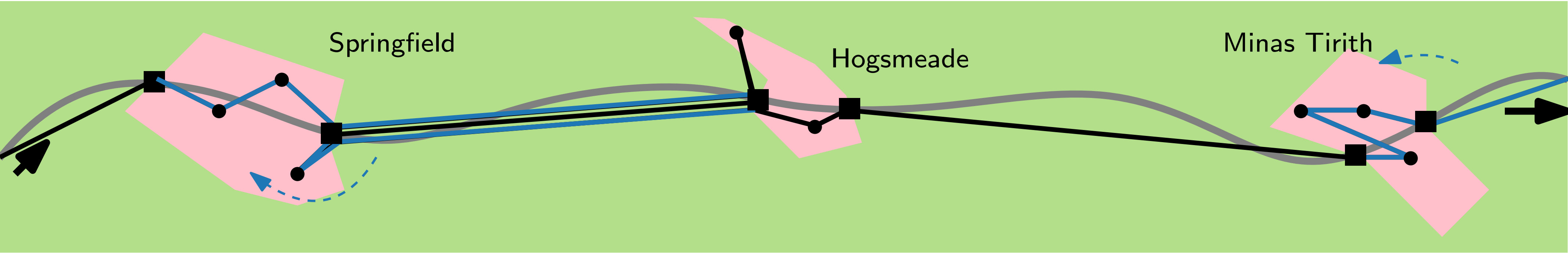
Let $\Upsilon(T, C_i) \in \mathbb{R}^+$ such that $\forall T: \sum_{i=1}^{q} \Upsilon(T, C_i) = c(T)$

Let $\Phi(C_i)$ be a lower bound on $\Upsilon(T^*, C_i)$.

**Theorem** (= Classifier): $\forall C_i: \Phi(C_i) = \Upsilon(\overrightarrow{T^*}, C_i) \Rightarrow T^* = \overrightarrow{T^*}$

# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.




Let $C_1, \ldots C_q$ be the clusters.

Let $\Upsilon(T, C_i) \in \mathbb{R}^+$ such that $\forall T\colon \sum_{i=1}^{q} \Upsilon(T, C_i) = c(T)$

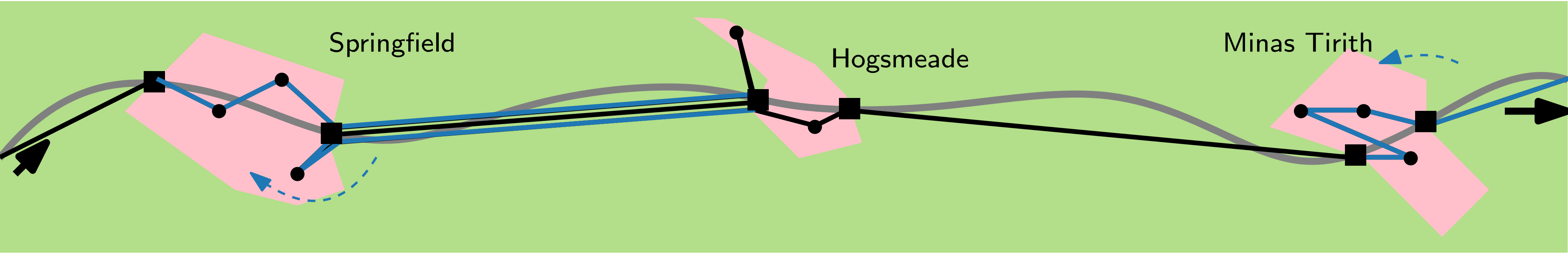
Let $\Phi(C_i)$ be a lower bound on $\Upsilon(T^*, C_i)$.

**Theorem** (= Classifier): $\forall C_i\colon \Phi(C_i) = \Upsilon(\overrightarrow{T^*}, C_i) \Rightarrow T^* = \overrightarrow{T^*}$

*Proof.* Via exchange argument. $\square$

# A Classifier

**Idea:** Distribute the costs of a tour to the clusters.




Let $C_1, \ldots C_q$ be the clusters.

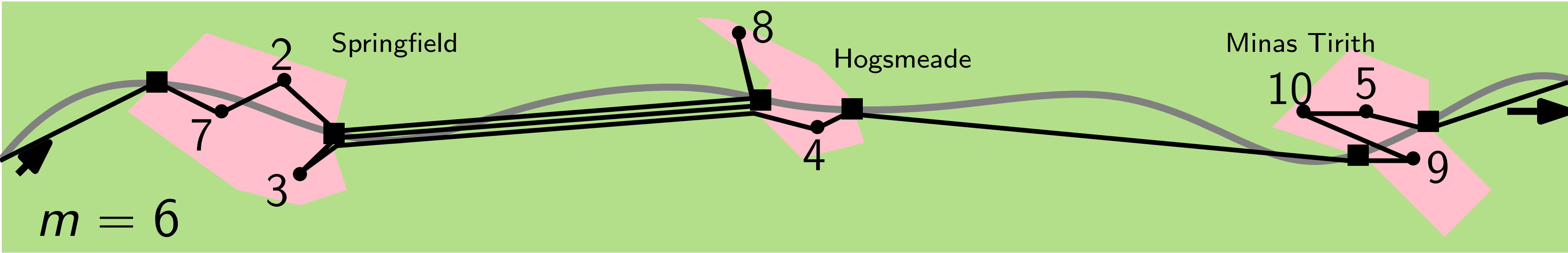
Let $\Upsilon(T, C_i) \in \mathbb{R}^+$ such that $\forall T\colon \sum_{i=1}^{q} \Upsilon(T, C_i) = c(T)$

Let $\Phi(C_i)$ be a lower bound on $\Upsilon(T^*, C_i)$. TODO!

**Theorem** (= Classifier): $\forall C_i\colon \Phi(C_i) = \Upsilon(\overrightarrow{T^*}, C_i) \Rightarrow T^* = \overrightarrow{T^*}$

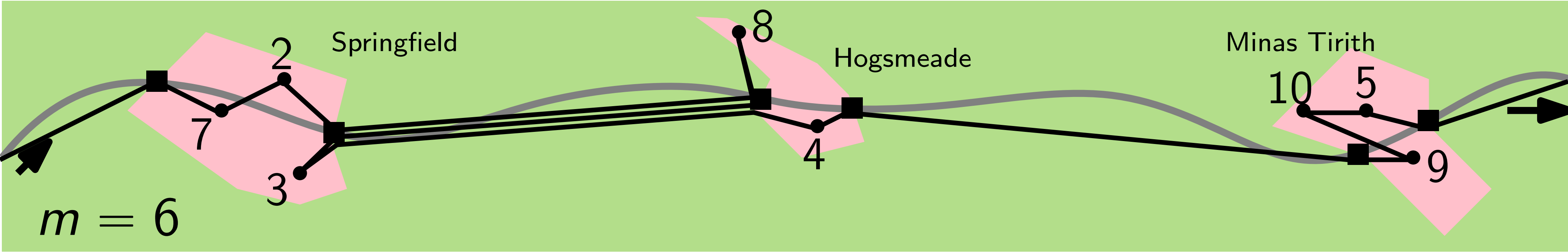
*Proof.* Via exchange argument. □

# Distribute Costs to Clusters



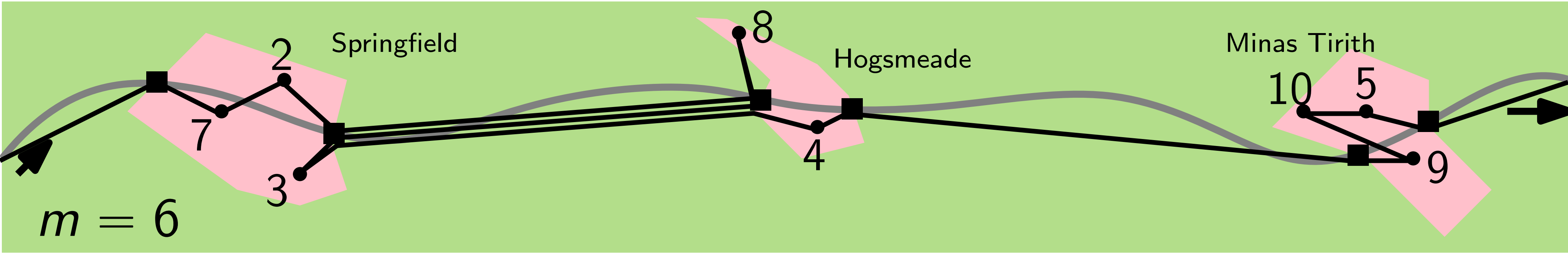
# Distribute Costs to Clusters

Assign the parts of a tour to clusters.



# Distribute Costs to Clusters

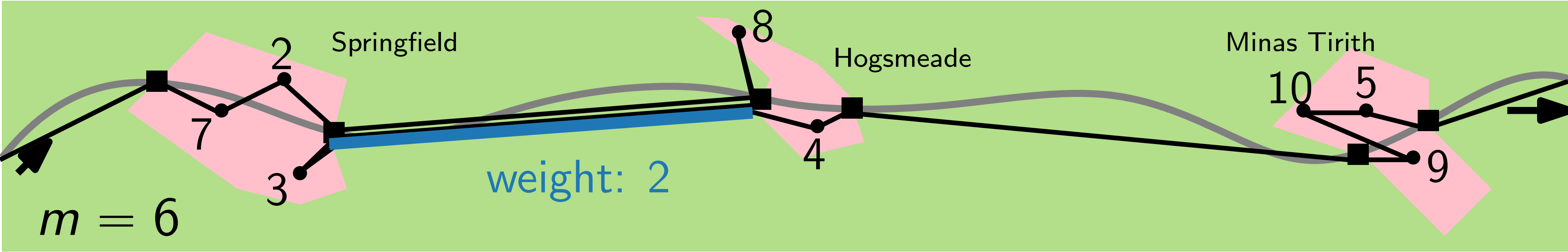
Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted.

# Distribute Costs to Clusters

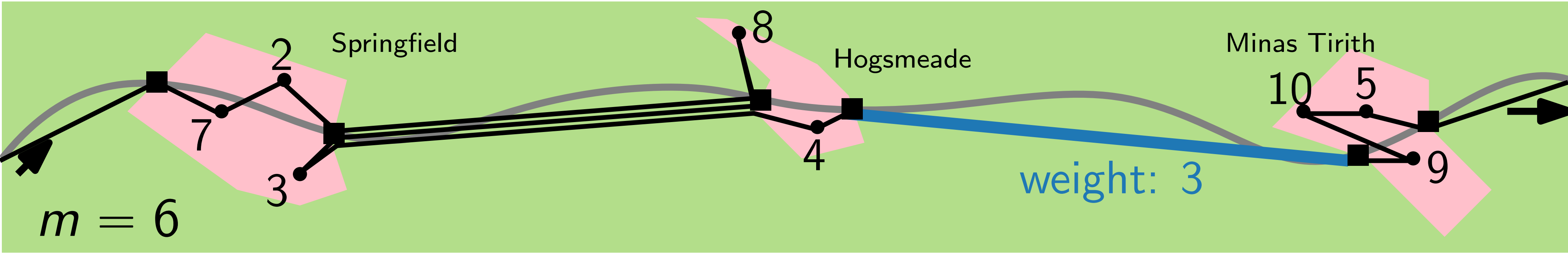
Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted.

# Distribute Costs to Clusters

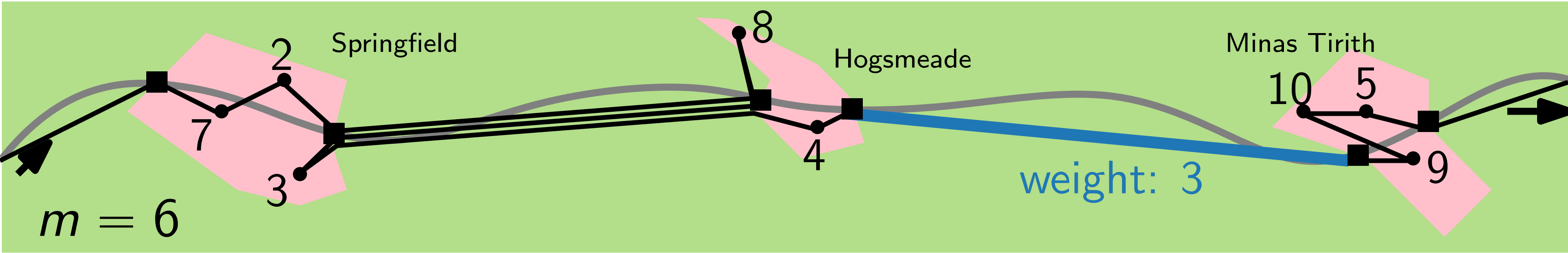
Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted.

# Distribute Costs to Clusters

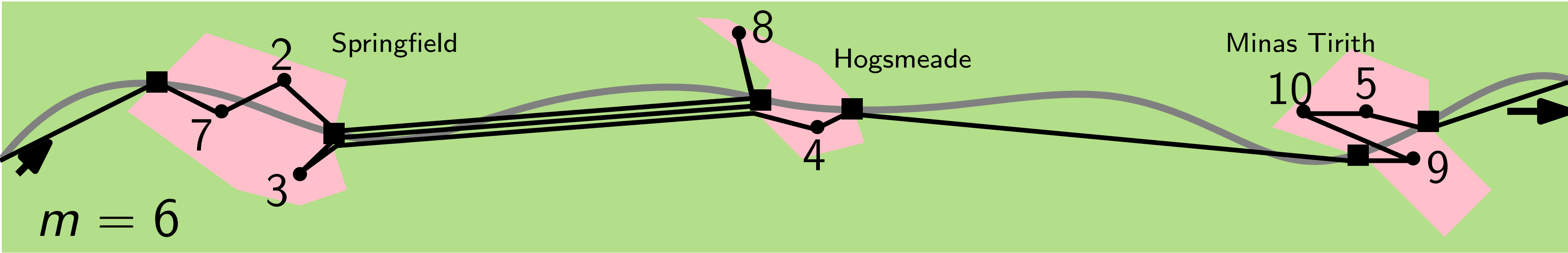
Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.

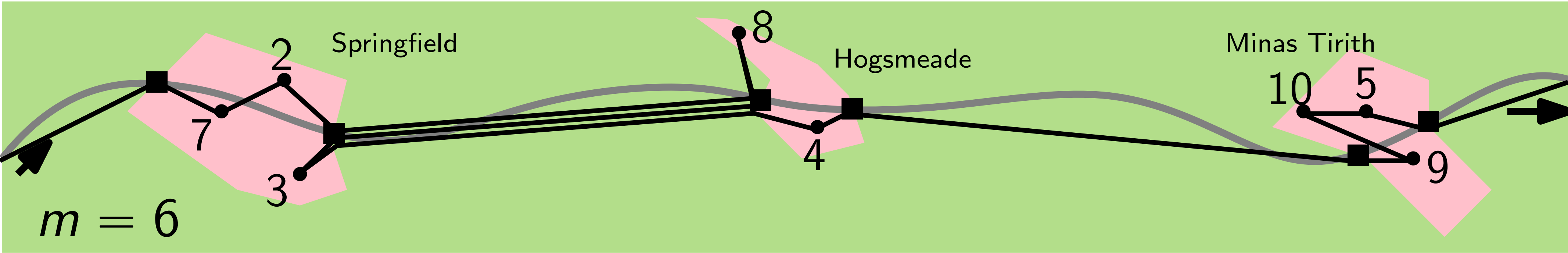



**Obs.:** Edges of a tour are weighted. $\rightarrow$ Count atomic journeys!

Every cluster $C_i$ has four counters:

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

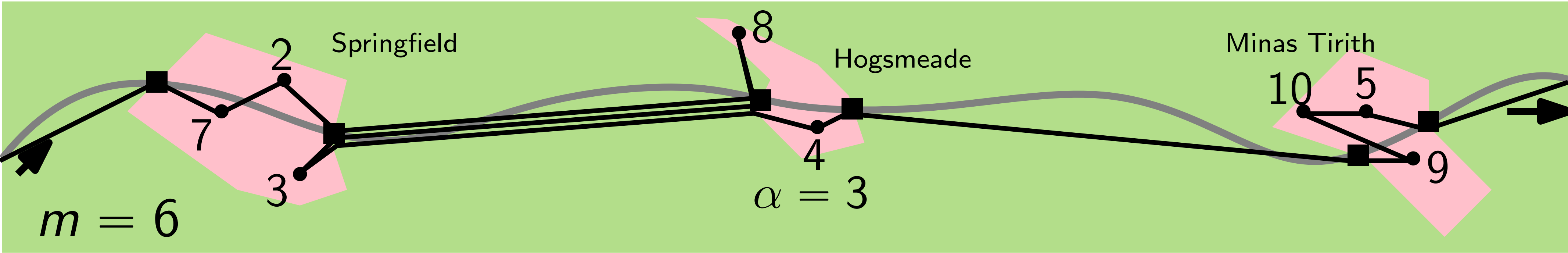
Every cluster $C_i$ has four counters:

pickup cluster of $r$

$\alpha := \#$rightbound persons with $p_r \leqslant i$.

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

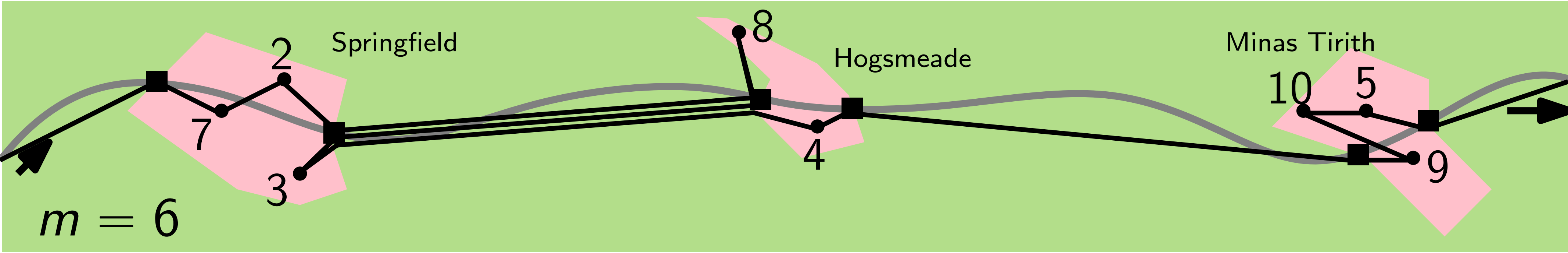
Every cluster $C_i$ has four counters:

pickup cluster of $r$

$\alpha :=$ #rightbound persons with $p_r \leqslant i$.

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

pickup cluster of $r$

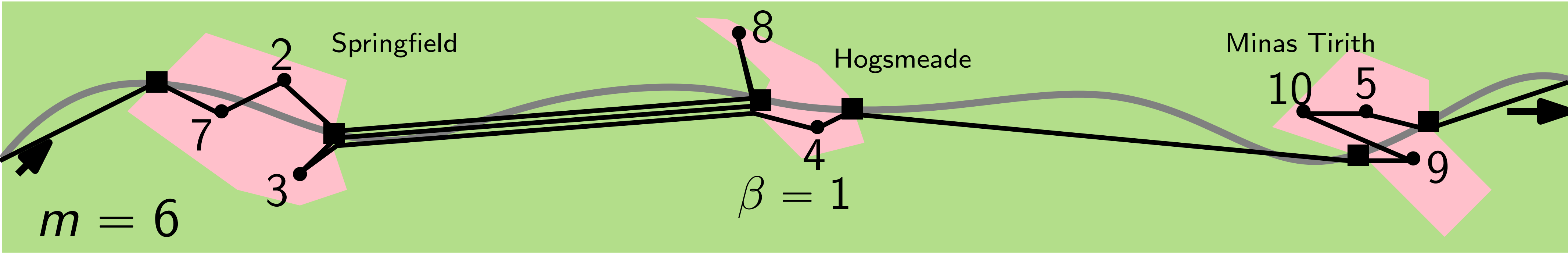
$\alpha :=$ #rightbound persons with $p_r \leqslant i$.

dropoff cluster of $r$

$\beta :=$ #leftbound persons with $d_r \geqslant i$.

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

pickup cluster of $r$

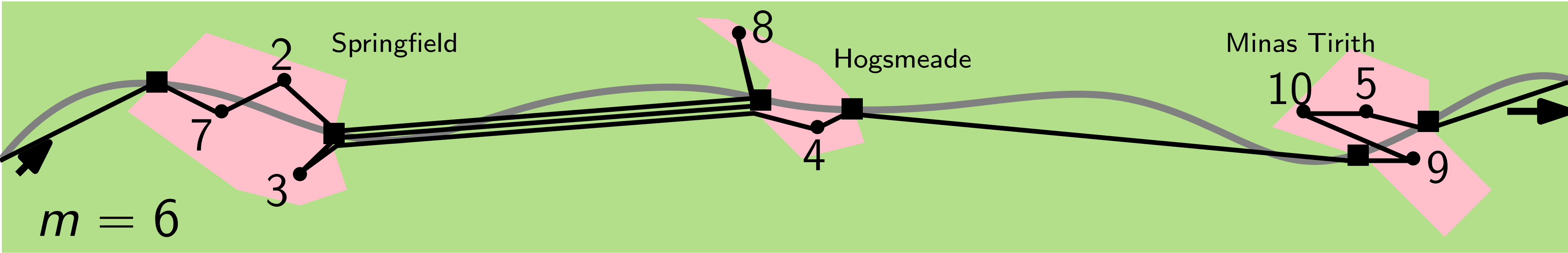
$\alpha :=$ #rightbound persons with $p_r \leqslant i$.

dropoff cluster of $r$

$\beta :=$ #leftbound persons with $d_r \geqslant i$.

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

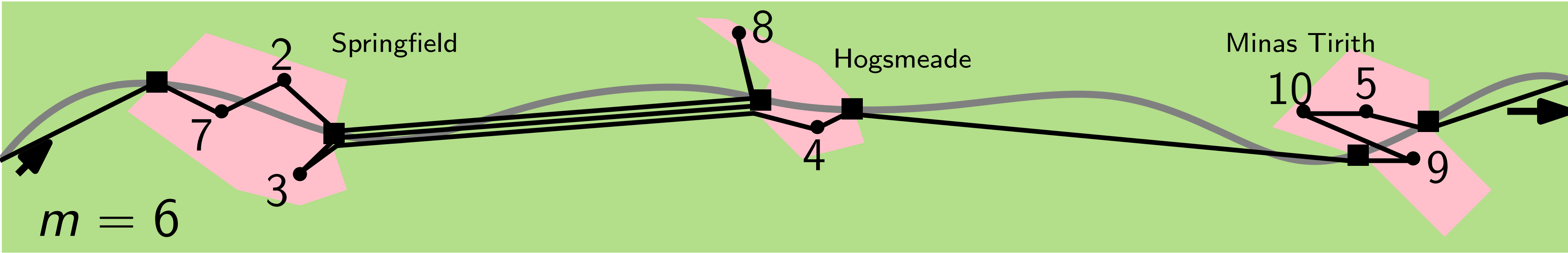
$\alpha :=$ #rightbound persons with $p_r \leqslant i$. (pickup cluster of $r$)

$\beta :=$ #leftbound persons with $d_r \geqslant i$. (dropoff cluster of $r$)

$\gamma :=$ #left-entering persons with $p_r \geqslant i$.

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

$\alpha :=$ #rightbound persons with $p_r \leqslant i$. (pickup cluster of $r$)

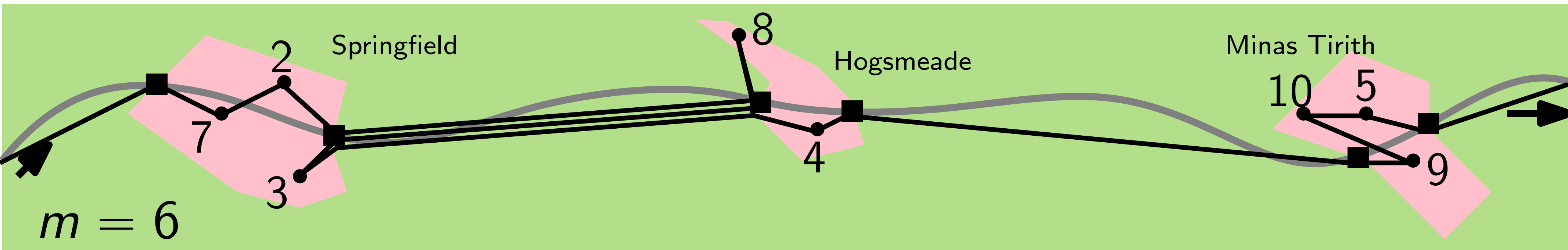
$\beta :=$ #leftbound persons with $d_r \geqslant i$. (dropoff cluster of $r$)

$\gamma :=$ #left-entering persons with $p_r \geqslant i$.

$\delta :=$ #right-entering persons with $d_r \leqslant i$.

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

pickup cluster of $r$

$\alpha :=$ #rightbound persons with $p_r \leqslant i$.

dropoff cluster of $r$

$\beta :=$ #leftbound persons with $d_r \geqslant i$.

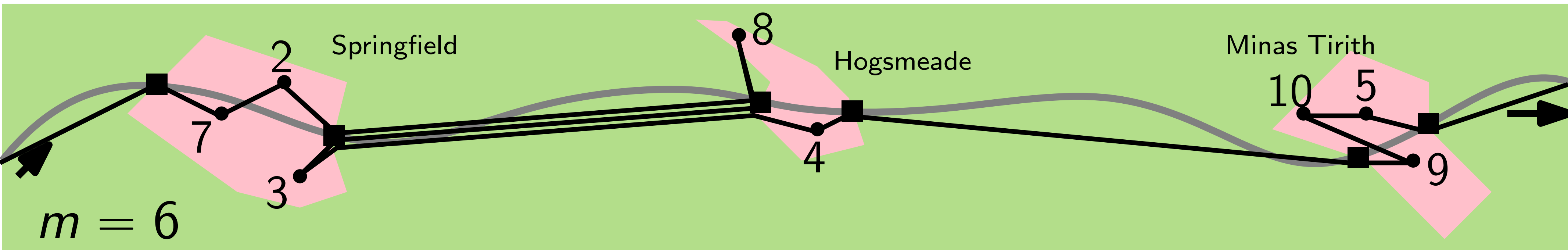
$\gamma :=$ #left-entering persons with $p_r \geqslant i$.

$\delta :=$ #right-entering persons with $d_r \leqslant i$.

$$\Upsilon(T, C_i) = \text{in}(C_i)$$

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

pickup cluster of $r$

$\alpha :=$ #rightbound persons with $p_r \leqslant i$.

dropoff cluster of $r$

$\beta :=$ #leftbound persons with $d_r \geqslant i$.

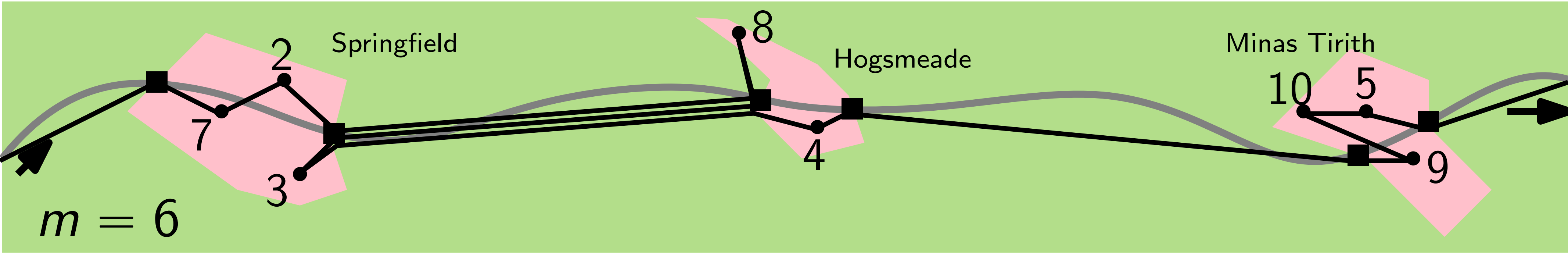
$\gamma :=$ #left-entering persons with $p_r \geqslant i$.

$\delta :=$ #right-entering persons with $d_r \leqslant i$.

$$\Upsilon(T, C_i) = \text{in}(C_i) + \alpha \overline{C_i C_{i+1}}$$

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

$\alpha :=$ #rightbound persons with $p_r \leqslant i$.

$\beta :=$ #leftbound persons with $d_r \geqslant i$.

$\gamma :=$ #left-entering persons with $p_r \geqslant i$.

$\delta :=$ #right-entering persons with $d_r \leqslant i$.

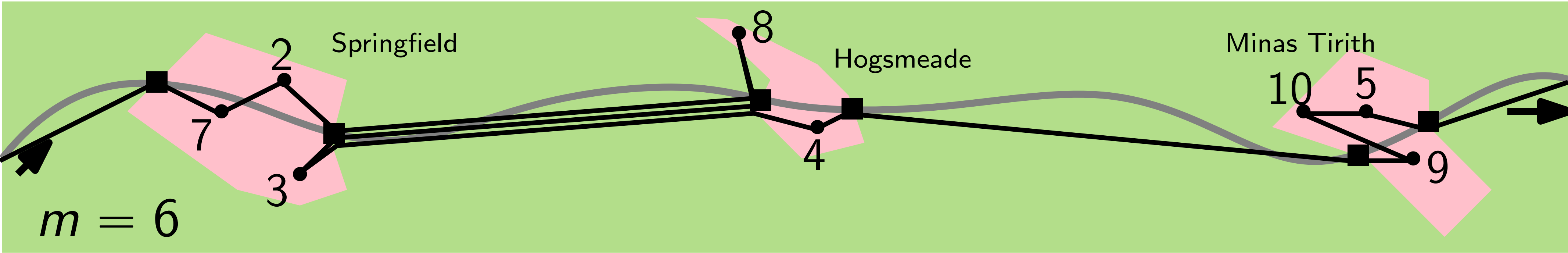
pickup cluster of $r$

dropoff cluster of $r$

$$\Upsilon(T, C_i) = \text{in}(C_i) + \alpha\overline{C_i C_{i+1}} + \beta\overline{C_i C_{i-1}}$$

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

$\alpha :=$ #rightbound persons with $p_r \leqslant i$. (pickup cluster of $r$)

$\beta :=$ #leftbound persons with $d_r \geqslant i$. (dropoff cluster of $r$)

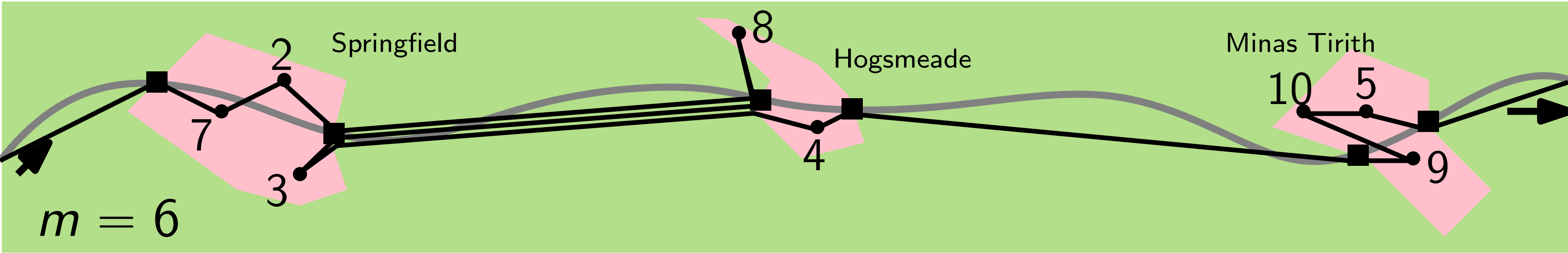
$\gamma :=$ #left-entering persons with $p_r \geqslant i$.

$\delta :=$ #right-entering persons with $d_r \leqslant i$.

$$\Upsilon(T, C_i) = \mathrm{in}(C_i) + \alpha\overline{C_i C_{i+1}} + \beta\overline{C_i C_{i-1}} + \gamma\overline{C_{i-1} C_i}$$

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

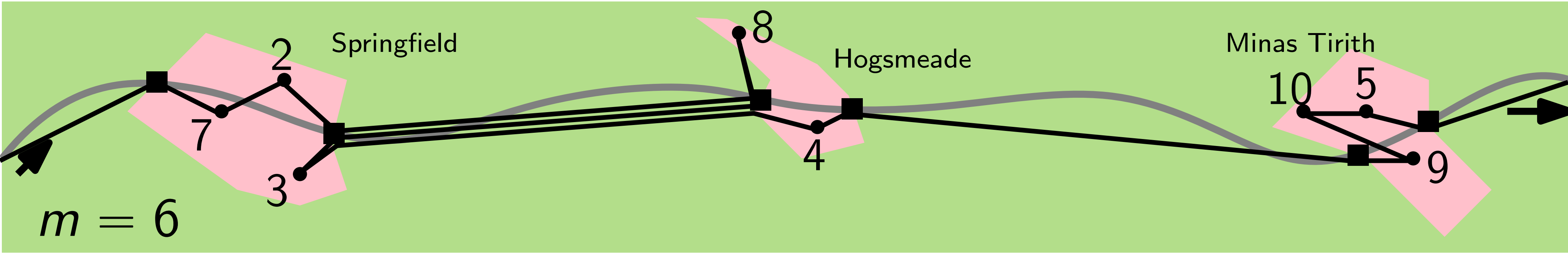
$\alpha :=$ #rightbound persons with $p_r \leqslant i$.
$\beta :=$ #leftbound persons with $d_r \geqslant i$.
$\gamma :=$ #left-entering persons with $p_r \geqslant i$.
$\delta :=$ #right-entering persons with $d_r \leqslant i$.

