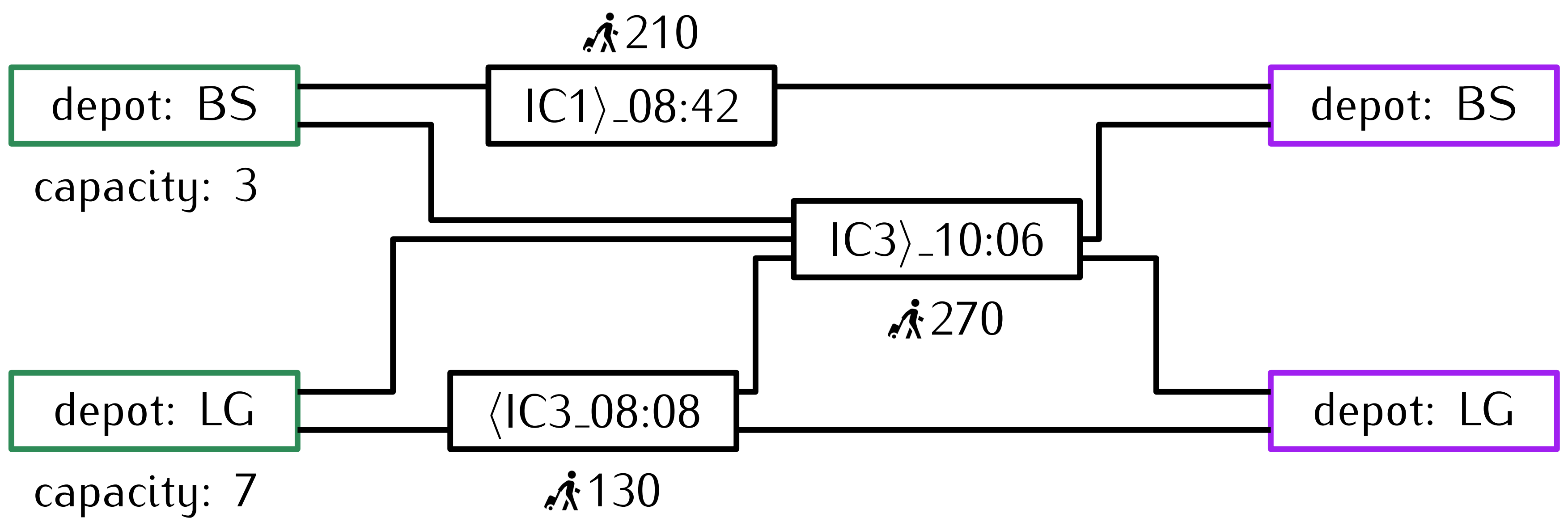
Collaboration SBB - ETH

RSSched

Rolling Stock Scheduling with Maintenance Regulations



by Leon Sering

Rolling Stock Scheduling

What does **rolling stock** mean?

rolling stock: "Vehicles that drive on rails."

for the project:

vehicle:	multiple wagen that can drive by itself but are never uncoupled (smallest unit)
formation:	one or more vehicles that are coupled and form a train
type:	vehicle can be of different types, only vehicles of the same type can form a formation

Rolling Stock Scheduling

What does **rolling stock** mean?

rolling stock: "Vehicles that drive on rails."

for the project:

vehicle:	multiple wagen that can drive by itself but are never uncoupled (smallest unit)
formation:	one or more vehicles that are coupled and form a train
type:	vehicle can be of different types, only vehicles of the same type can form a formation

What is a **schedule** in this context?

schedule:	collection of **tours**, one for each vehicle
tour:	sequence of **activities** covering one day

• **start:**	spawning at a depot in the morning
• **service trip:**	brings customers from a to b
• **dead-head trip:**	driving from a to b without customers
• **maintenance:**	maintenance check after given distance
• **end:**	de-spawning at a depot in the evening

Rolling Stock Scheduling

What does **rolling stock** mean?

rolling stock: "Vehicles that drive on rails."

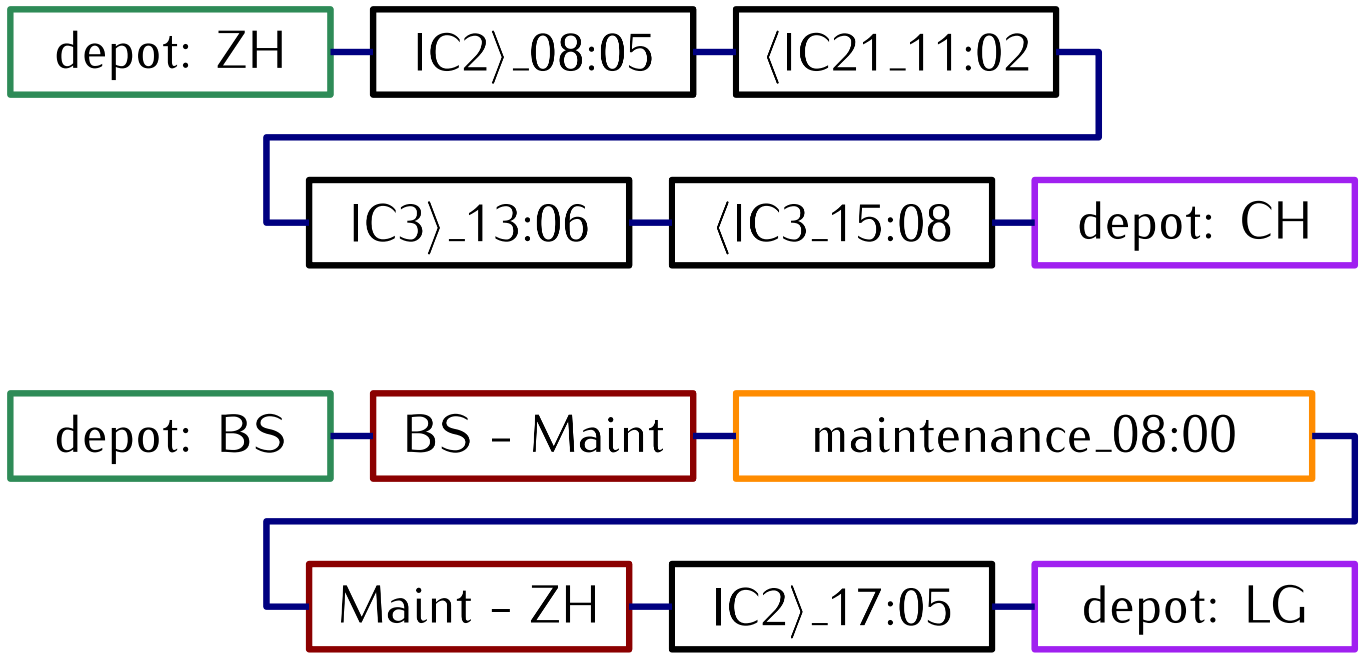
for the project:

vehicle:	multiple wagen that can drive by itself but are never uncoupled (smallest unit)
formation:	one or more vehicles that are coupled and form a train
type:	vehicle can be of different types, only vehicles of the same type can form a formation

What is a **schedule** in this context?

schedule:	collection of **tours**, one for each vehicle
tour:	sequence of **activities** covering one day

- **start:** spawning at a depot in the morning
- **service trip:** brings customers from a to b
- **dead-head trip:** driving from a to b without customers
- **maintenance:** maintenance check after given distance
- **end:** de-spawning at a depot in the evening



Phase 1
Basic Scheduling

Phase 1 – Task

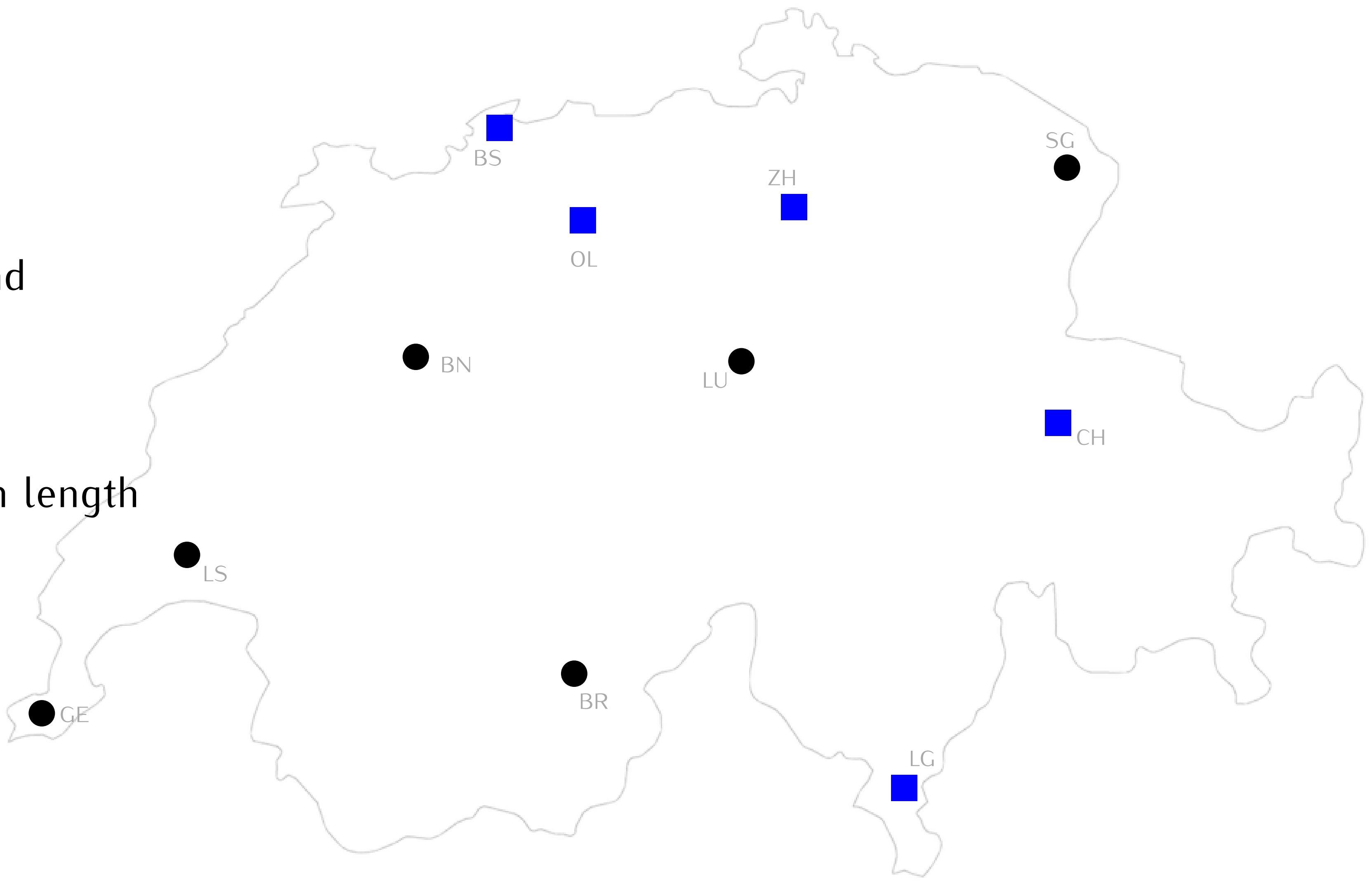
input:

- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L \times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L \times L}$
- **vehicle type infos** passenger capacity, max formation length

Phase 1 – Task

input:

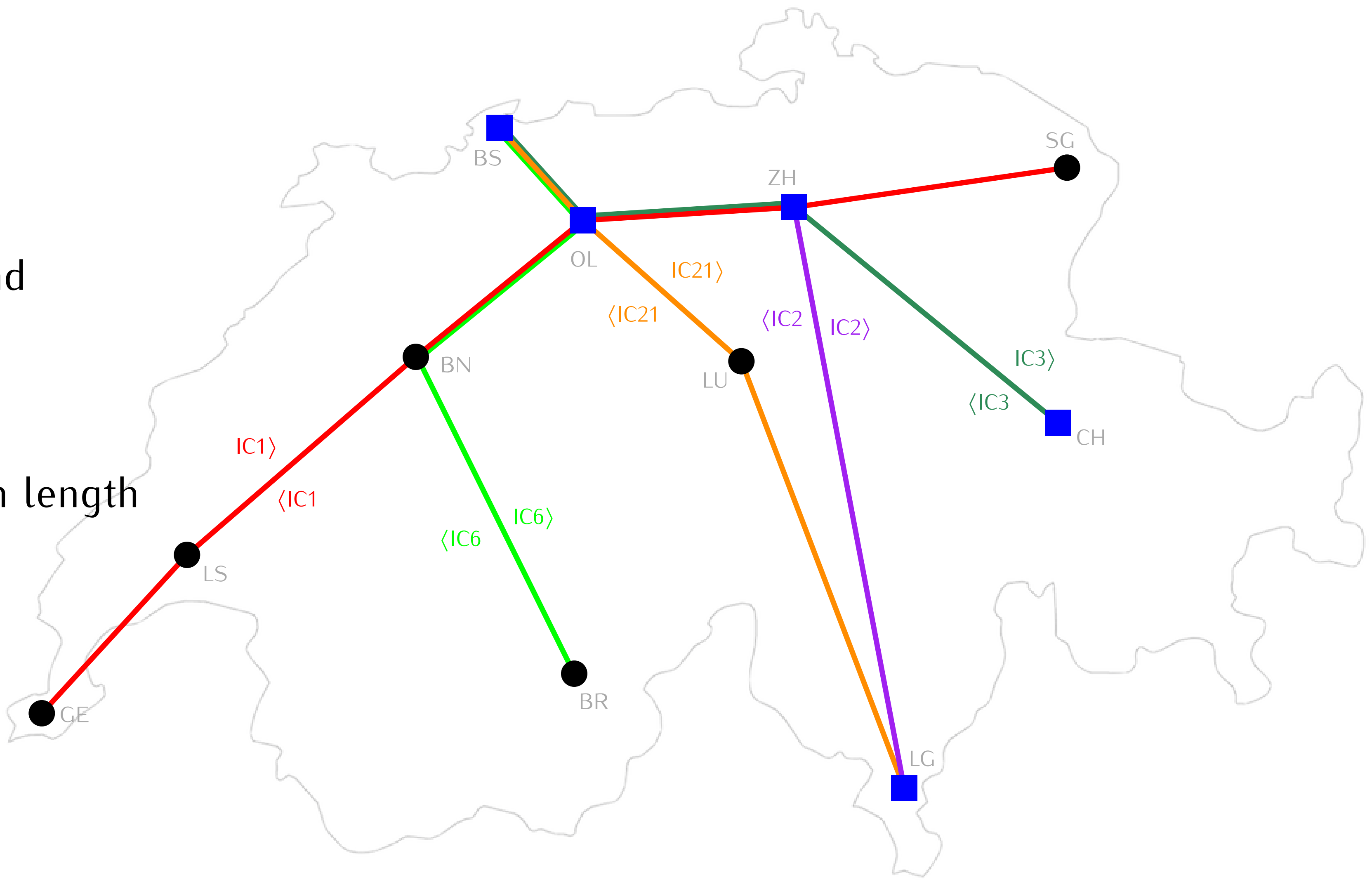
- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L \times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L \times L}$
- **vehicle type infos** passenger capacity, max formation length



Phase 1 – Task

input:

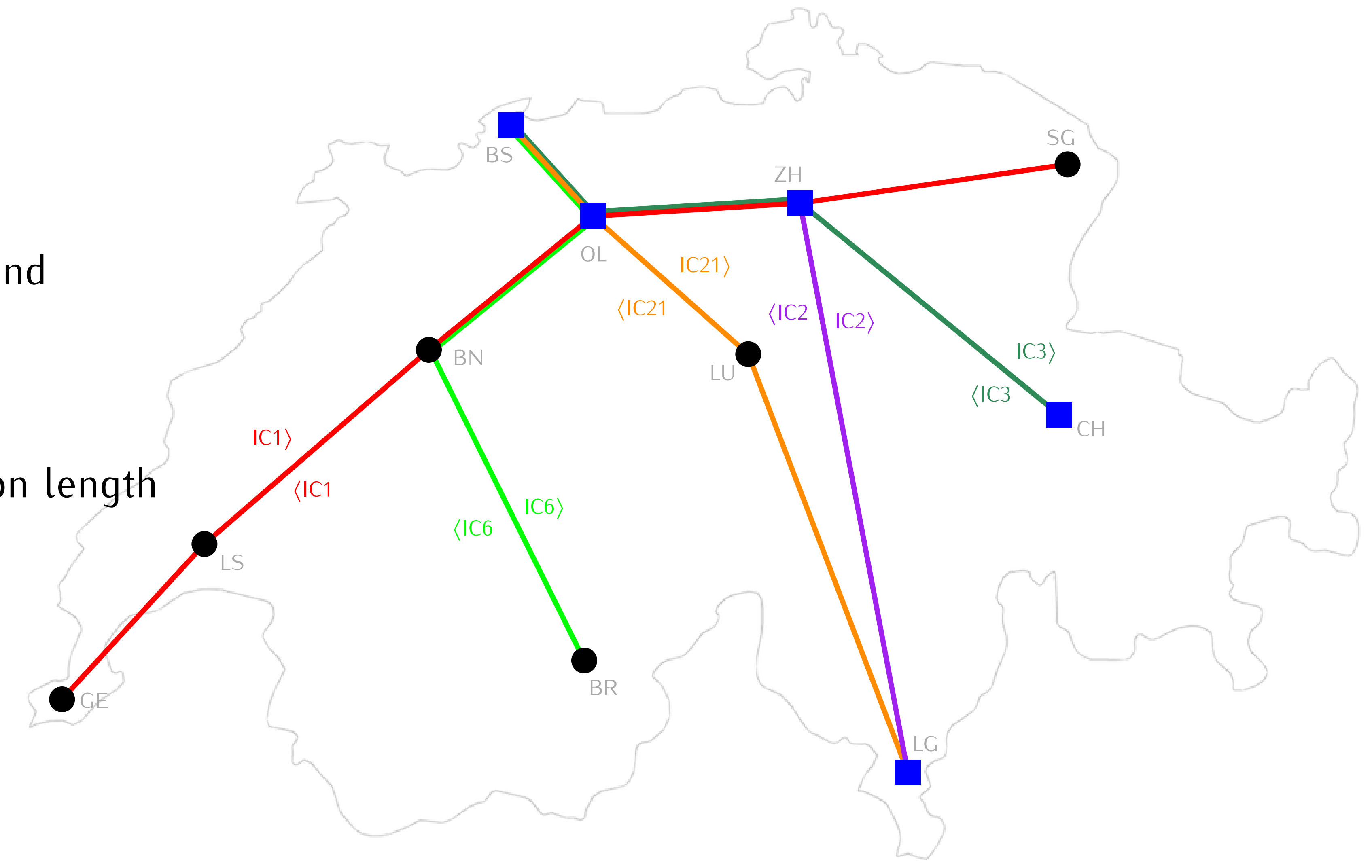
- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L\times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L\times L}$
- **vehicle type infos** passenger capacity, max formation length