See thesis for proof of $c(T) = \sum \Upsilon(T, C_i)$.

$$\Upsilon(T, C_i) = \mathrm{in}(C_i) + \alpha \overline{C_i C_{i+1}} + \beta \overline{C_i C_{i-1}} + \gamma \overline{C_{i-1} C_i} + \delta \overline{C_{i+1} C_i}$$

# Distribute Costs to Clusters

Assign the parts of a tour to clusters.




**Obs.:** Edges of a tour are weighted. → Count atomic journeys!

Every cluster $C_i$ has four counters:

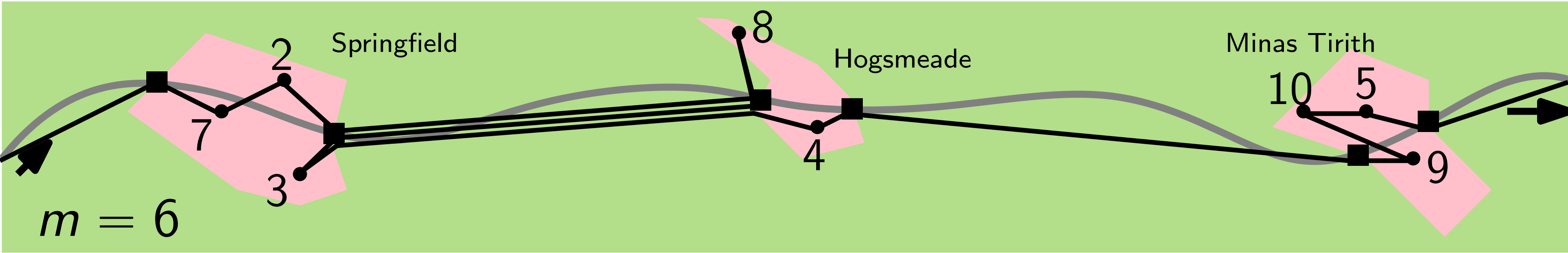
$\alpha :=$ #rightbound persons with $p_r \leqslant i$.
$\beta :=$ #leftbound persons with $d_r \geqslant i$.
$\gamma :=$ #left-entering persons with $p_r \geqslant i$.
$\delta :=$ #right-entering persons with $d_r \leqslant i$.

See thesis for proof of $c(T) = \sum \Upsilon(T, C_i)$.

$$\Upsilon(T, C_i) = \mathrm{in}(C_i) + \alpha \overline{C_i C_{i+1}} + \beta \overline{C_i C_{i-1}} + \gamma \overline{C_{i-1} C_i} + \delta \overline{C_{i+1} C_i}$$

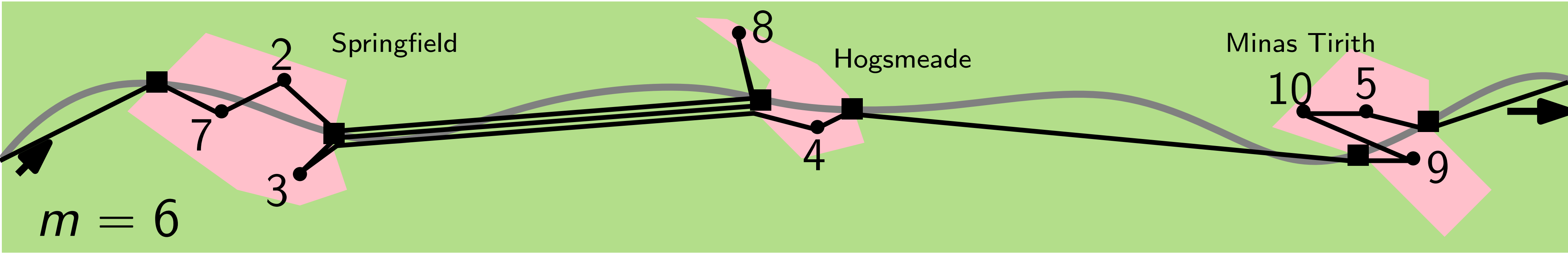
Todo: $\Phi(C_i) \leqslant \Upsilon(T^*, C_i)$

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)



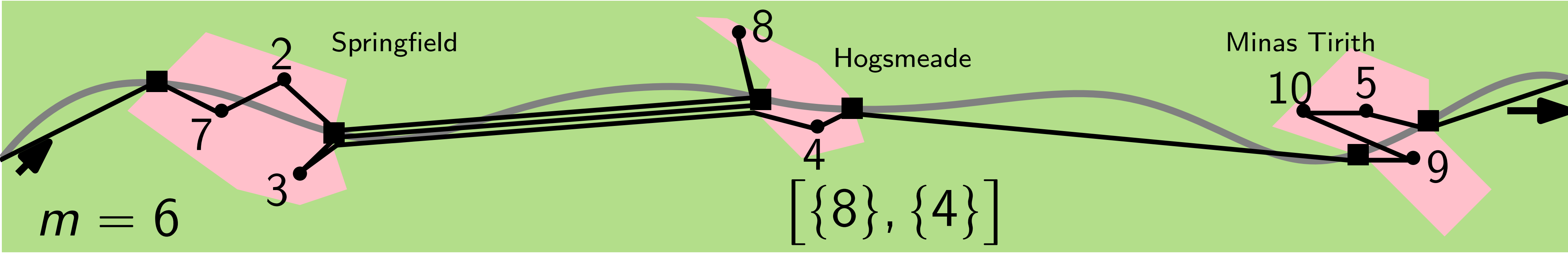
# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.



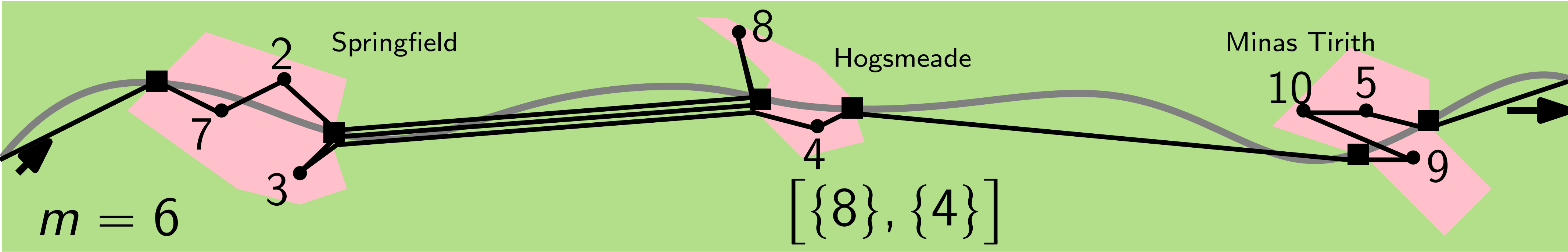
# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.



# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

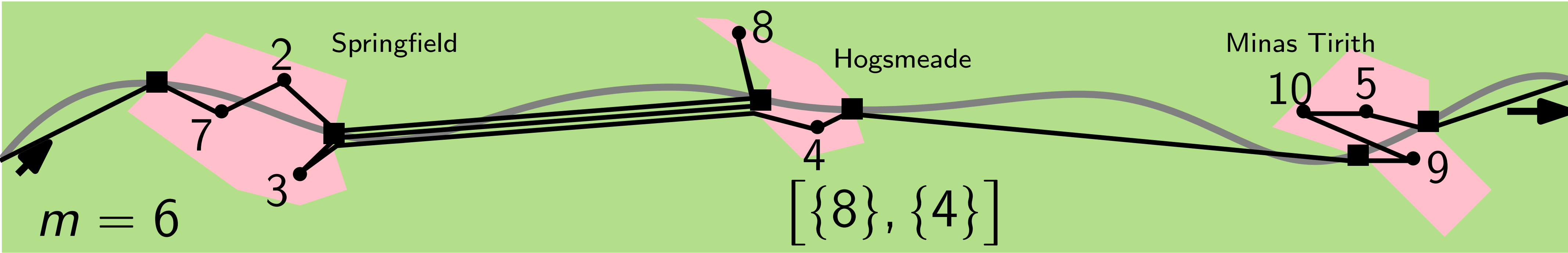
**Idea:** Any $T$ induces an ordered partition on every cluster.




Other Possibilities?

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

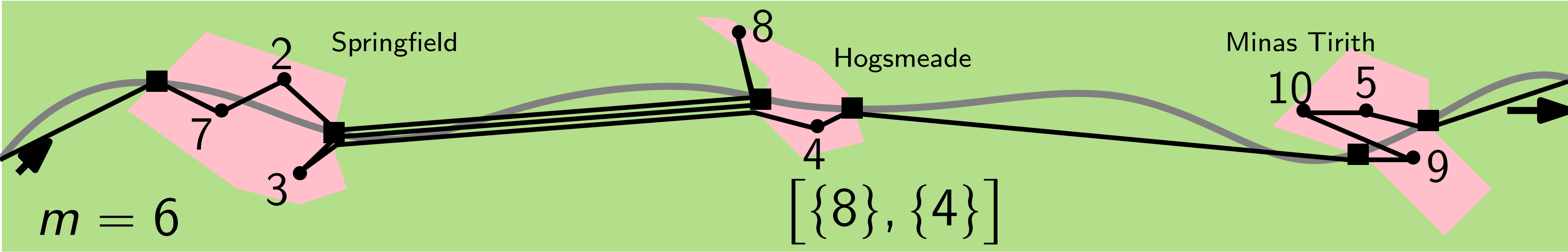
**Idea:** Any $T$ induces an ordered partition on every cluster.




Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.

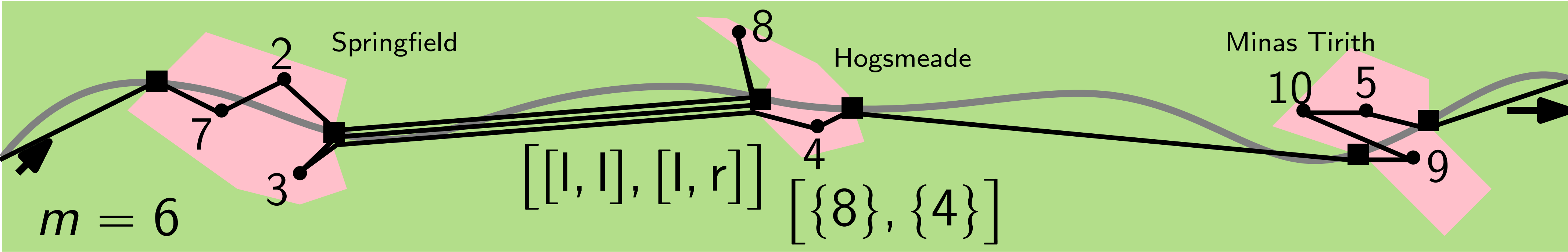



Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

Additionally: List of Portals $P$.

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.

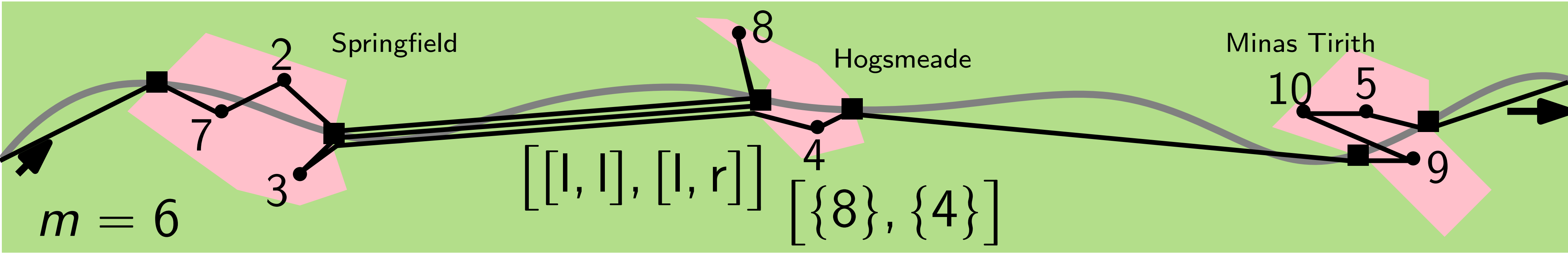



Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

Additionally: List of Portals $P$.

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.



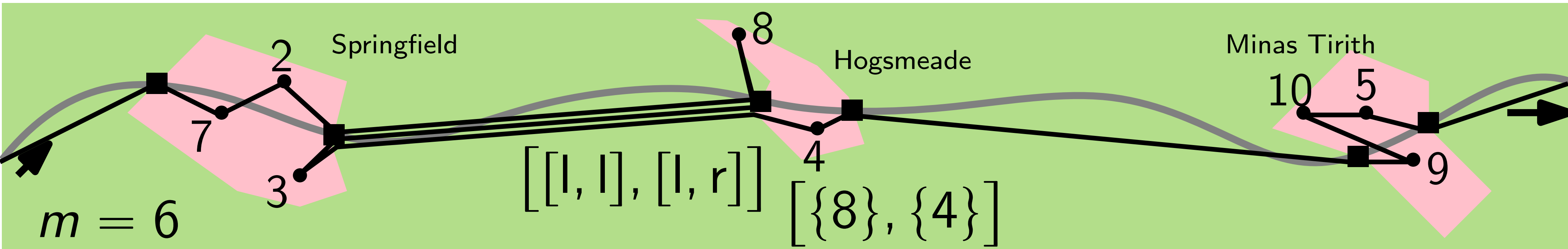

Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

Additionally: List of Portals $P$.

Given $\mathcal{S}$ and $P$ the lower bound can be estimated.

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.




Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

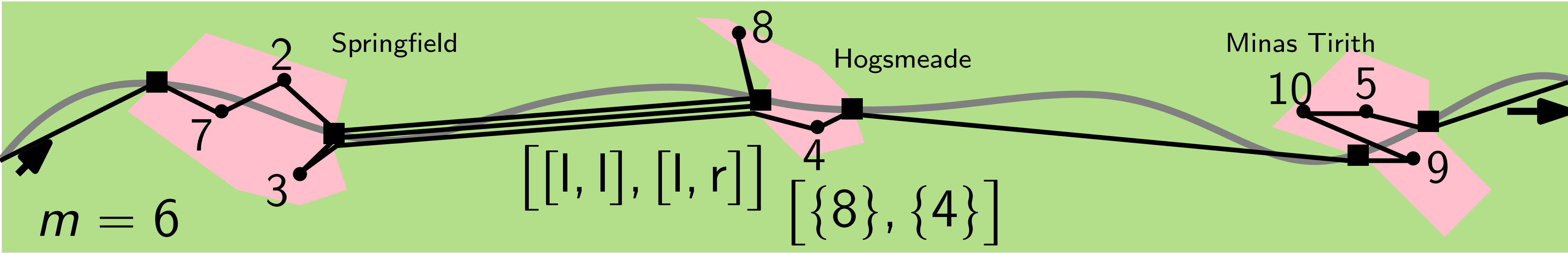
Additionally: List of Portals $P$.

Given $\mathcal{S}$ and $P$ the lower bound can be estimated.

Solve internal tours.

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.




Other Possibilities? $\mathcal{S} = [\{4\},\{8\}]$ $\mathcal{S} = [\{4,8\}]$

Additionally: List of Portals $P$.

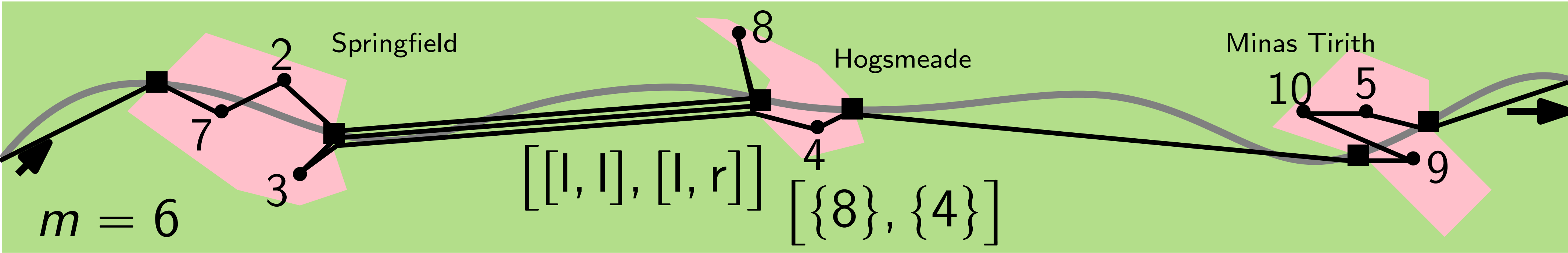
Given $\mathcal{S}$ and $P$ the lower bound can be estimated.

Solve internal tours.

Compute lower bounds for $\alpha, \beta, \gamma$ and $\delta$.

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.




Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

Additionally: List of Portals $P$.

Given $\mathcal{S}$ and $P$ the lower bound can be estimated.

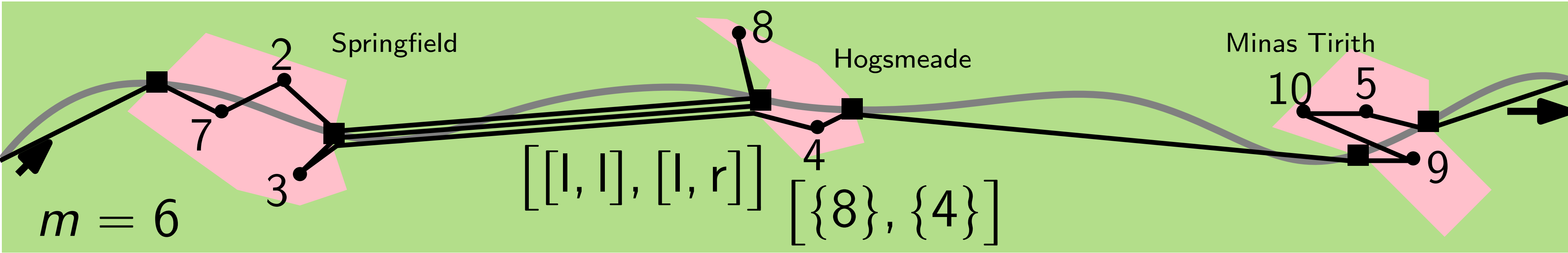
Solve internal tours.

Compute lower bounds for $\alpha, \beta, \gamma$ and $\delta$.

Add costs up and obtain lower bound $\Phi_{\mathcal{S},P}(C_i)$.

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.




Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

Additionally: List of Portals $P$.

Given $\mathcal{S}$ and $P$ the lower bound can be estimated.

Solve internal tours.

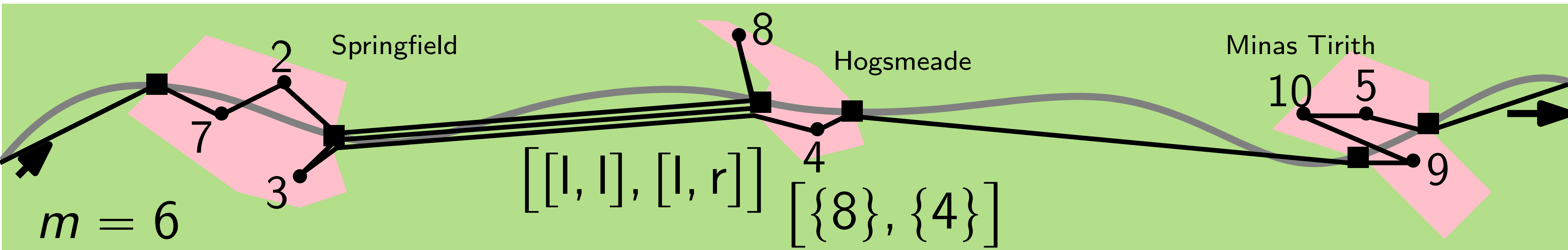
Compute lower bounds for $\alpha, \beta, \gamma$ and $\delta$.

Add costs up and obtain lower bound $\Phi_{\mathcal{S},P}(C_i)$.

$\Rightarrow \min \Phi(C_i)_{\mathcal{S},P} = \Phi(C_i) \leqslant \Upsilon(T^*, C_i)$

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.




Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

Additionally: List of Portals $P$.

Given $\mathcal{S}$ and $P$ the lower bound can be estimated.

$|C_i| = 6$: 5 227 236 choices

Solve internal tours.

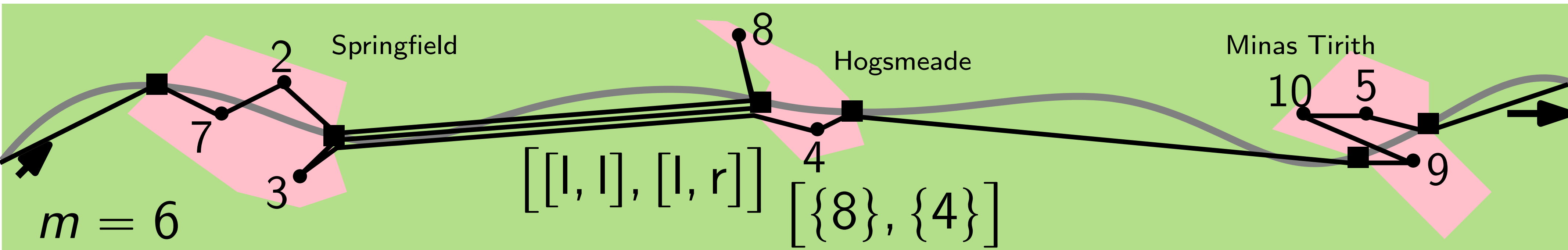
Compute lower bounds for $\alpha, \beta, \gamma$ and $\delta$.

Add costs up and obtain lower bound $\Phi_{\mathcal{S},P}(C_i)$.

$\Rightarrow \min \Phi(C_i)_{\mathcal{S},P} = \Phi(C_i) \leqslant \Upsilon(T^*, C_i)$

# Lower Bound on $\Upsilon(T^*, C_i)$ (Sketch)

**Idea:** Any $T$ induces an ordered partition on every cluster.




Other Possibilities? $\mathcal{S} = [\{4\}, \{8\}]$ $\mathcal{S} = [\{4, 8\}]$

Additionally: List of Portals $P$.

Given $\mathcal{S}$ and $P$ the lower bound can be estimated.

$|C_i| = 6$: 5 227 236 choices

Practical Limit!

Solve internal tours.

Compute lower bounds for $\alpha, \beta, \gamma$ and $\delta$.

Add costs up and obtain lower bound $\Phi_{\mathcal{S},P}(C_i)$.

$\Rightarrow \min \Phi(C_i)_{\mathcal{S},P} = \Phi(C_i) \leqslant \Upsilon(T^*, C_i)$

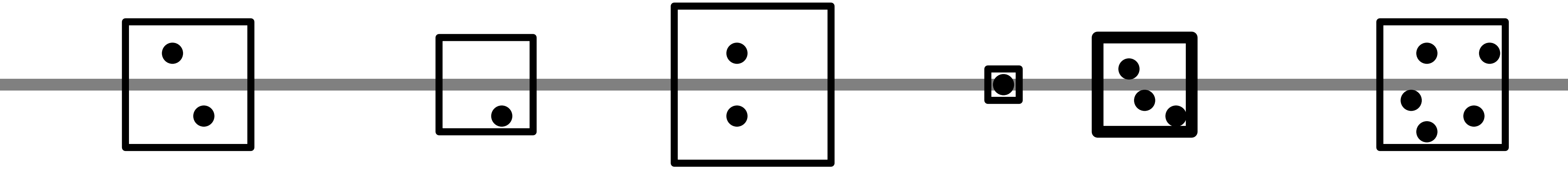
# Evaluation

# Evaluation

→ First artificial instances, then realistic instances.

# Evaluation

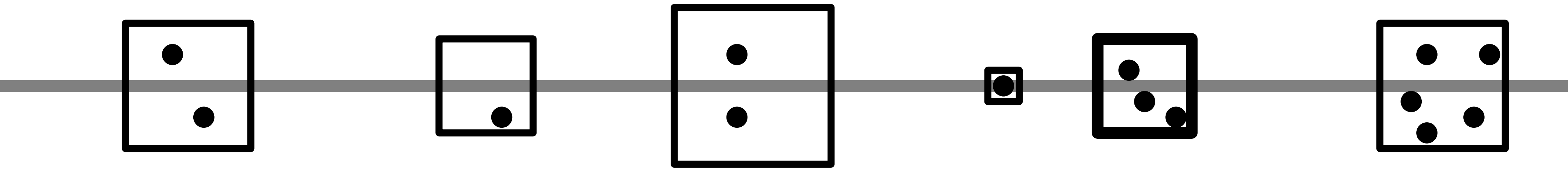
→ First artificial instances, then realistic instances.



# Evaluation

for $n = 12$

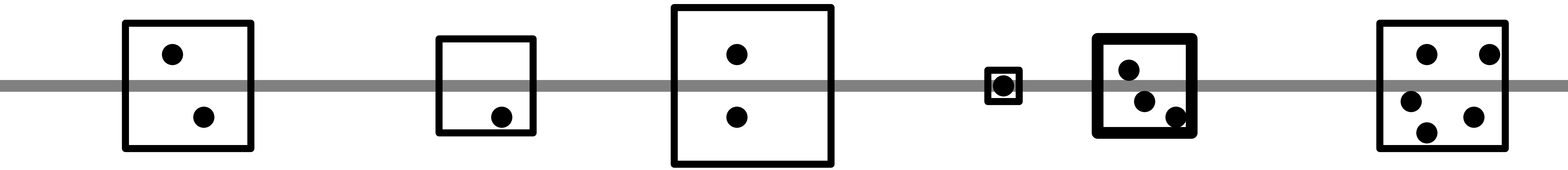
→ First artificial instances, then realistic instances.



# Evaluation

for $n = 12$

→ First artificial instances, then realistic instances.

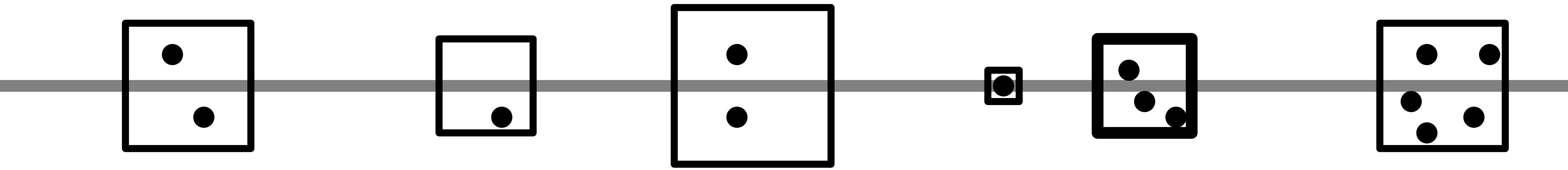


**Runtimes:**

# Evaluation

for $n = 12$

→ First artificial instances, then realistic instances.



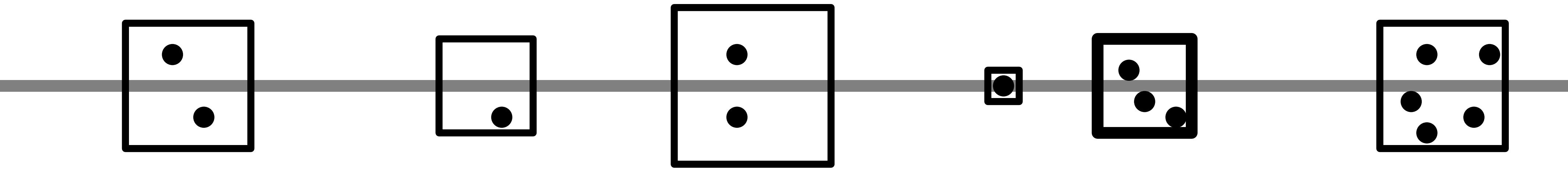
**Runtimes:**

**Exact:** 120 s  $\overrightarrow{T^*}$-**Algorithm**: 3 ms  **Classifier:** 4 s

# Evaluation

for $n = 12$

→ First artificial instances, then realistic instances.



**Runtimes:**

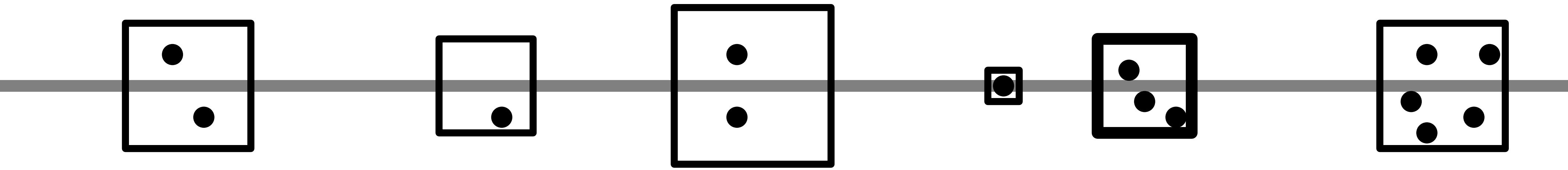
**Exact:** 120 s $\overrightarrow{T^*}$-**Algorithm**: 3 ms **Classifier:** 4 s

**Classifier's Accuracy**:

# Evaluation

for $n = 12$

→ First artificial instances, then realistic instances.



**Runtimes:**

**Exact:** 120 s $\overrightarrow{T^*}$-**Algorithm**: 3 ms **Classifier:** 4 s

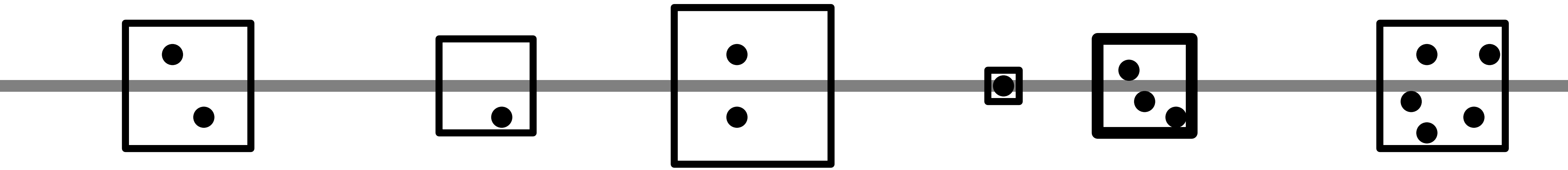
**Classifier's Accuracy**:

$$\text{Ratio } T^* = \overrightarrow{T^*}$$

# Evaluation

for $n = 12$

$\rightarrow$ First artificial instances, then realistic instances.



**Runtimes:**

**Exact:** 120 s $\overrightarrow{T^*}$-**Algorithm**: 3 ms **Classifier:** 4 s

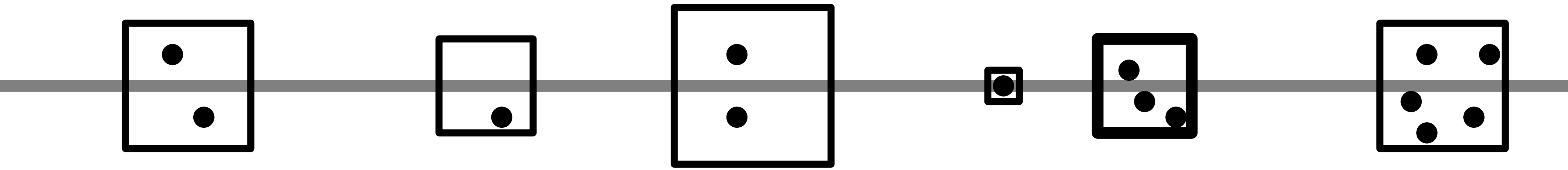
**Classifier's Accuracy**:

| | Ratio $T^* = \overrightarrow{T^*}$ |
|---|---|
| Clusters close together ($\sim$ 6km): | 59 % |

# Evaluation

for $n = 12$

$\rightarrow$ First artificial instances, then realistic instances.



**Runtimes:**

**Exact:** 120 s $\qquad$ $\overrightarrow{T^*}$-**Algorithm**: 3 ms $\qquad$ **Classifier:** 4 s

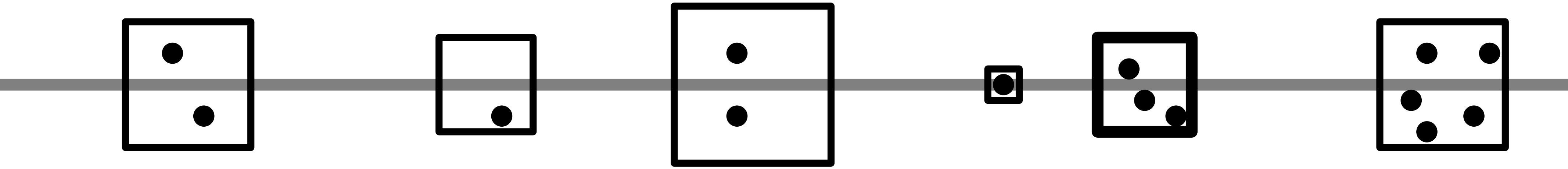
**Classifier's Accuracy**:

| | Ratio $T^* = \overrightarrow{T^*}$ |
|---|---|
| Clusters close together ($\sim$ 6km): | 59 % |
| far apart ($\geqslant$ 16 km): | 100 % |

# Evaluation

for $n = 12$

$\rightarrow$ First artificial instances, then realistic instances.