Phase 1 – Task

input:

- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L \times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L \times L}$
- **vehicle type infos** passenger capacity, max formation length

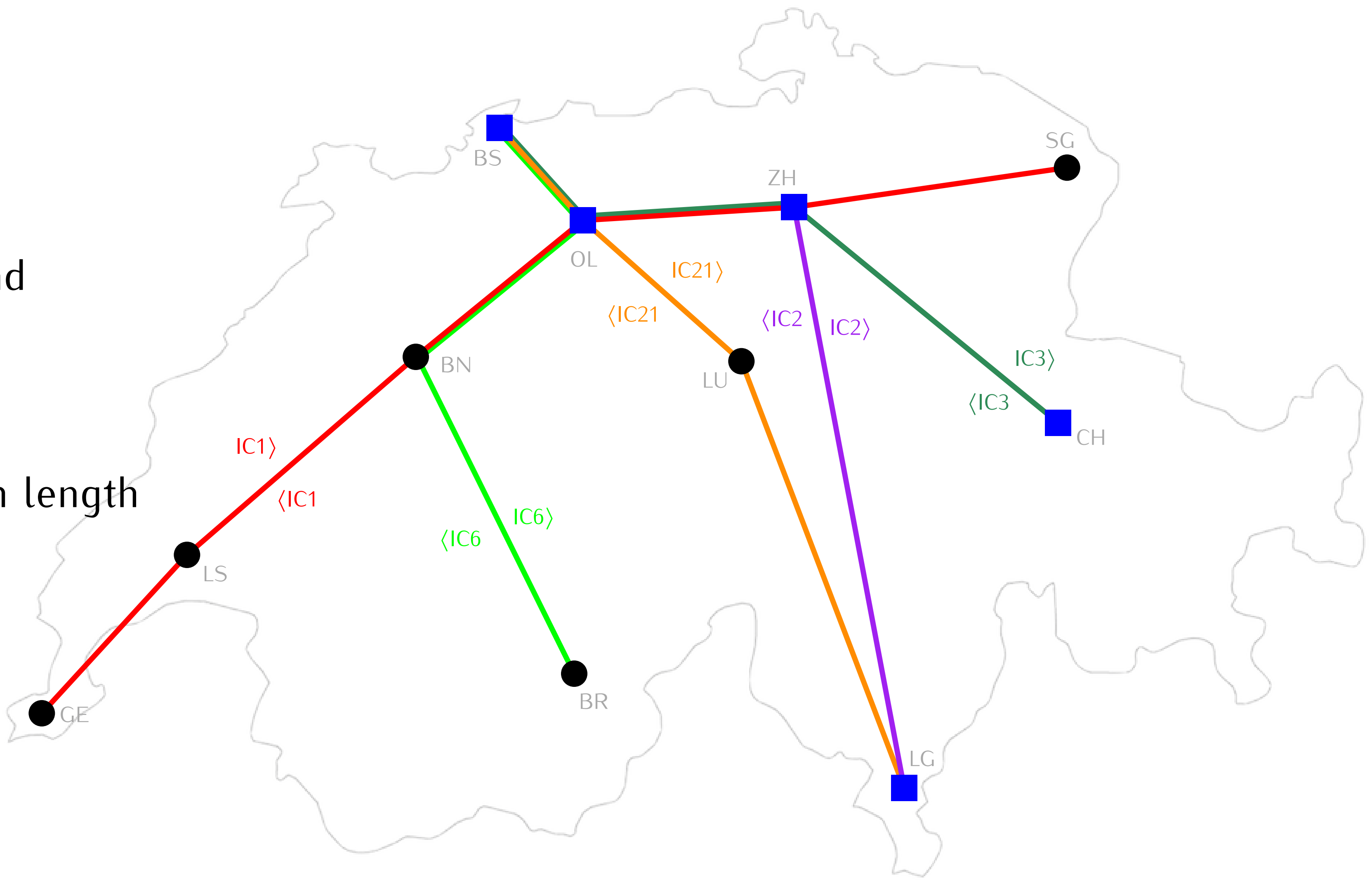


IC1⟩_05:42	⟨IC1_05:07	. . .	⟨IC21_07:02
IC1⟩_06:42	⟨IC1_06:07		⟨IC21_08:02
IC1⟩_07:42	⟨IC1_07:07		⟨IC21_09:02
⋮	⋮		⋮
IC1⟩_20:42	⟨IC1_21:07	. . .	⟨IC21_22:02

Phase 1 – Task

input:

- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L \times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L \times L}$
- **vehicle type infos** passenger capacity, max formation length



IC1⟩_05:42 🚶 80	⟨IC1_05:07 🚶 20	. . .	⟨IC21_07:02 🚶 70
IC1⟩_06:42 🚶210	⟨IC1_06:07 🚶130		⟨IC21_08:02 🚶130
IC1⟩_07:42 🚶180	⟨IC1_07:07 🚶310		⟨IC21_09:02 🚶150
⋮	⋮		⋮
IC1⟩_20:42 🚶100	⟨IC1_21:07 🚶110	. . .	⟨IC21_22:02 🚶140

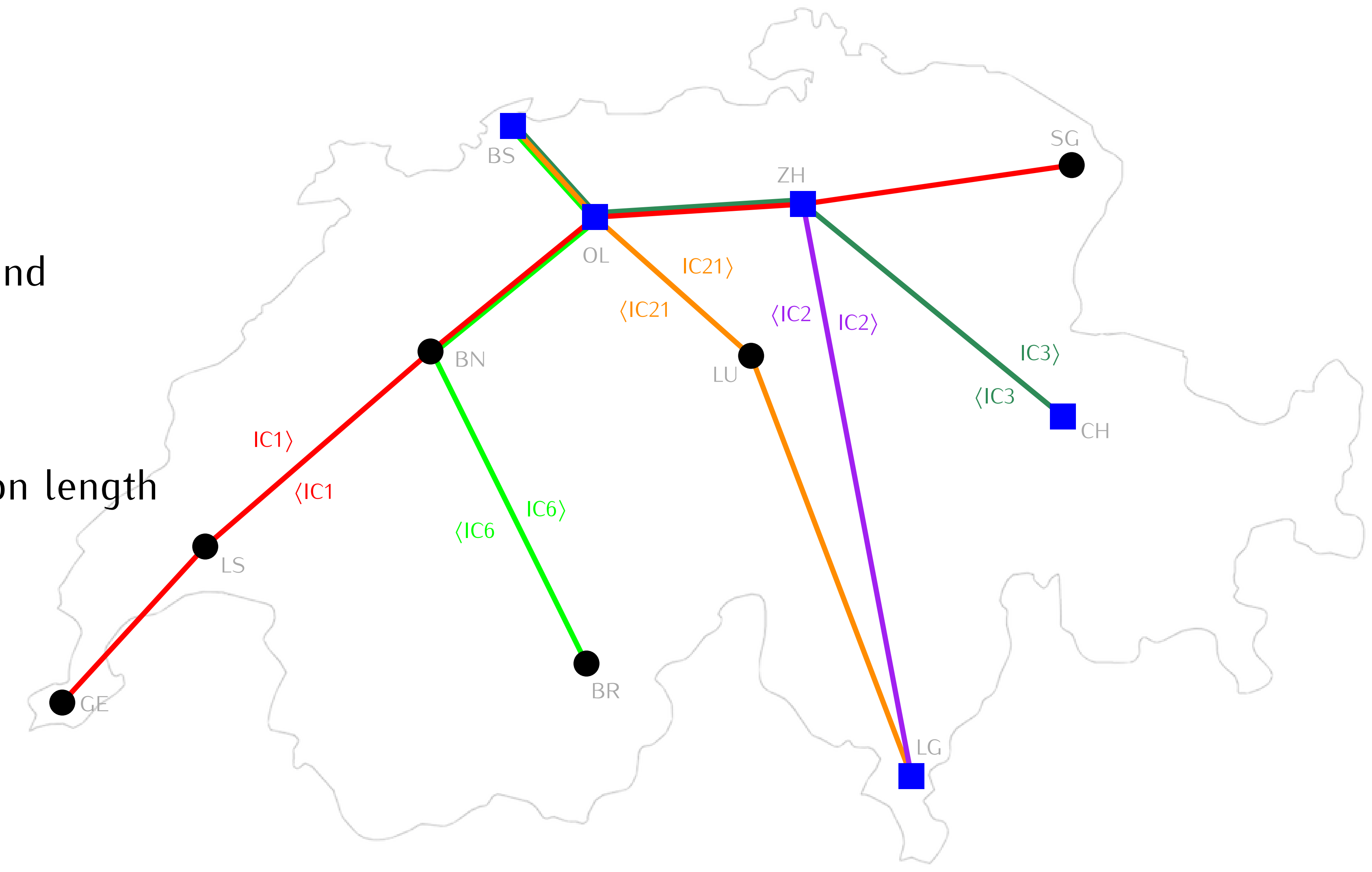
Phase 1 – Task

input:

- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L \times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L \times L}$
- **vehicle type infos** passenger capacity, max formation length

output (schedule):

- a list of tours starting and ending at a depot
- cyclic (every day the same schedule)
- no maintenance yet



IC1⟩_05:42 🚶 80	⟨IC1_05:07 🚶 20	...	⟨IC21_07:02 🚶 70
IC1⟩_06:42 🚶210	⟨IC1_06:07 🚶130		⟨IC21_08:02 🚶130
IC1⟩_07:42 🚶180	⟨IC1_07:07 🚶310		⟨IC21_09:02 🚶150
⋮	⋮		⋮
IC1⟩_20:42 🚶100	⟨IC1_21:07 🚶110	...	⟨IC21_22:02 🚶140

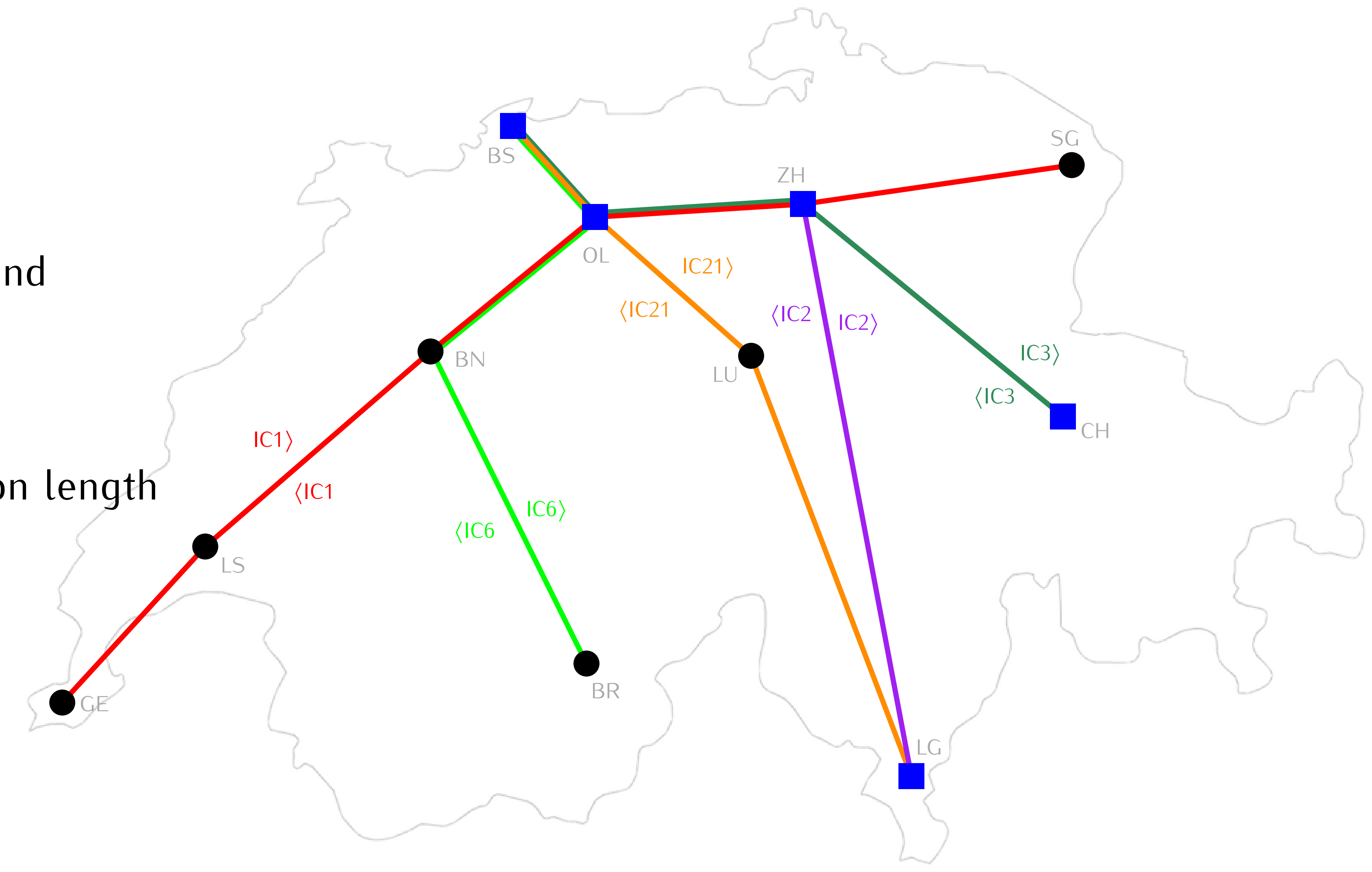
Phase 1 – Task

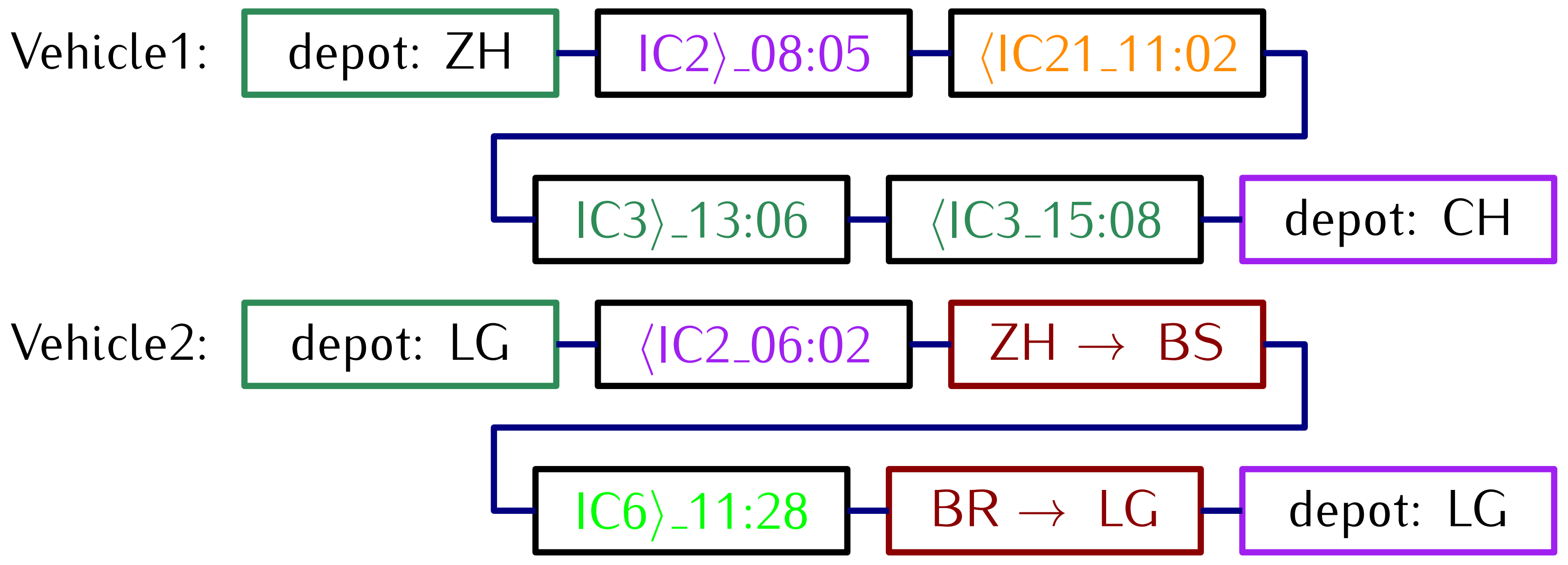
input:

- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L\times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L\times L}$
- **vehicle type infos** passenger capacity, max formation length

output (schedule):

- a list of tours starting and ending at a depot
- cyclic (every day the same schedule)
- no maintenance yet





Phase 1 – Task

input:

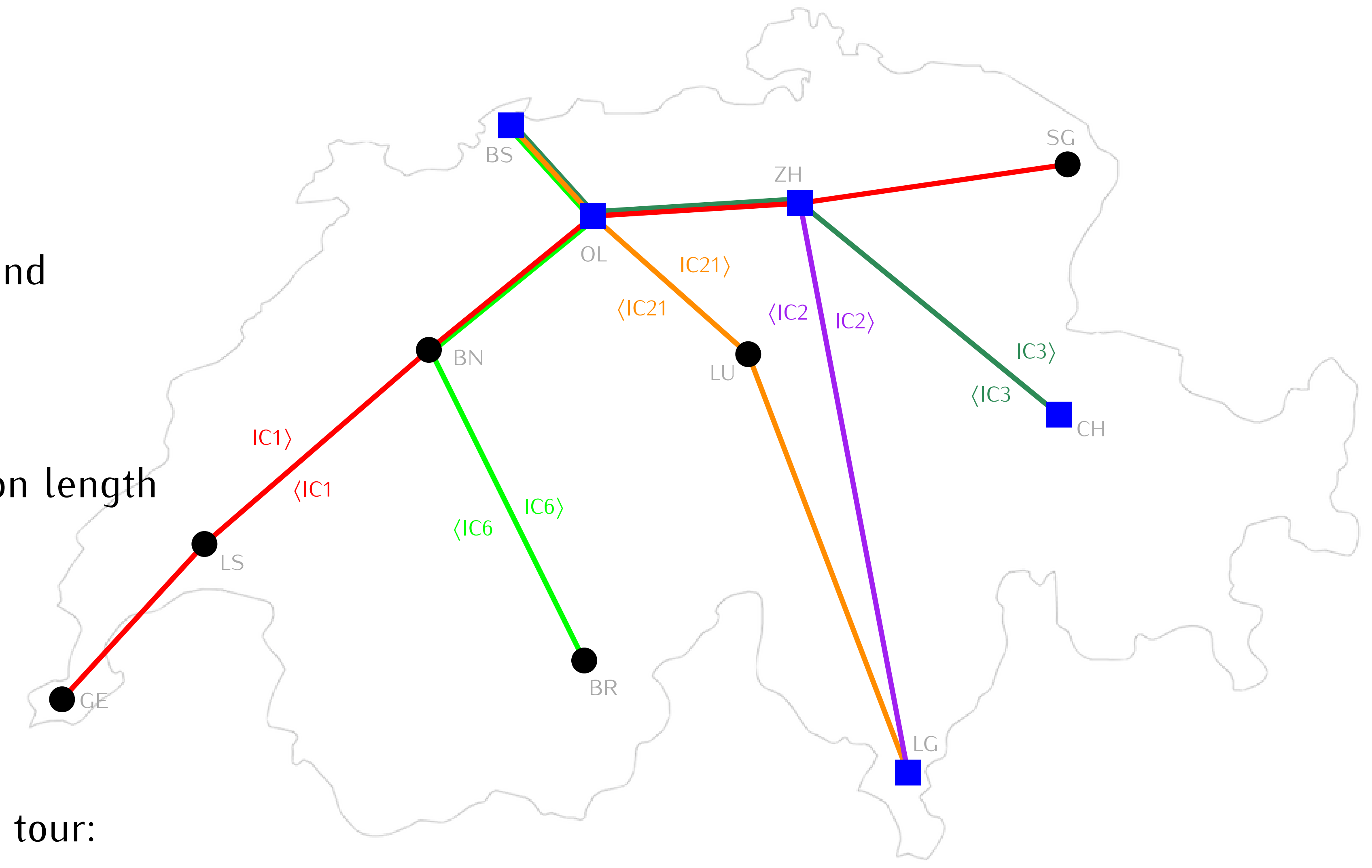
- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L \times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L \times L}$
- **vehicle type infos** passenger capacity, max formation length

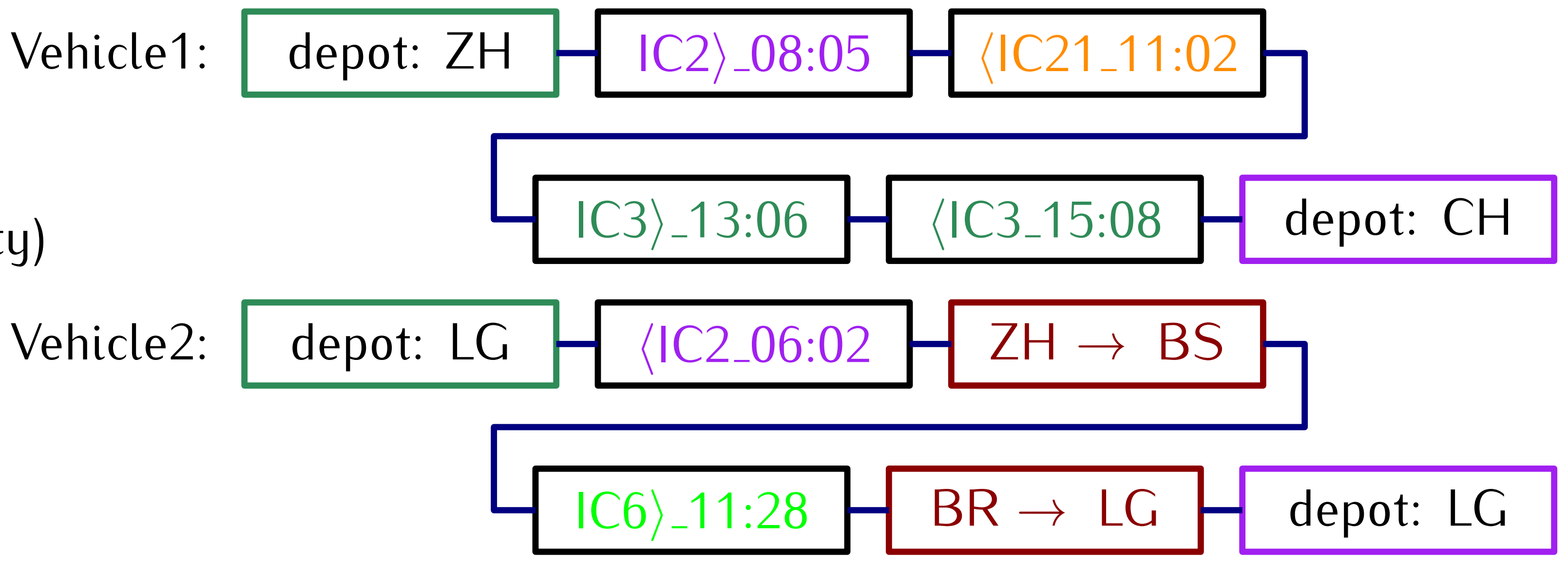
output (schedule):

- a list of tours starting and ending at a depot
- cyclic (every day the same schedule)
- no maintenance yet

feasibility:

1. for each pair of consecutive activities a_1, a_2 in same tour:
   - a_1.endLocation $=$ a_2.startLocation
   - a_1.endTime $\leq$ a_2.startTime $-$gapTime
2. for each depot:
   - # spawning vehicles $\leq$ depots capacity
   - # spawning vehicles $=$ # de-spawning vehicles (cyclicity)

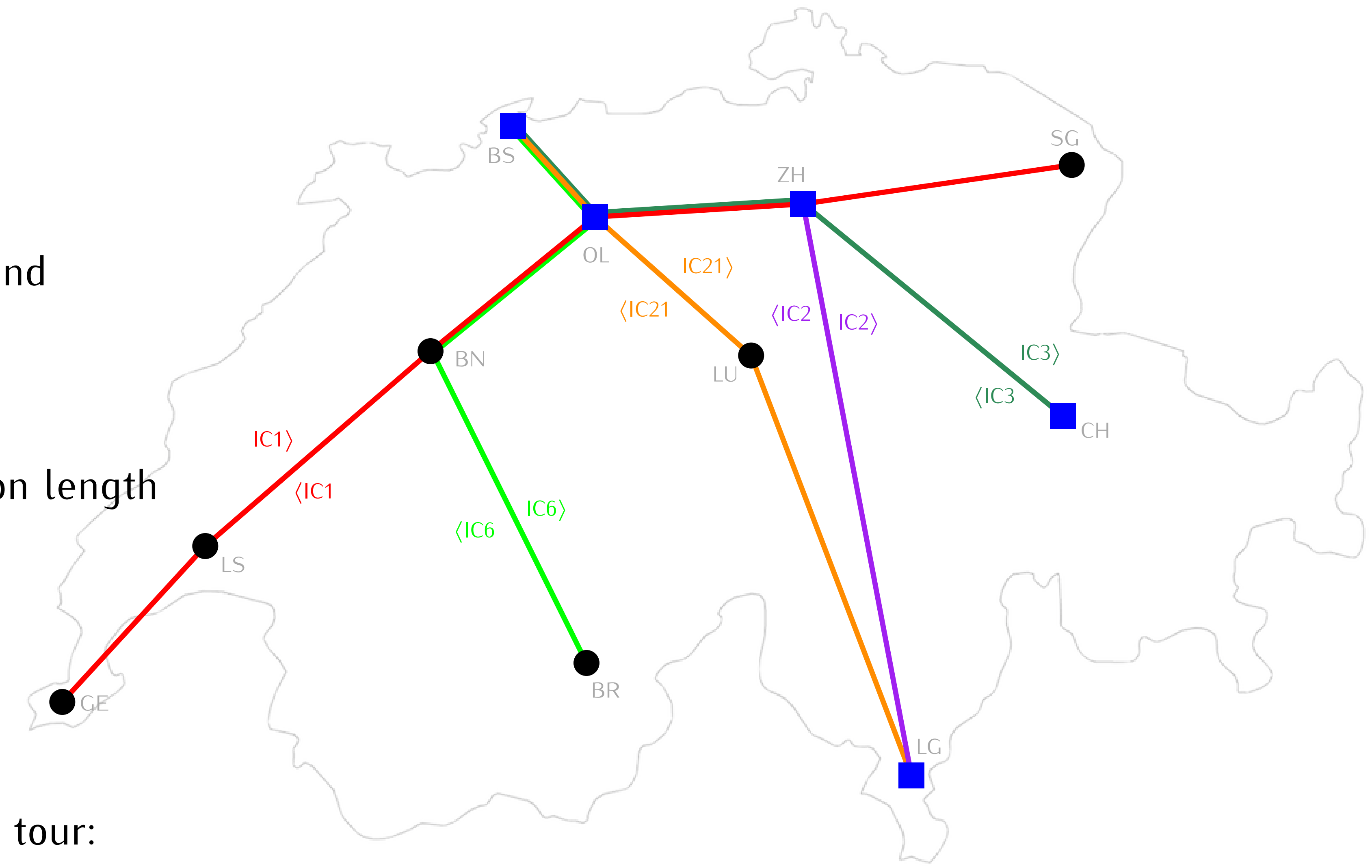




Phase 1 – Task

input:

- **locations** (finite set L)
- **depots:** location, capacity
- **routes** origin, destination, distance, duration
- **service trips** route, departure time, passenger demand
- **dead head trips:**
  - distance matrix $\mathbb{R}_{\geq 0}^{L \times L}$
  - travel-time matrix $\mathbb{R}_{\geq 0}^{L \times L}$
- **vehicle type infos** passenger capacity, max formation length

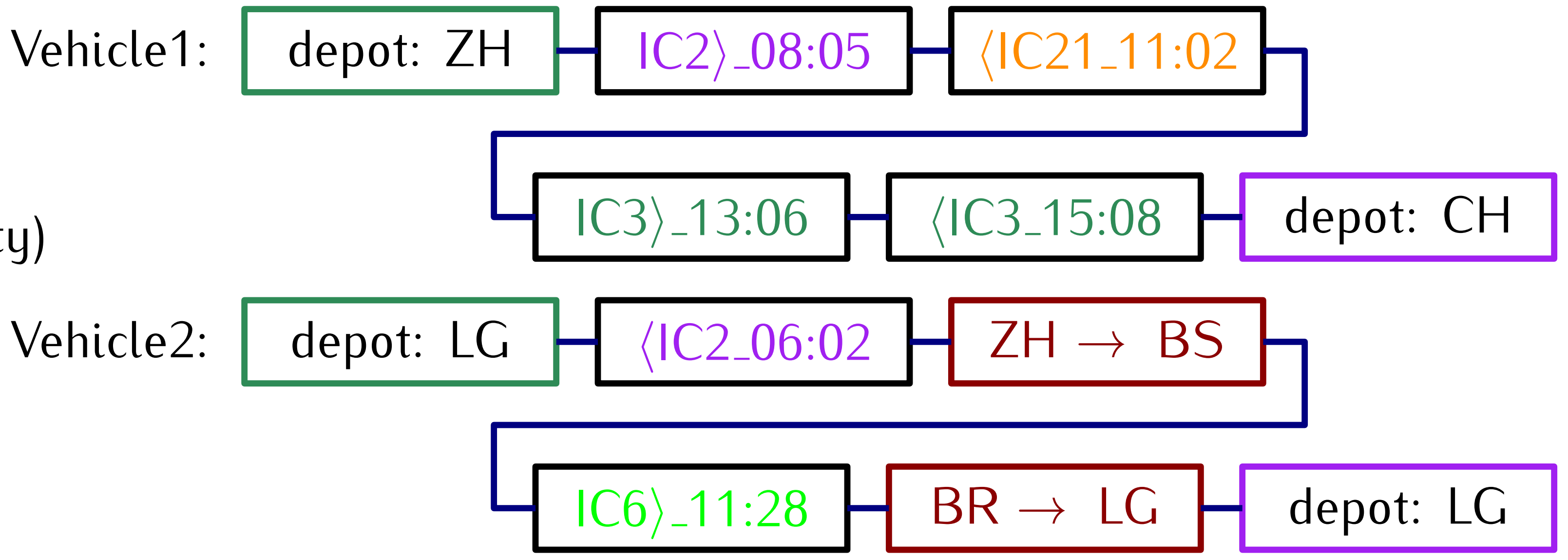


output (schedule):

- a list of tours starting and ending at a depot
- cyclic (every day the same schedule)
- no maintenance yet

feasibility:

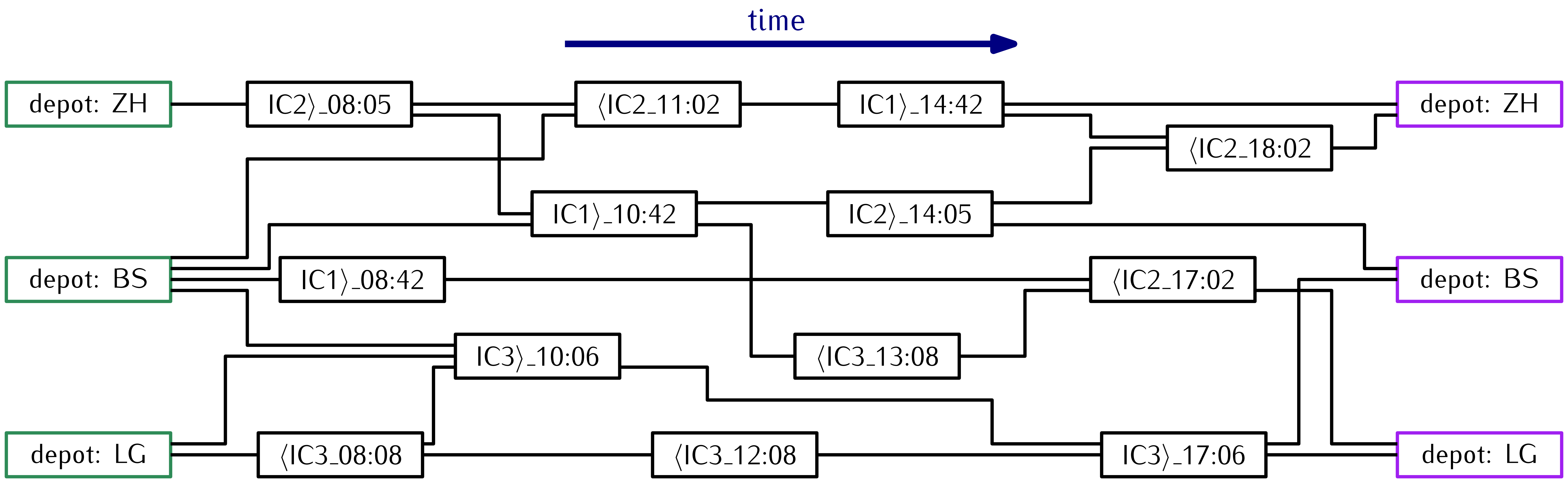
1. for each pair of consecutive activities a_1, a_2 in same tour:
   - a_1.endLocation $=$ a_2.startLocation
   - a_1.endTime $\leq$ a_2.startTime $-$gapTime
2. for each depot:
   - # spawning vehicles $\leq$ depots capacity
   - # spawning vehicles $=$ # de-spawning vehicles (cyclicity)



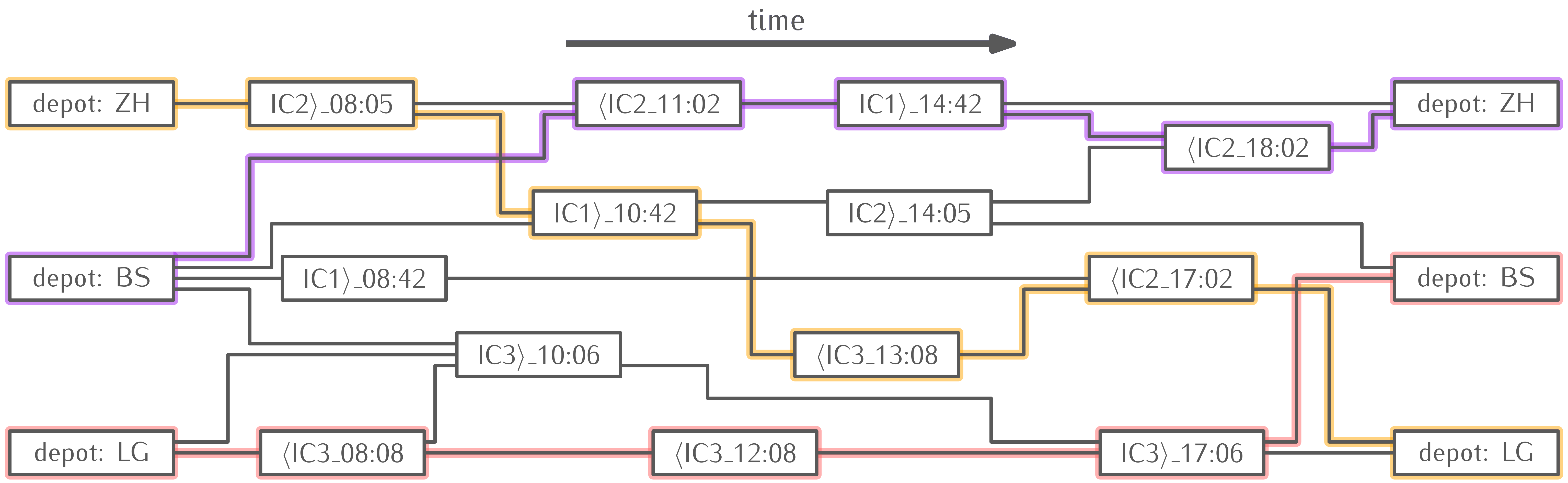
hierachical objective: (to be minimized from top to bottom)

1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Phase 1 – Modeling

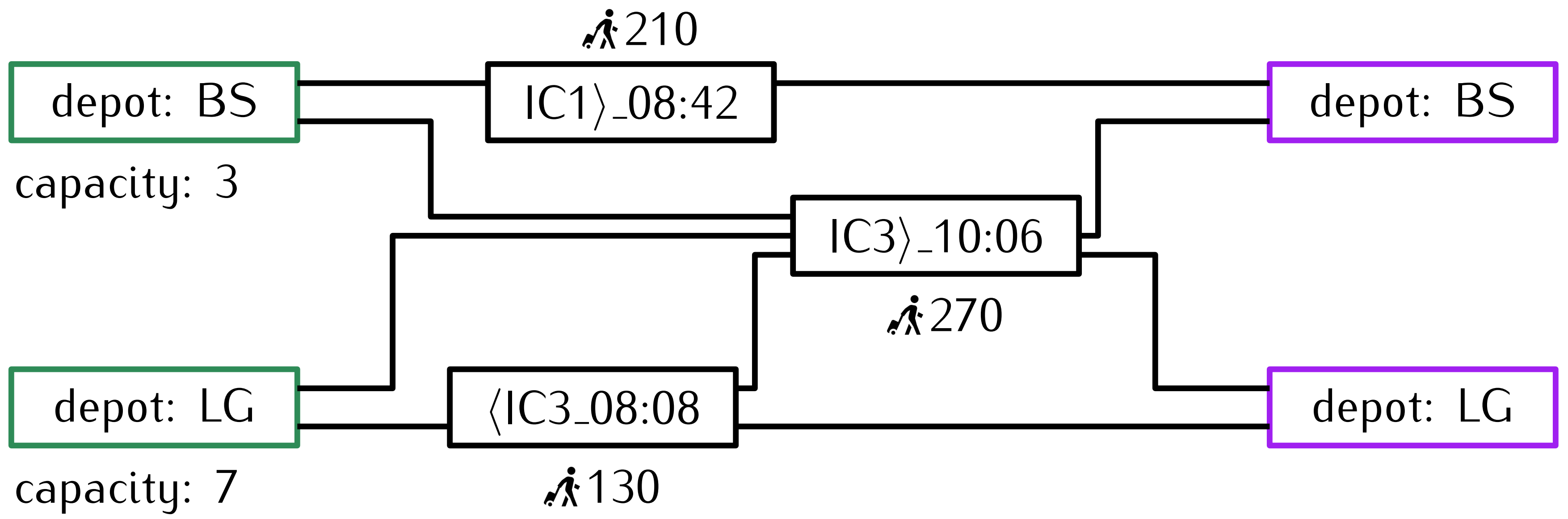


Phase 1 – Modeling



Phase 1 – Algorithm

Model



train info:

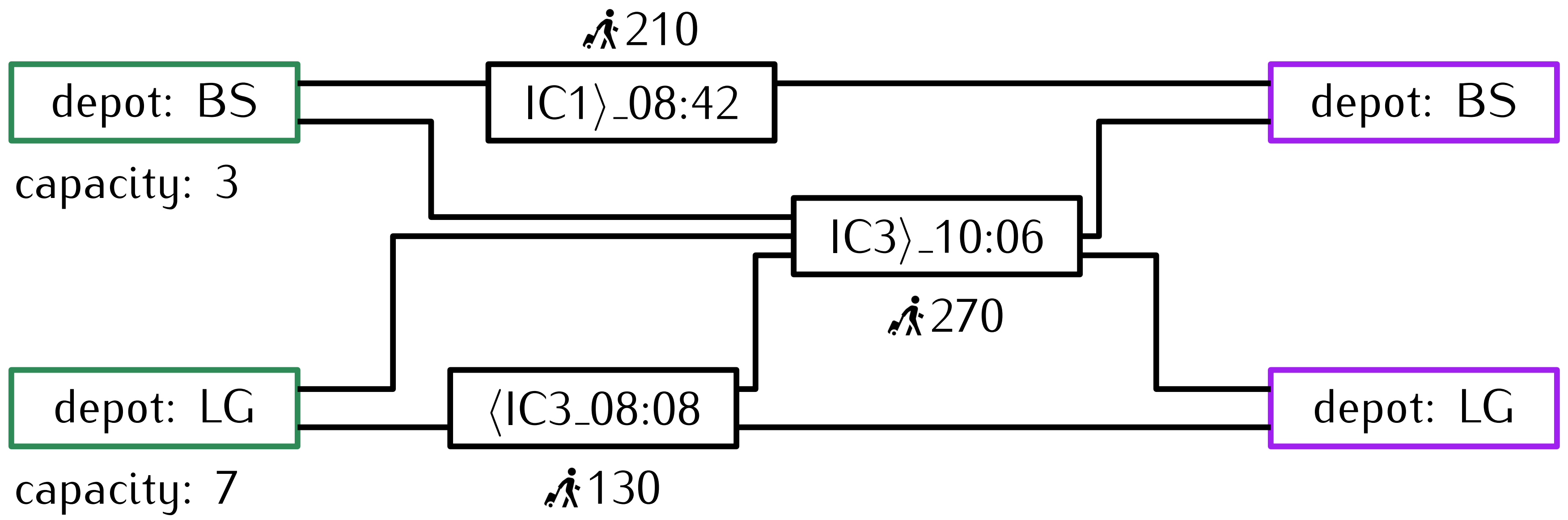
- passenger capacity per train: 200
- maximal vehicle in formation: 5

hierachical objective:

1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Phase 1 - Algorithm

Model



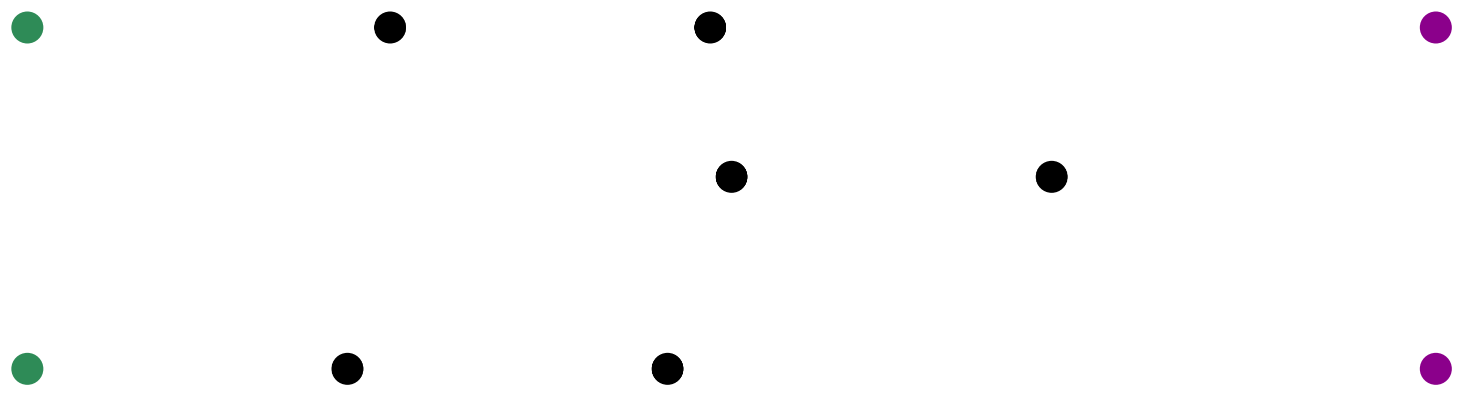
train info:

- passenger capacity per train: 200
- maximal vehicle in formation: 5

hierachical objective:

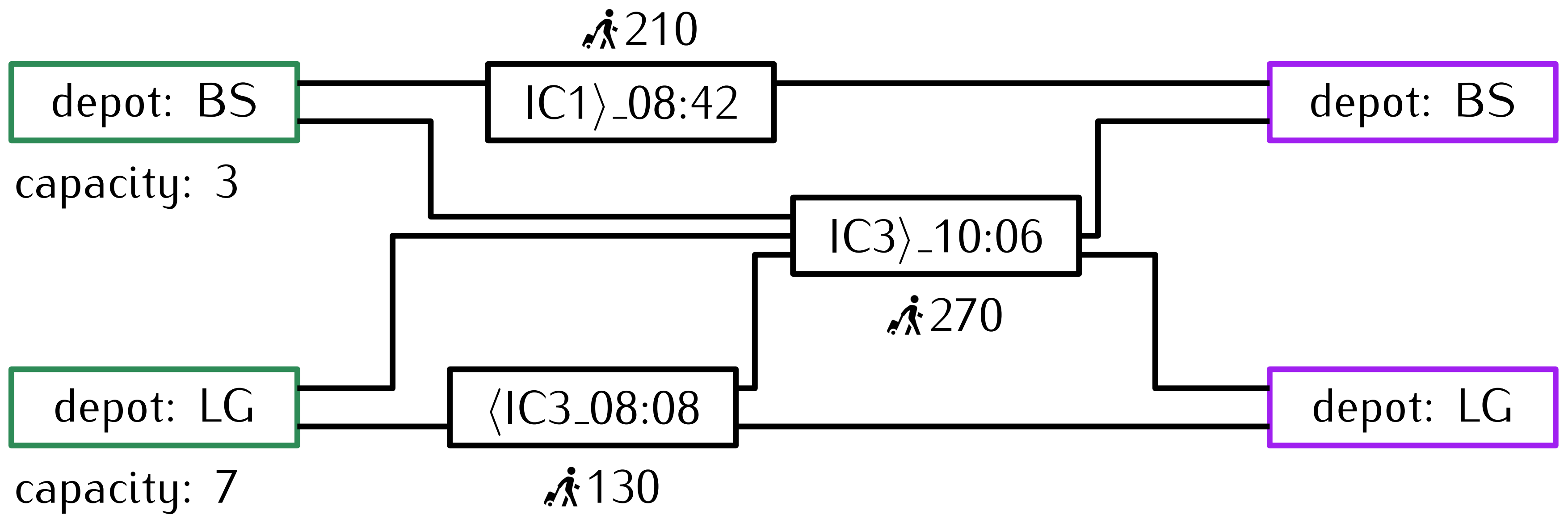
1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Min-Cost-Circulation



Phase 1 - Algorithm

Model



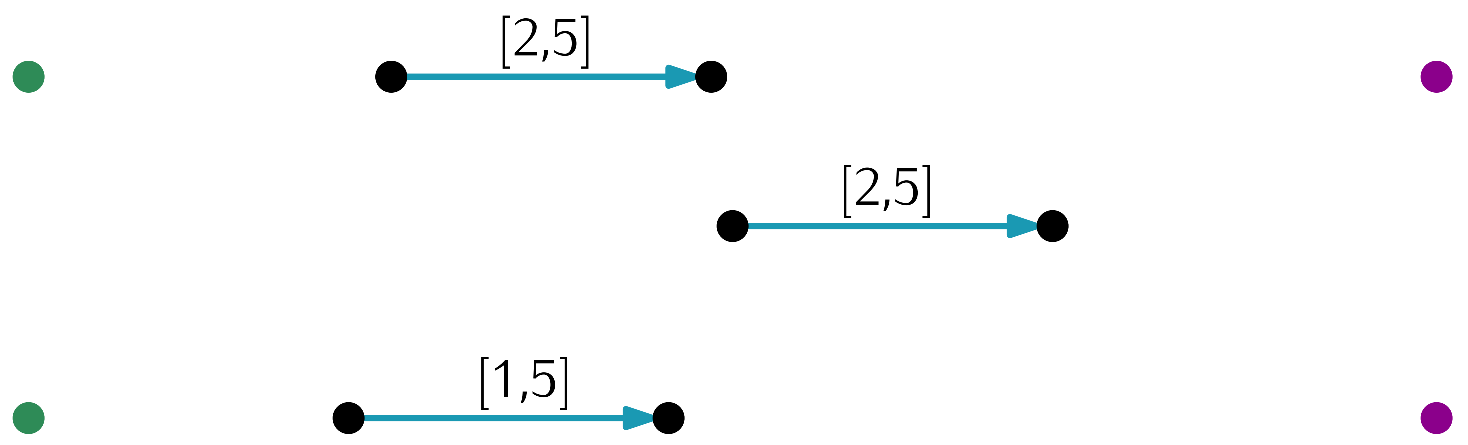
train info:

- passenger capacity per train: 200
- maximal vehicle in formation: 5

hierachical objective:

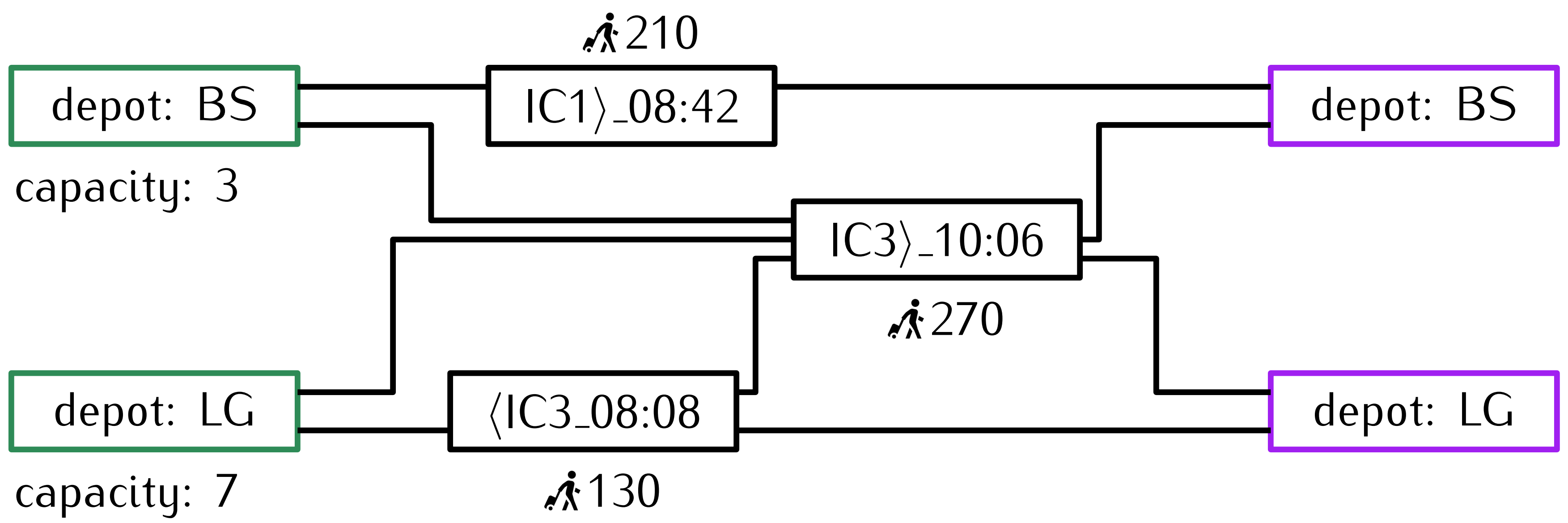
1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Min-Cost-Circulation



Phase 1 - Algorithm

Model



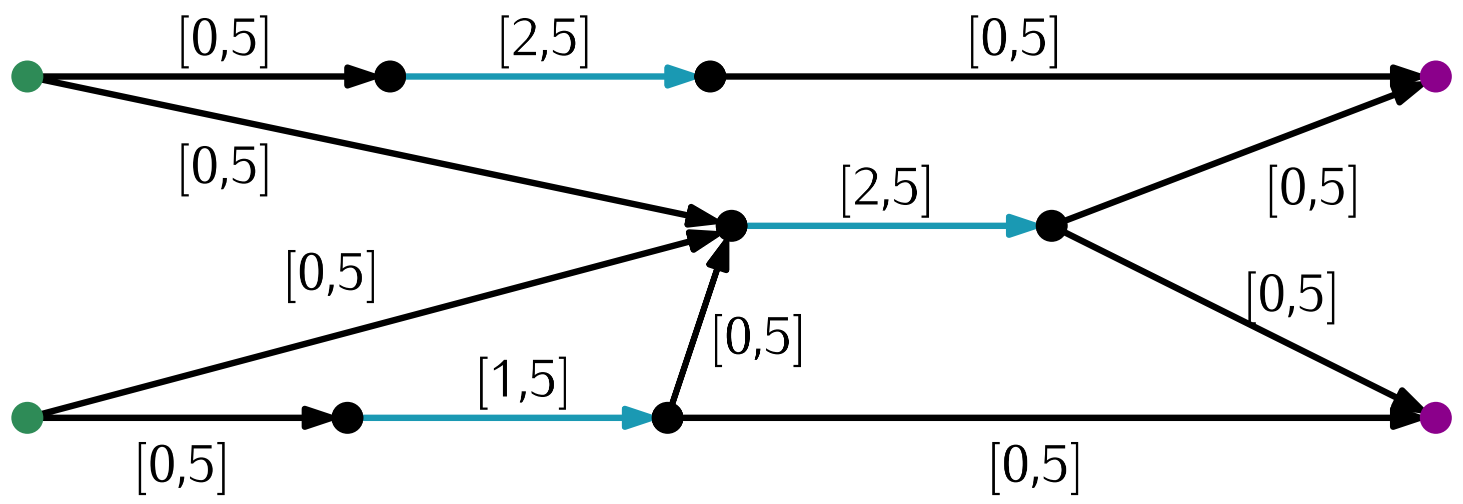
train info:

- passenger capacity per train: 200
- maximal vehicle in formation: 5

hierachical objective:

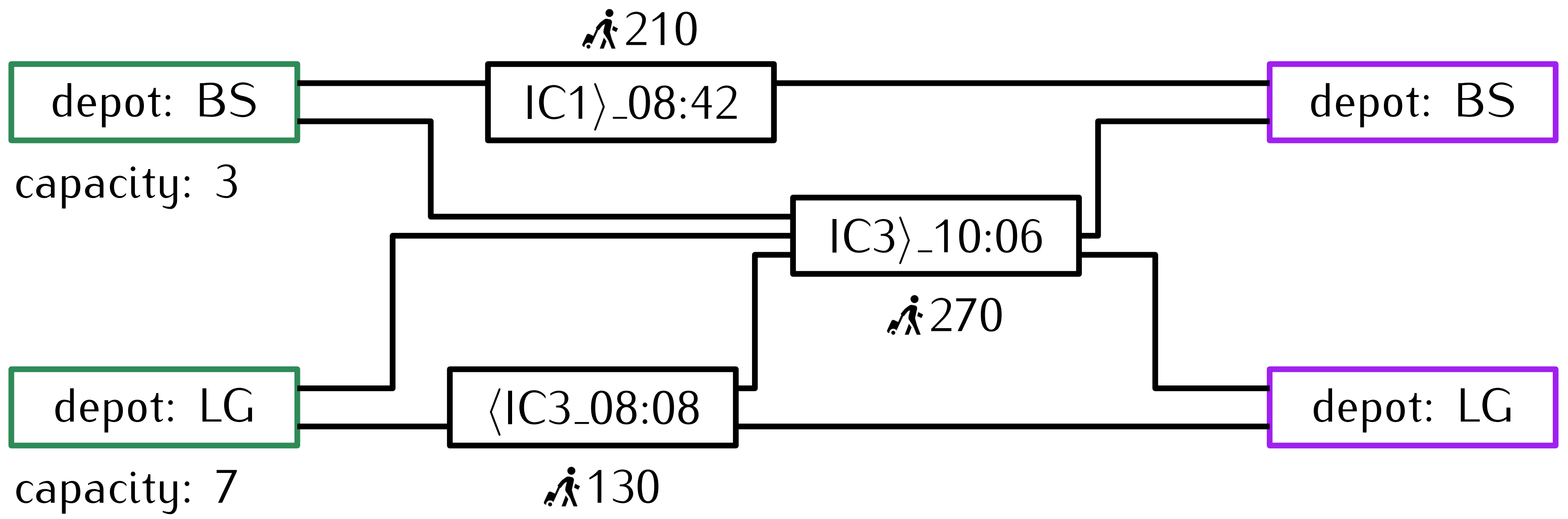
1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Min-Cost-Circulation