**Runtimes:**

**Exact:** 120 s $\overrightarrow{T^*}$-**Algorithm**: 3 ms **Classifier:** 4 s

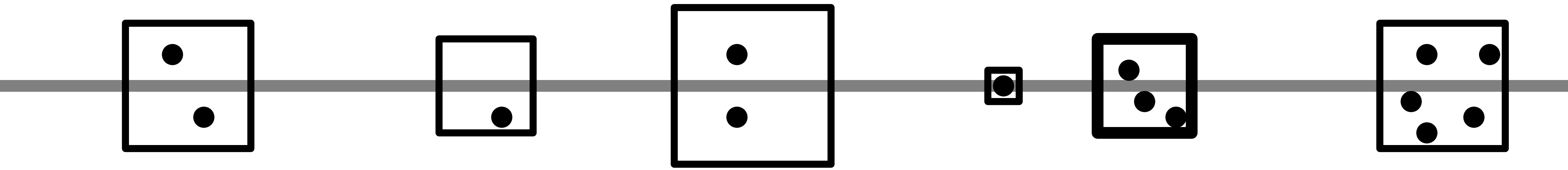
**Classifier's Accuracy**:

| | Ratio $T^* = \overrightarrow{T^*}$ | Recall |
|---|---|---|
| Clusters close together ($\sim$ 6km): | 59 % | |
| far apart ($\geqslant$ 16 km): | 100 % | |

# Evaluation

for $n = 12$

→ First artificial instances, then realistic instances.



**Runtimes:**

**Exact:** 120 s — $\overrightarrow{T^*}$-**Algorithm**: 3 ms — **Classifier:** 4 s

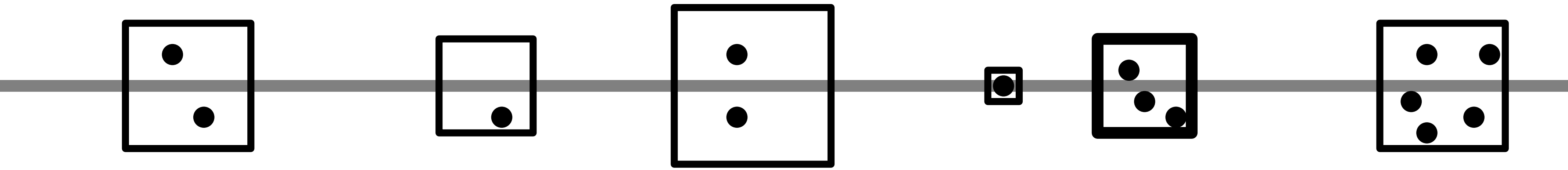
**Classifier's Accuracy**:

| | Ratio $T^* = \overrightarrow{T^*}$ | Recall |
|---|---|---|
| Clusters close together ($\sim$ 6km): | 59 % | 0.4 |
| far apart ($\geqslant$ 16 km): | 100 % | |

# Evaluation

for $n = 12$

→ First artificial instances, then realistic instances.



**Runtimes:**

**Exact:** 120 s $\qquad$ $\overrightarrow{T^*}$-**Algorithm**: 3 ms $\qquad$ **Classifier:** 4 s

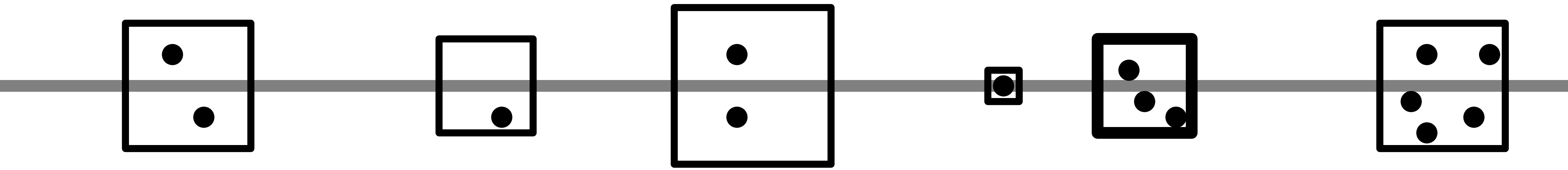
**Classifier's Accuracy**:

| | Ratio $T^* = \overrightarrow{T^*}$ | Recall |
|---|---|---|
| Clusters close together ($\sim$ 6km): | 59 % | 0.4 |
| far apart ($\geqslant$ 16 km): | 100 % | 0.9 |

# Evaluation

for $n = 12$

→ First artificial instances, then realistic instances.



**Runtimes:**

**Exact:** 120 s $\overrightarrow{T^*}$-**Algorithm**: 3 ms **Classifier:** 4 s

**Classifier's Accuracy**:

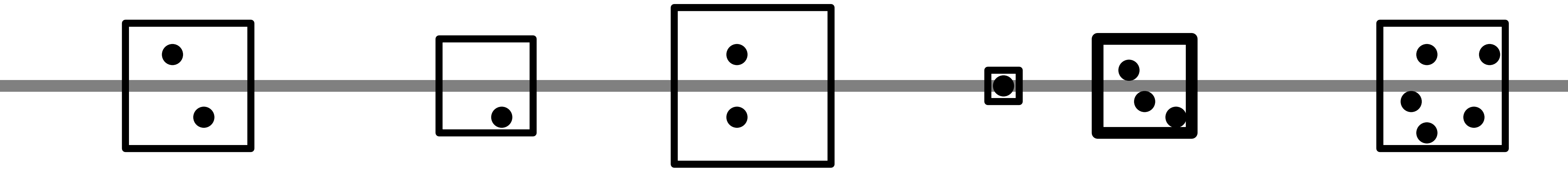
| | Ratio $T^* = \overrightarrow{T^*}$ | Recall |
|---|---|---|
| Clusters close together ($\sim$ 6km): | 59 % | 0.4 |
| far apart ($\geqslant$ 16 km): | 100 % | 0.9 |

$\overrightarrow{T^*}$-**Algorithm as Heuristic:**

# Evaluation

for $n = 12$

→ First artificial instances, then realistic instances.



**Runtimes:**

**Exact:** 120 s   $\overrightarrow{T^*}$-**Algorithm**: 3 ms   **Classifier:** 4 s

**Classifier's Accuracy**:

| | Ratio $T^* = \overrightarrow{T^*}$ | Recall |
|---|---|---|
| Clusters close together ($\sim$ 6km): | 59 % | 0.4 |
| far apart ($\geqslant$ 16 km): | 100 % | 0.9 |

$\overrightarrow{T^*}$-**Algorithm as Heuristic:**

Approximation Quality (empiric): $\leqslant 1.1$

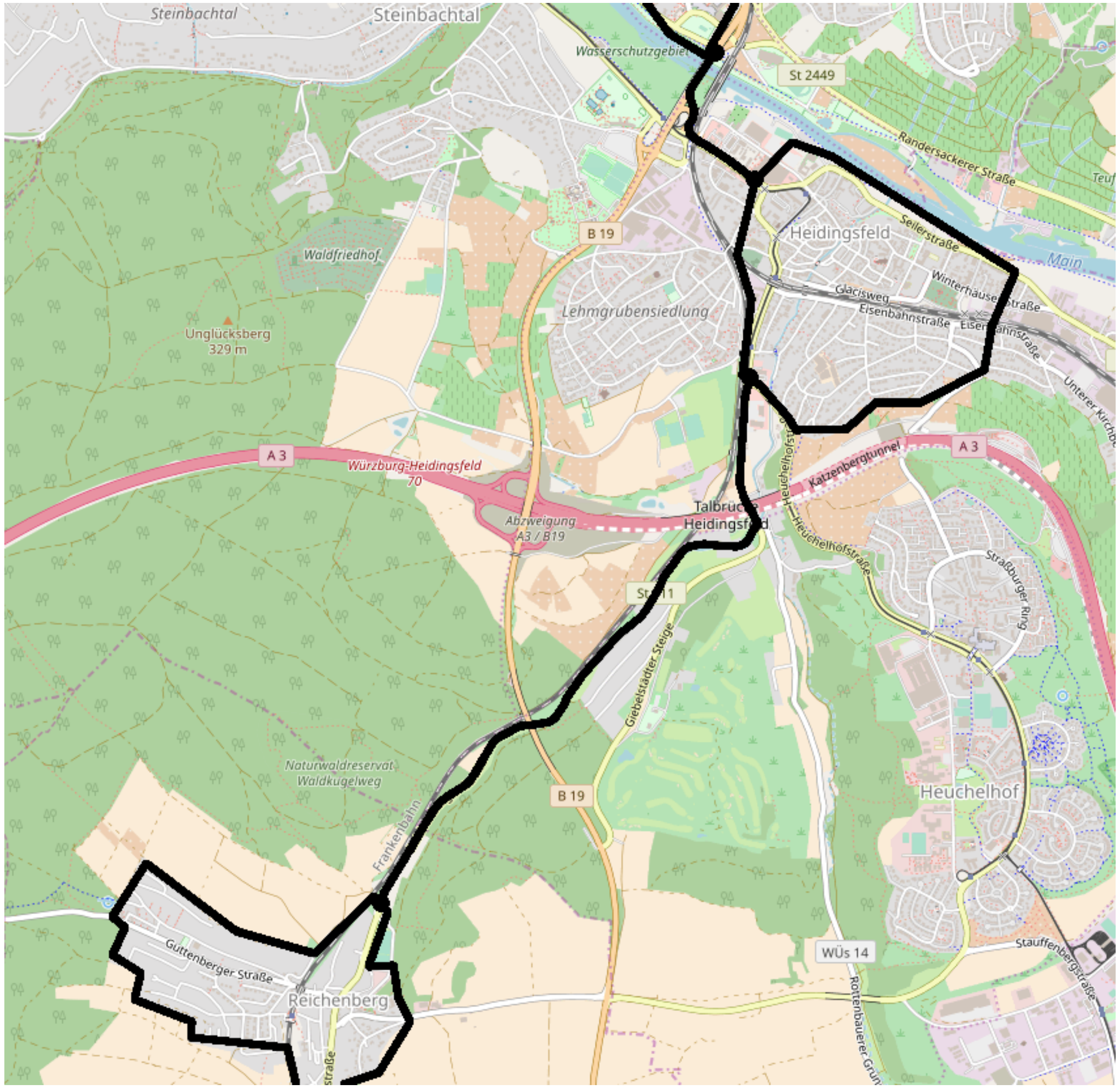
# Topology of Street Networks

# Topology of Street Networks

Street Networks often do not meet the assumptions.

# Topology of Street Networks

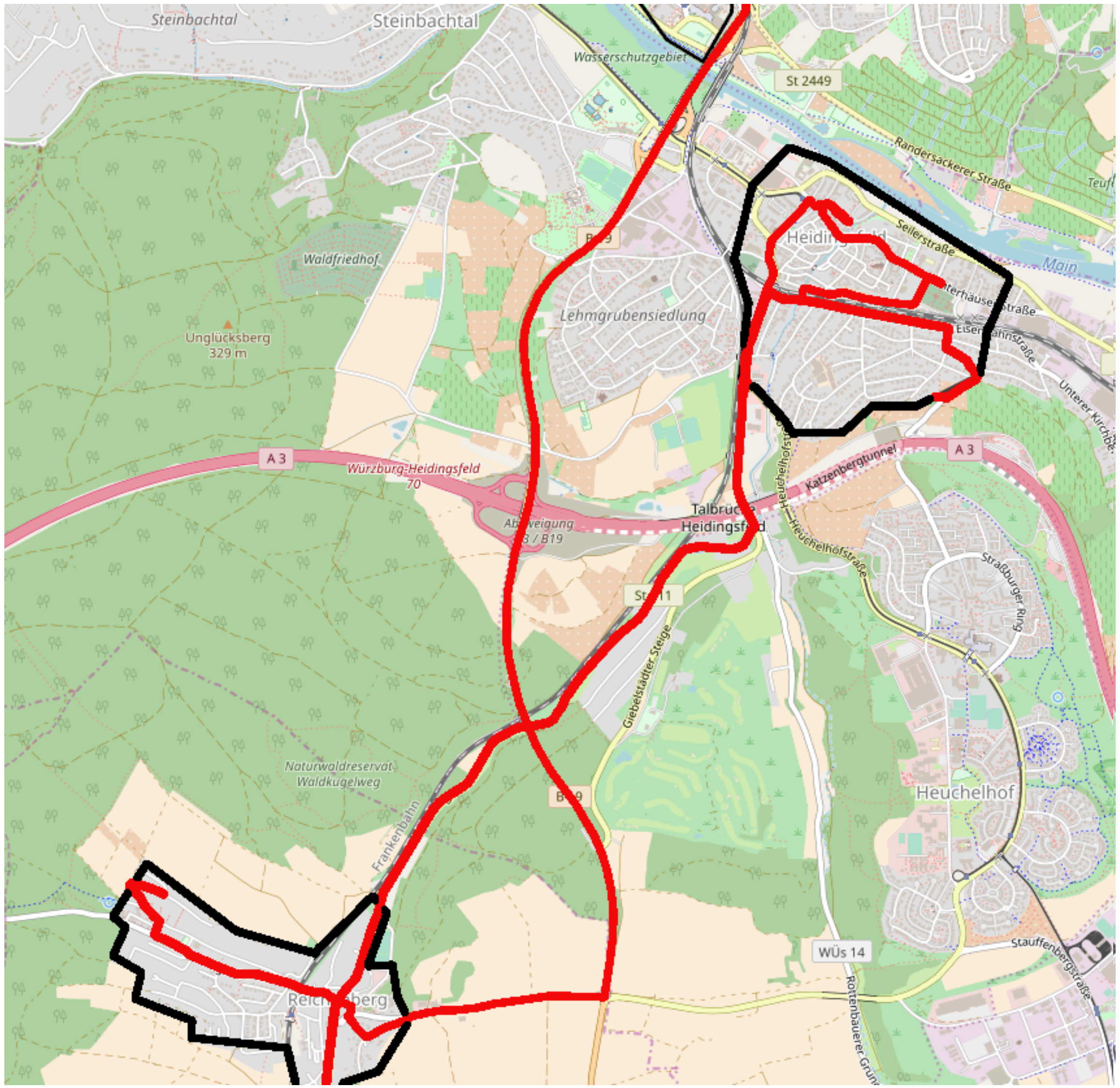
Street Networks often do not meet the assumptions.




Example #1:
Rural Instance

# Topology of Street Networks

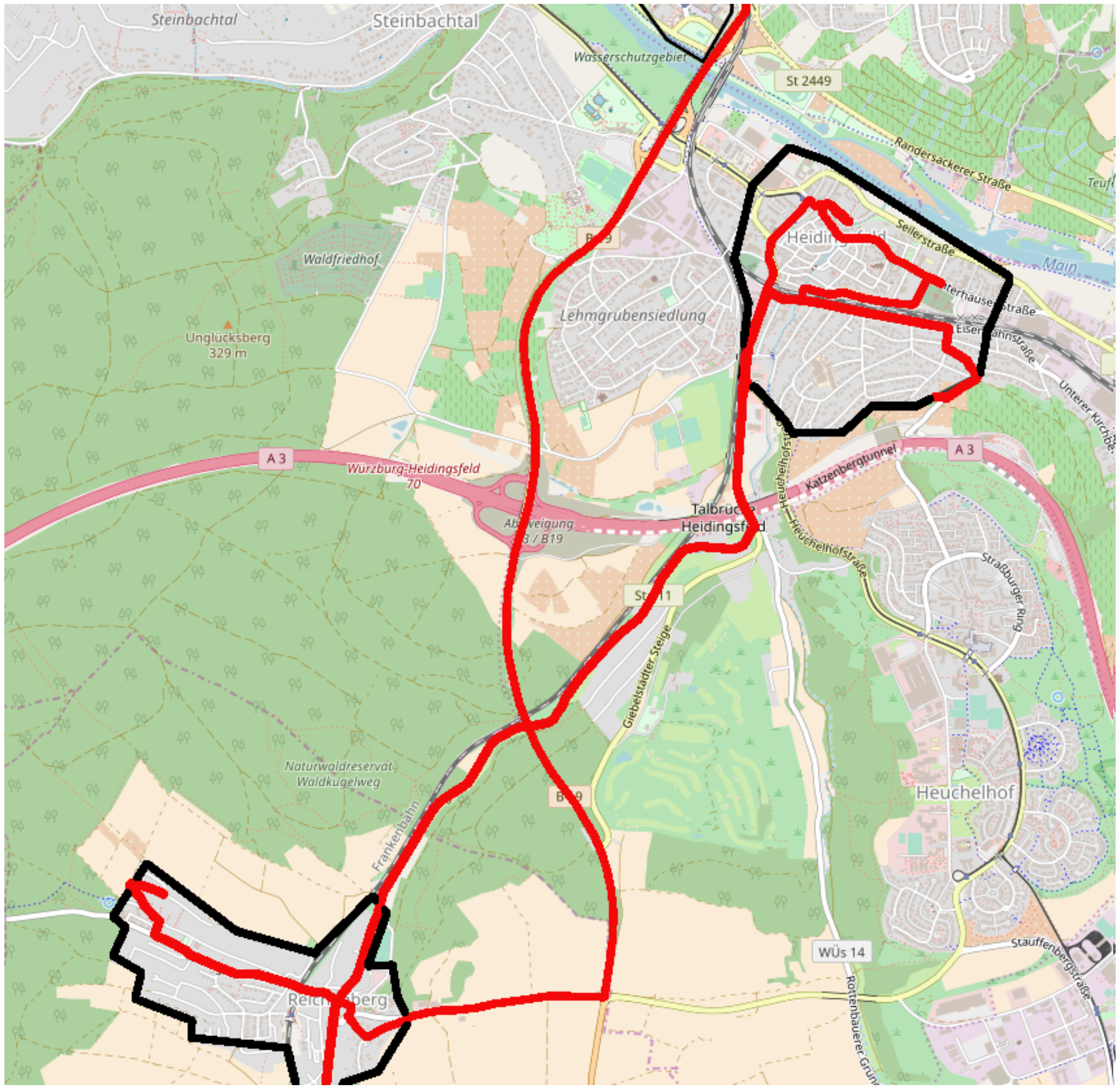
Street Networks often do not meet the assumptions.




Example #1:
Rural Instance

# Topology of Street Networks

Street Networks often do not meet the assumptions.

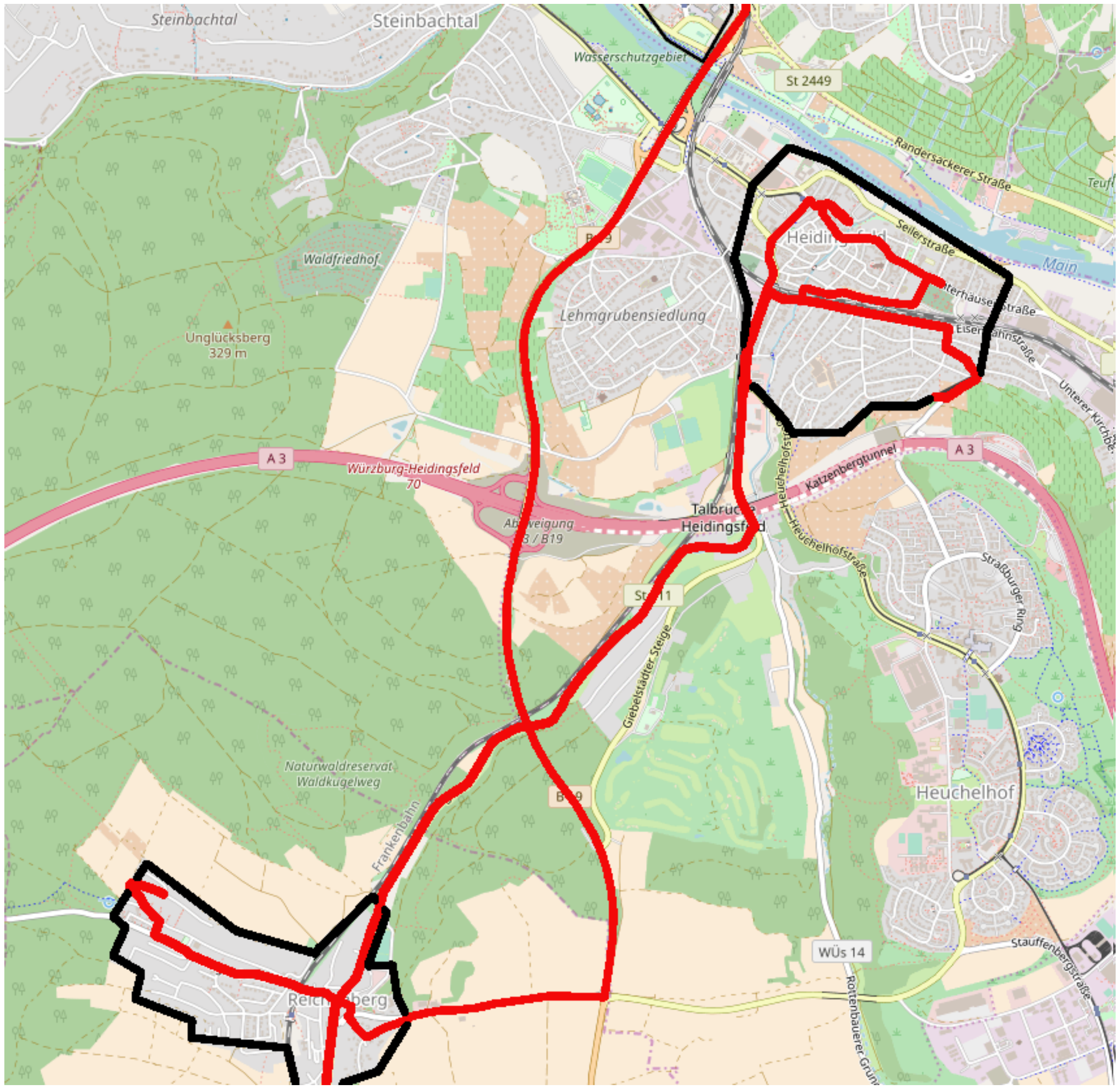



Example #1:
Rural Instance

$T^*$ bypasses a cluster!

# Topology of Street Networks

Street Networks often do not meet the assumptions.



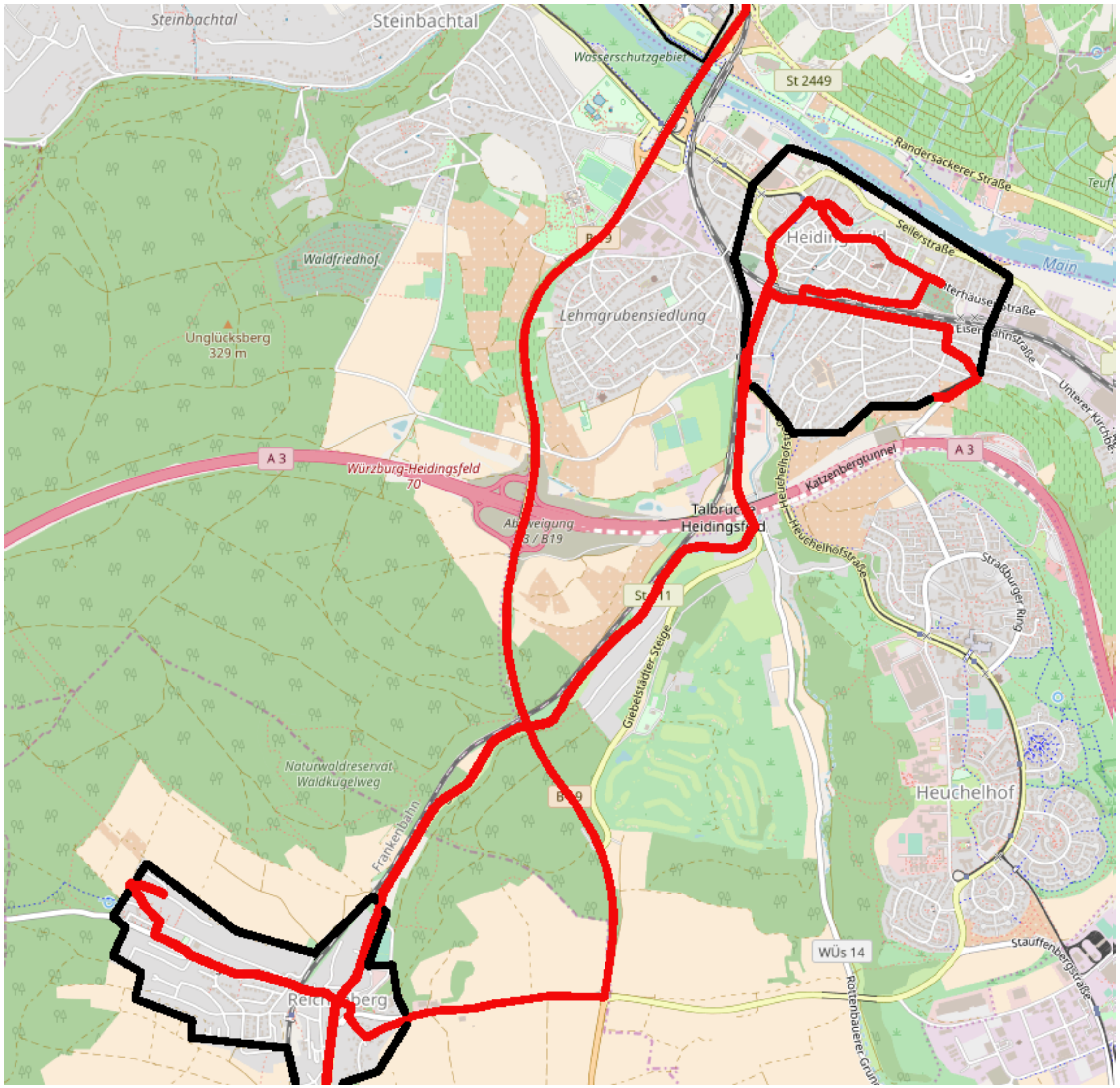

Example #1:
Rural Instance

$T^*$ bypasses a cluster!

Yet, no false positive.

# Topology of Street Networks

Street Networks often do not meet the assumptions.




Example #1:
Rural Instance

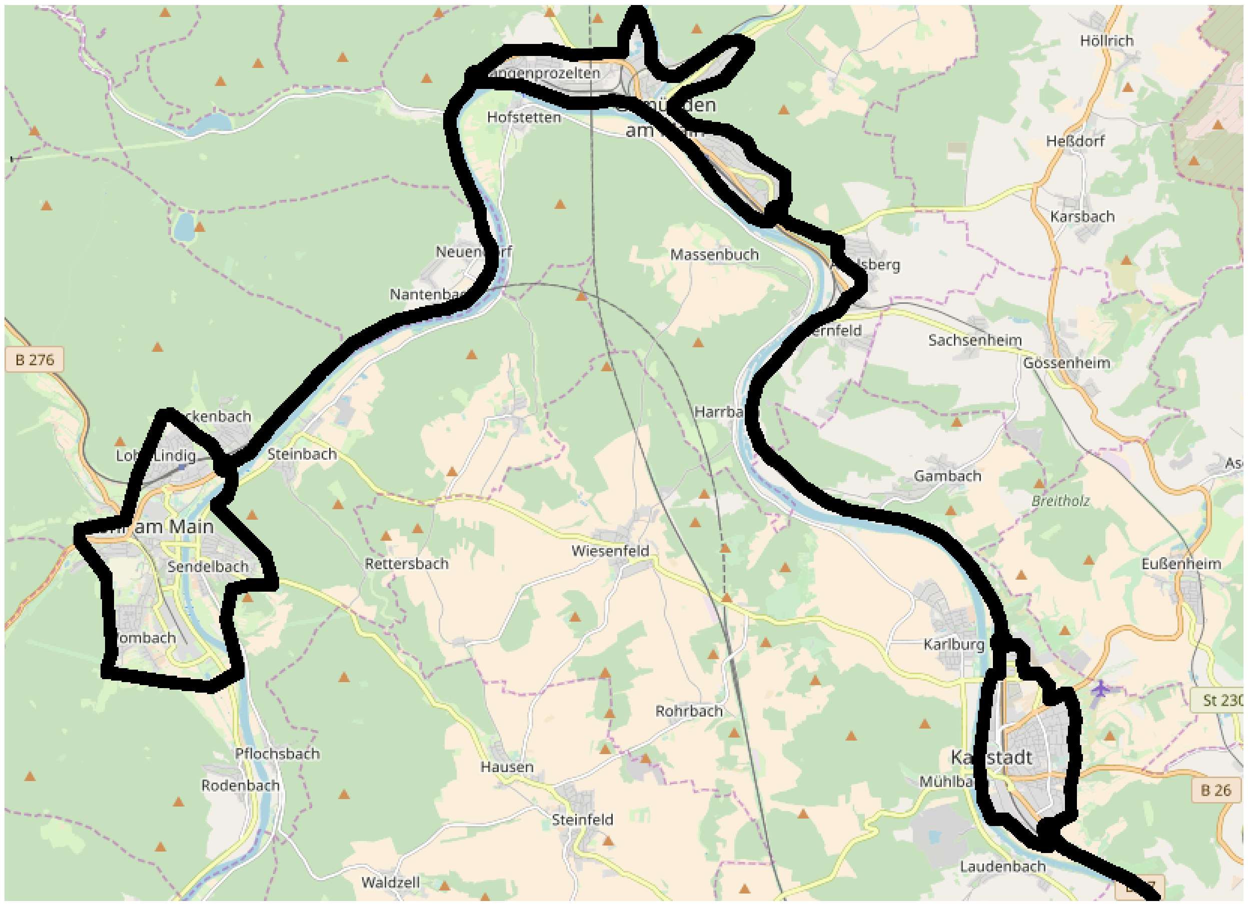
$T^*$ bypasses a cluster!

Yet, no false positive.

$\Rightarrow$ Classifier is robust to some extent.

# Topology of Street Networks

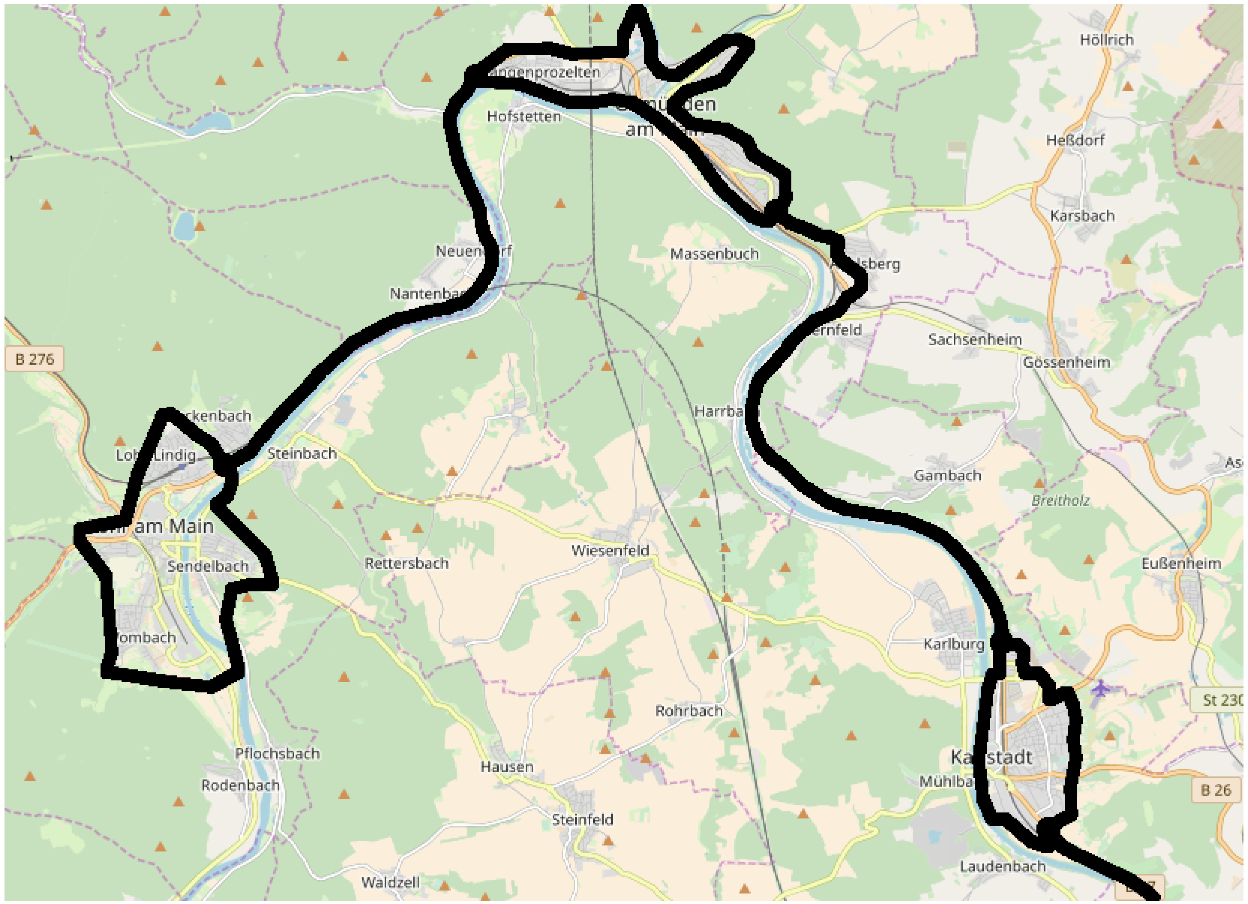
Street Networks often do not meet the assumptions.