Phase 1 - Algorithm

Model



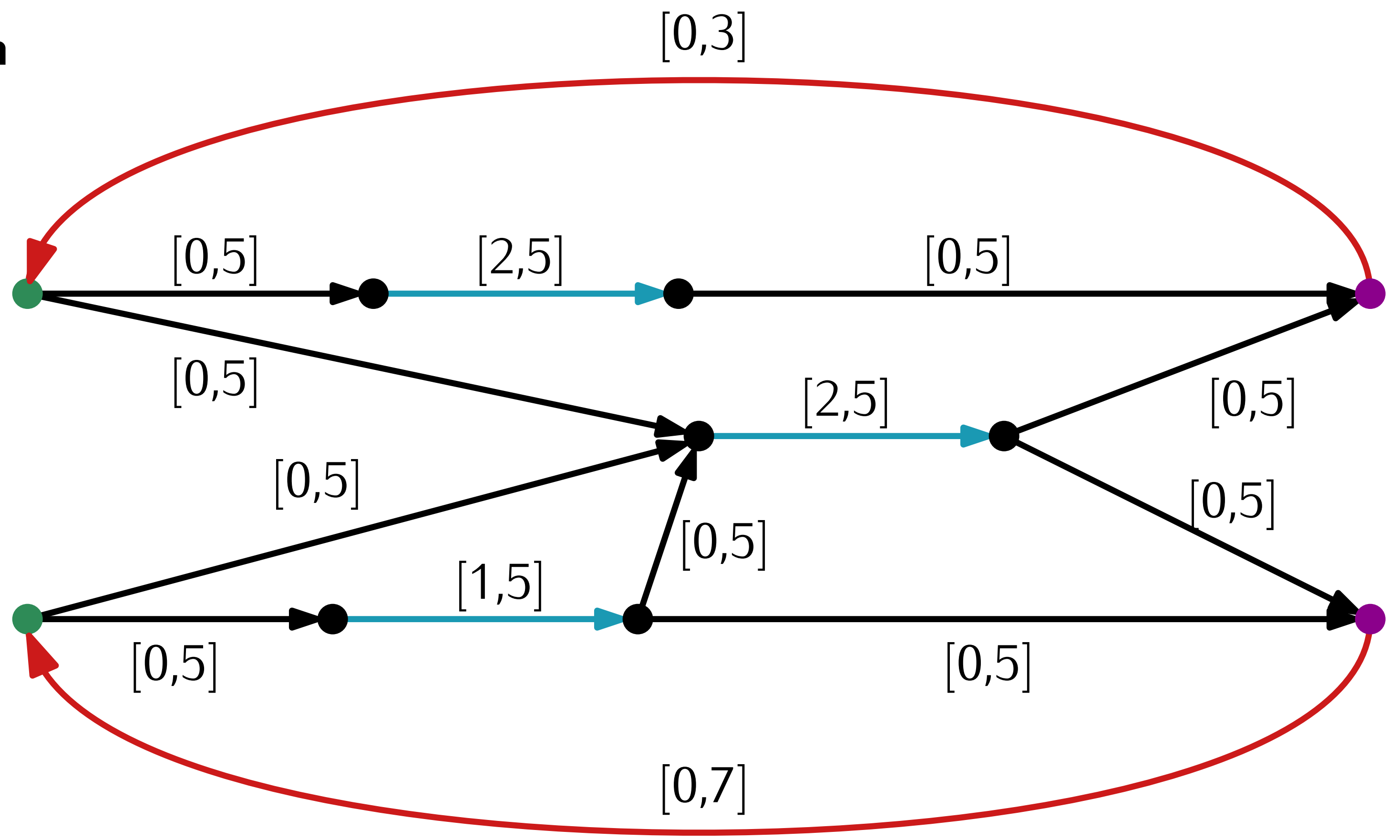
train info:

- passenger capacity per train: 200
- maximal vehicle in formation: 5

hierachical objective:

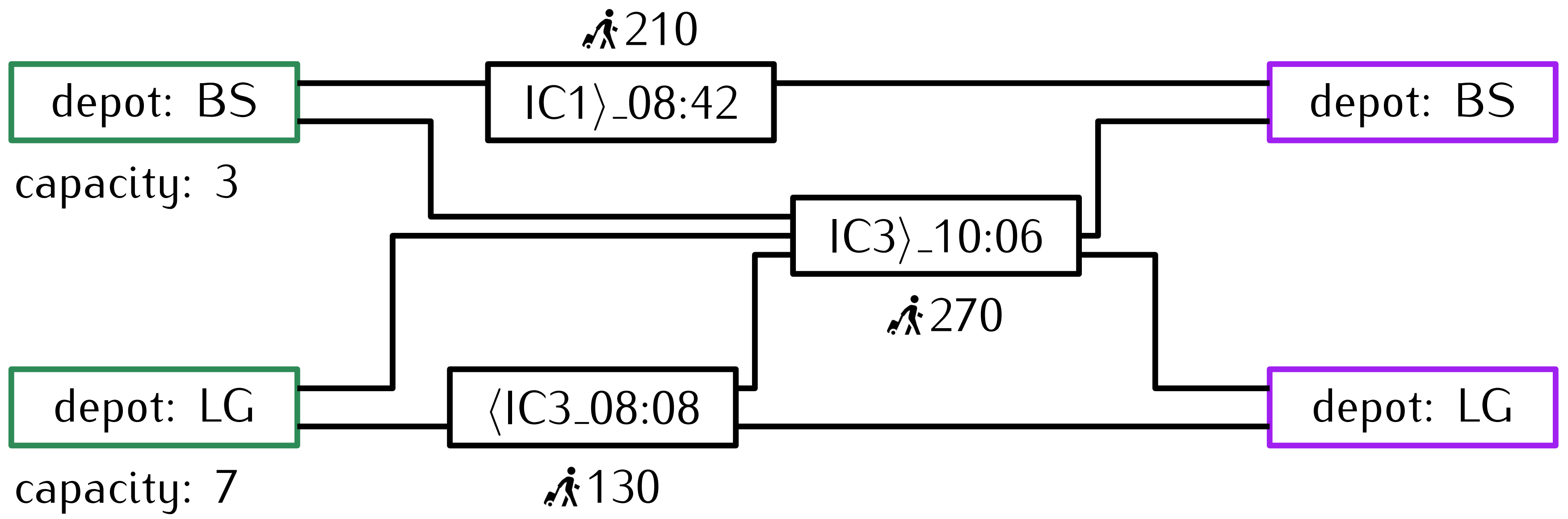
1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Min-Cost-Circulation



Phase 1 - Algorithm

Model



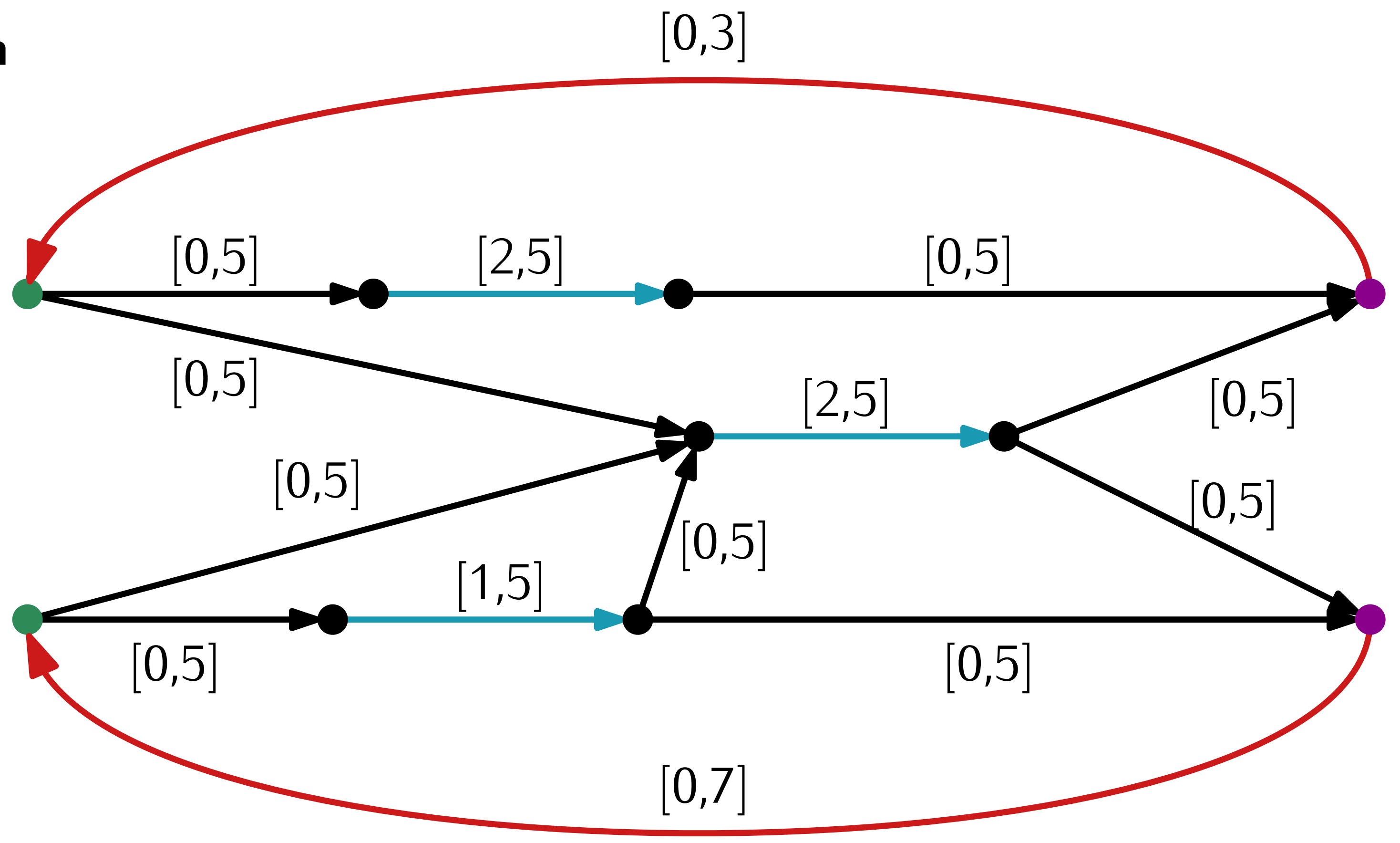
train info:

- passenger capacity per train: 200
- maximal vehicle in formation: 5

hierachical objective:

1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Min-Cost-Circulation

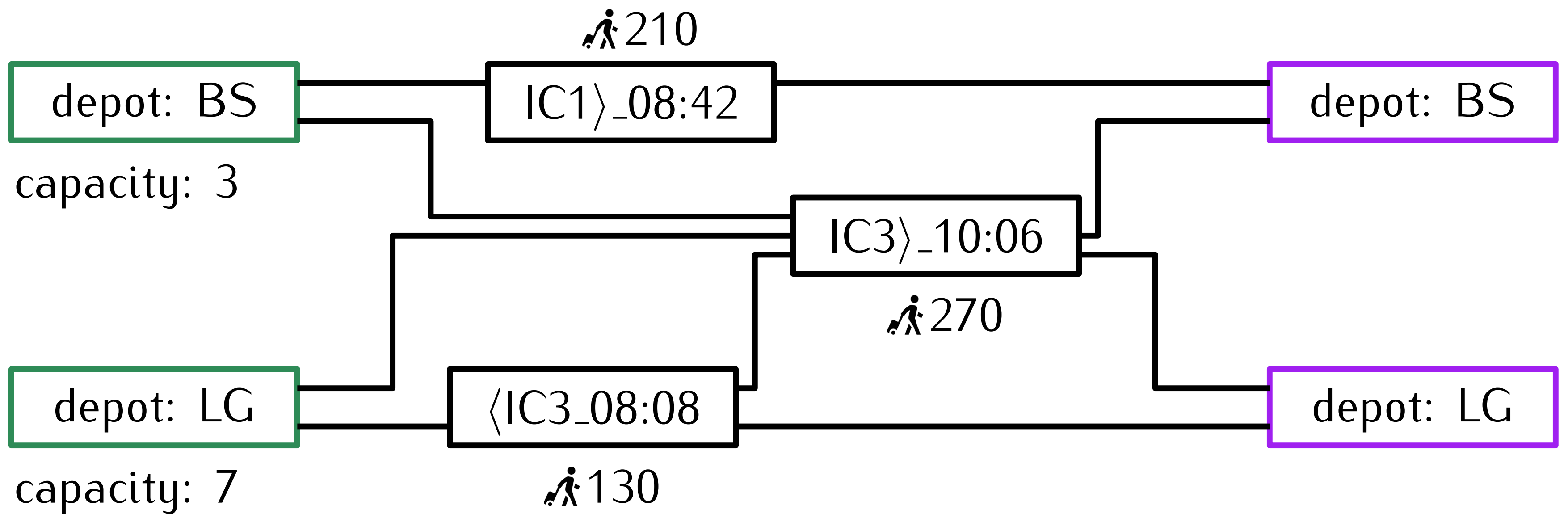


costs:

1. forward arcs: distance traveled
2. depot arcs: sum of all distances

Phase 1 - Algorithm

Model



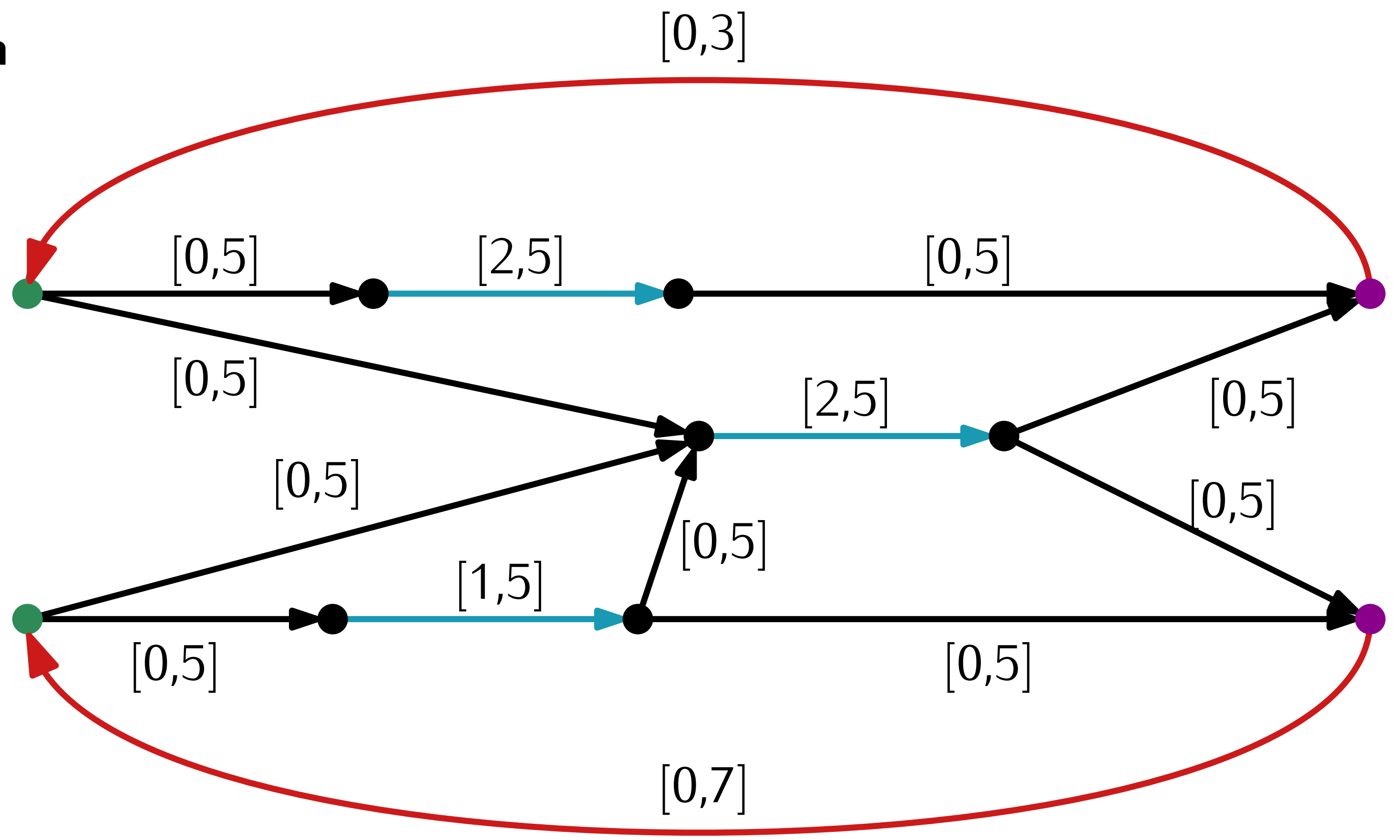
train info:

- passenger capacity per train: 200
- maximal vehicle in formation: 5

hierachical objective:

1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Min-Cost-Circulation



costs:

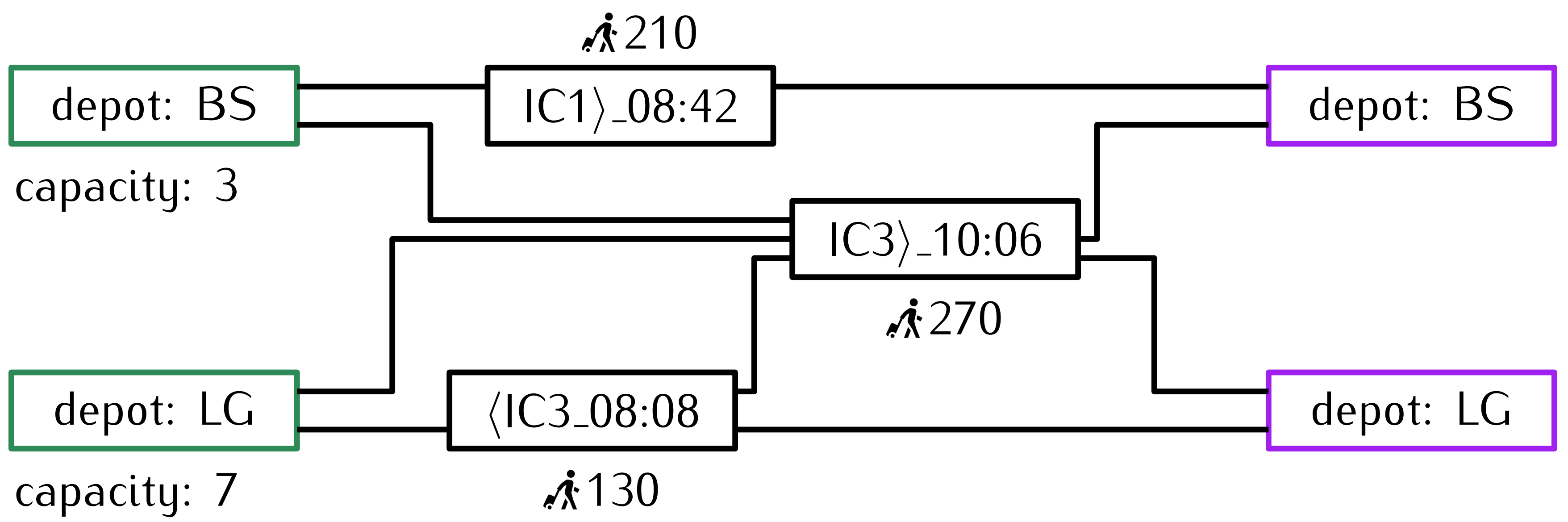
1. forward arcs: distance traveled
2. depot arcs: sum of all distances

algorithm:

1. create flow network
2. use network simplex (rs_graph crate) to compute min-cost-circulation
3. do arbitrary path-decomposition

Phase 1 – Algorithm

Model



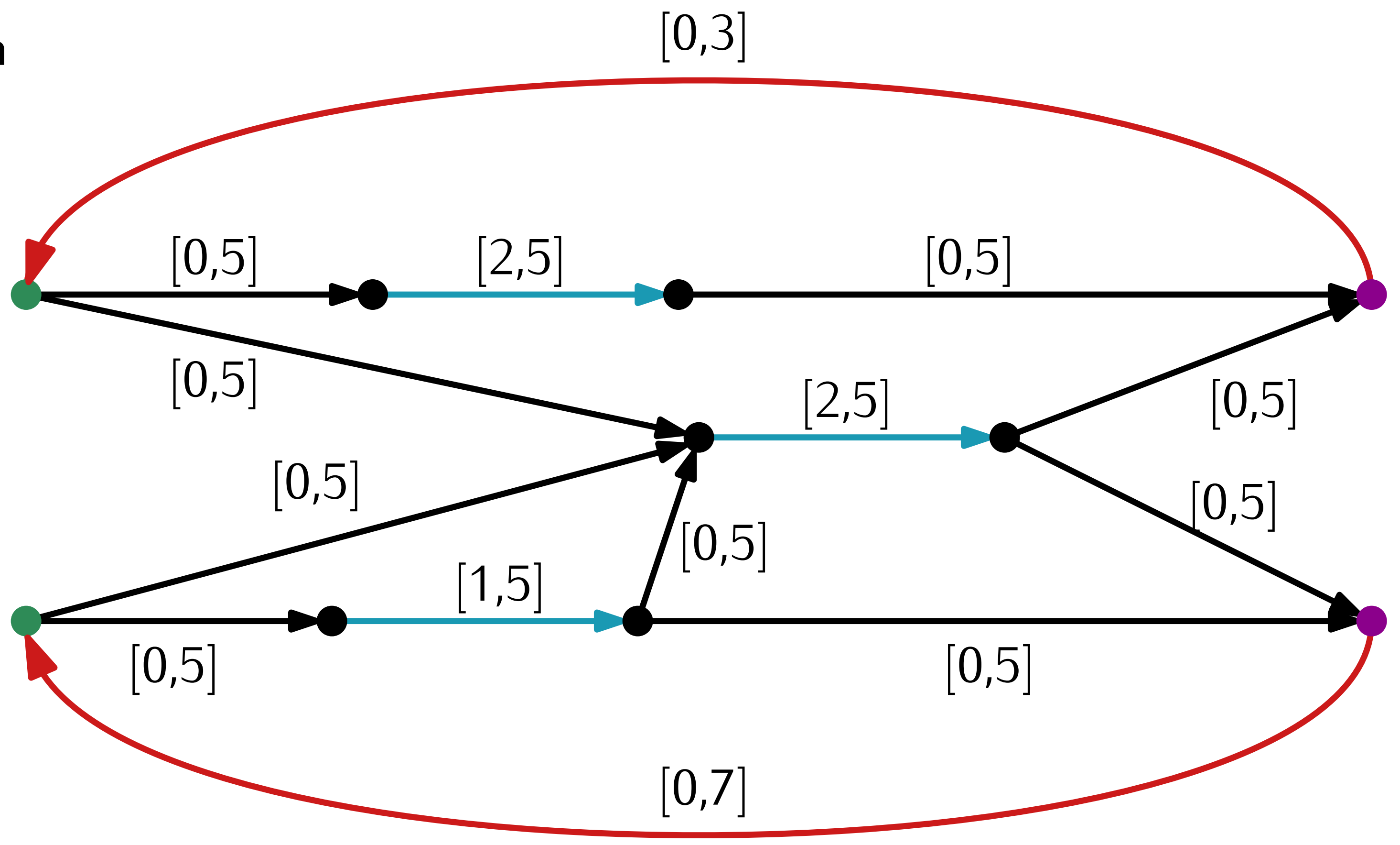
train info:

- passenger capacity per train: 200
- maximal vehicle in formation: 5

hierachical objective:

1. # unserved passengers
2. # number of vehicles
3. total distance traveled

Min-Cost-Circulation



costs:

1. forward arcs: distance traveled
2. depot arcs: sum of all distances

algorithm:

1. create flow network
2. use network simplex (rs_graph crate) to compute min-cost-circulation
3. do arbitrary path-decomposition

running time

- 1200 service trips
- 120 vehicle
- Lenovo X1 Yoga

⇒ **0.5 seconds**

Phase 2
Model Extensions

Phase 2 – New Aspects

seated / standing passengers

- assumption: passengers travelling $<$ 15 minutes can stand (no seat needed)

Phase 2 – New Aspects

seated / standing passengers

- assumption: passengers travelling $<$ 15 minutes can stand (no seat needed)

strengthen / weaken train formation during a trip

- split service trips into segments
- within a segment: no coupling allowed
- between segments: vehicles can be (de-)coupled

Phase 2 - New Aspects

seated / standing passengers

- assumption: passengers travelling $<$ 15 minutes can stand (no seat needed)

strengthen / weaken train formation during a trip

- split service trips into segments
- within a segment: no coupling allowed
- between segments: vehicles can be (de-)coupled

"hitch-hiking"

- vehicles can be towed unused on a service trips (saves staff cost)

Phase 2 - New Aspects

seated / standing passengers

- assumption: passengers travelling $<$ 15 minutes can stand (no seat needed)

strengthen / weaken train formation during a trip

- split service trips into segments
- within a segment: no coupling allowed
- between segments: vehicles can be (de-)coupled

"hitch-hiking"

- vehicles can be towed unused on a service trips (saves staff cost)

vehicle types

- only vehicles of the same type can be coupled
- service trips are served by exactly one type

Phase 2 - New Aspects

seated / standing passengers

- assumption: passengers travelling < 15 minutes can stand (no seat needed)

strengthen / weaken train formation during a trip

- split service trips into segments
- within a segment: no coupling allowed
- between segments: vehicles can be (de-)coupled

"hitch-hiking"

- vehicles can be towed unused on a service trips (saves staff cost)

vehicle types

- only vehicles of the same type can be coupled
- service trips are served by exactly one type

maintenance

- maintenance stations with multiple maintenance slots
- maintenance slots can be used by each type
- vehicles must go to maintenance every 15 000 km ($\sim$ every 20 days)
- schedule is a single day repeated each day
- → map vehicles arriving at a depot in the evening to vehicles departure early in the morning (on the next day)

$$\begin{tikzcd}
\text{vehicle 1} \arrow[dr] & \text{vehicle 1} \\
\text{vehicle 2} \arrow[ur] & \text{vehicle 2} \\
\text{vehicle 3} \arrow[r] & \text{vehicle 3}
\end{tikzcd}$$

Phase 2 - New Aspects

seated / standing passengers

- assumption: passengers travelling $<$ 15 minutes can stand (no seat needed)

strengthen / weaken train formation during a trip

- split service trips into segments
- within a segment: no coupling allowed
- between segments: vehicles can be (de-)coupled

"hitch-hiking"

- vehicles can be towed unused on a service trips (saves staff cost)

vehicle types

- only vehicles of the same type can be coupled
- service trips are served by exactly one type

maintenance

- maintenance stations with multiple maintenance slots
- maintenance slots can be used by each type
- vehicles must go to maintenance every 15 000 km ($\sim$ every 20 days)
- schedule is a single day repeated each day
- $\rightarrow$ map vehicles arriving at a depot in the evening to vehicles departure early in the morning (on the next day)

old hierachical objective:

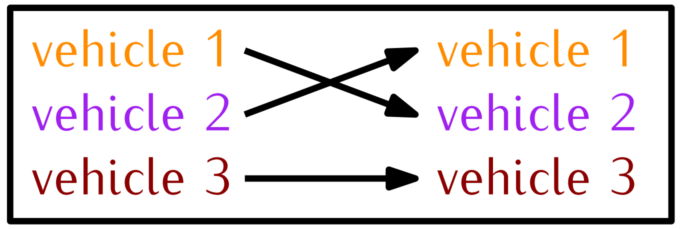
1. # unserved passengers
2. # number of vehicles
3. total distance traveled

new hierachical objective:

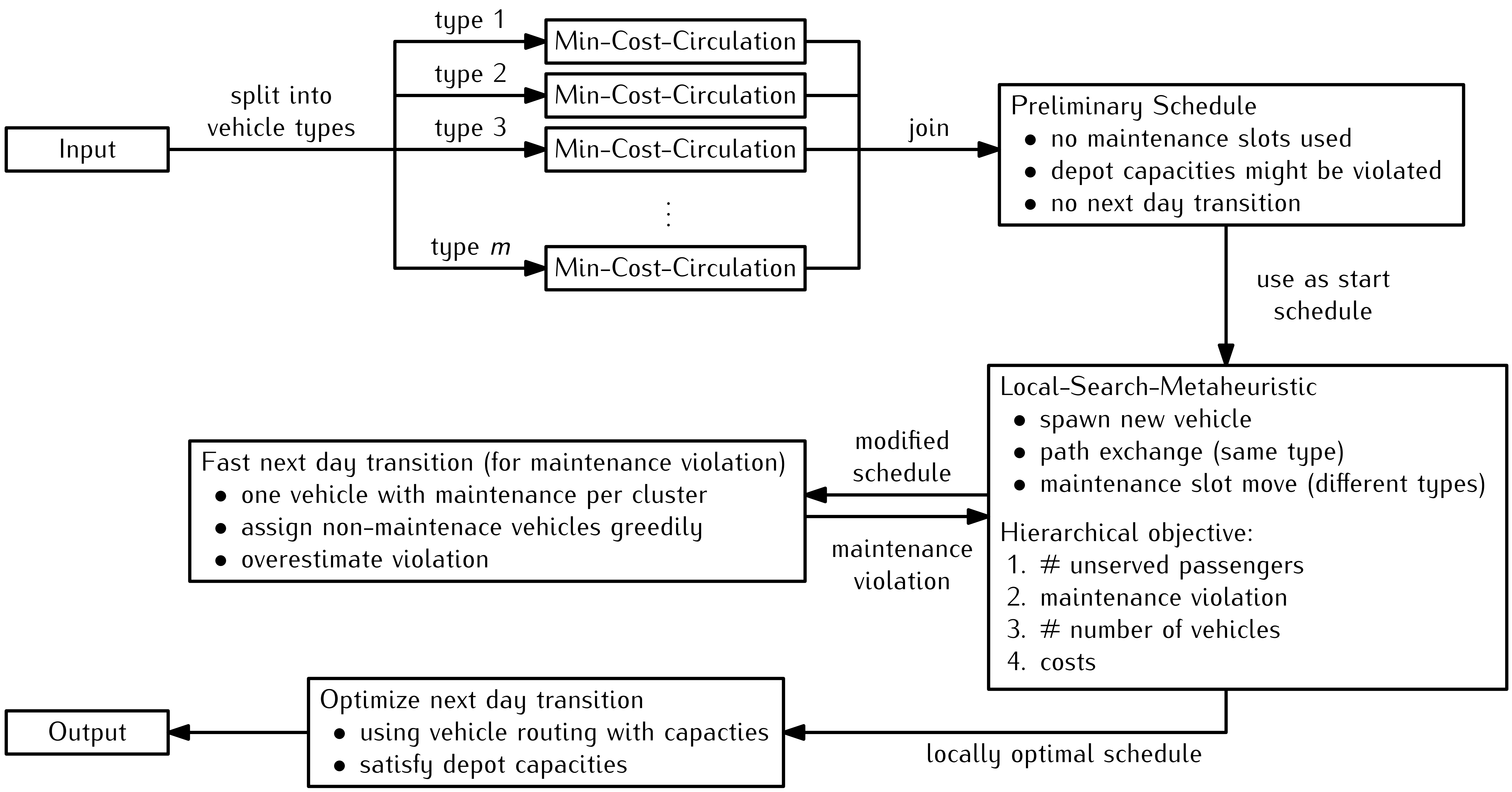
1. # unserved passengers
2. maintenance violation
3. # number of vehicles
4. costs

costs are a linear combination of

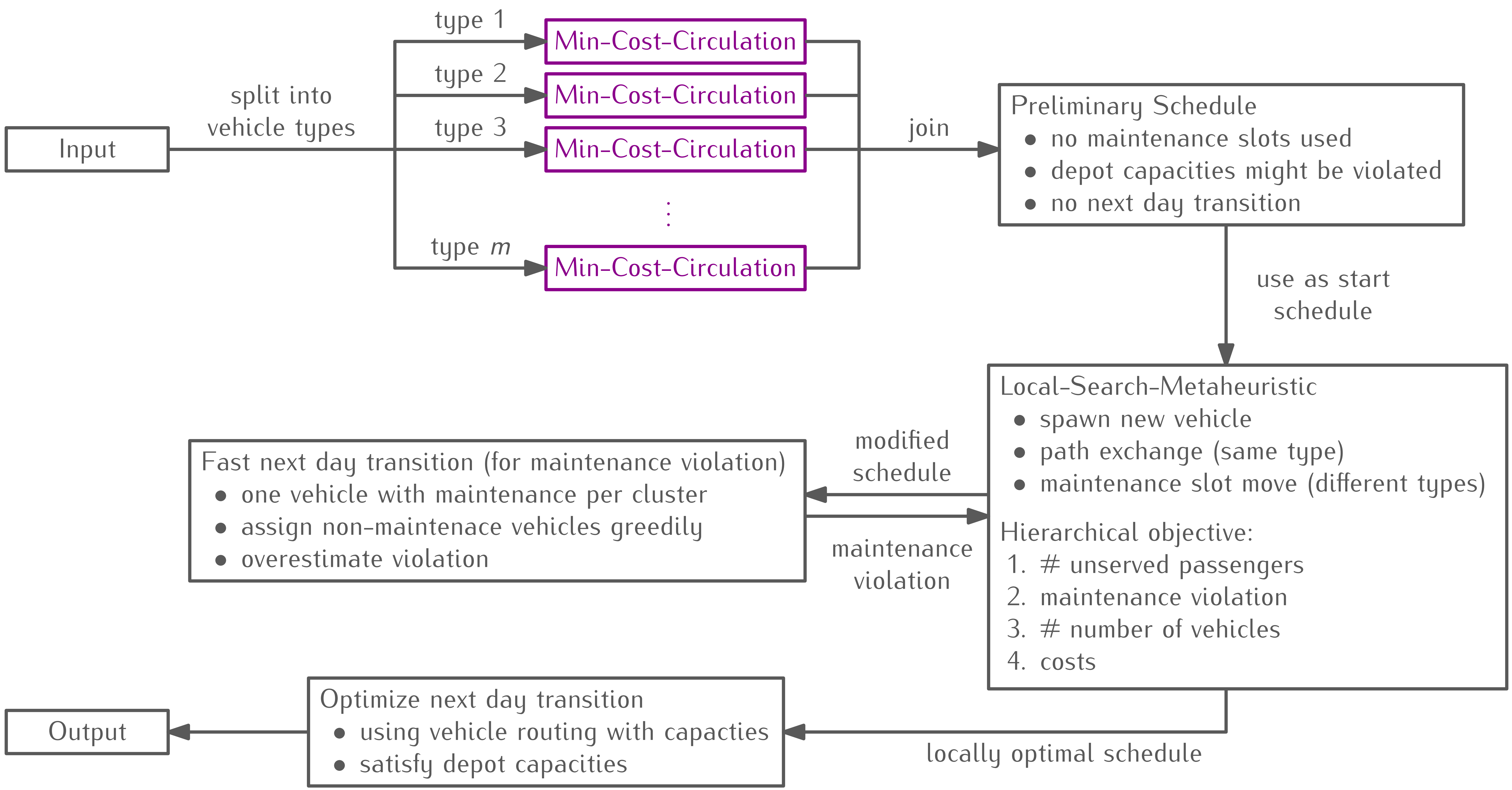
- total service trip duration
- total dead head trip duration
- mainteance duration
- idle duration
- staff cost (each train formation pays this only once)


$$\begin{tikzcd}
\text{vehicle 1} \arrow[dr] & \text{vehicle 1} \\
\text{vehicle 2} \arrow[ur] & \text{vehicle 2} \\
\text{vehicle 3} \arrow[r] & \text{vehicle 3}
\end{tikzcd}$$

Phase 2 – Algorithm Overview

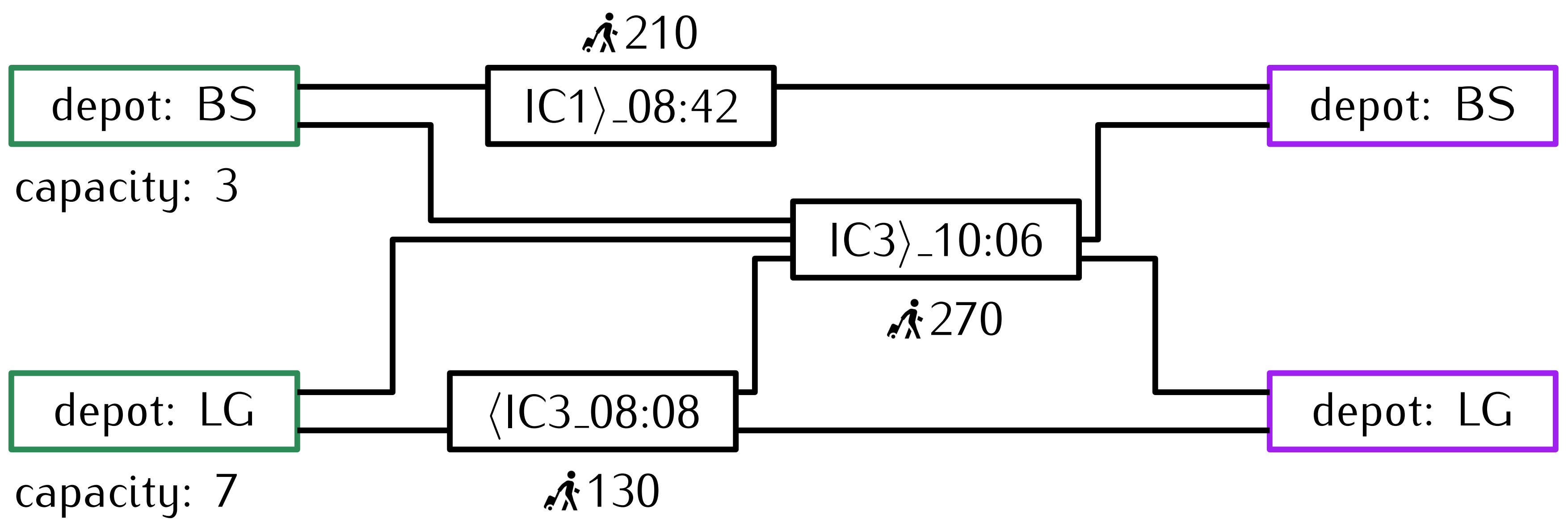


Phase 2 - Algorithm Overview

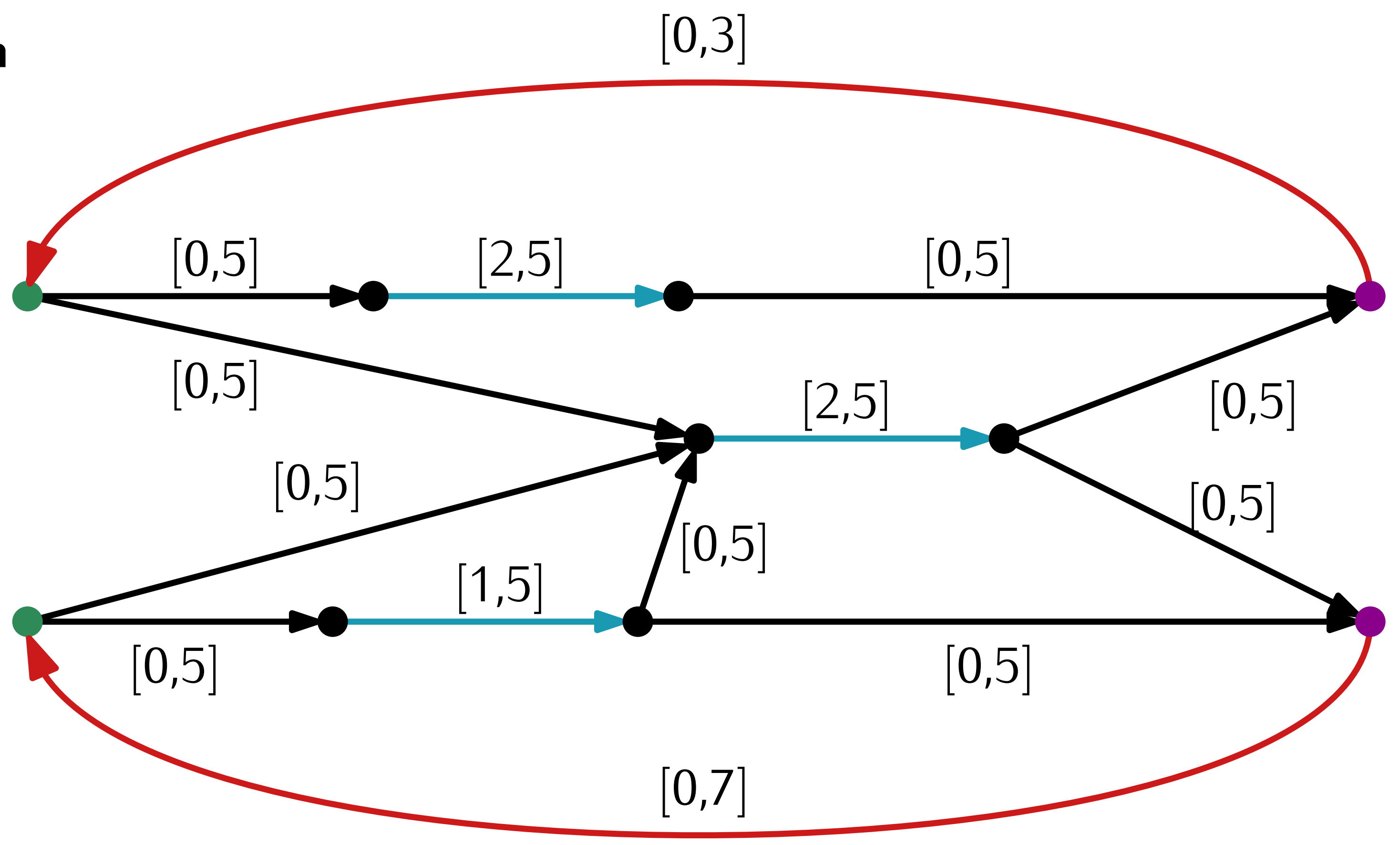


Min-Cost-Circulation

Model



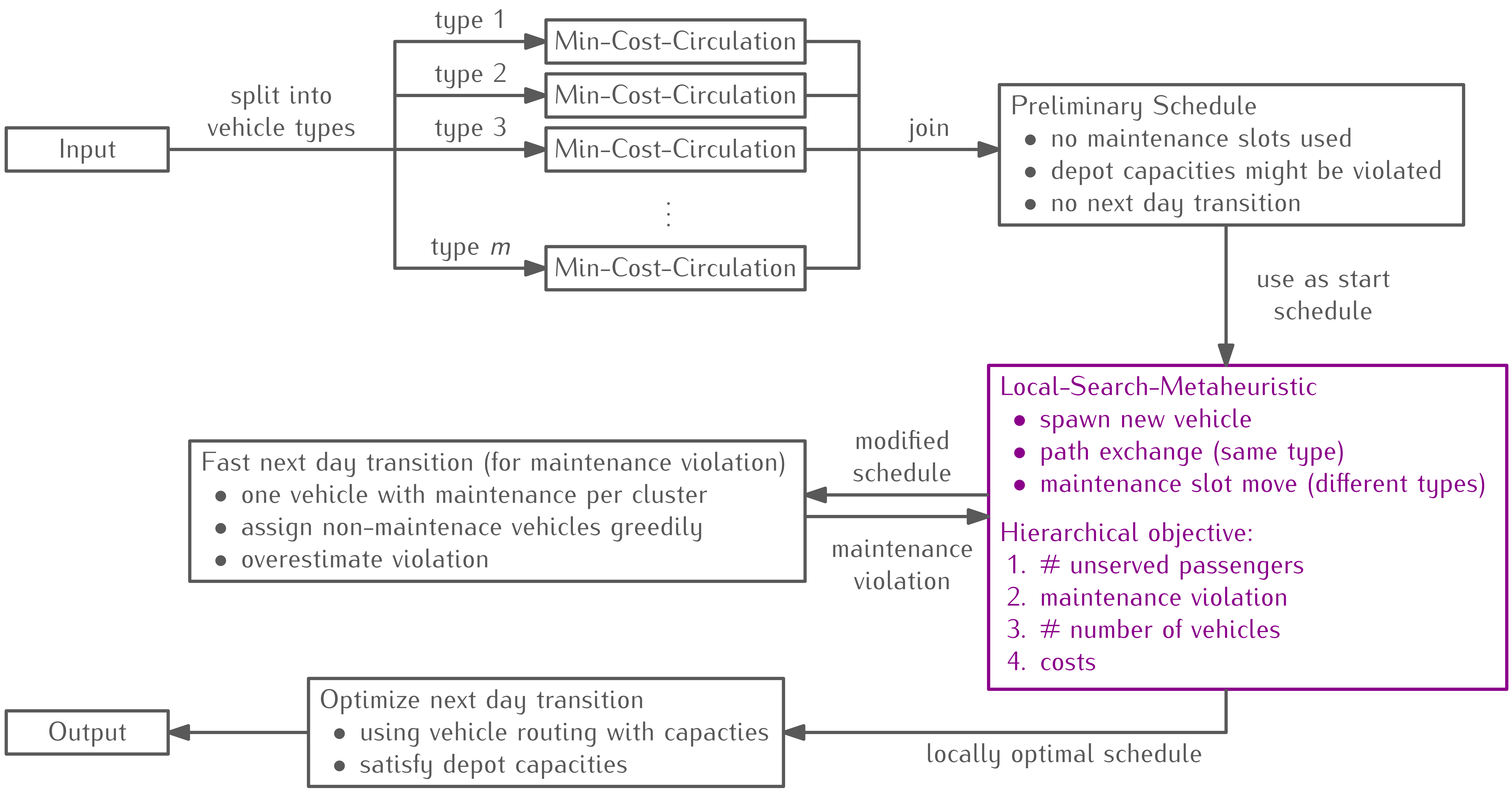
Min-Cost-Circulation



~~**costs:**~~
1. ~~forward arcs: distance traveled~~
2. ~~depot arcs: sum of all distances~~