Example #2:
Regional Instance

# Topology of Street Networks

Street Networks often do not meet the assumptions.

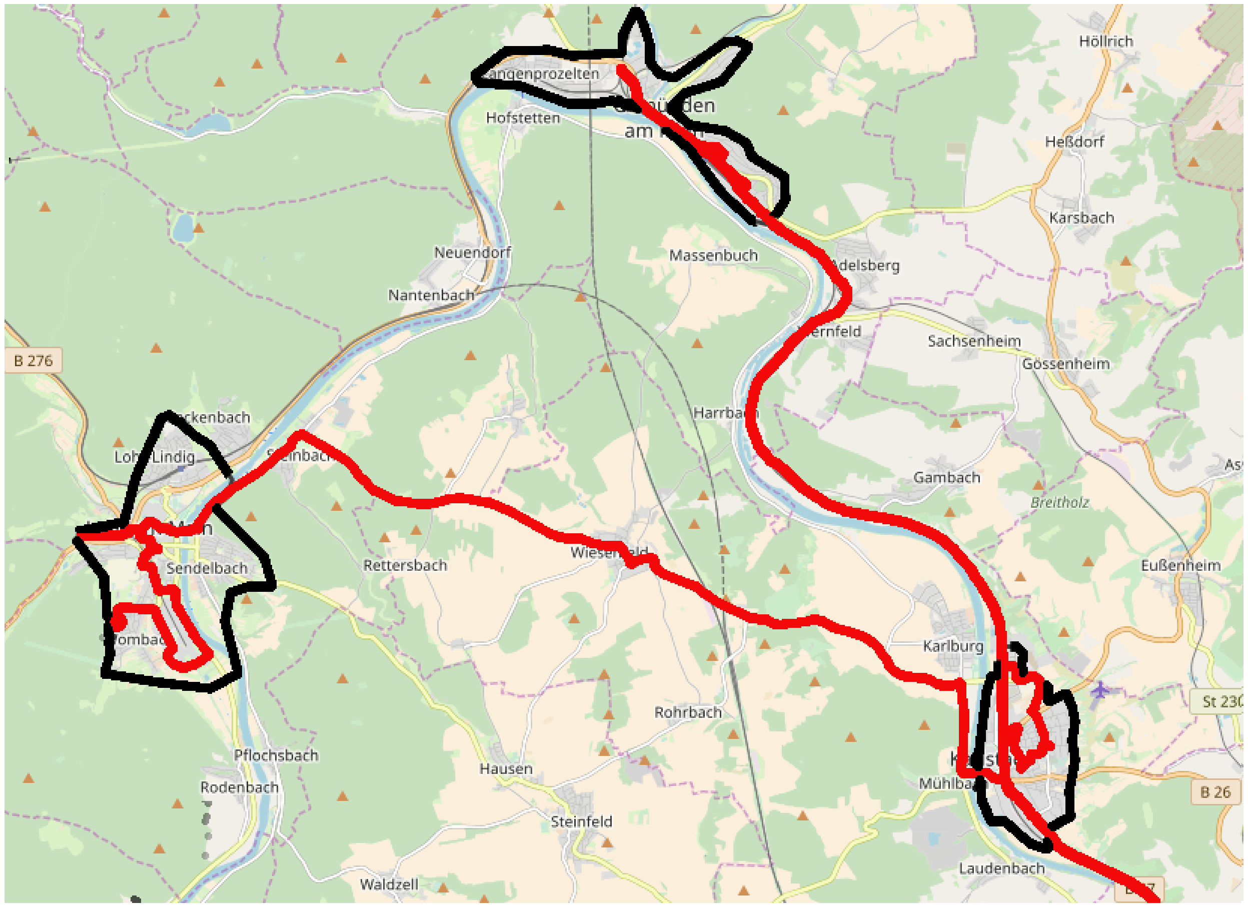



Example #2:
Regional Instance

Really hard scenario . . .

# Topology of Street Networks

Street Networks often do not meet the assumptions.

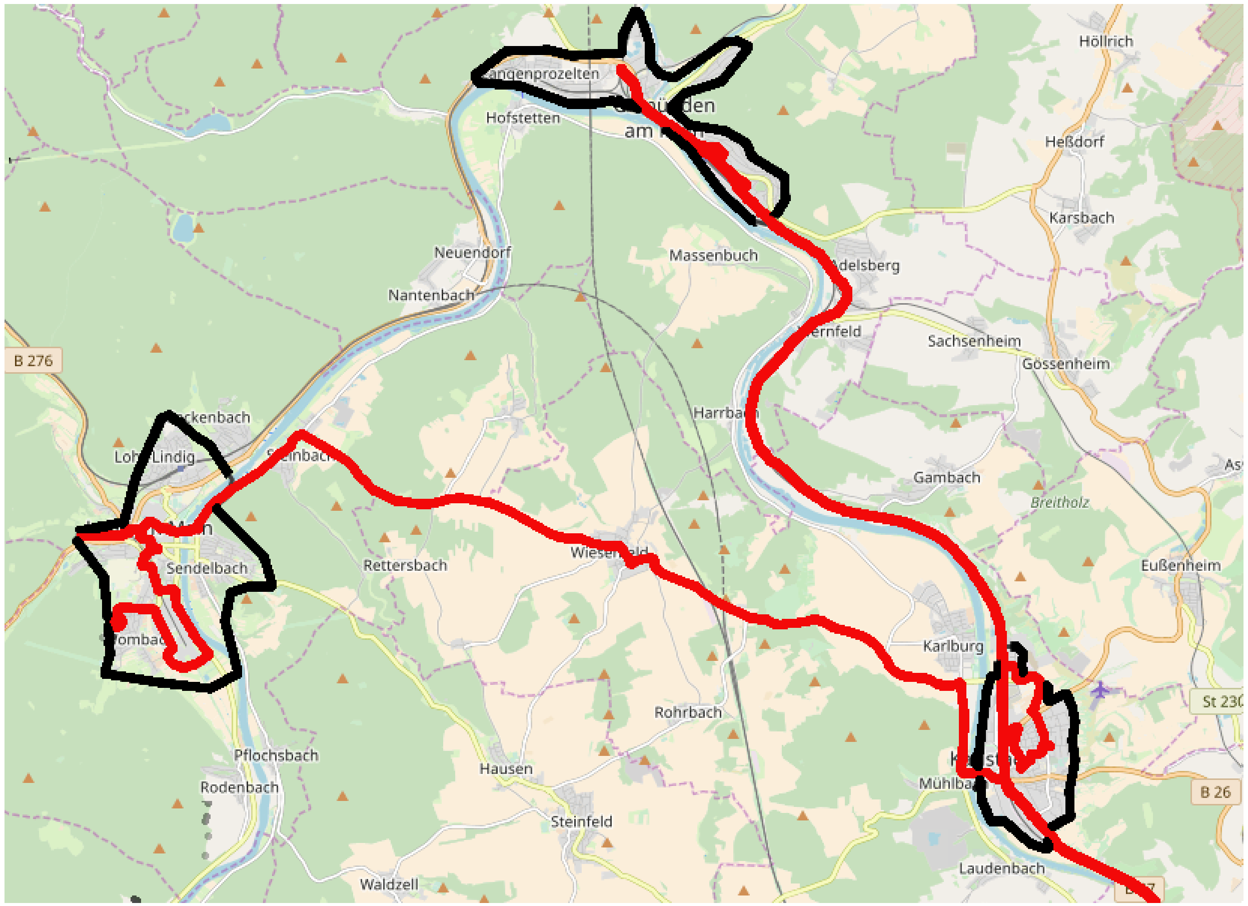



Example #2:
Regional Instance

Really hard scenario . . .

# Topology of Street Networks

Street Networks often do not meet the assumptions.

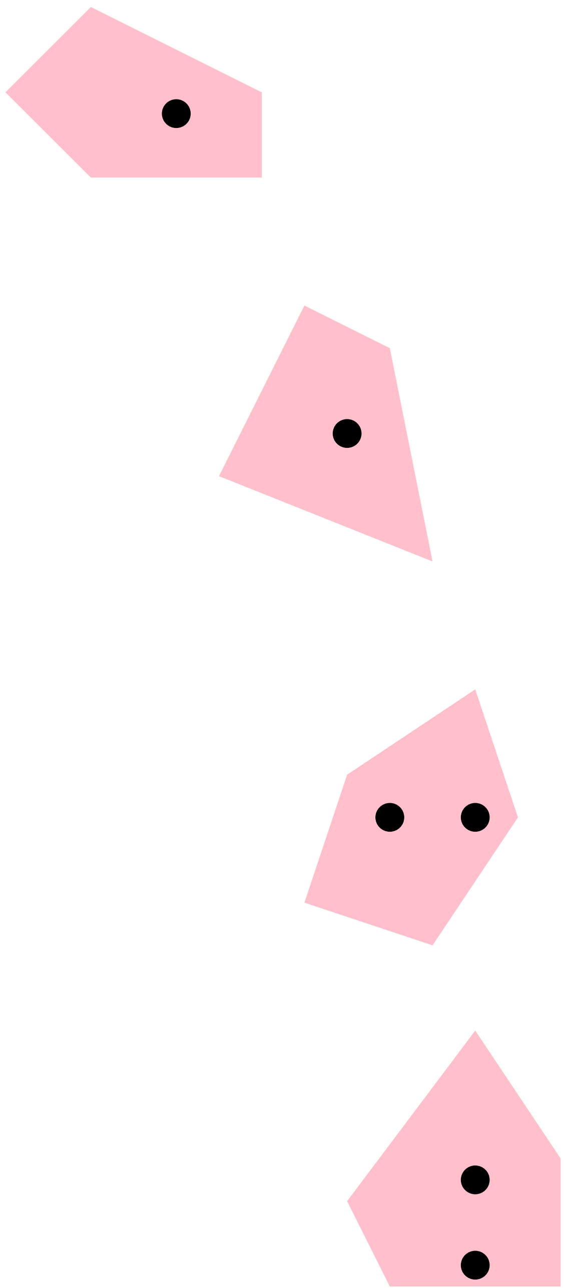



Example #2:
Regional Instance

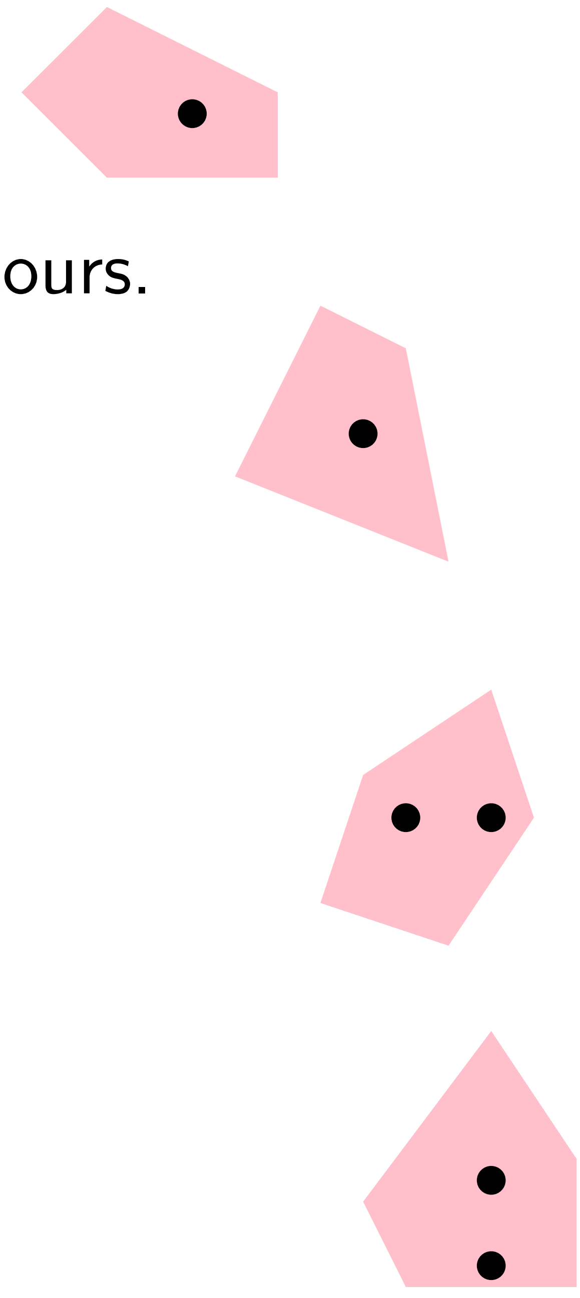
Really hard scenario . . .

False positives are to be expected in this case.

# Conclusion



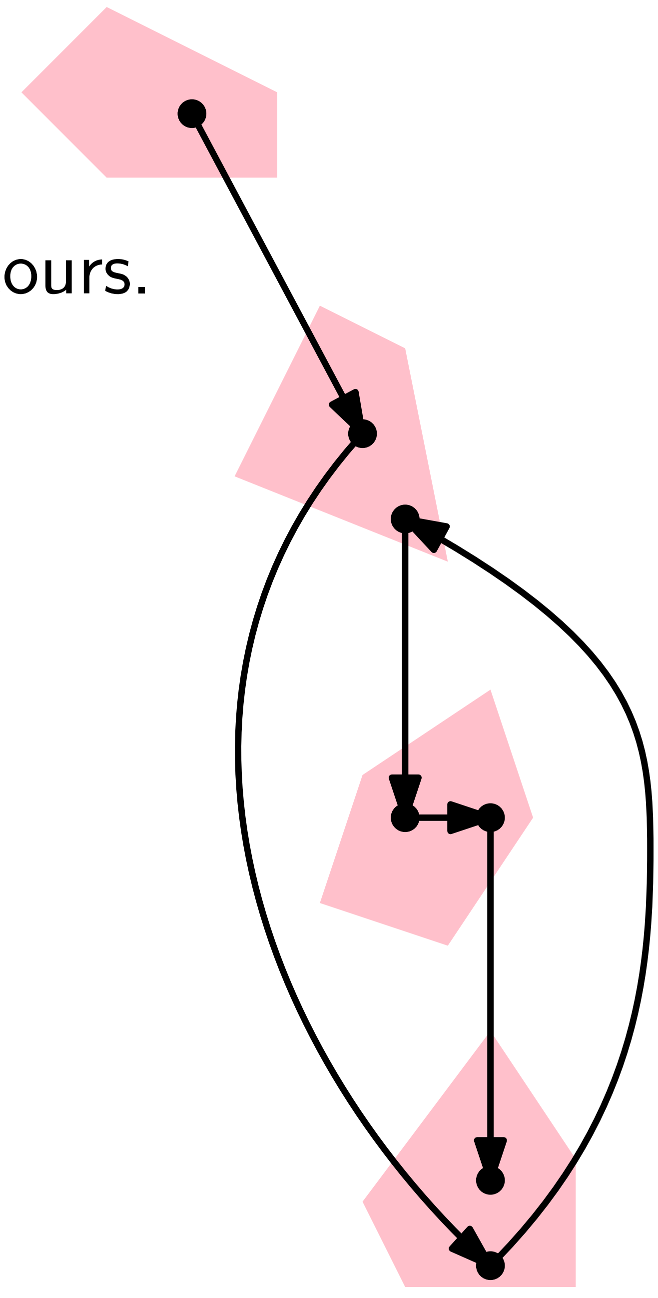
# Conclusion



The Exact Algorithm considers unsensible tours.

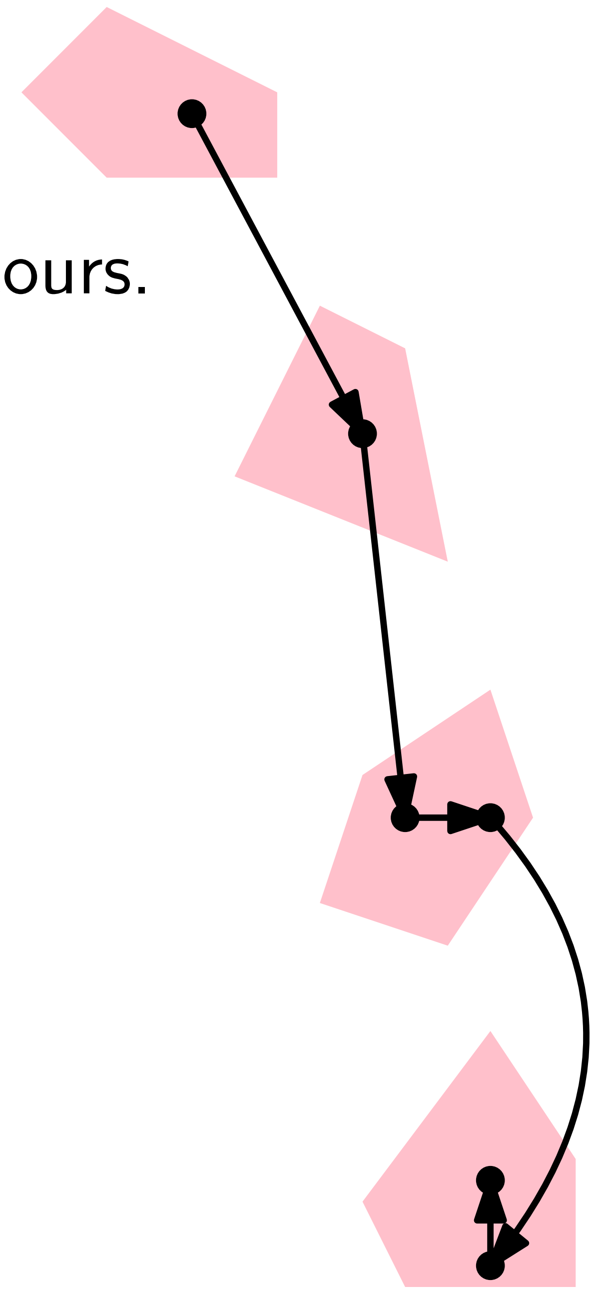
# Conclusion

The Exact Algorithm considers unsensible tours.



# Conclusion

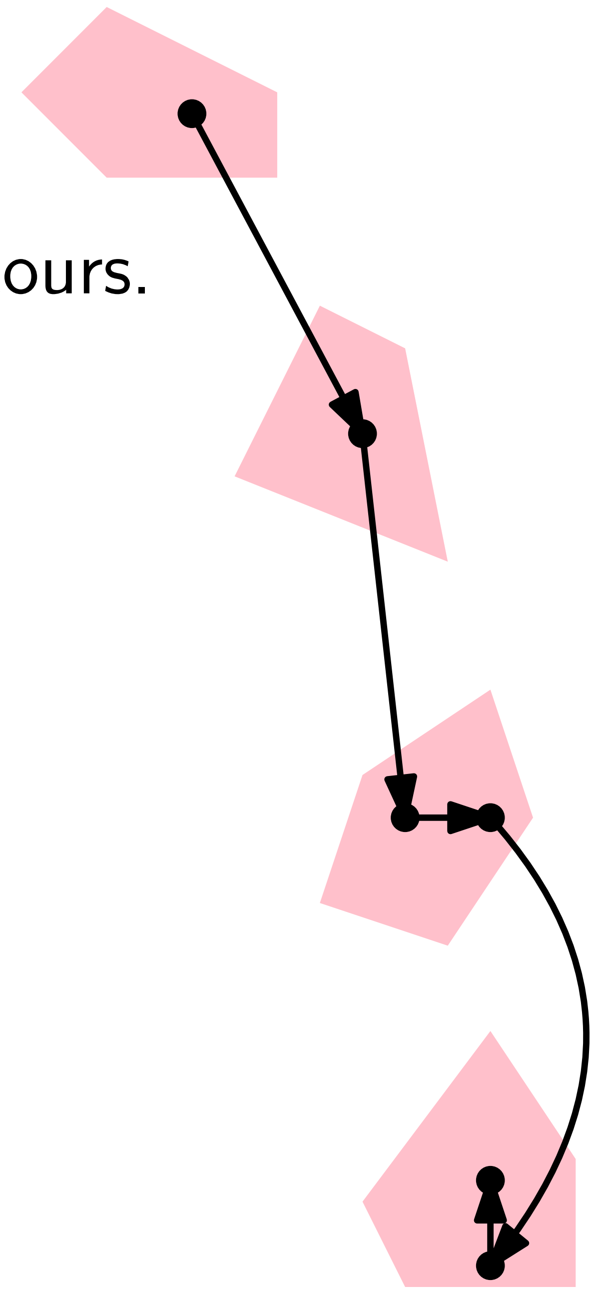
The Exact Algorithm considers unsensible tours.



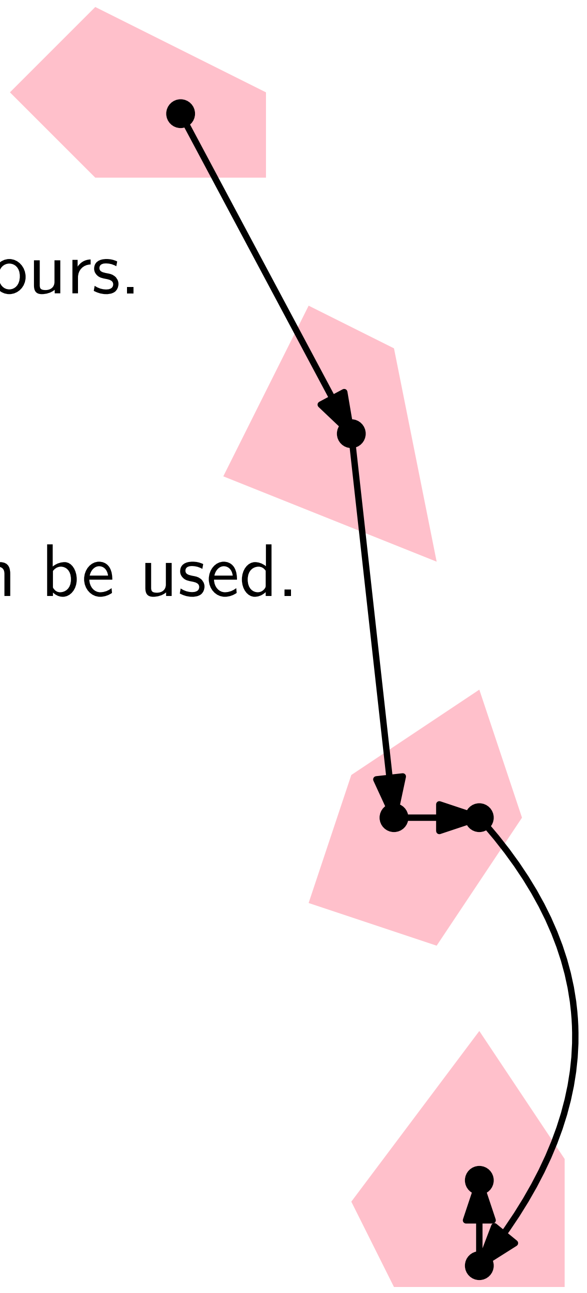
# Conclusion

The Exact Algorithm considers unsensible tours.

Intuition yields the $\overrightarrow{T^*}$-algorithm.



# Conclusion

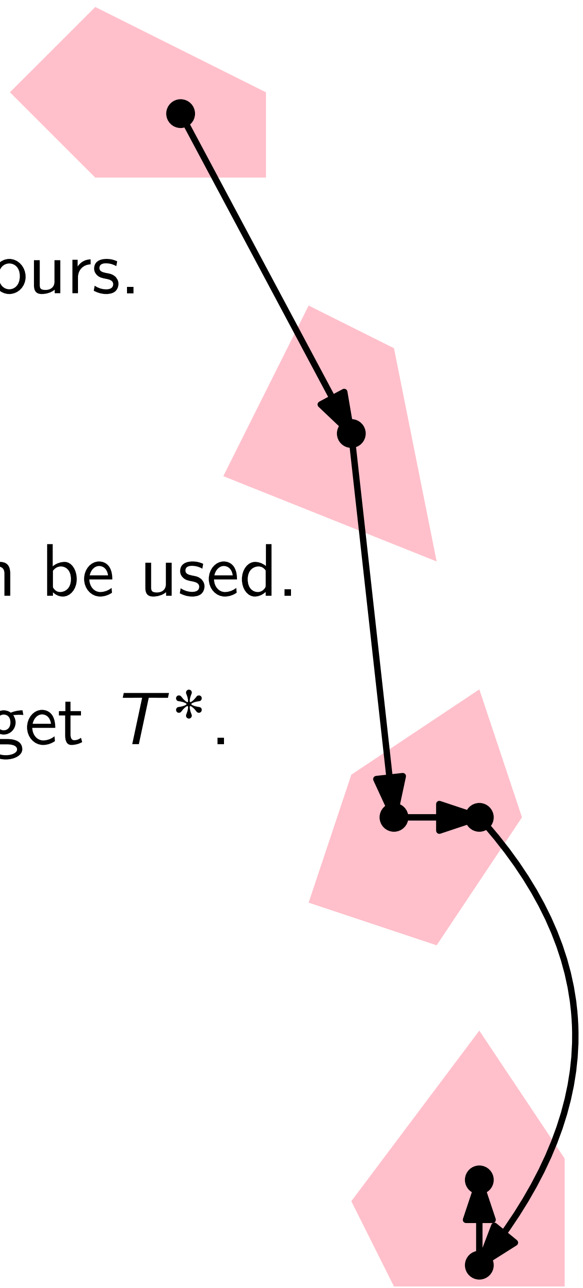


The Exact Algorithm considers unsensible tours.

Intuition yields the $\overrightarrow{T*}$-algorithm.

A **classifier** decides if the $\overrightarrow{T*}$-algorithm can be used.

# Conclusion



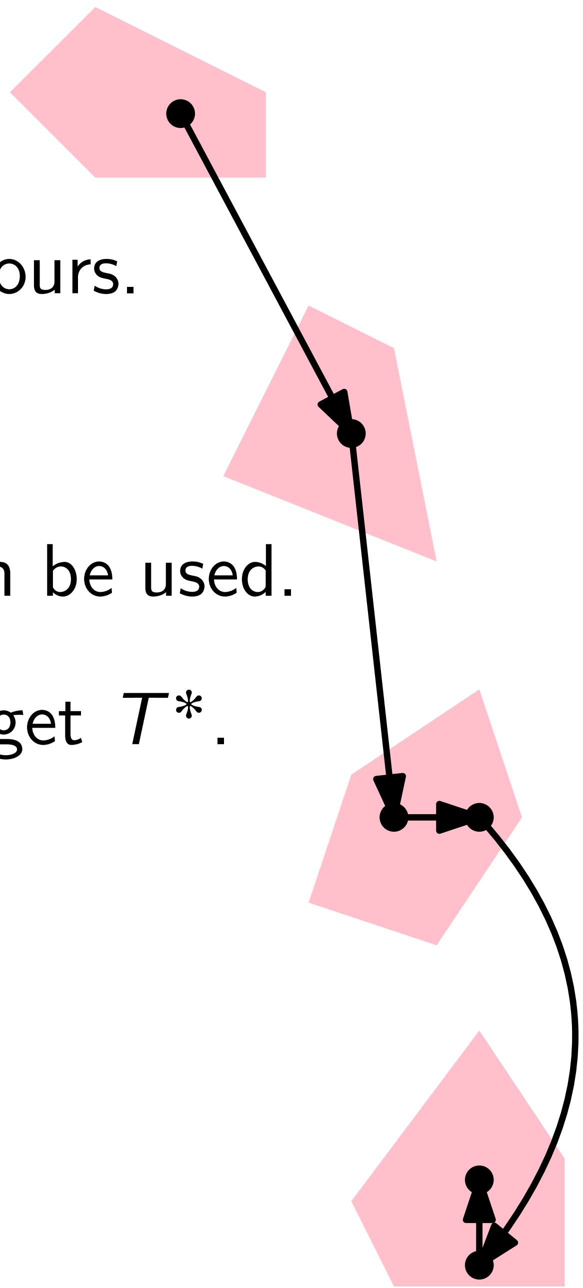
The Exact Algorithm considers unsensible tours.

Intuition yields the $\overrightarrow{T*}$-algorithm.

A **classifier** decides if the $\overrightarrow{T*}$-algorithm can be used.

If **yes**, only a fraction of time is needed to get $T*$.

# Conclusion



The Exact Algorithm considers unsensible tours.

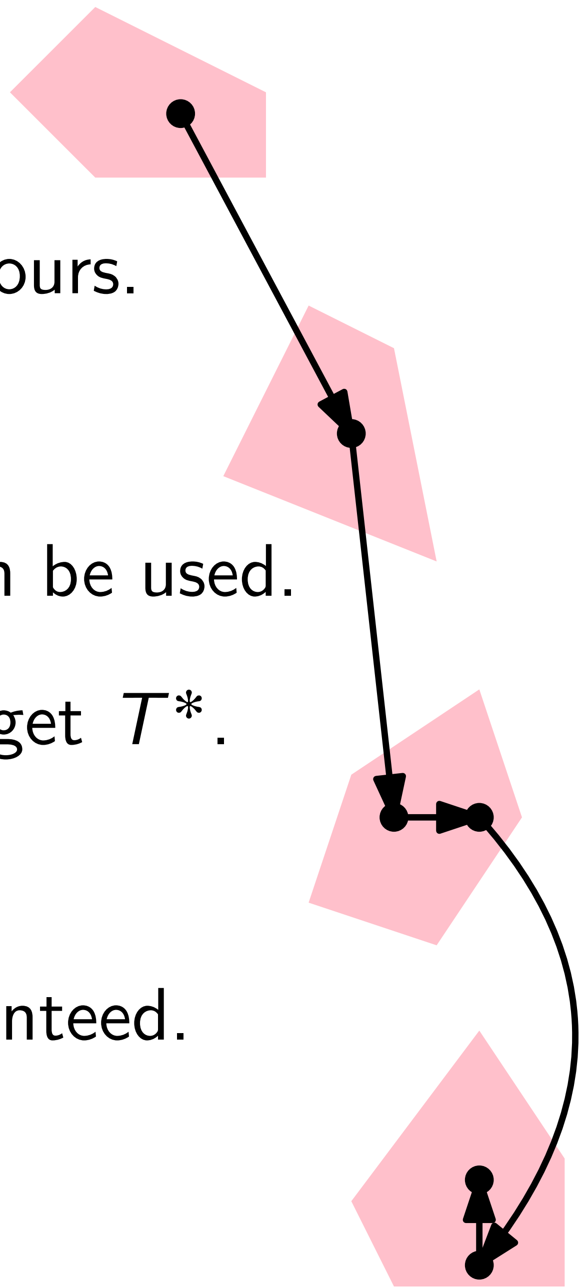
Intuition yields the $\overrightarrow{T*}$-algorithm.

A **classifier** decides if the $\overrightarrow{T*}$-algorithm can be used.

If **yes**, only a fraction of time is needed to get $T*$.

If **no**, virtually no time is wasted.

# Conclusion



The Exact Algorithm considers unsensible tours.

Intuition yields the $\overrightarrow{T*}$-algorithm.

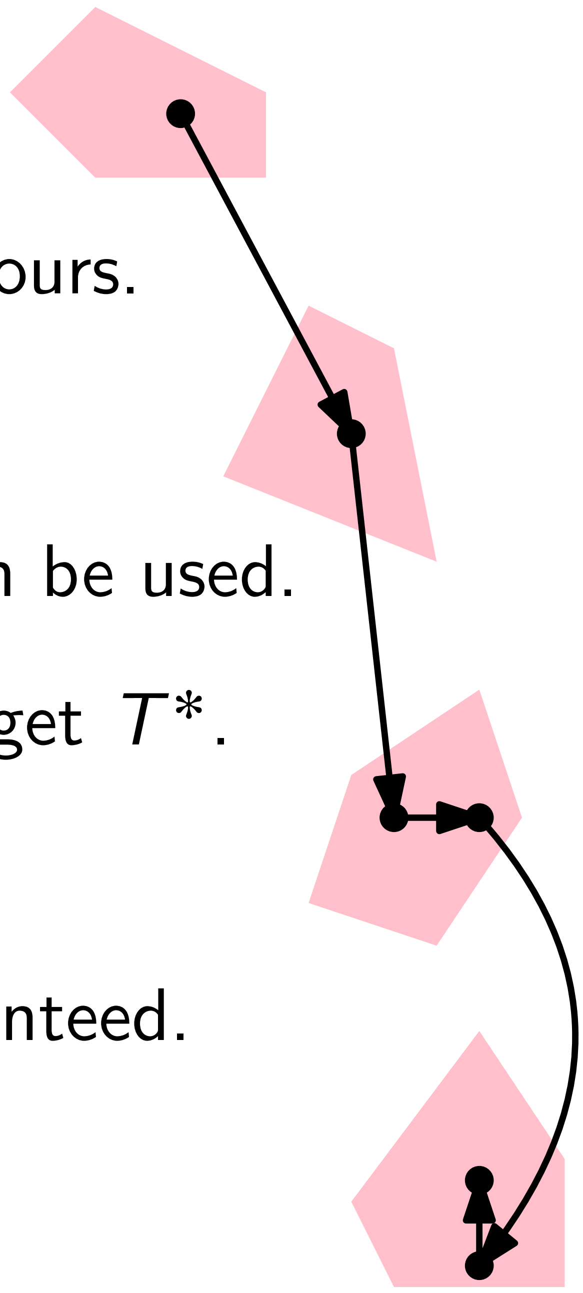
A **classifier** decides if the $\overrightarrow{T*}$-algorithm can be used.

If **yes**, only a fraction of time is needed to get $T*$.

If **no**, virtually no time is wasted.

**No false-positives:** Optimal route is guaranteed.

# Conclusion



The Exact Algorithm considers unsensible tours.

Intuition yields the $\overrightarrow{T*}$-algorithm.

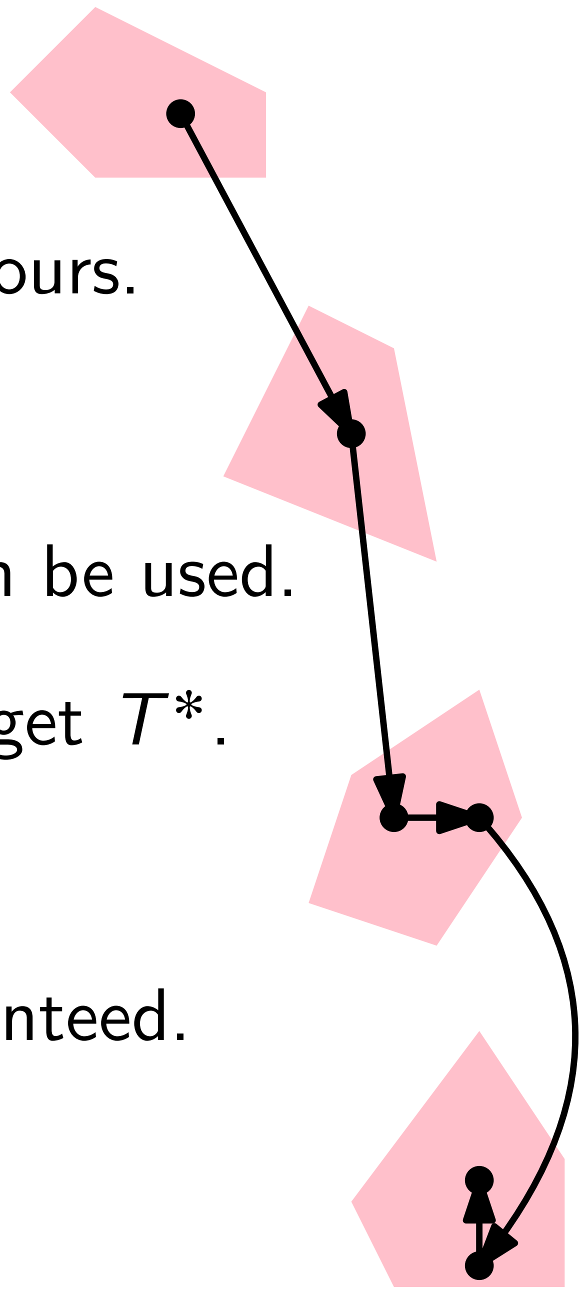
A **classifier** decides if the $\overrightarrow{T*}$-algorithm can be used.

If **yes**, only a fraction of time is needed to get $T*$.

If **no**, virtually no time is wasted.

**No false-positives:** Optimal route is guaranteed.

# Conclusion



The Exact Algorithm considers unsensible tours.

Intuition yields the $\overrightarrow{T*}$-algorithm.

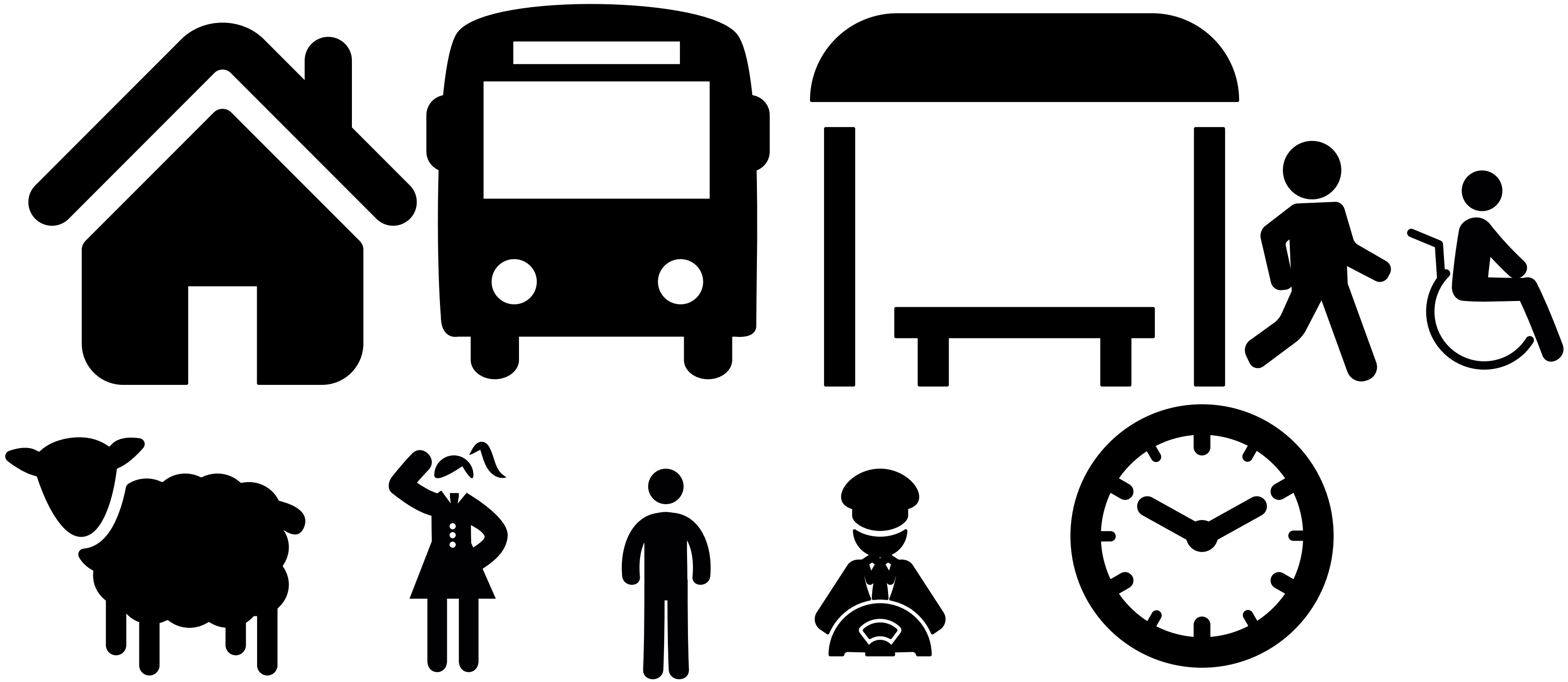
A **classifier** decides if the $\overrightarrow{T*}$-algorithm can be used.

If **yes**, only a fraction of time is needed to get $T*$.

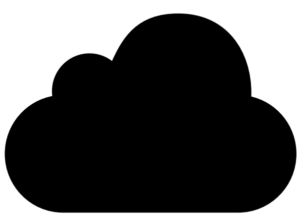
If **no**, virtually no time is wasted.

**No false-positives:** Optimal route is guaranteed.

# Attributions



The above icons are made by Freepik from flaticon.com

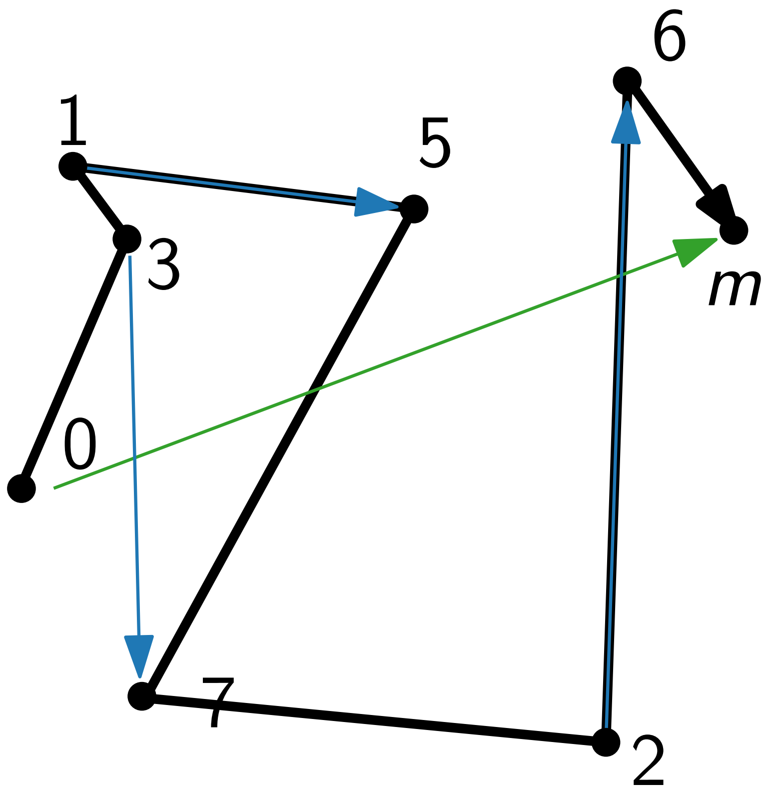


← CC 3.0 BY by SimpleIcon from flaticon.com

(c) Map Images from OpenStreetMap (osm.org)

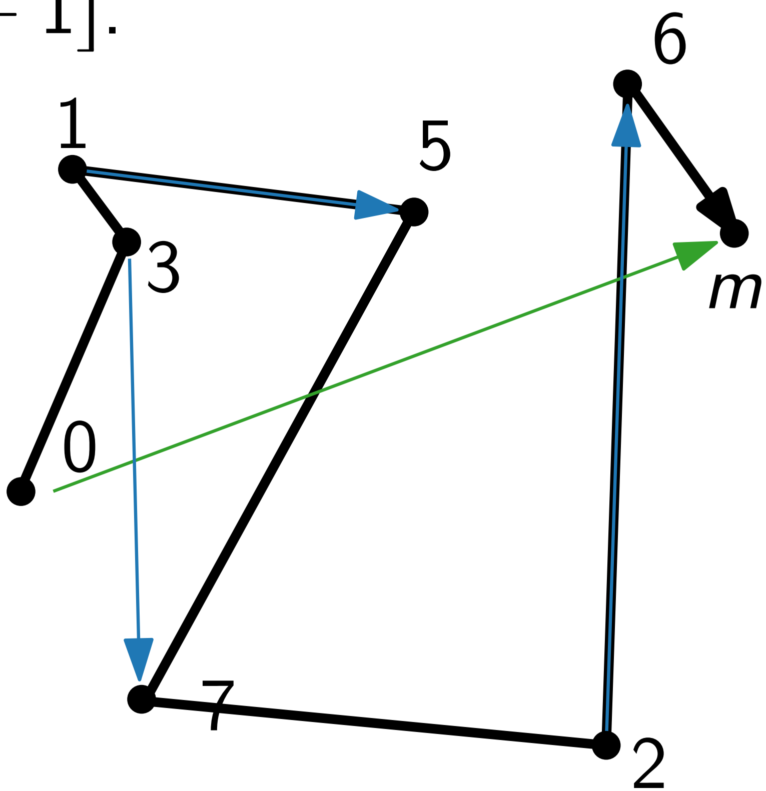
The following slides were abandoned at some point and not officially shown at the presentation. They may contain errors or are incomplete. Maybe they help you nonetheless.

# The Objective Function



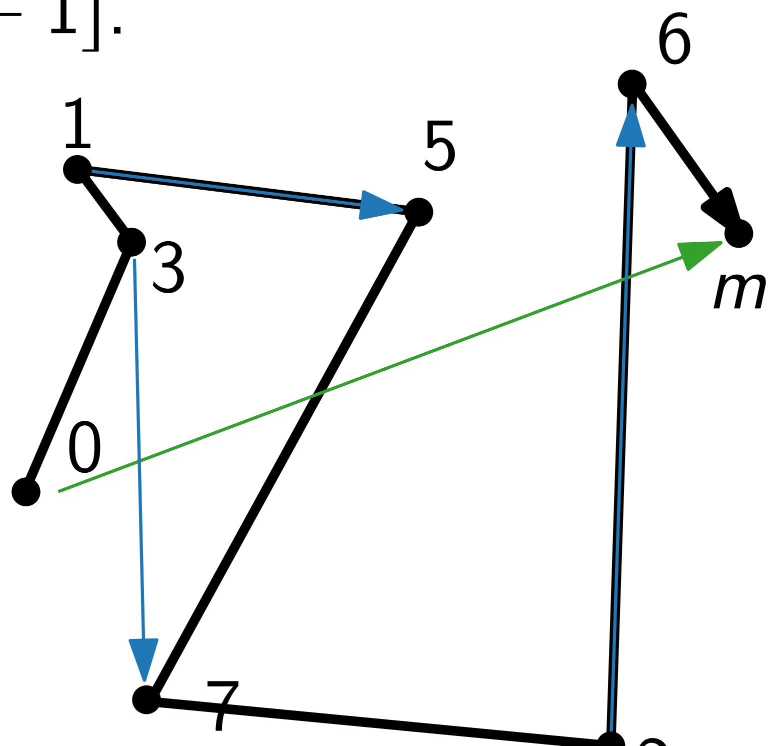
# The Objective Function

A *tour* $T$ is a permutation of $[0, 2m - 1]$.



# The Objective Function

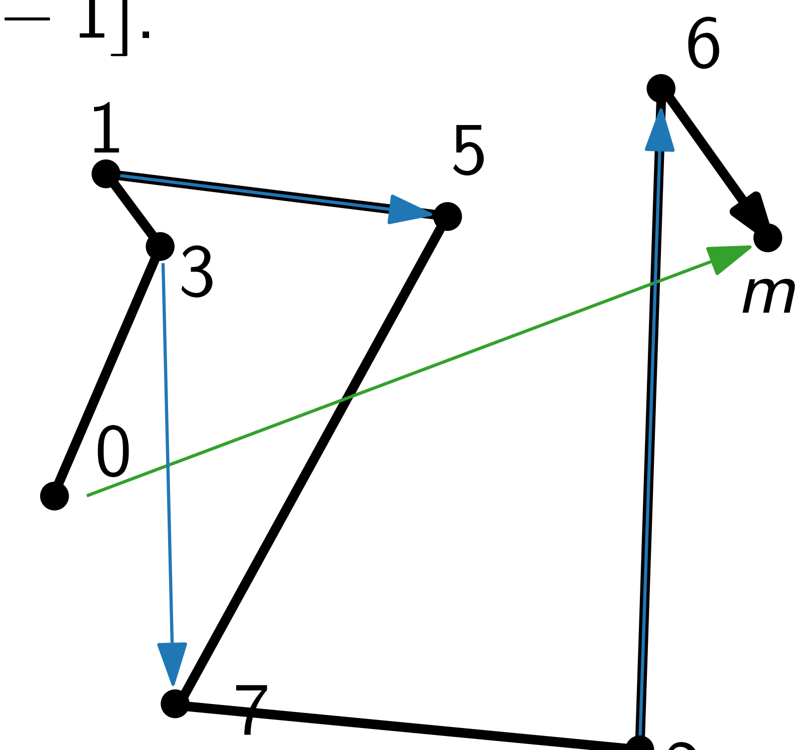
A *tour* $T$ is a permutation of $[0, 2m-1]$.




$T = [0, 3, 1, 5, 7, 2, 6, 4]$

# The Objective Function

A *tour* $T$ is a permutation of $[0, 2m-1]$.



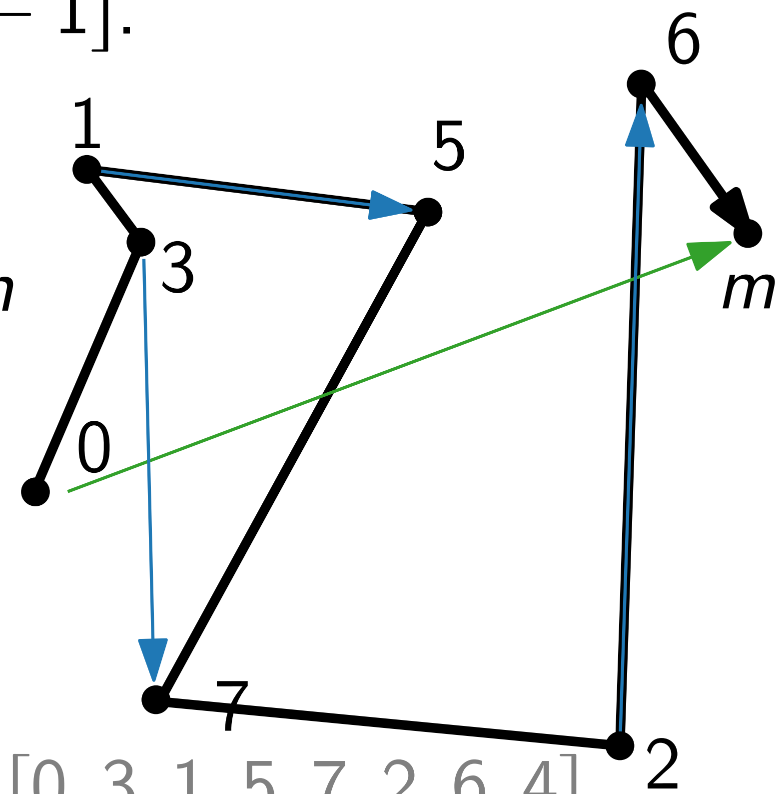

$T$ feasible $\Leftrightarrow$

$T = [0, 3, 1, 5, 7, 2, 6, 4]$

# The Objective Function

A *tour* $T$ is a permutation of $[0, 2m-1]$.

$T$ feasible $\Leftrightarrow T[1] = 0$ & $T[2m] = m$

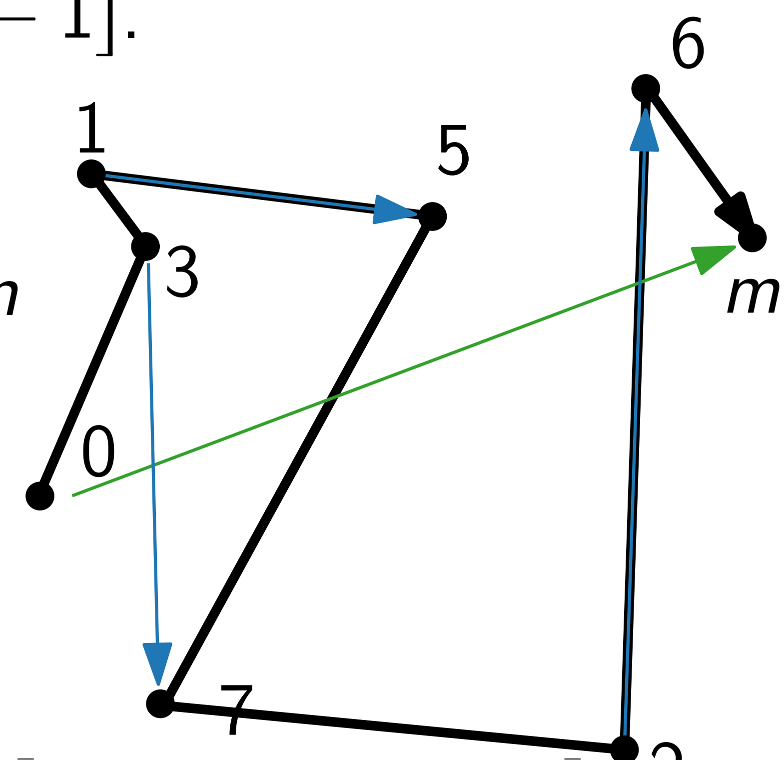



$T = [0, 3, 1, 5, 7, 2, 6, 4]$

# The Objective Function

A *tour* $T$ is a permutation of $[0, 2m-1]$.

$T$ feasible $\Leftrightarrow T[1] = 0$ & $T[2m] = m$
& precedences obeyed

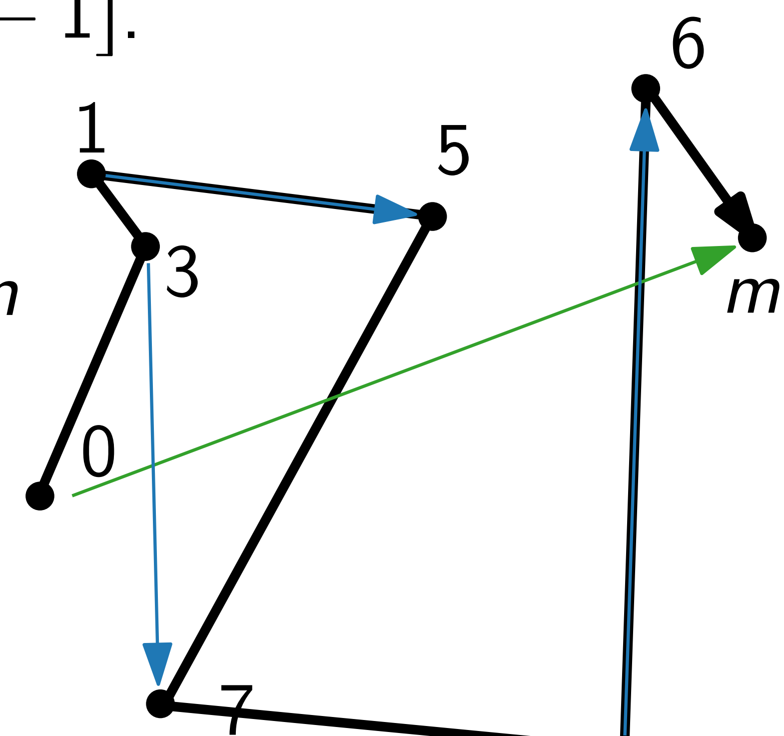



$T = [0, 3, 1, 5, 7, 2, 6, 4]$

# The Objective Function

A *tour* $T$ is a permutation of $[0, 2m-1]$.

$T$ feasible $\Leftrightarrow T[1] = 0$ & $T[2m] = m$
& precedences obeyed
& $S$ not violated

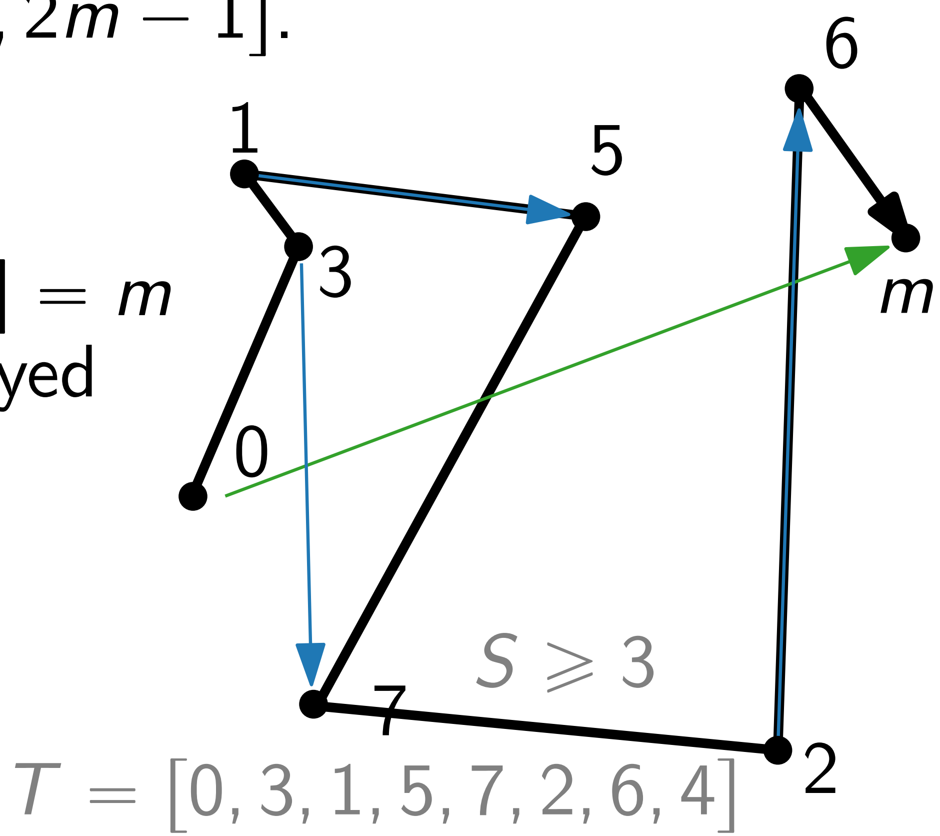



$T = [0, 3, 1, 5, 7, 2, 6, 4]$

# The Objective Function

A *tour* $T$ is a permutation of $[0, 2m-1]$.

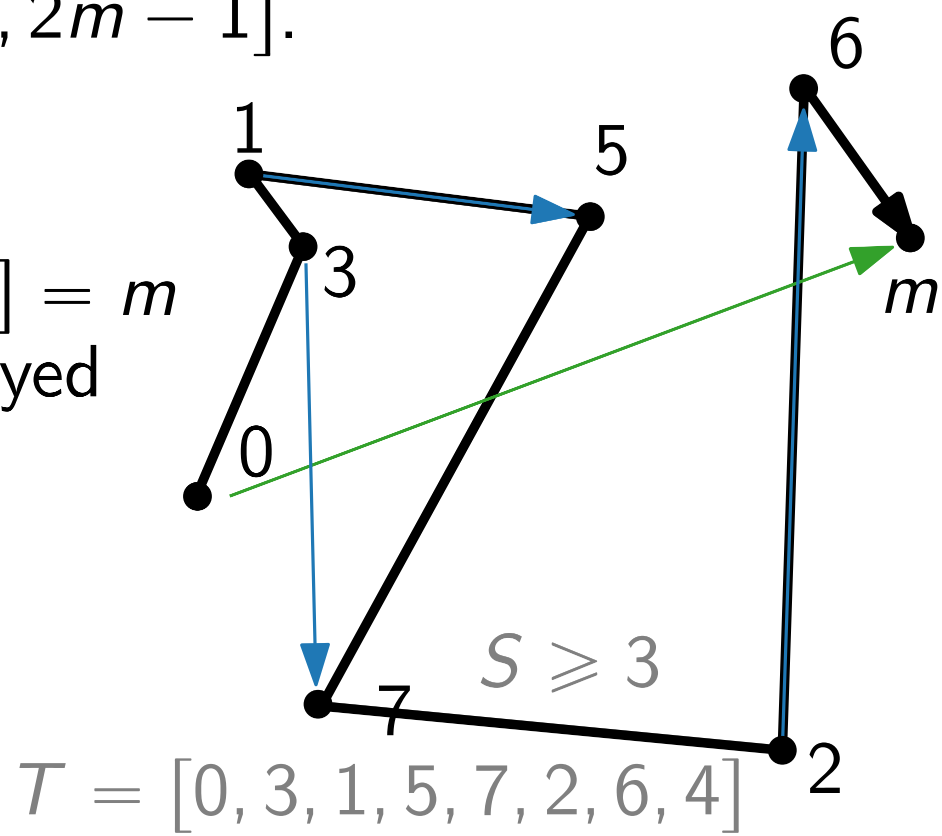
$T$ feasible $\Leftrightarrow$ $T[1] = 0$ & $T[2m] = m$
& precedences obeyed
& $S$ not violated



# The Objective Function

A *tour* $T$ is a permutation of $[0, 2m-1]$.

$T$ feasible $\Leftrightarrow T[1] = 0$ & $T[2m] = m$
& precedences obeyed
& $S$ not violated



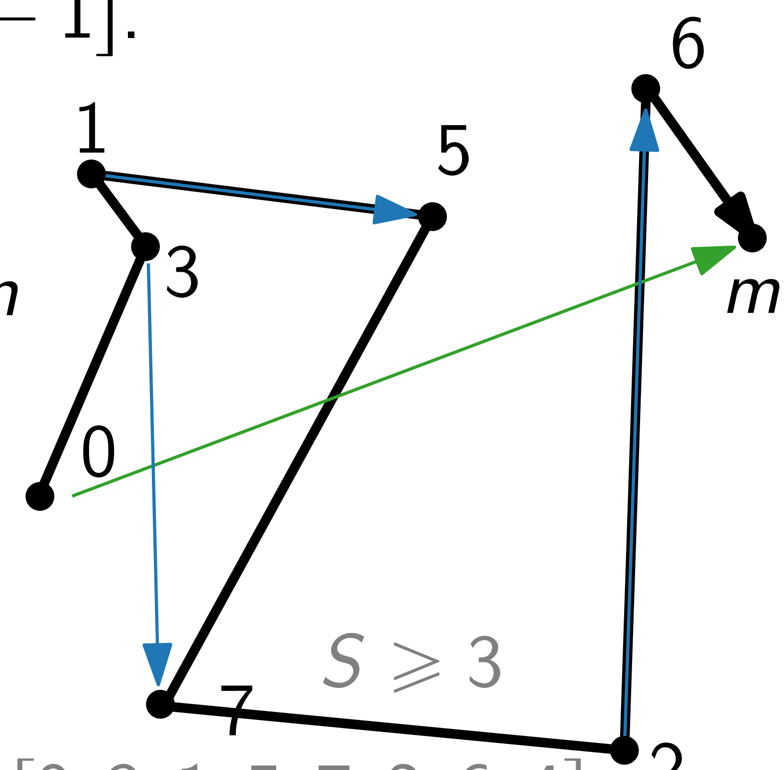

Objective:

$$\min_{T \text{ feasible}} \sum_{i=2}^{2m} k(i-1) \cdot d\Big[T[i-1], T[i]\Big]$$

# The Objective Function

A *tour* $T$ is a permutation of $[0, 2m-1]$.

$T$ feasible $\Leftrightarrow T[1] = 0$ & $T[2m] = m$
& precedences obeyed
& $S$ not violated



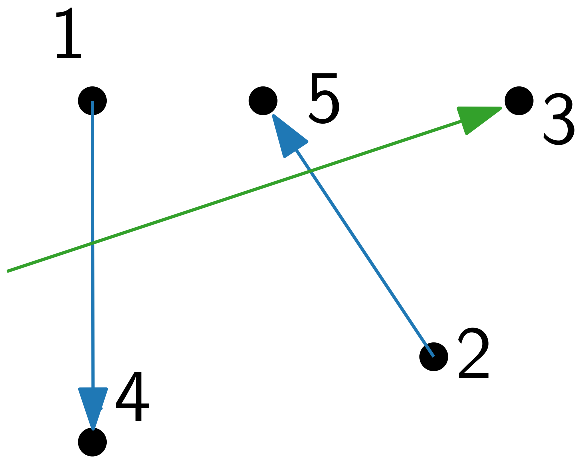

Objective:

$$\min_{T \text{ feasible}} \sum_{i=2}^{2m} k(i-1) \cdot d\Big[T[i-1], T[i]\Big]$$

$k(j)$ is the number of persons after step $j$ of $T$.

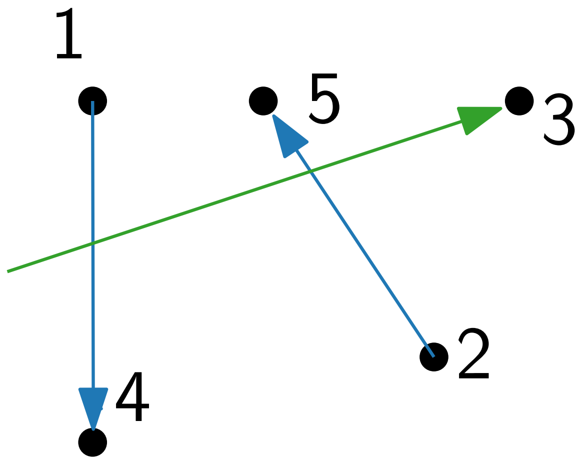
# An Exact Algorithm

# An Exact Algorithm



# An Exact Algorithm

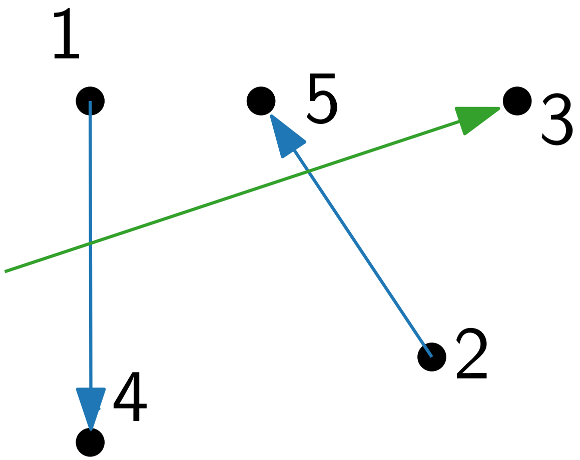
Find a tour with 6 steps:



# An Exact Algorithm

Find a tour with 6 steps:

1 2 3 4 5

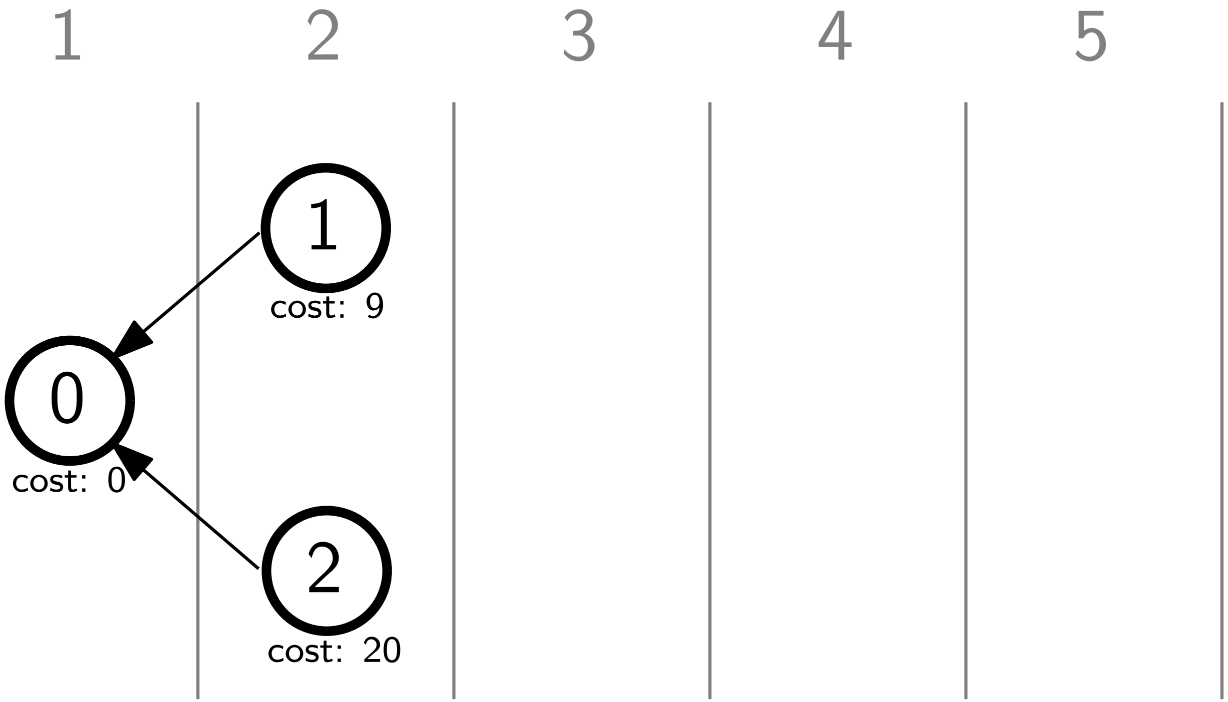


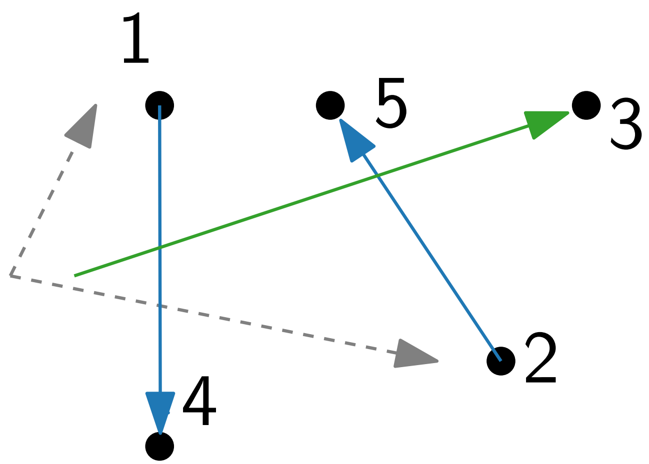
# An Exact Algorithm

Find a tour with 6 steps:

# An Exact Algorithm

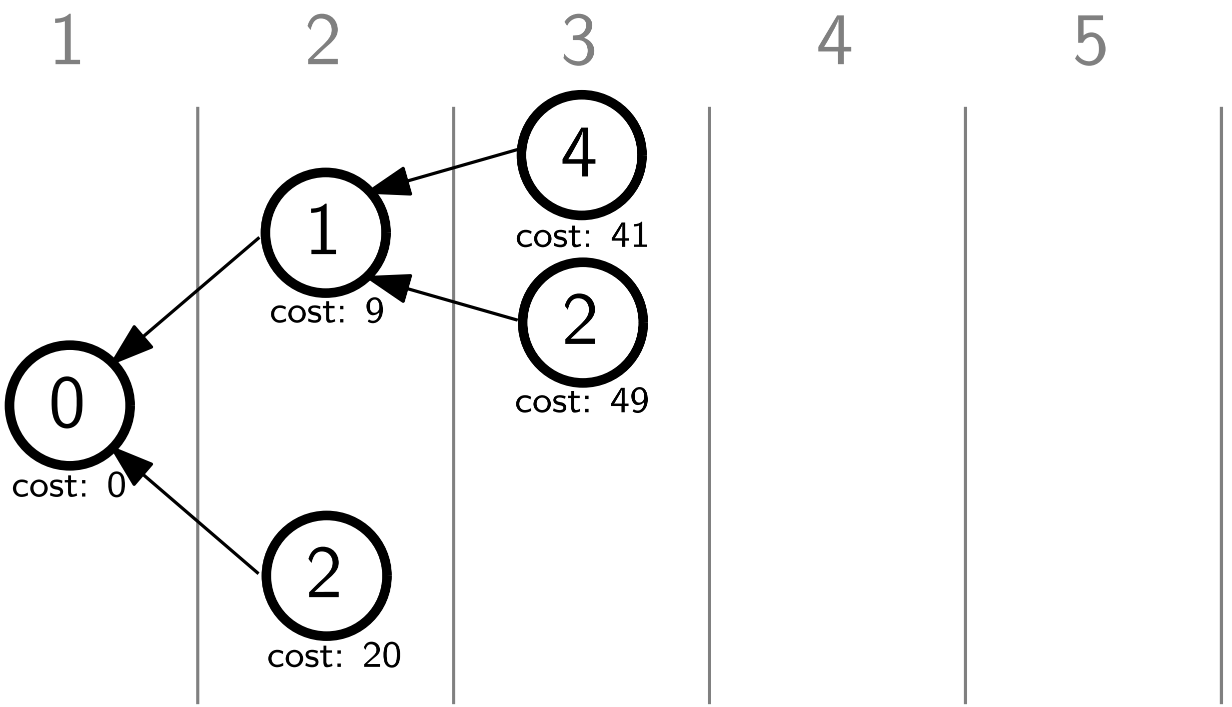
Find a tour with 6 steps:

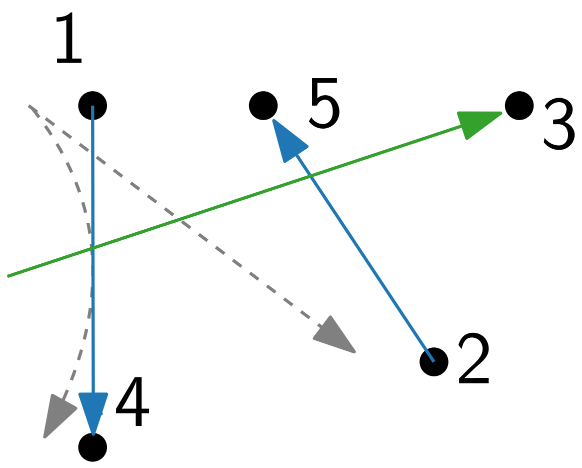


# An Exact Algorithm

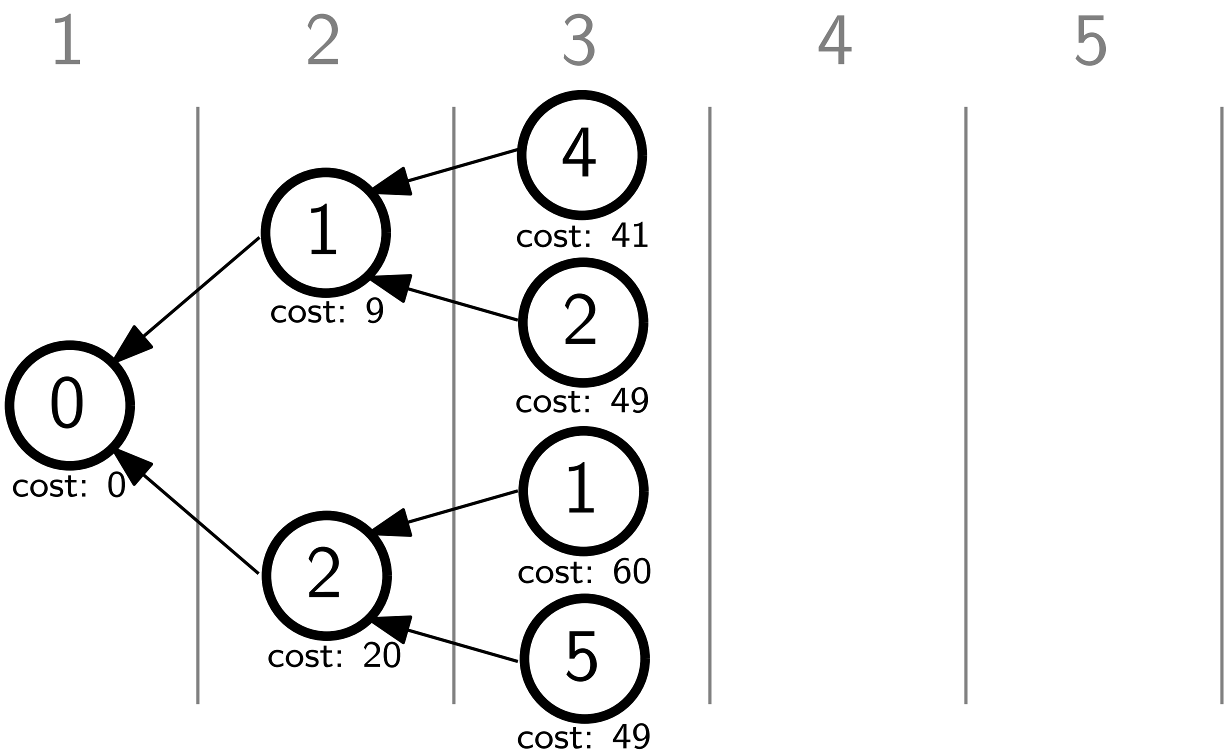
Find a tour with 6 steps:

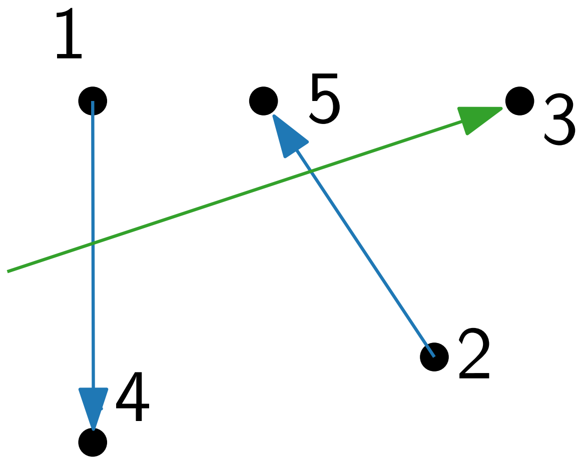


# An Exact Algorithm

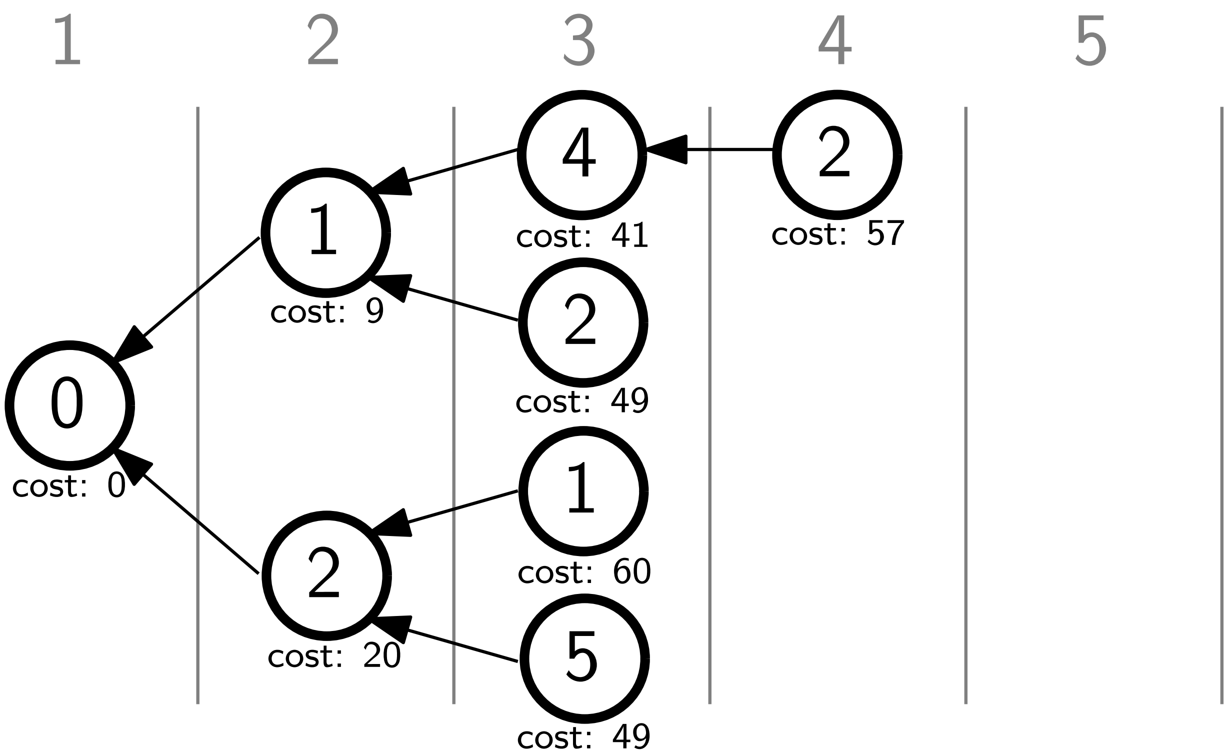
Find a tour with 6 steps:

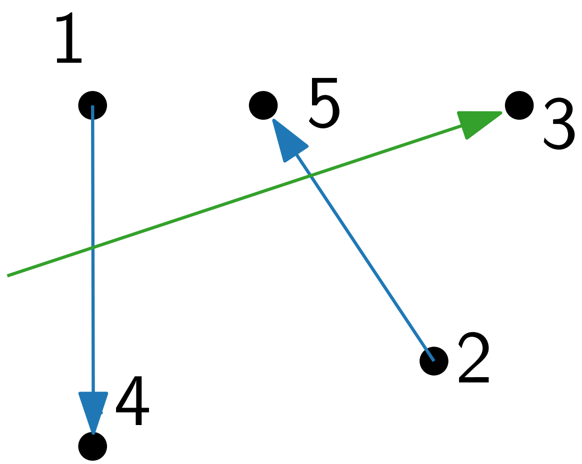


# An Exact Algorithm

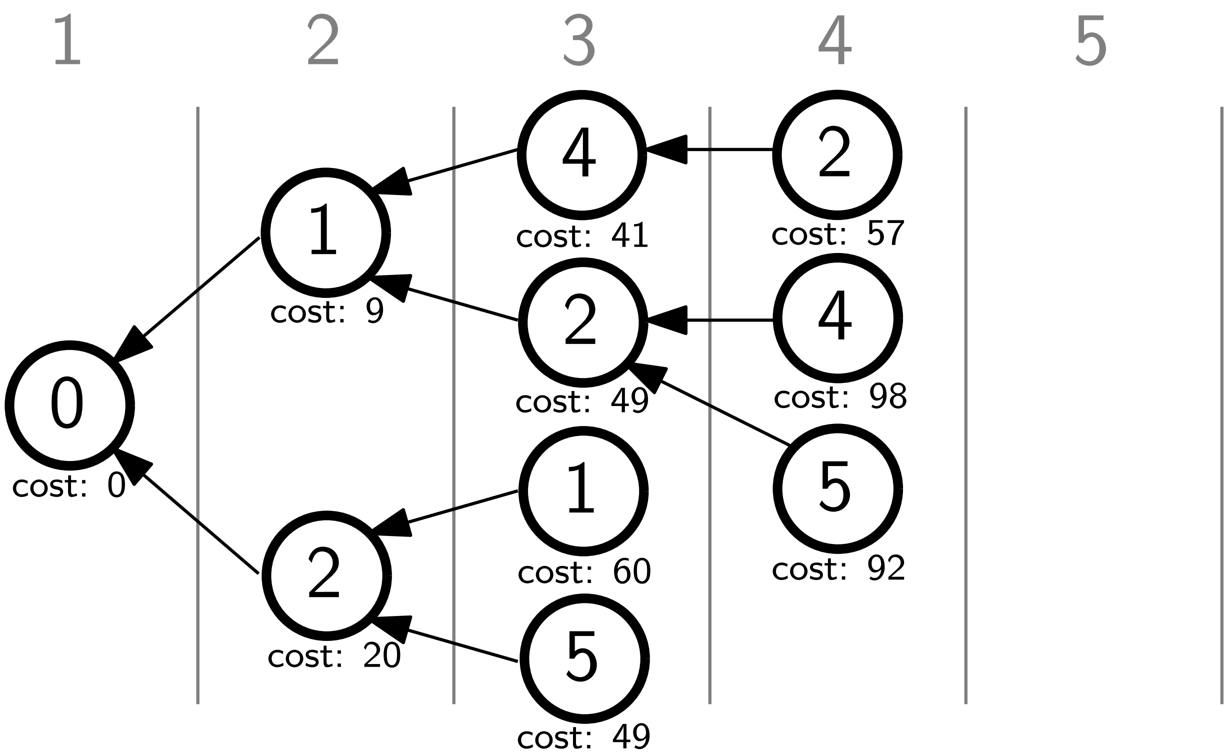
Find a tour with 6 steps:

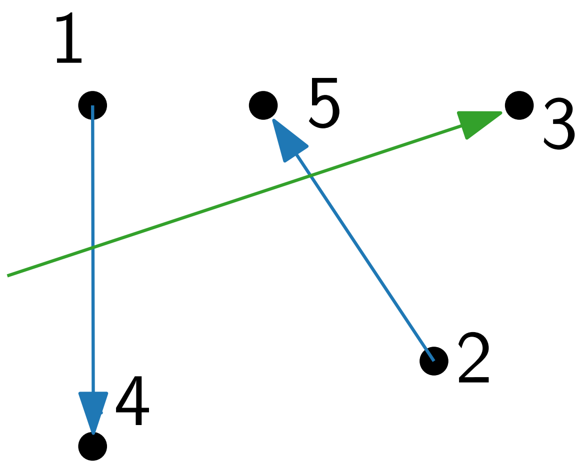


# An Exact Algorithm

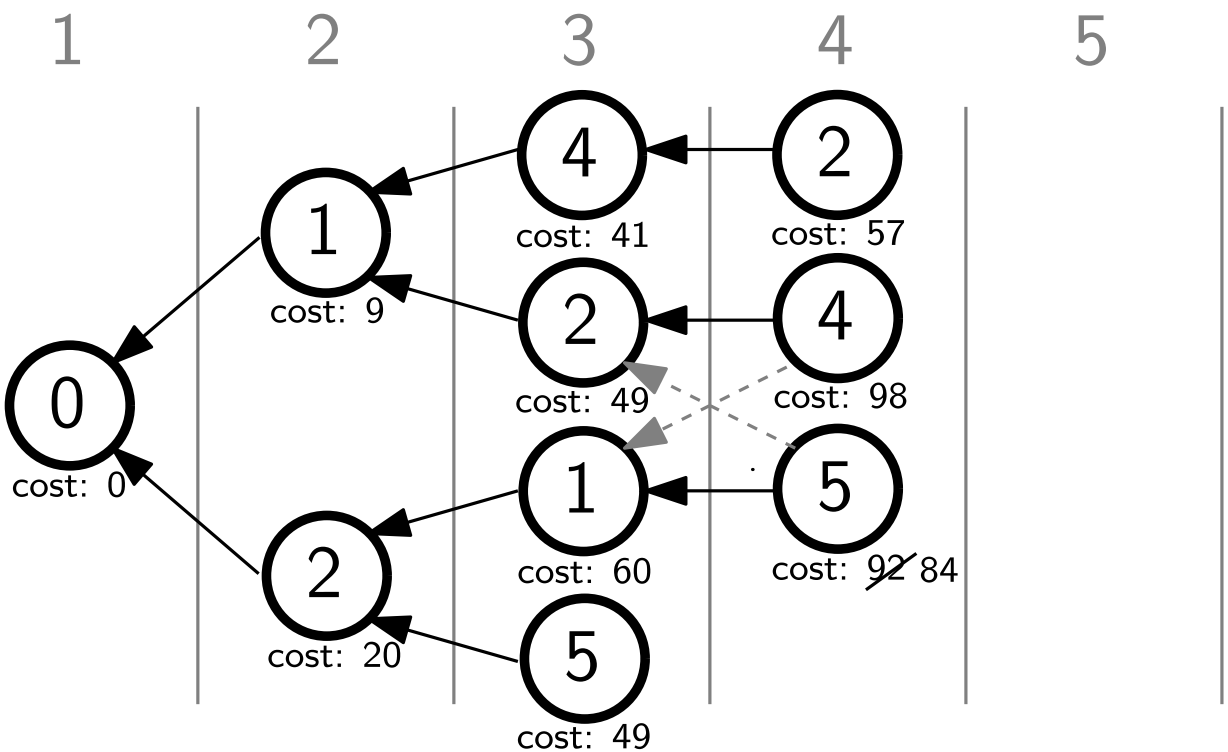
Find a tour with 6 steps:

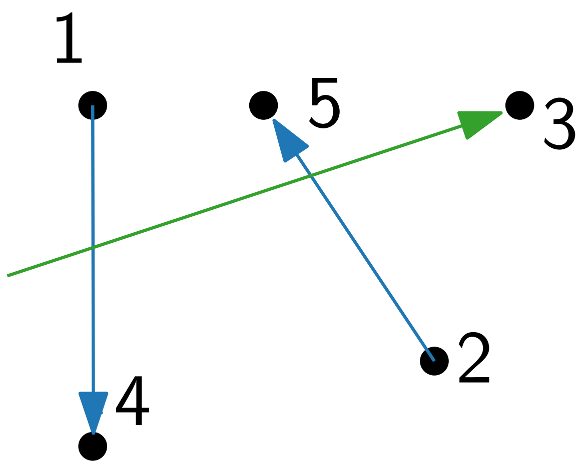


# An Exact Algorithm

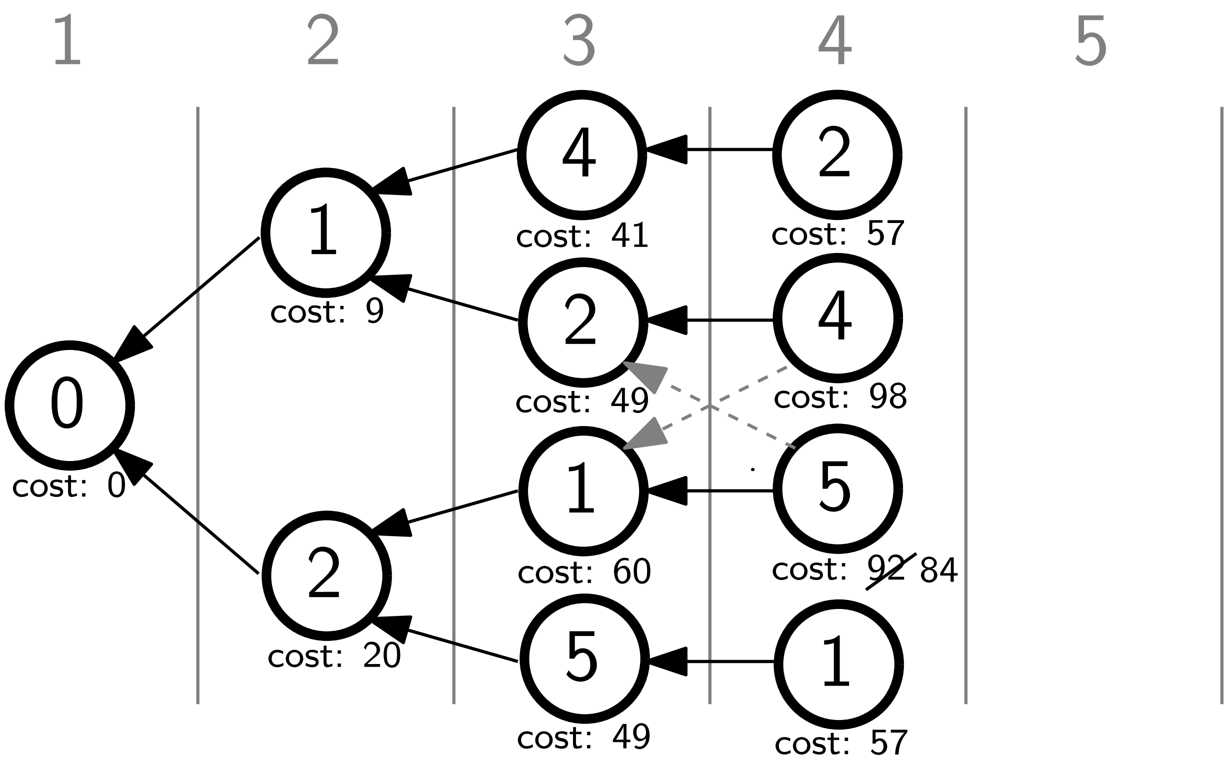
Find a tour with 6 steps:

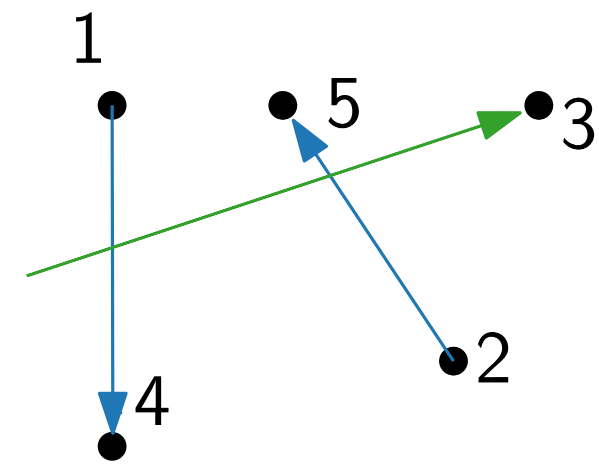


# An Exact Algorithm

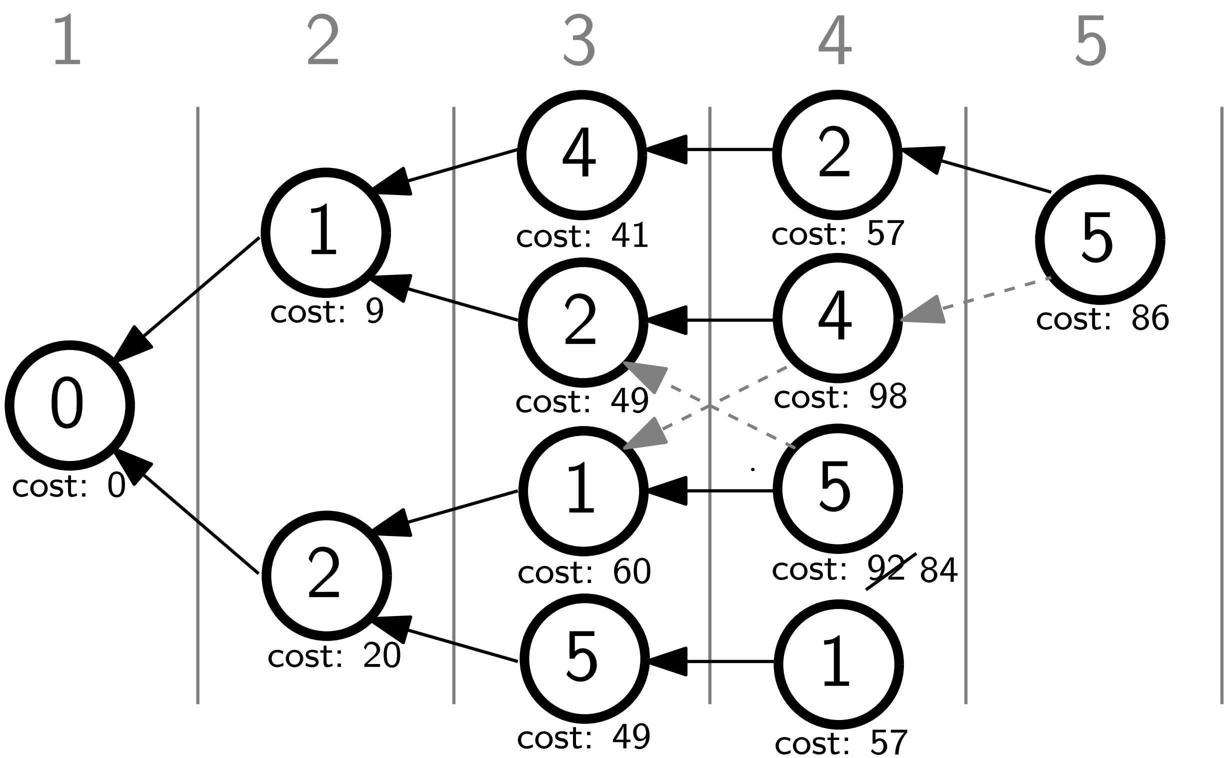
Find a tour with 6 steps:

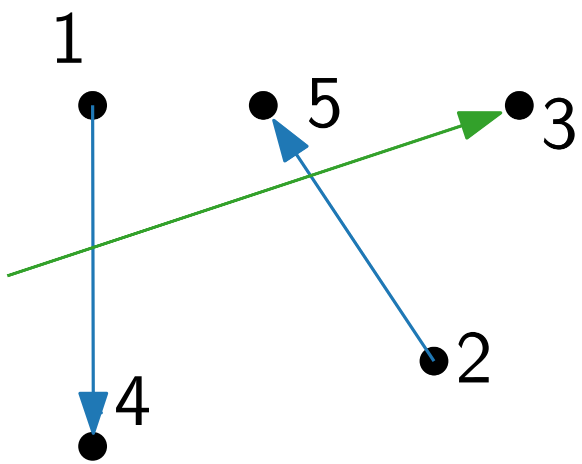


# An Exact Algorithm

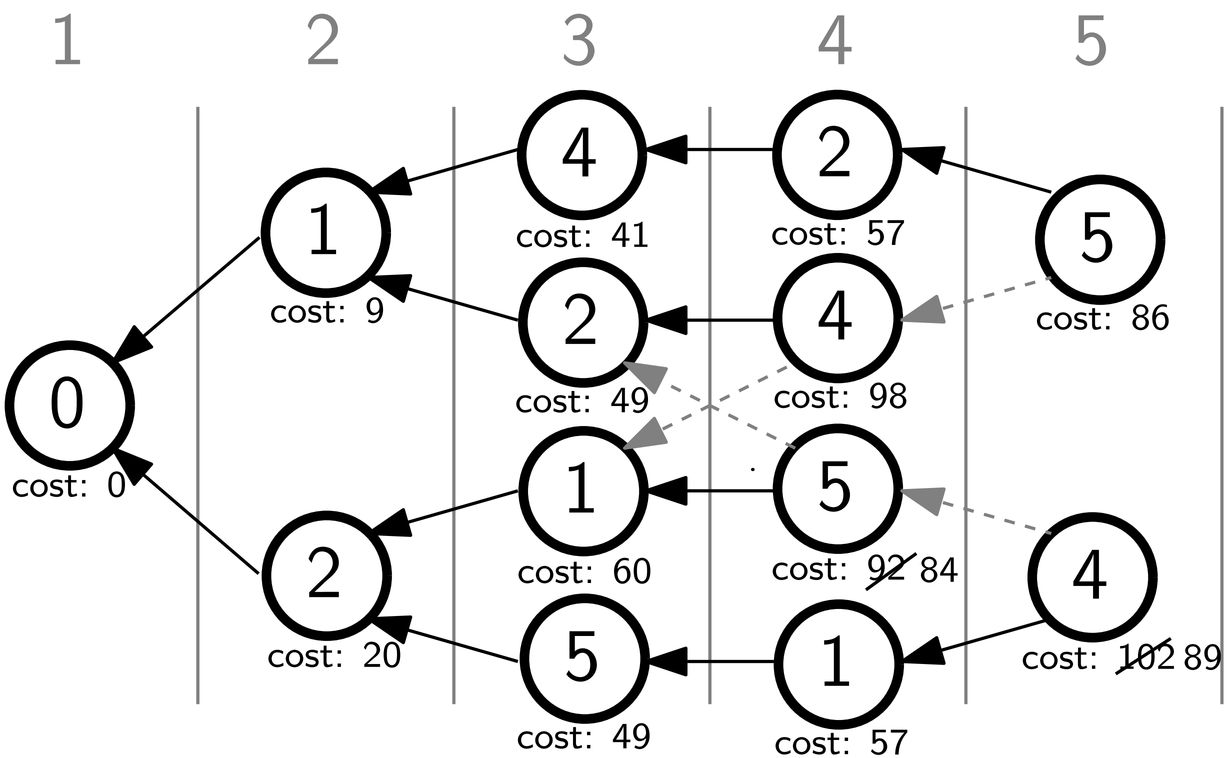
Find a tour with 6 steps:

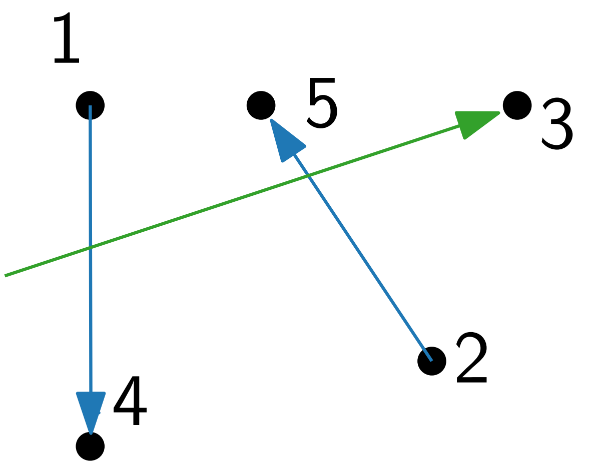


# An Exact Algorithm

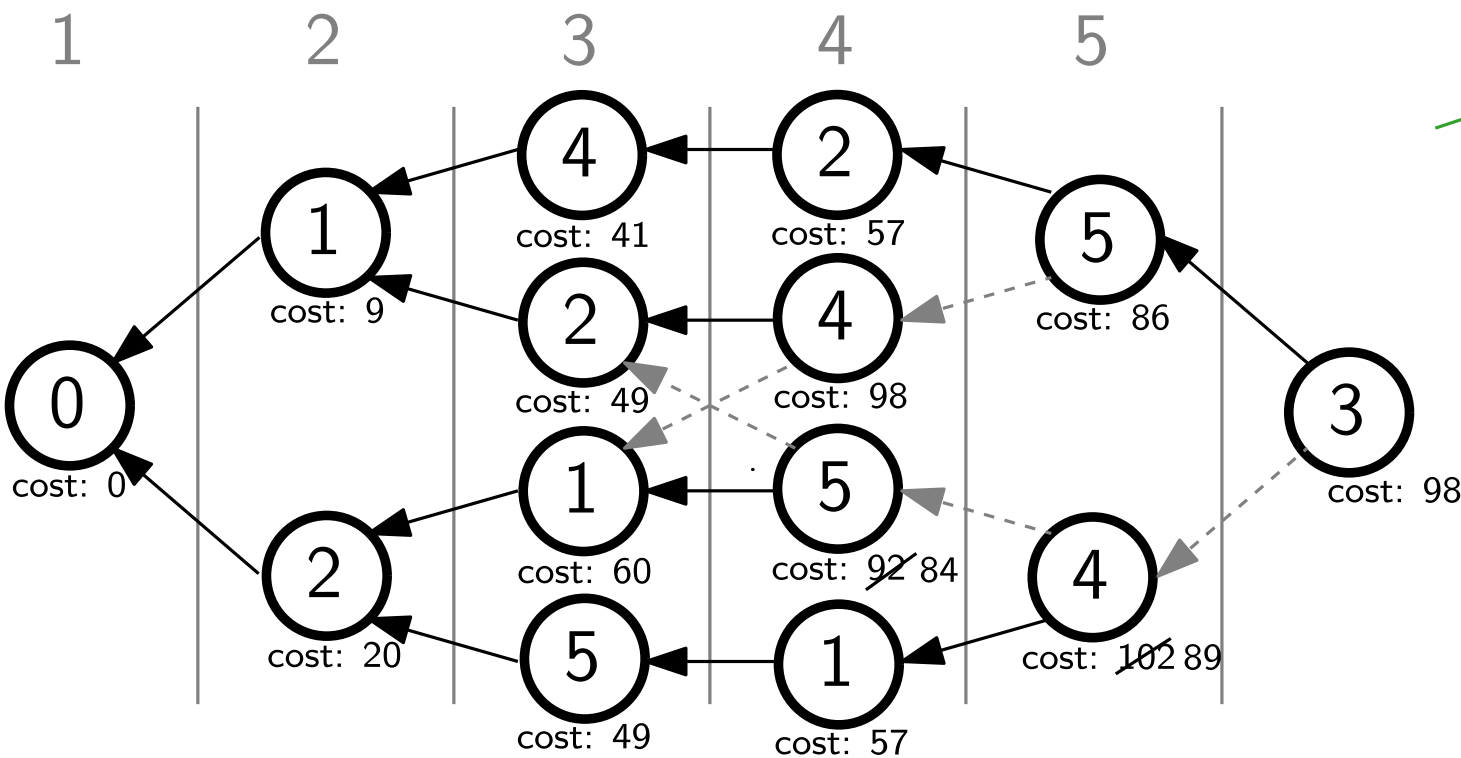
Find a tour with 6 steps:

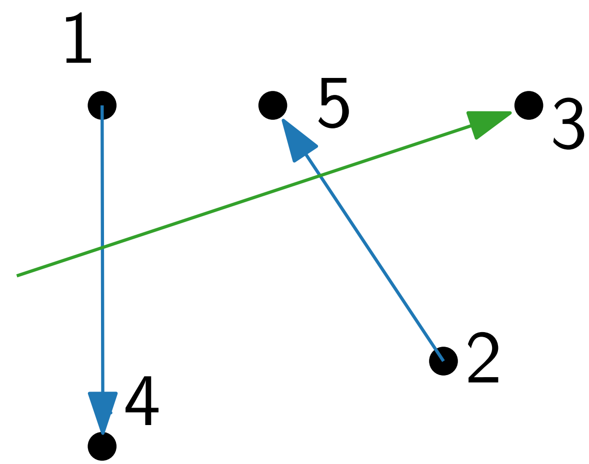


# An Exact Algorithm

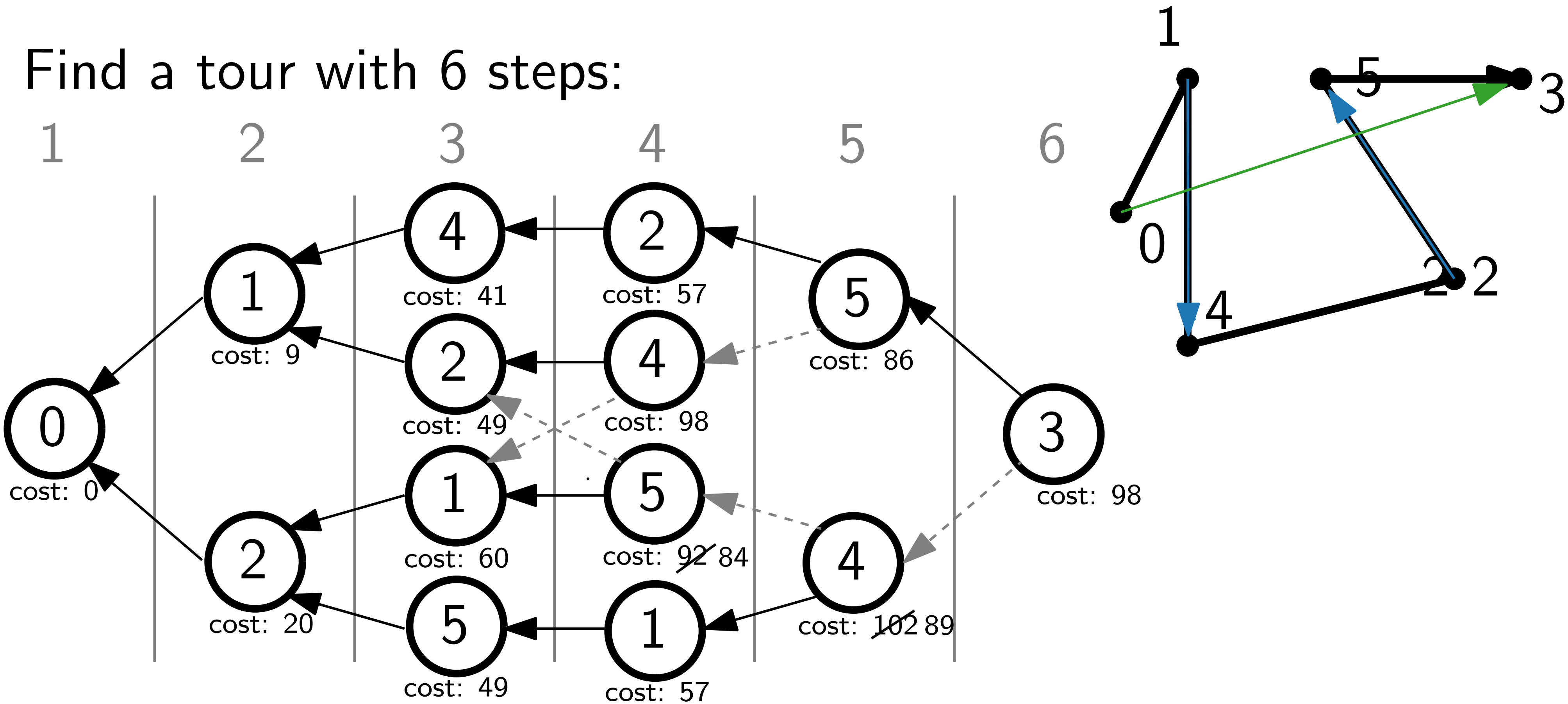
Find a tour with 6 steps:



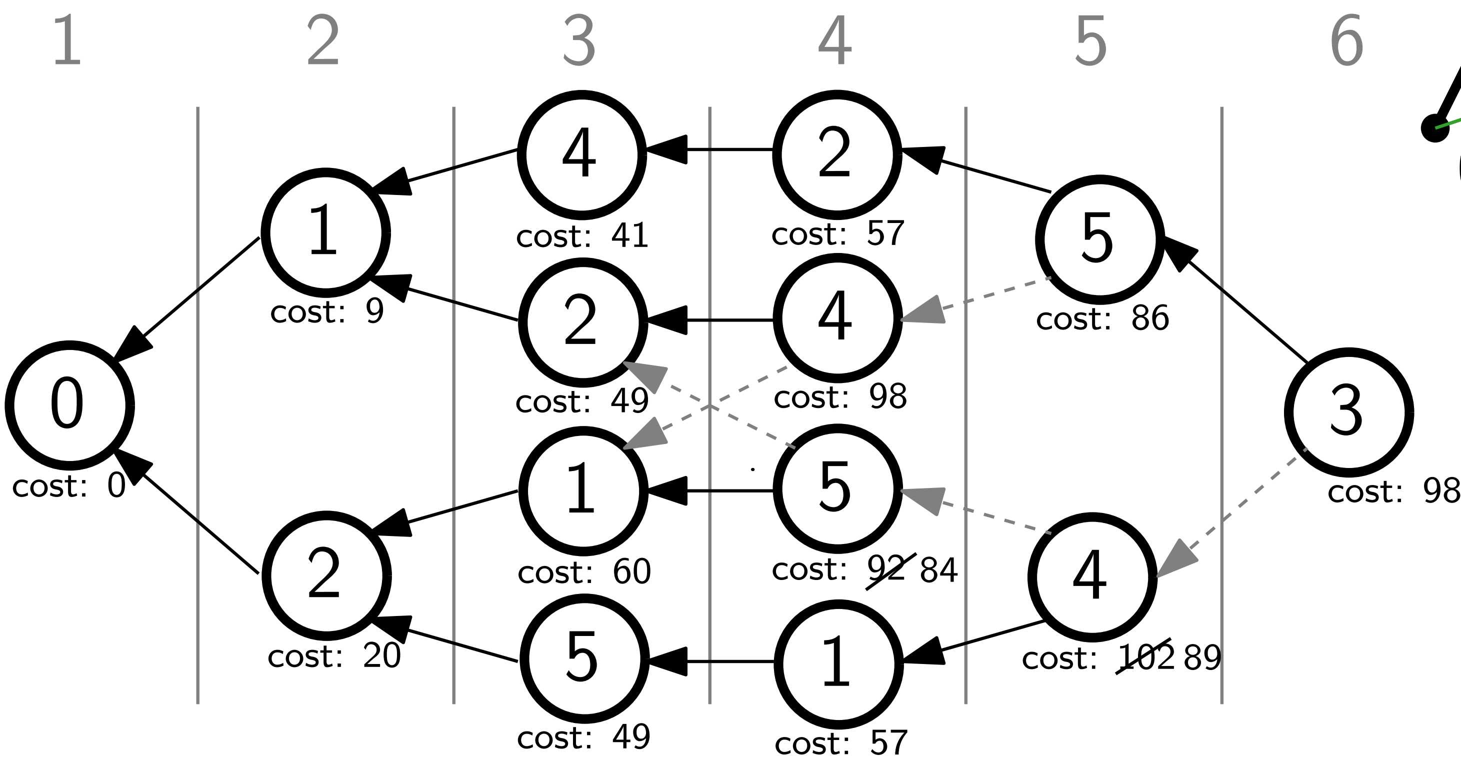
# An Exact Algorithm

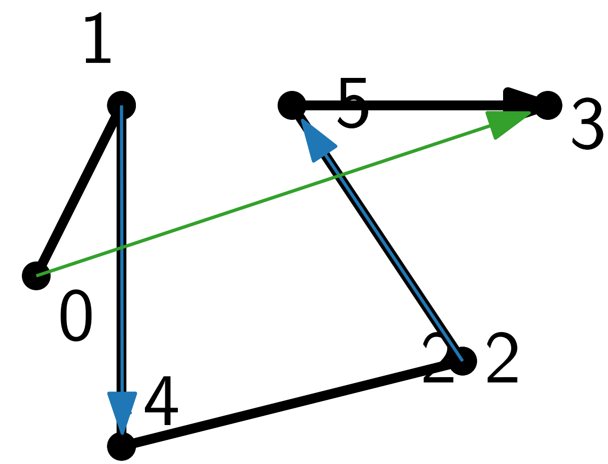
Find a tour with 6 steps:



# An Exact Algorithm

Find a tour with 6 steps:

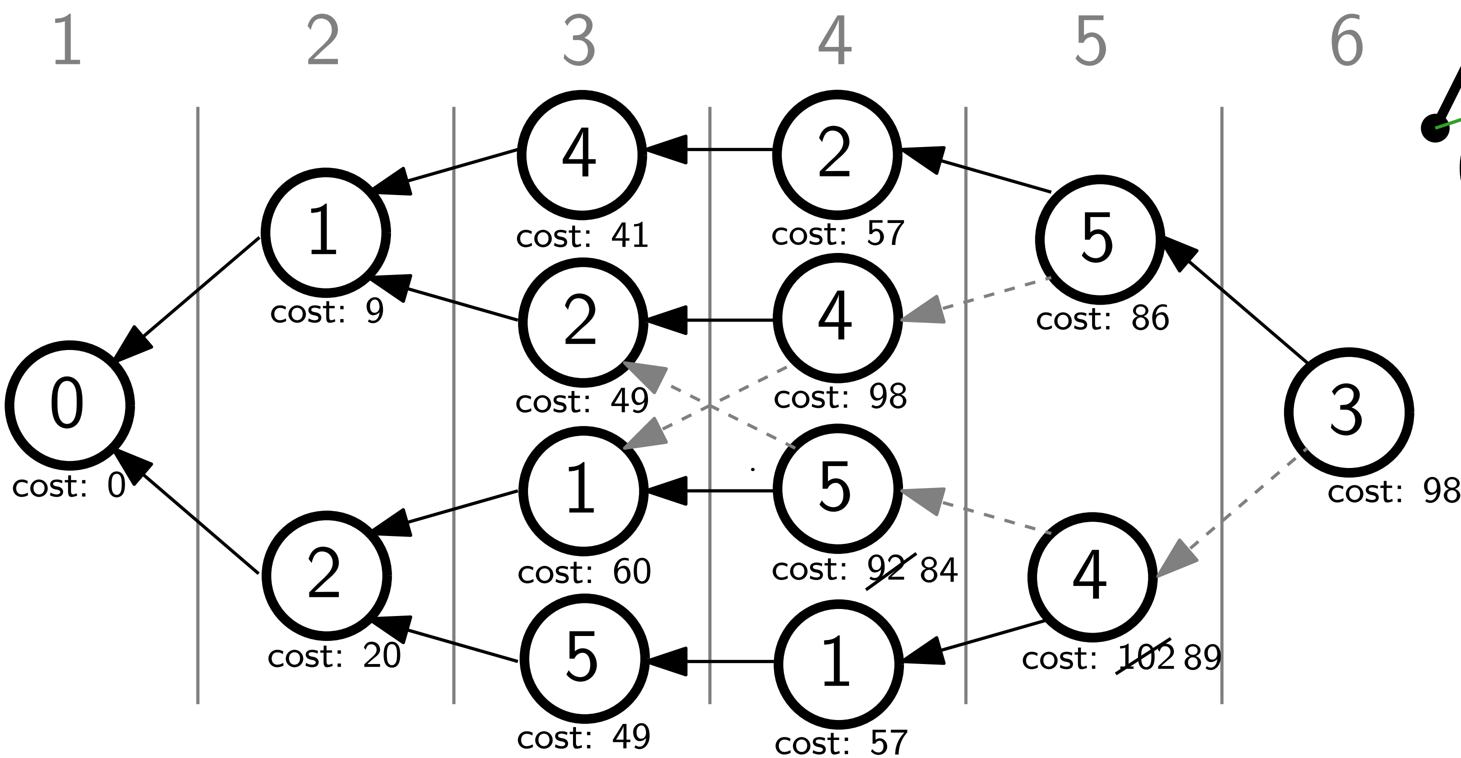


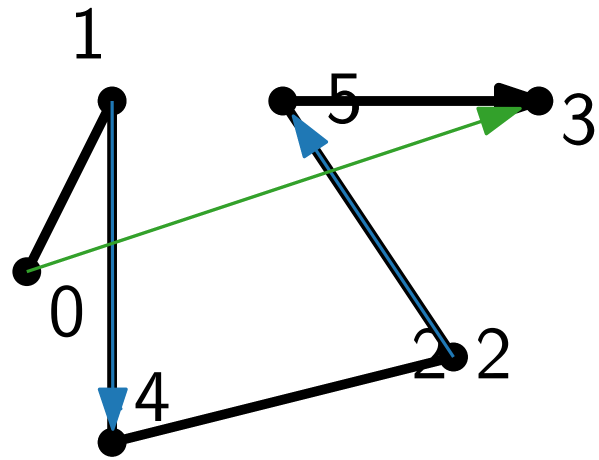




→ Generalizes to an algorithm with exchangeable objective

# An Exact Algorithm

Find a tour with 6 steps:



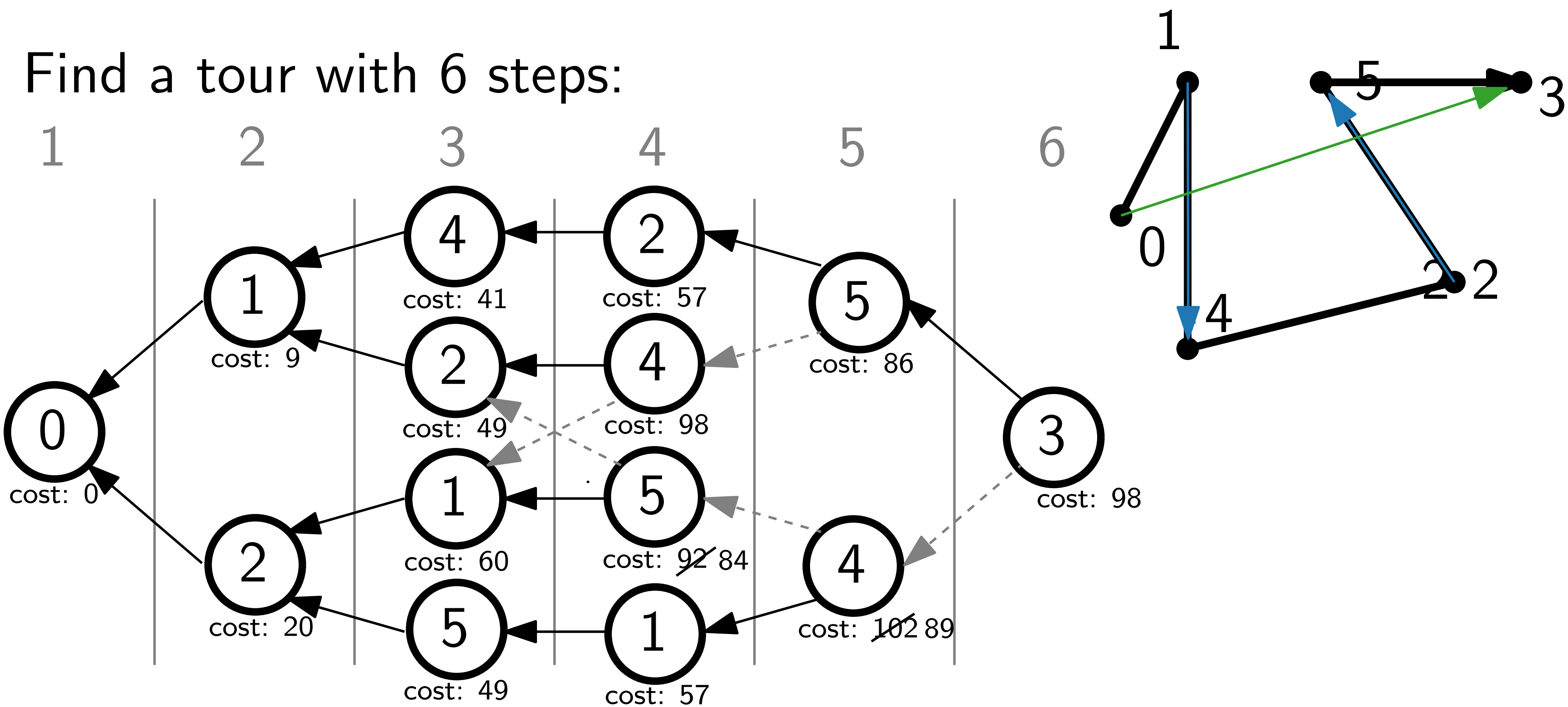




→ Generalizes to an algorithm with exchangeable objective

→ DFS-like traversal also possible [Psaraftis 1980]

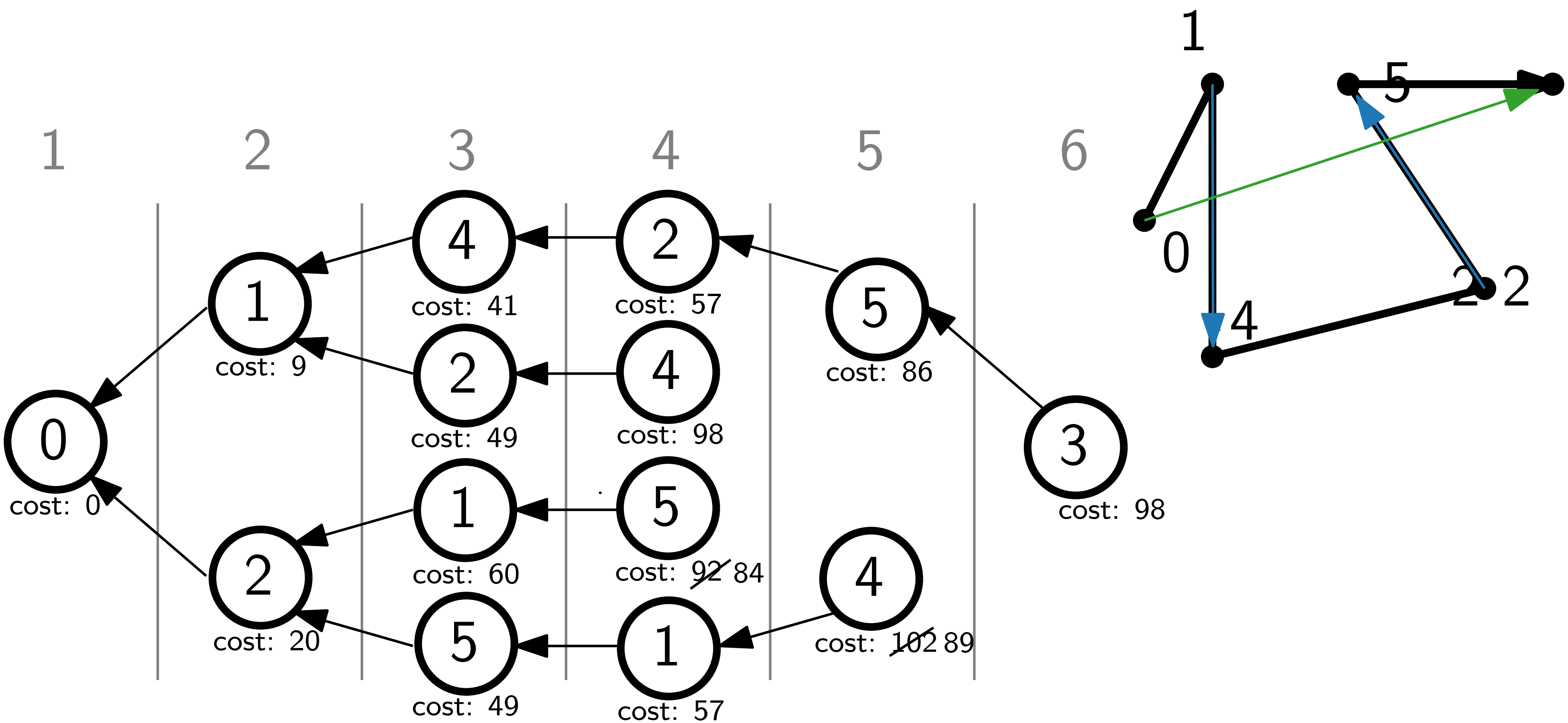
# An Exact Algorithm

Find a tour with 6 steps:

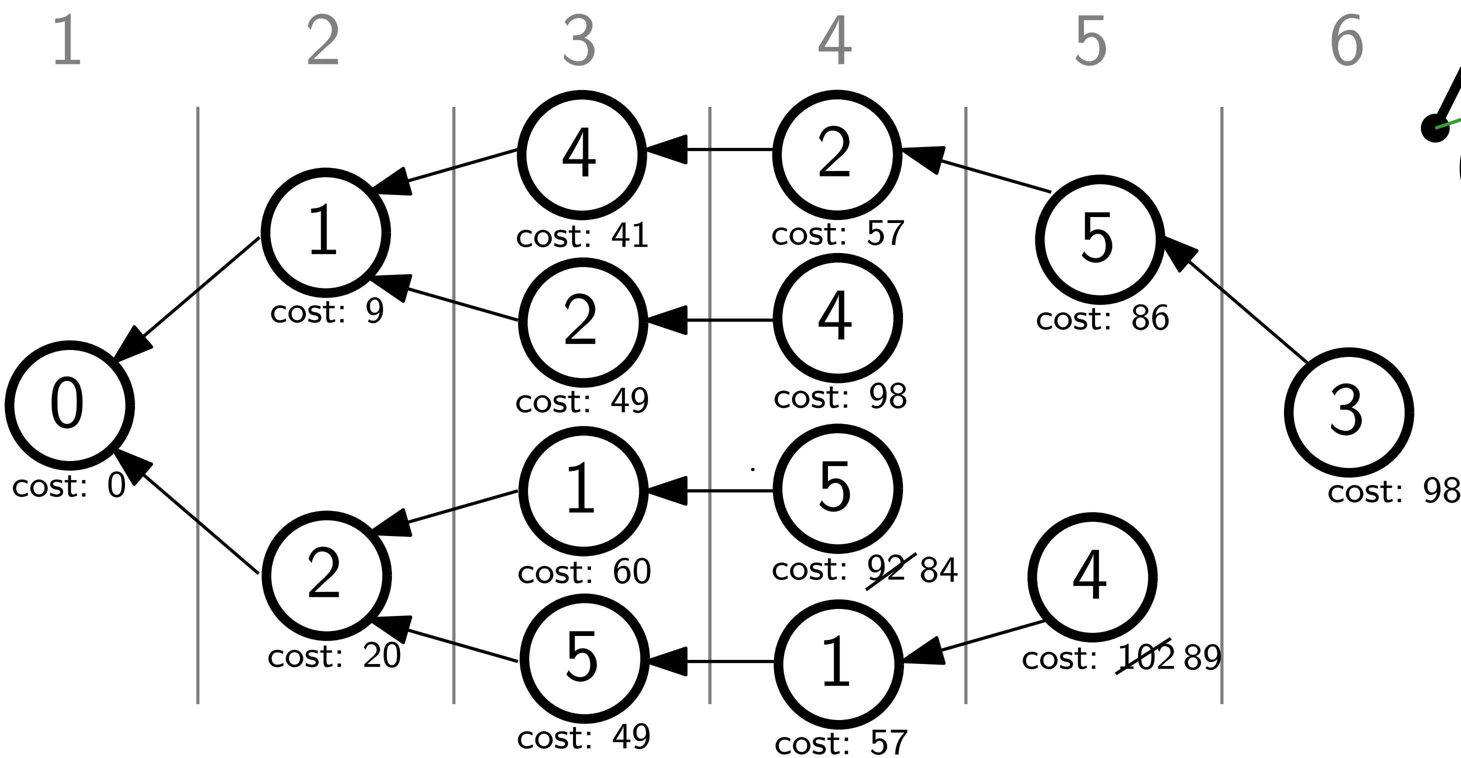


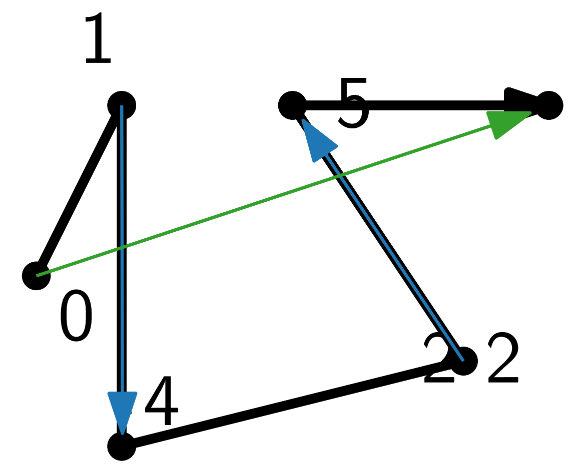

→ Generalizes to an algorithm with exchangeable objective
→ DFS-like traversal also possible [Psaraftis 1980]
→ BFS-like traversal can save storage

# Running Time and Partial Execution



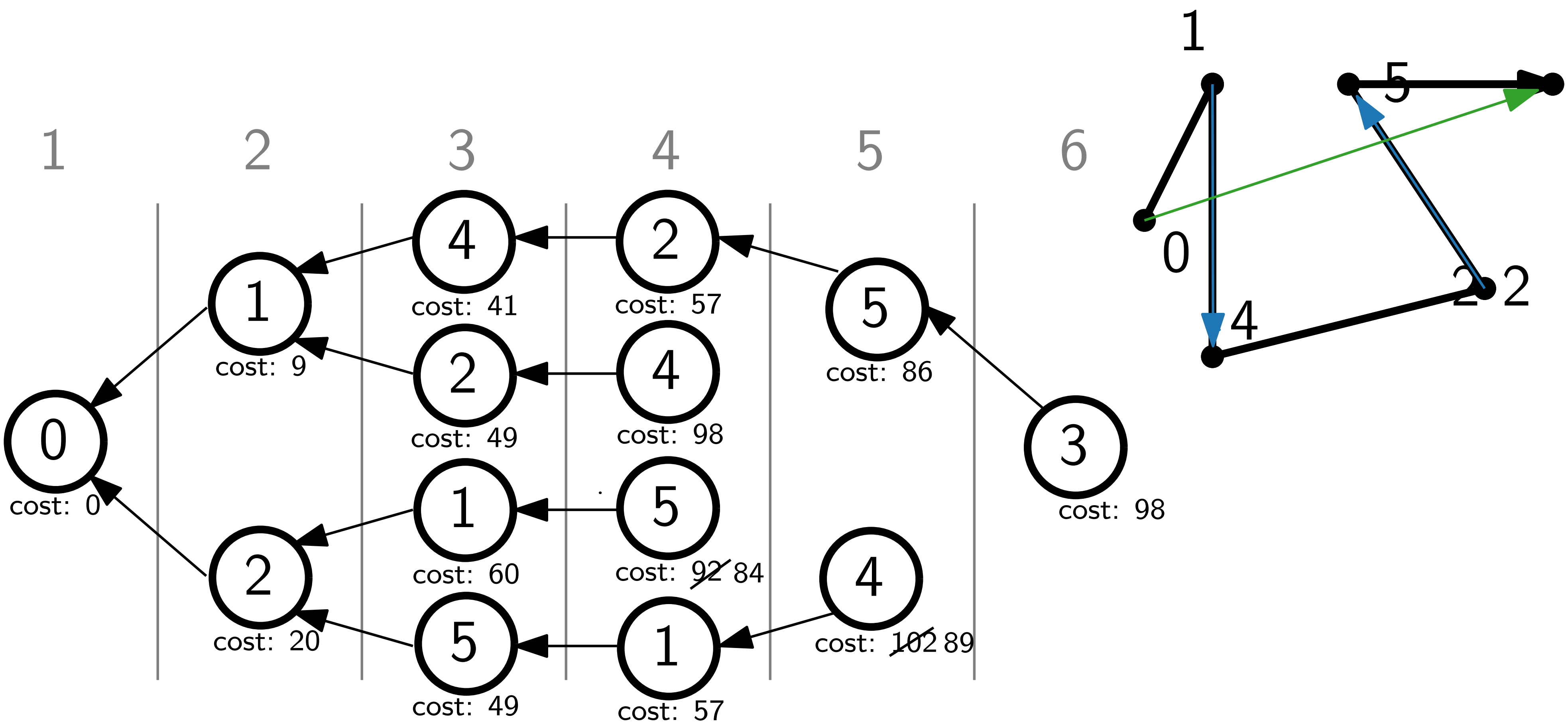
# Running Time and Partial Execution




At every step, a rider can have three steps: *wait*, *travel*, *finish*.

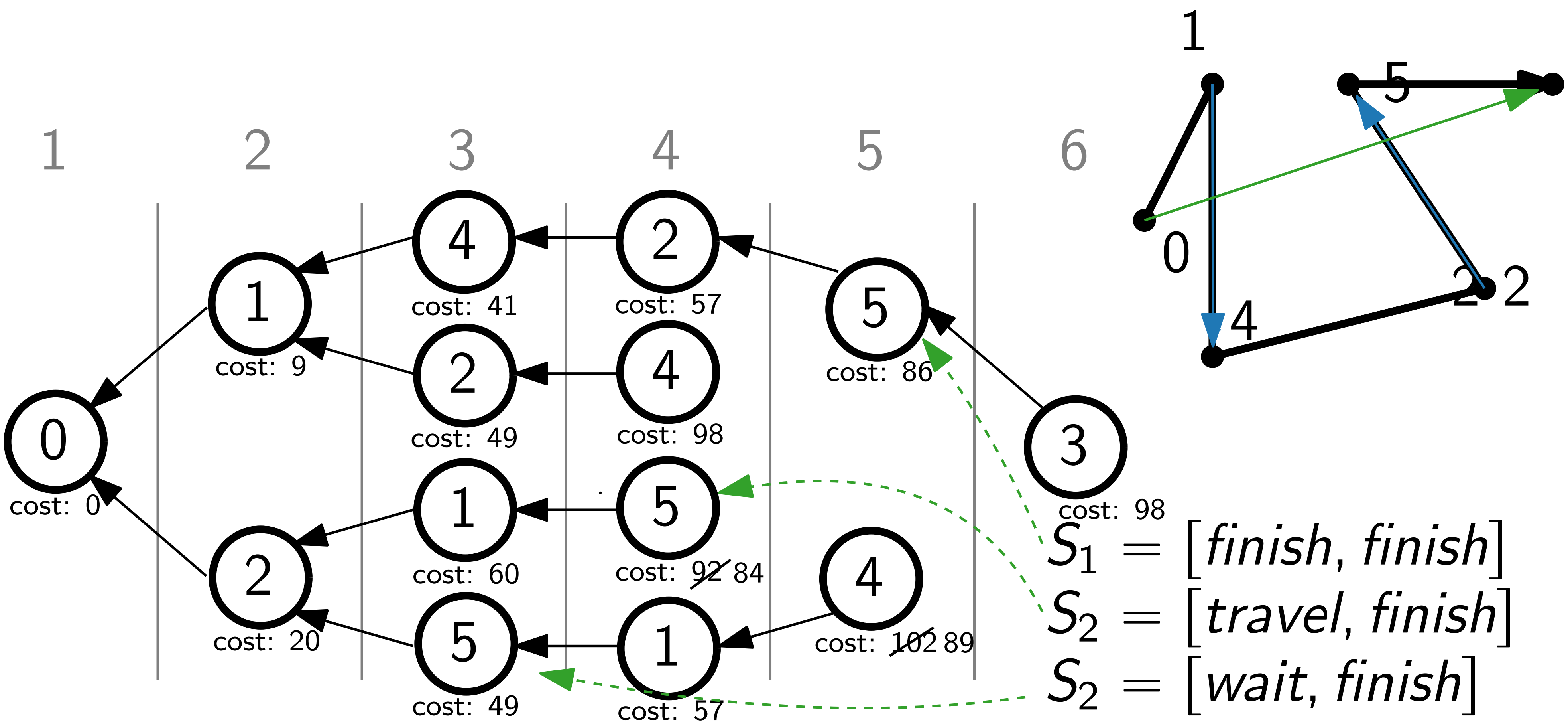
# Running Time and Partial Execution




At every step, a rider can have three steps: *wait*, *travel*, *finish*.

$\Rightarrow$ for a fixed location $\notin \{0, m\}$ there are $3^{n-1}$ states.

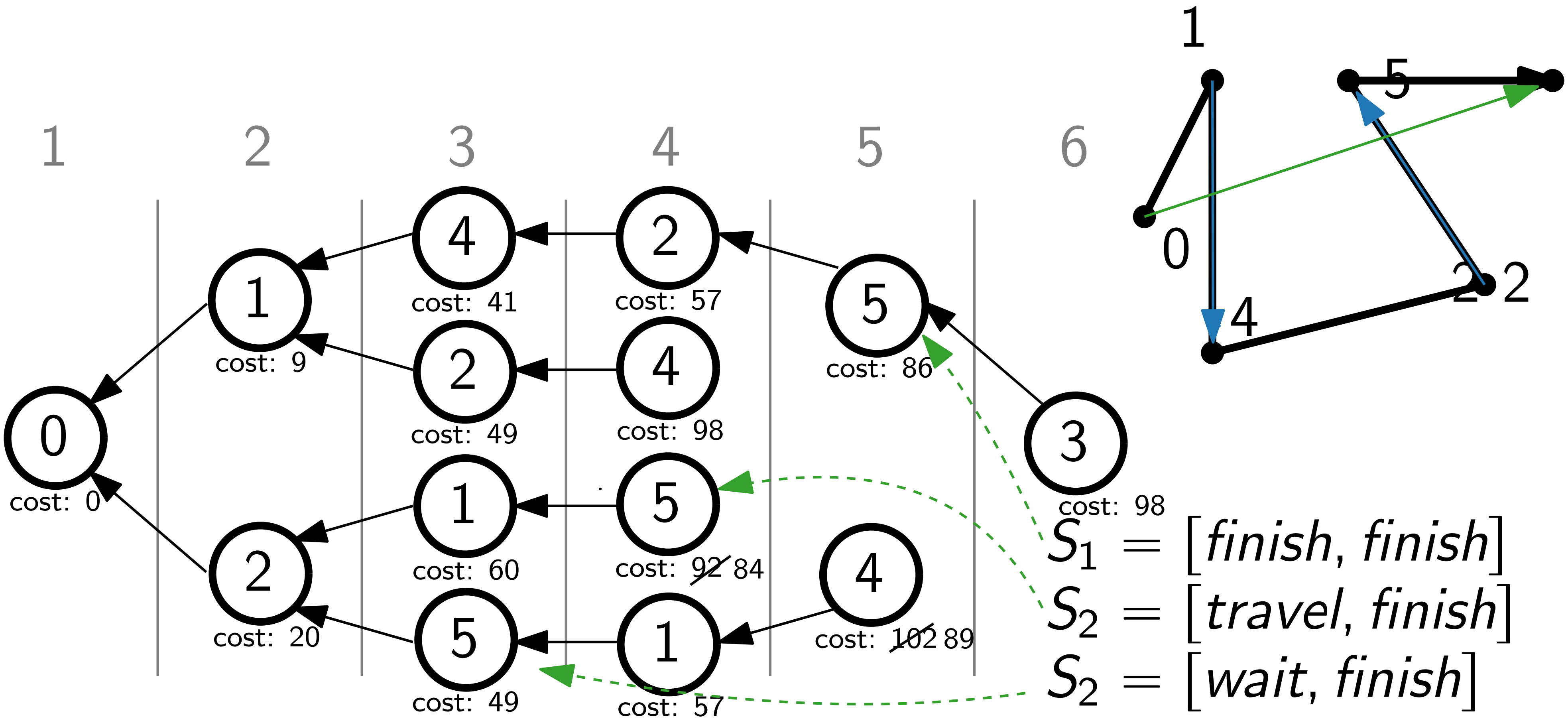
# Running Time and Partial Execution




At every step, a rider can have three steps: *wait*, *travel*, *finish*.

$\Rightarrow$ for a fixed location $\notin \{0, m\}$ there are $3^{n-1}$ states.

# Running Time and Partial Execution

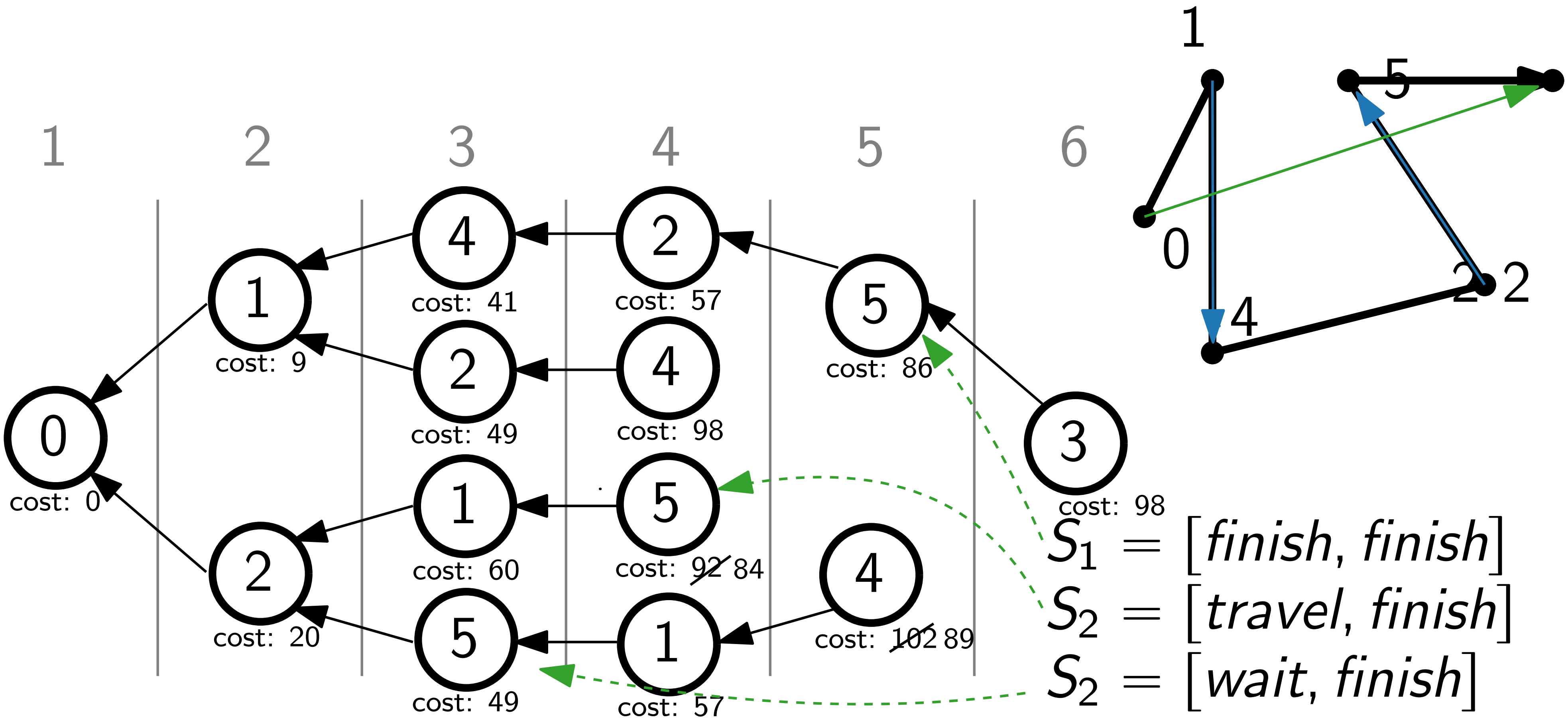



At every step, a rider can have three steps: *wait*, *travel*, *finish*.

$\Rightarrow$ for a fixed location $\notin \{0, m\}$ there are $3^{n-1}$ states.

$\Rightarrow 2n3^{n-1} + 2$ vertices, which yields $O^*(3^{n-1})$ running time.

# Running Time and Partial Execution



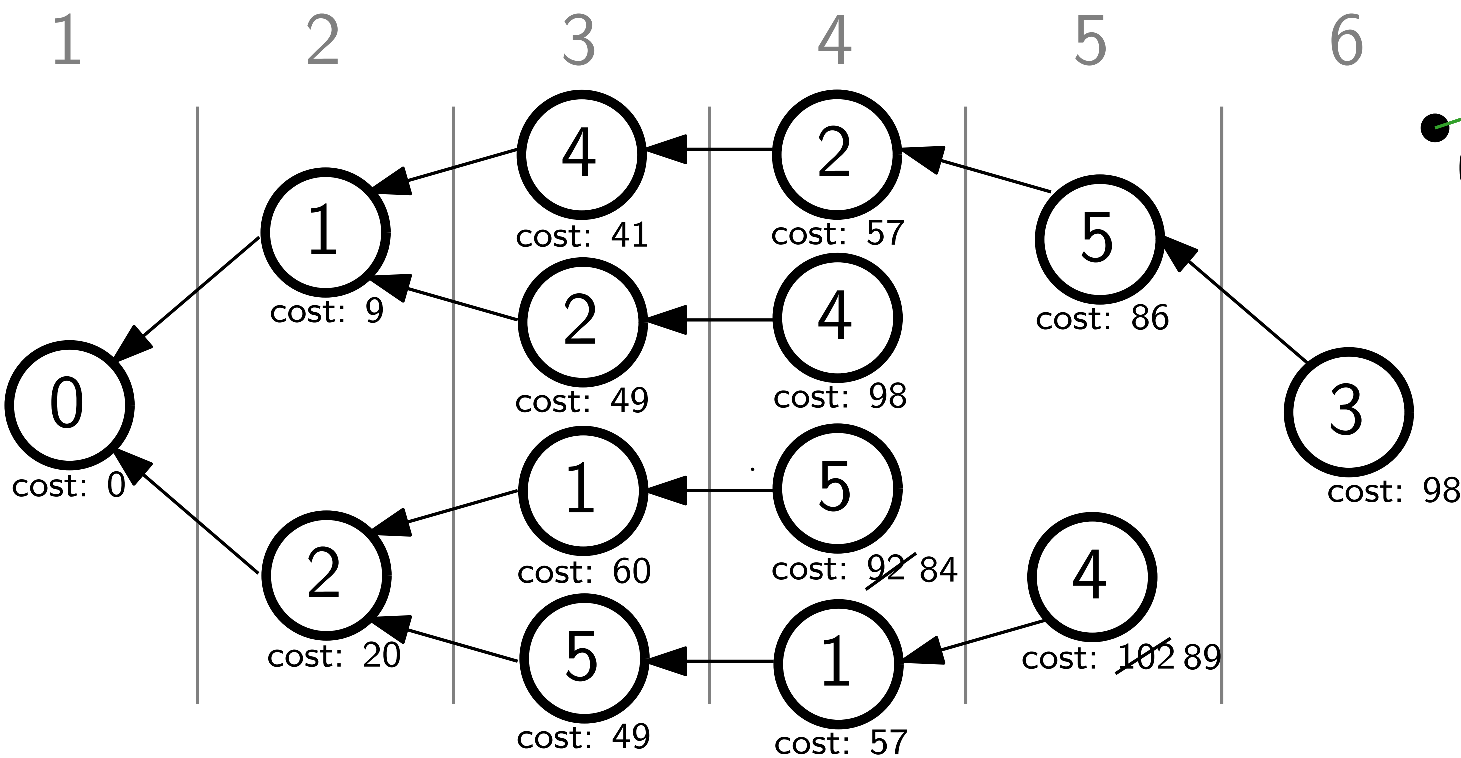

At every step, a rider can have three steps: *wait*, *travel*, *finish*.

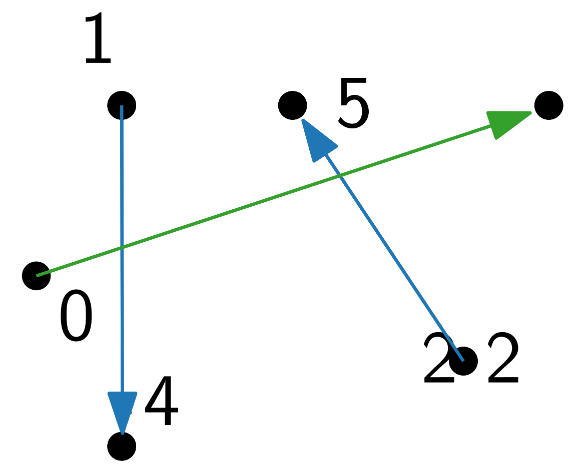
$\Rightarrow$ for a fixed location $\notin \{0, m\}$ there are $3^{n-1}$ states.

$\Rightarrow 2n3^{n-1} + 2$ vertices, which yields $O^*(3^{n-1})$ running time.

Given $S$ and $S'$, the instance can be solved *partially*.

# Running Time and Partial Execution



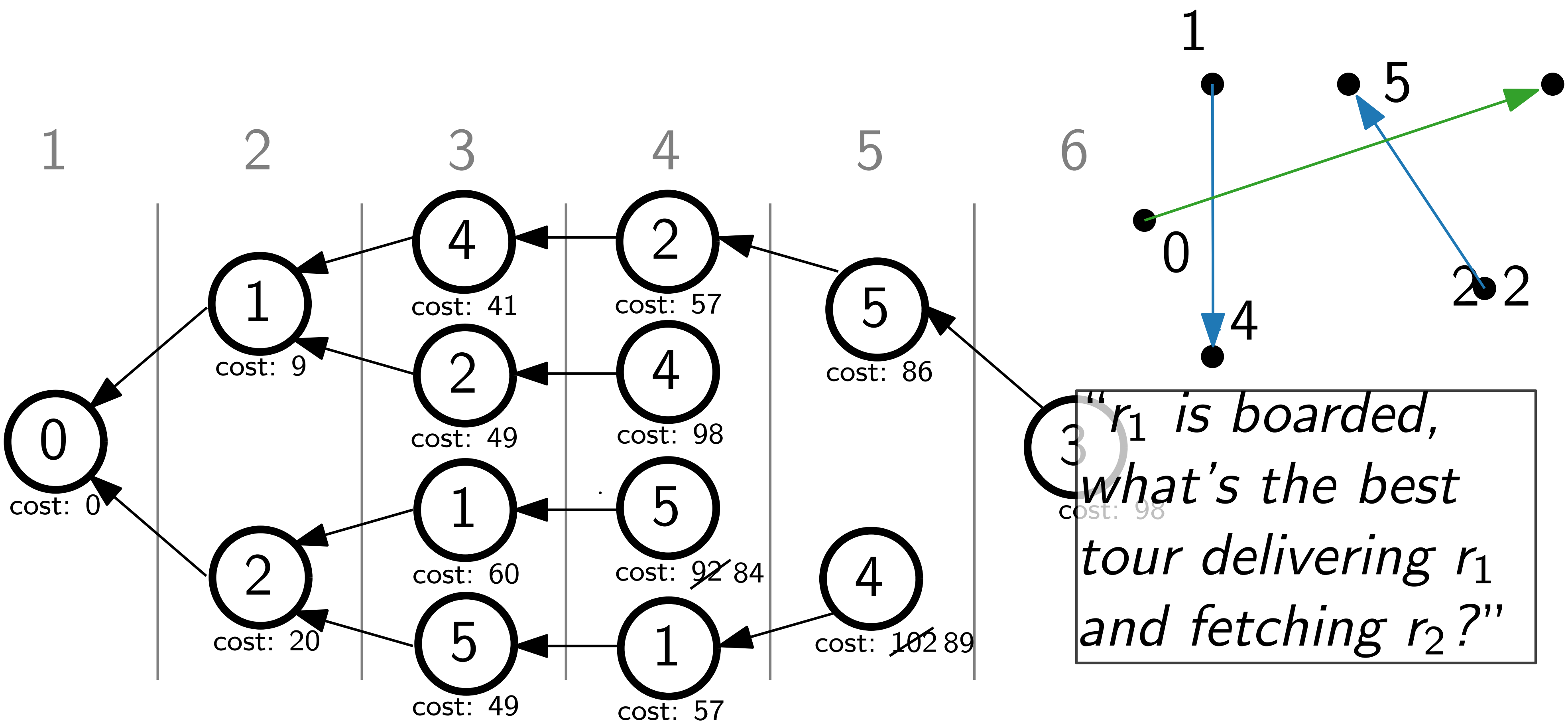

At every step, a rider can have three steps: *wait*, *travel*, *finish*.

$\Rightarrow$ for a fixed location $\notin \{0, m\}$ there are $3^{n-1}$ states.

$\Rightarrow$ $2n3^{n-1} + 2$ vertices, which yields $O^*(3^{n-1})$ running time.

Given $S$ and $S'$, the instance can be solved *partially*.

# Running Time and Partial Execution



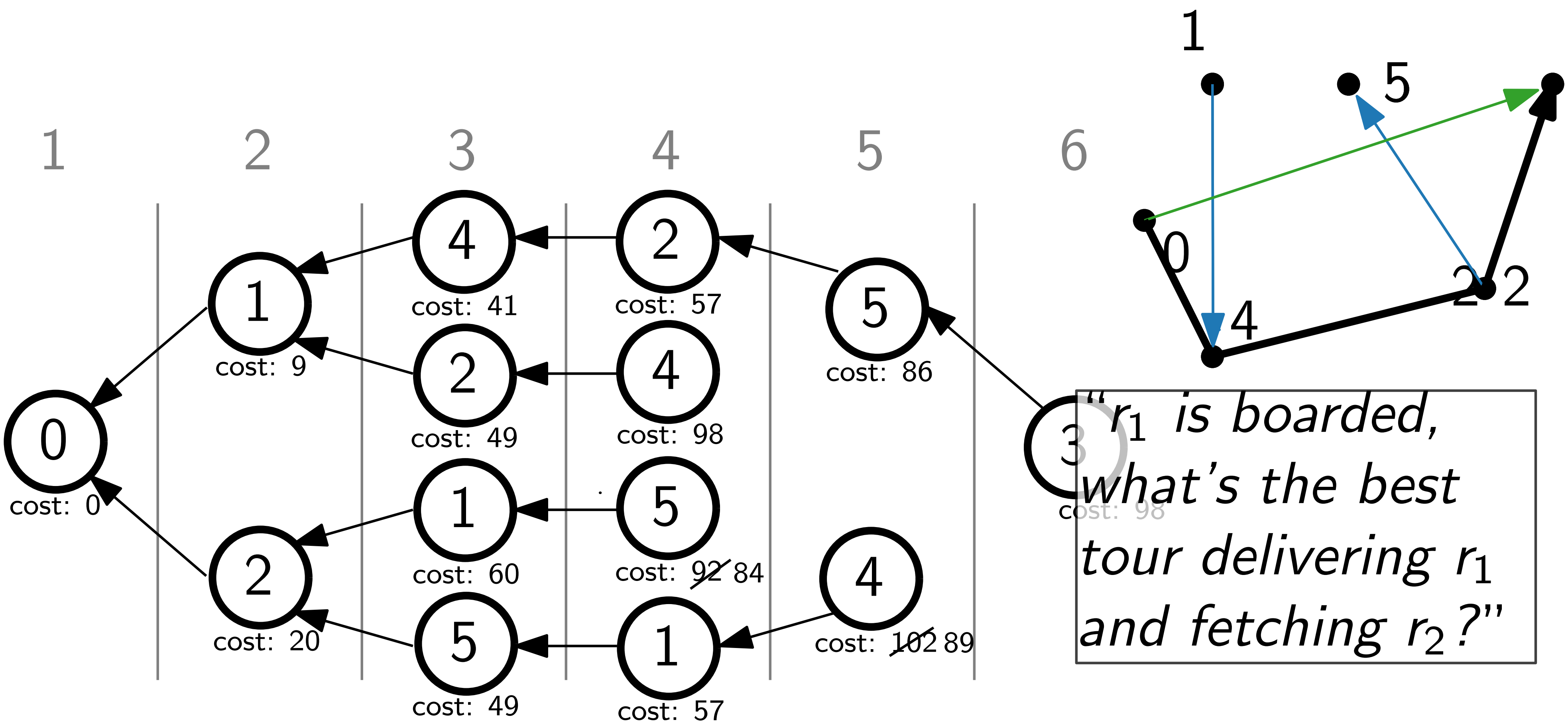

At every step, a rider can have three steps: *wait*, *travel*, *finish*.

$\Rightarrow$ for a fixed location $\notin \{0, m\}$ there are $3^{n-1}$ states.

$\Rightarrow 2n3^{n-1} + 2$ vertices, which yields $O^*(3^{n-1})$ running time.

Given $S$ and $S'$, the instance can be solved *partially*.

# Running Time and Partial Execution



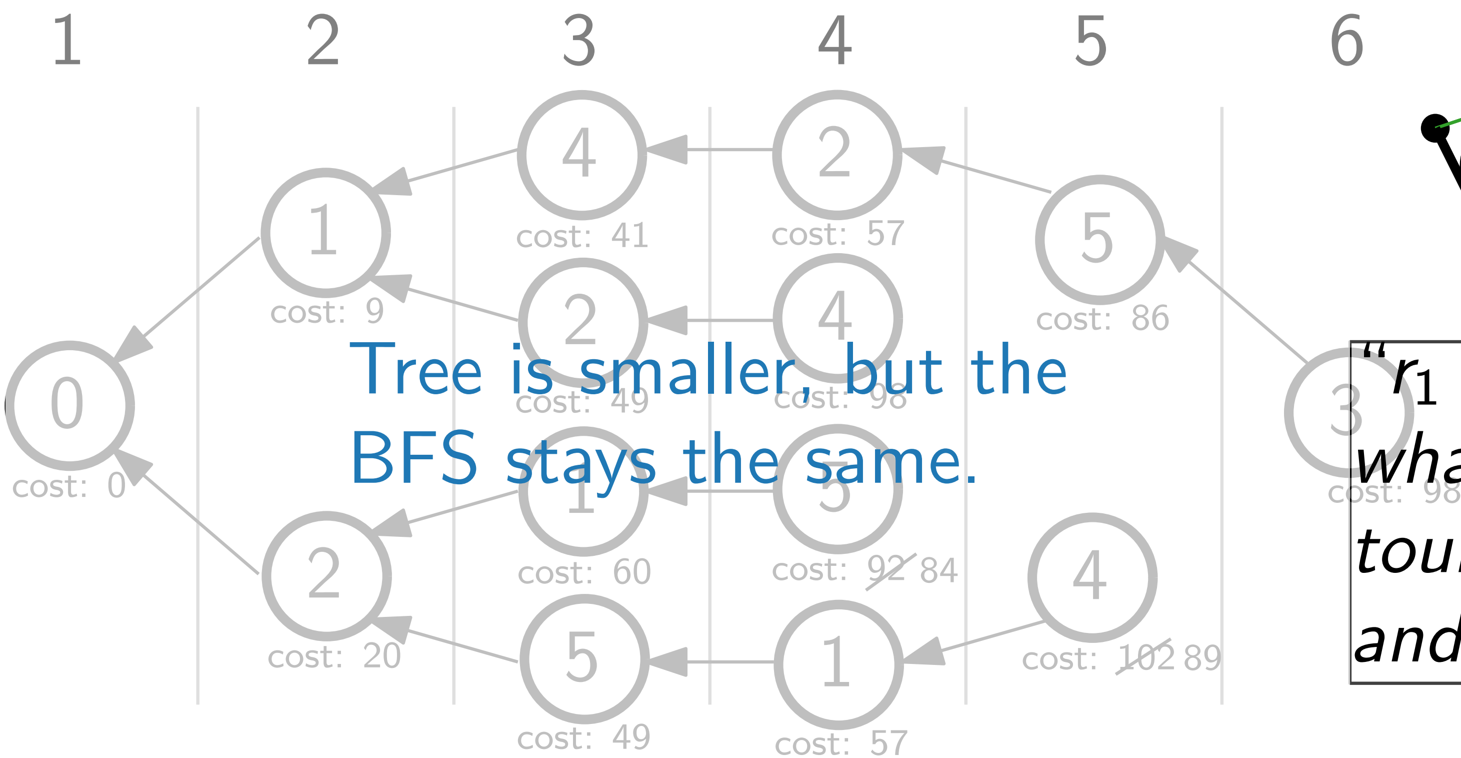

At every step, a rider can have three steps: *wait*, *travel*, *finish*.

$\Rightarrow$ for a fixed location $\notin \{0, m\}$ there are $3^{n-1}$ states.

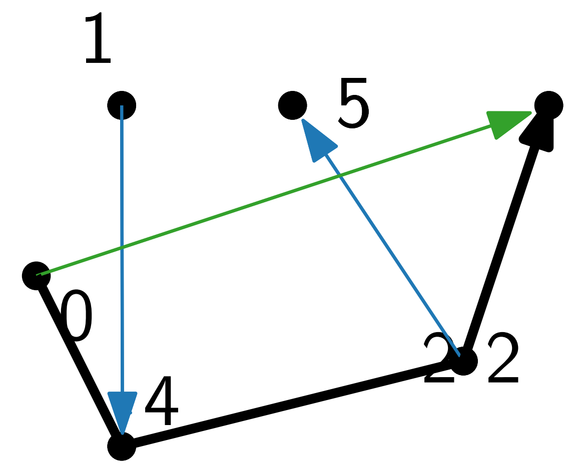
$\Rightarrow 2n3^{n-1} + 2$ vertices, which yields $O^*(3^{n-1})$ running time.

Given $S$ and $S'$, the instance can be solved *partially*.

# Running Time and Partial Execution




Tree is smaller, but the BFS stays the same.




"*$r_1$ is boarded, what's the best tour delivering $r_1$ and fetching $r_2$?*"

At every step, a rider can have three steps: *wait*, *travel*, *finish*.

$\Rightarrow$ for a fixed location $\notin \{0, m\}$ there are $3^{n-1}$ states.

$\Rightarrow 2n3^{n-1} + 2$ vertices, which yields $O^*(3^{n-1})$ running time.

Given $S$ and $S'$, the instance can be solved *partially*.

# Evalution of Realistic Examples

# Evalution of Realistic Examples

**Rural Scenario**

# Evalution of Realistic Examples

**Rural Scenario**

**Regional Scenario**

# Evalution of Realistic Examples

**Rural Scenario**

**Regional Scenario**

**Intercity Scenario**

# Evalution of Realistic Examples

**Rural Scenario**

Six small villages with $\varnothing$1.2 km distance.

**Regional Scenario**

**Intercity Scenario**

# Evalution of Realistic Examples

**Rural Scenario**

Six small villages with ∅1.2 km distance.

**Regional Scenario**

Six small towns with ∅7.2 km distance.

**Intercity Scenario**

# Evalution of Realistic Examples

**Rural Scenario**
Six small villages with ∅1.2 km distance.

**Regional Scenario**

Six small towns with ∅7.2 km distance.

**Intercity Scenario**
Six major german cities with ∅129 km distance.

# Evalution of Realistic Examples

**Rural Scenario**

Six small villages with $\varnothing$1.2 km distance.

**Regional Scenario**

Six small towns with $\varnothing$7.2 km distance.

**Intercity Scenario**

Six major german cities with $\varnothing$129 km distance.

All optimal tours are unidirectional, recall $> 0.9$.

# Evalution of Realistic Examples

**Rural Scenario**

Six small villages with ∅1.2 km distance.

$>$ 70% optimal tours unidirectional, recall $<$ 0.1.

**Regional Scenario**

Six small towns with ∅7.2 km distance.

**Intercity Scenario**

Six major german cities with ∅129 km distance.

All optimal tours are unidirectional, recall $>$ 0.9.

# Evalution of Realistic Examples

**Rural Scenario**
Six small villages with ∅1.2 km distance.
$>$ 70% optimal tours unidirectional, recall $<$ 0.1.

Bad, distances are too small.

**Regional Scenario**

Six small towns with ∅7.2 km distance.

**Intercity Scenario**
Six major german cities with ∅129 km distance.

All optimal tours are unidirectional, recall $>$ 0.9.

# Evalution of Realistic Examples

**Rural Scenario**

Six small villages with ∅1.2 km distance.

$>$ 70% optimal tours unidirectional, recall $<$ 0.1.

Bad, distances are too small.

**Regional Scenario**

Six small towns with ∅7.2 km distance.

$>$ 50% optimal tours unidir., recall $>$ 0.55, precision 0.61.

**Intercity Scenario**

Six major german cities with ∅129 km distance.

All optimal tours are unidirectional, recall $>$ 0.9.

# Evalution of Realistic Examples

**Rural Scenario**

Six small villages with ∅1.2 km distance.

$>$ 70% optimal tours unidirectional, recall $<$ 0.1.

Bad, distances are too small.

**Regional Scenario**

Six small towns with ∅7.2 km distance.

$>$ 50% optimal tours unidir., recall $>$ 0.55, precision 0.61.

Wait ... What?!

**Intercity Scenario**

Six major german cities with ∅129 km distance.

All optimal tours are unidirectional, recall $>$ 0.9.