costs:
1. forward arcs: new costs
2. depot arcs: sum of all costs

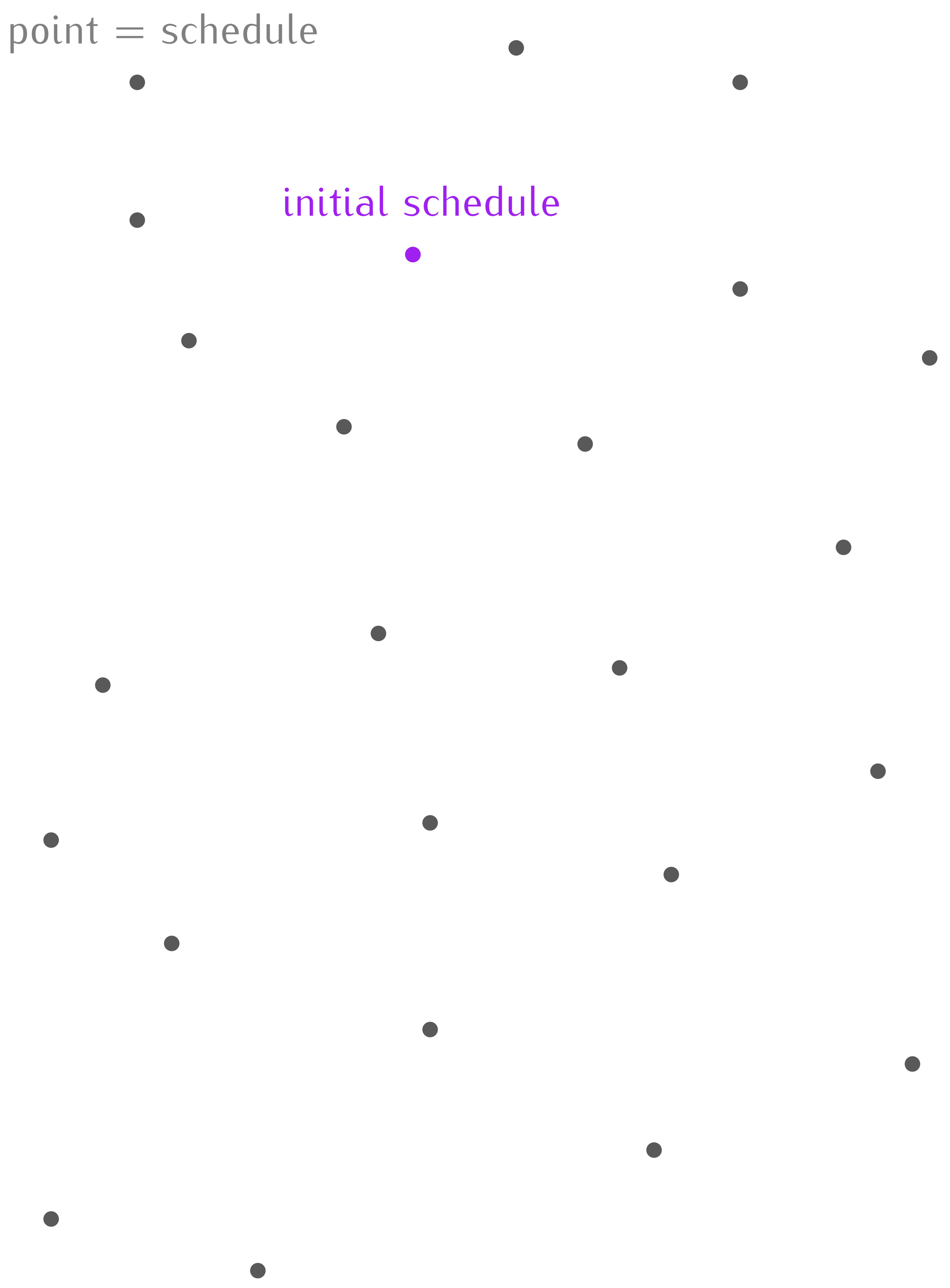
Phase 2 - Algorithm Overview



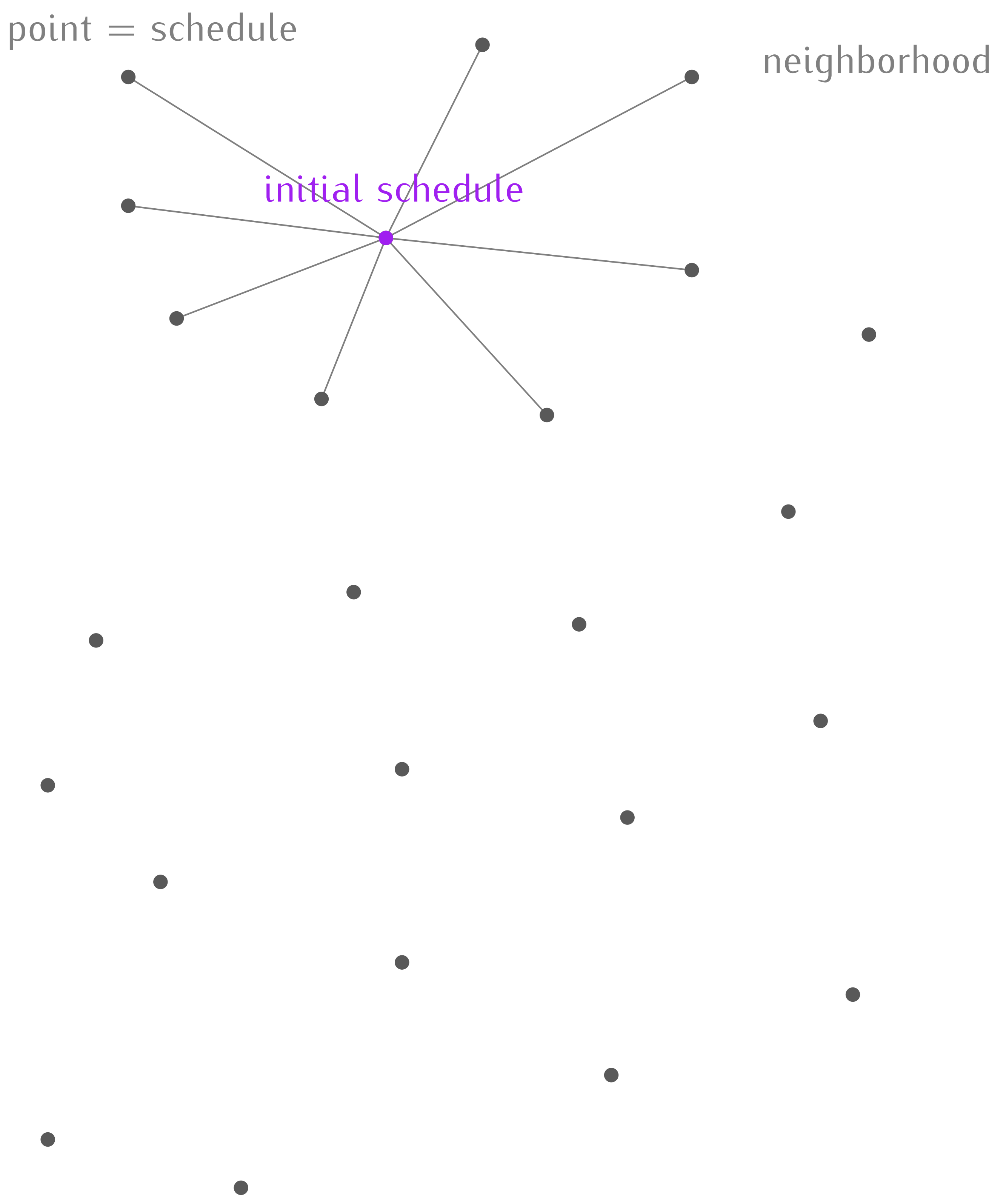
Local Search

point = schedule

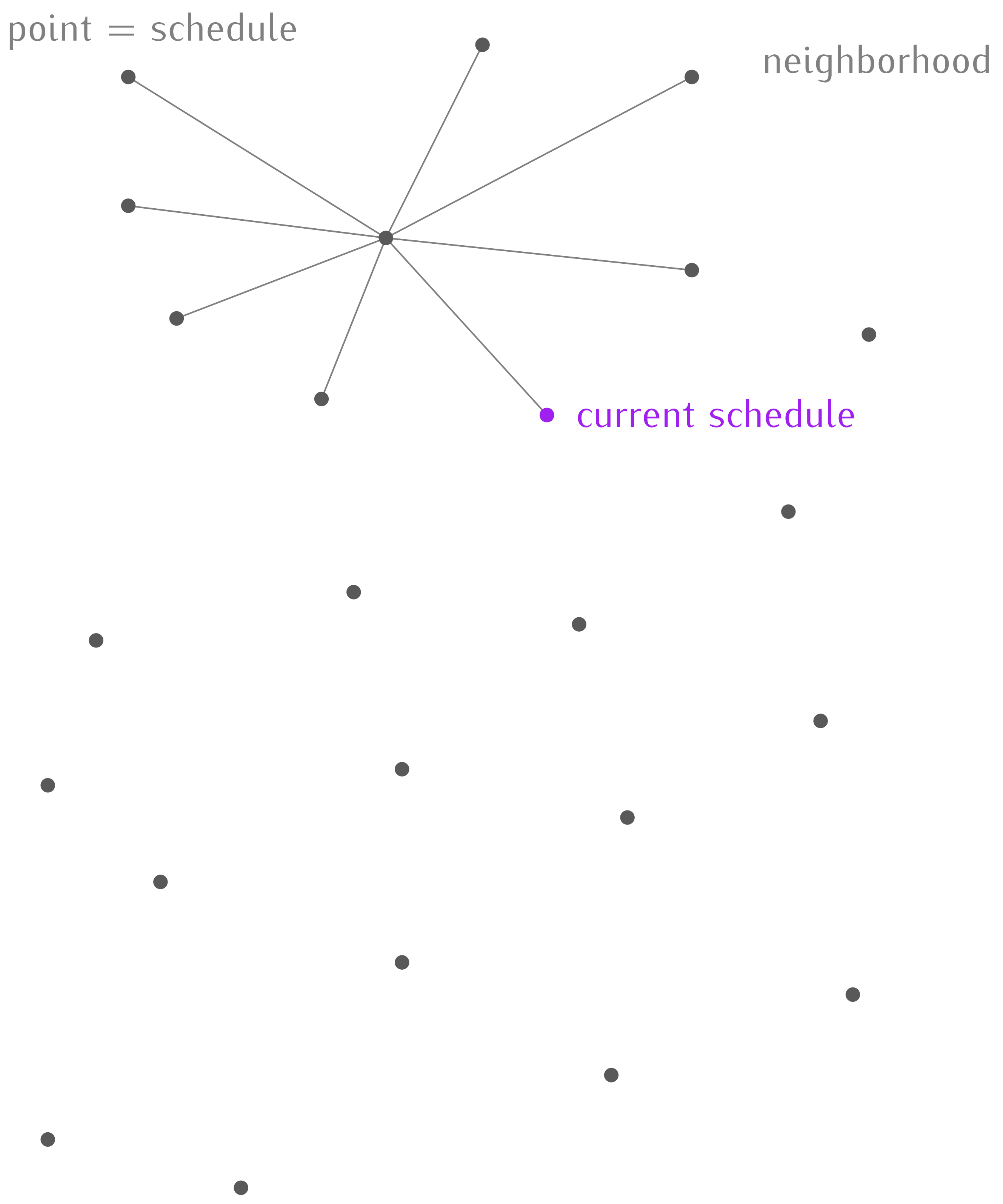
Local Search



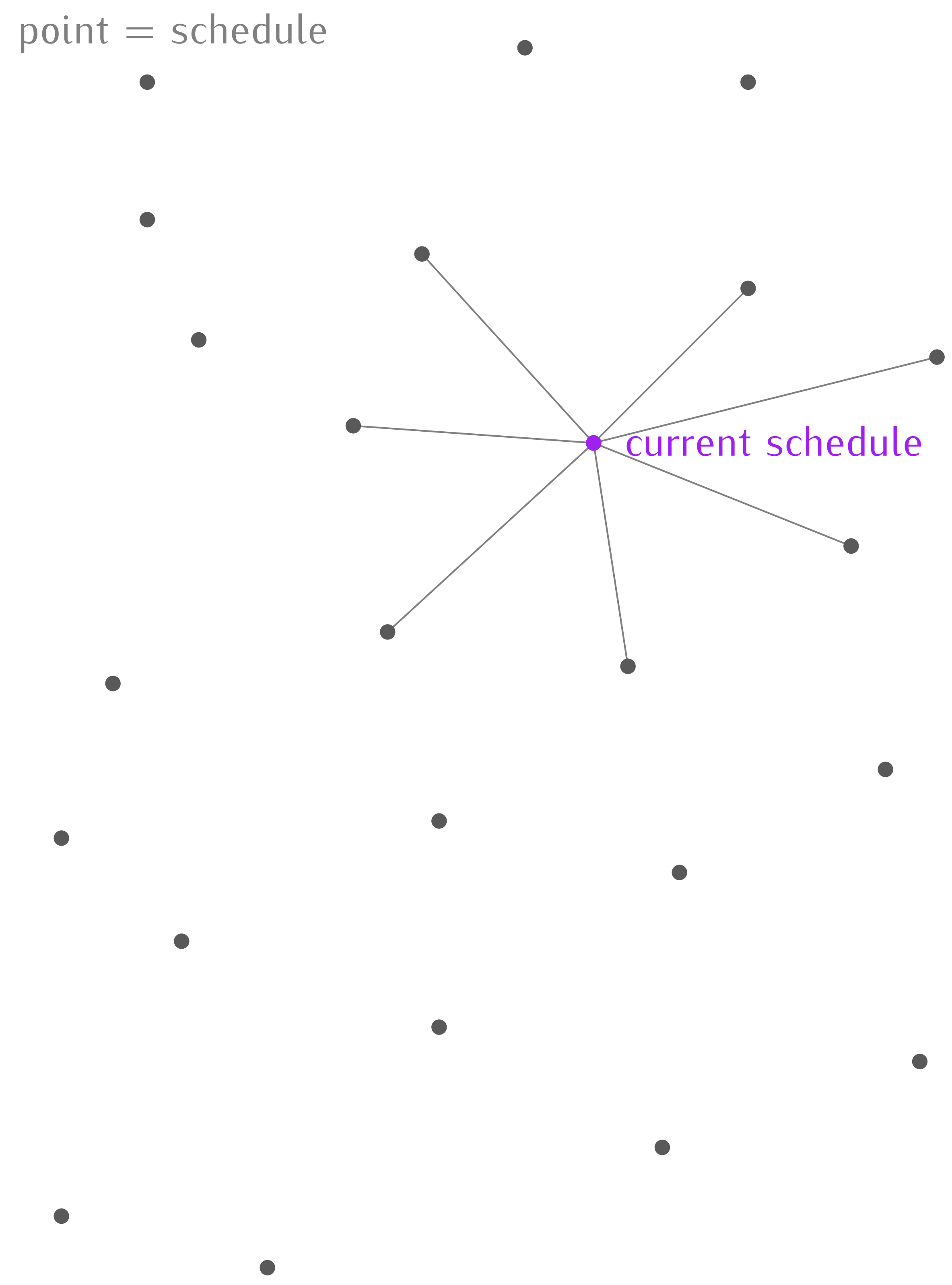
Local Search



Local Search

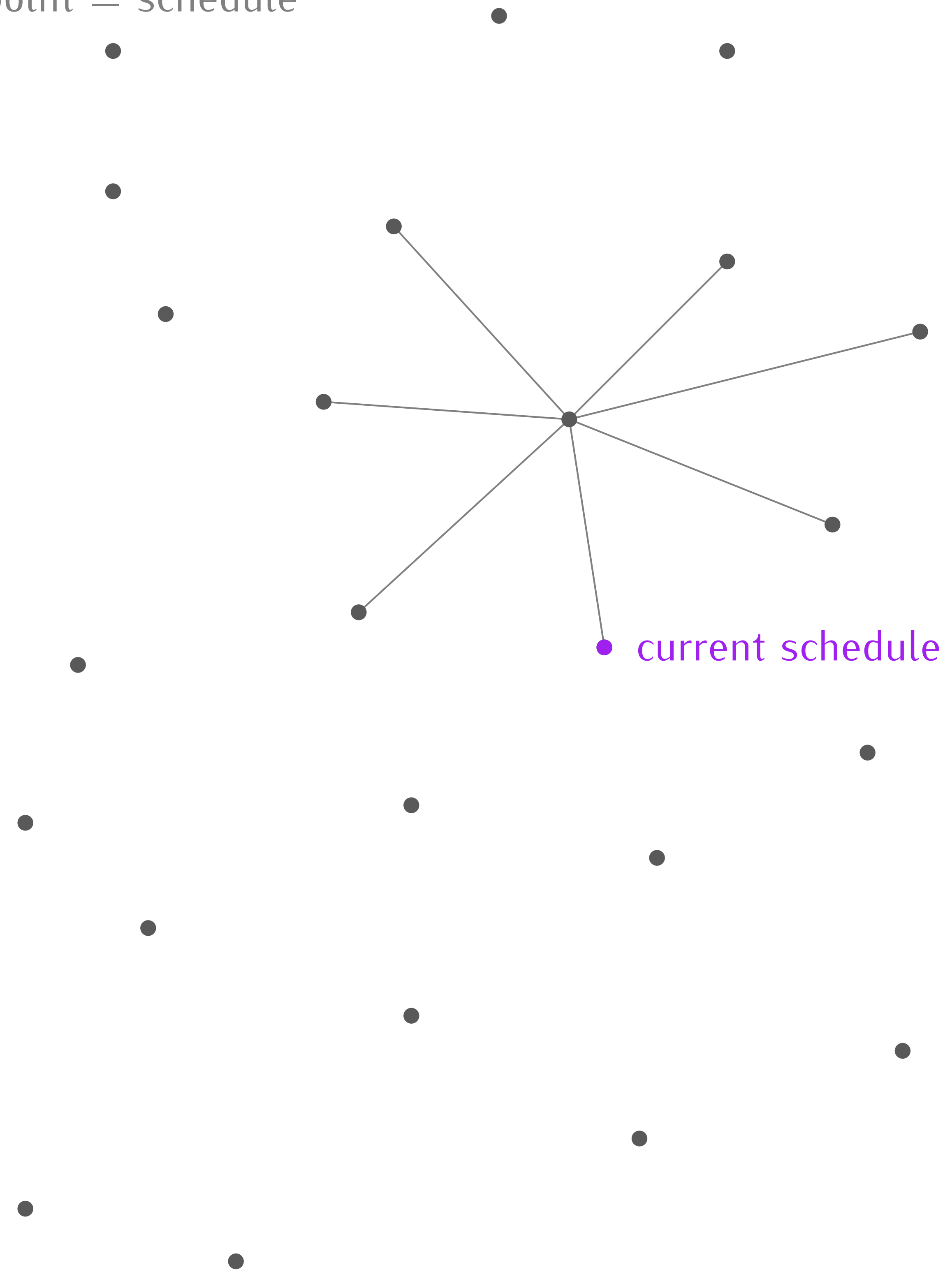


Local Search

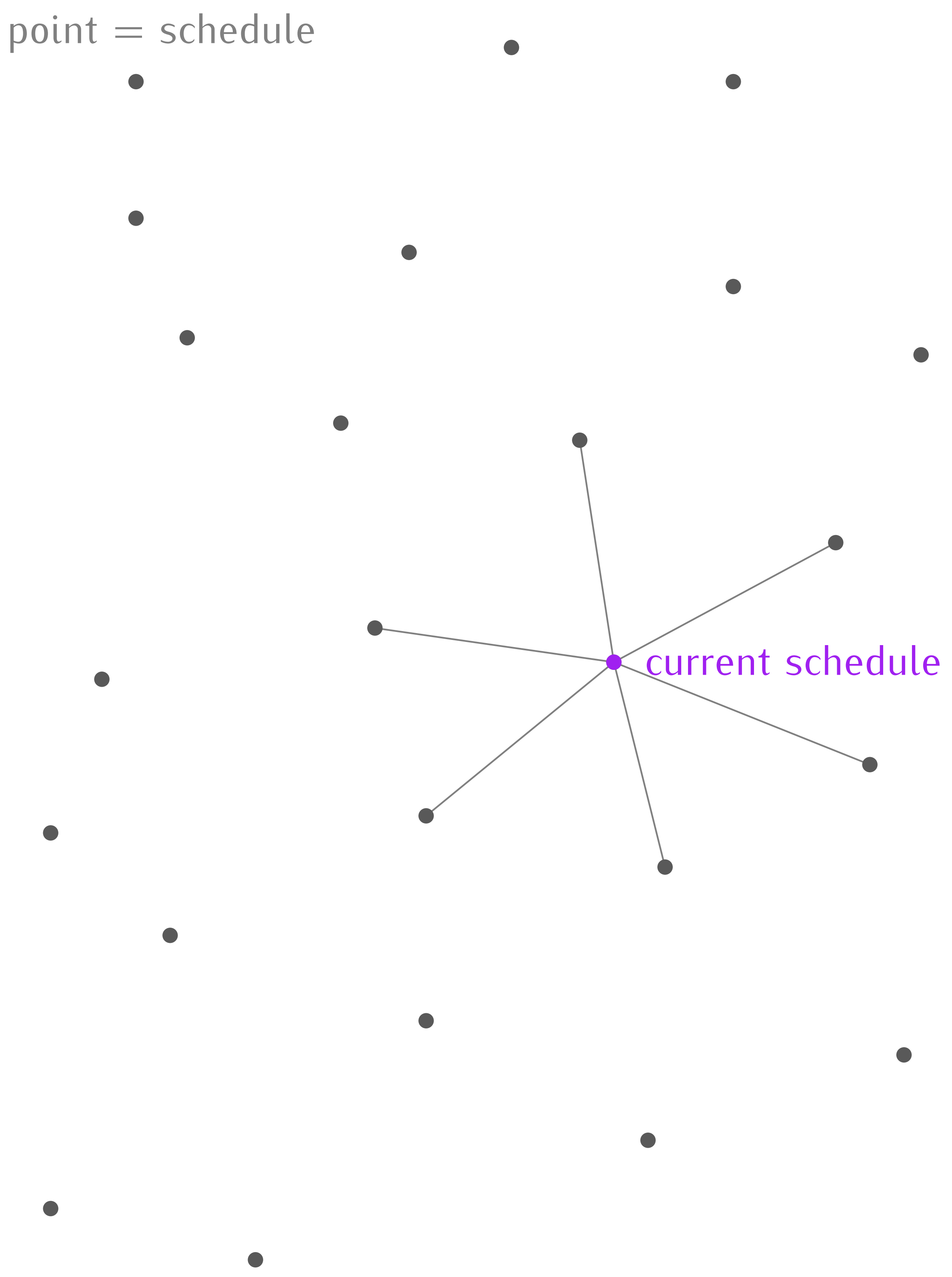


Local Search

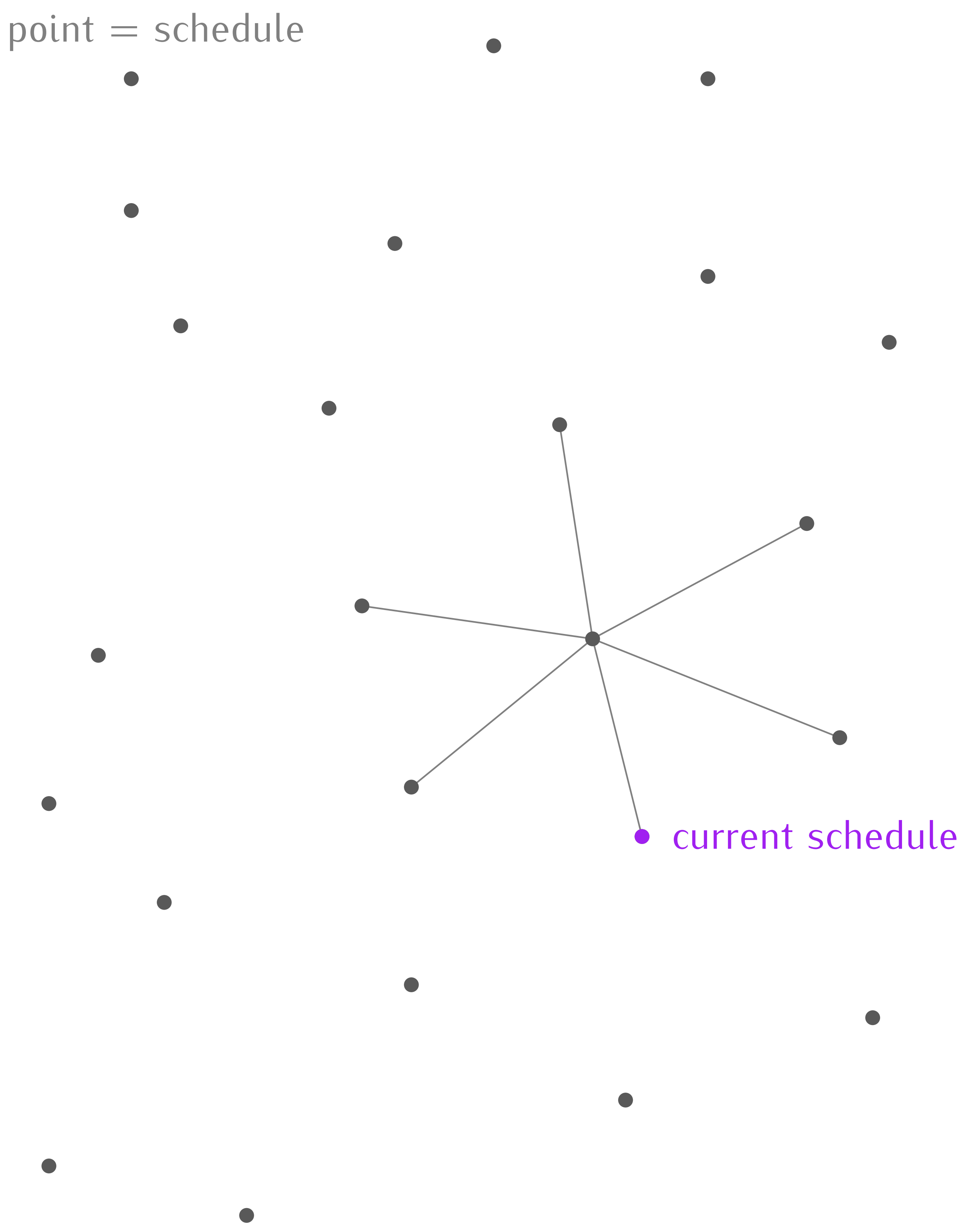
point = schedule



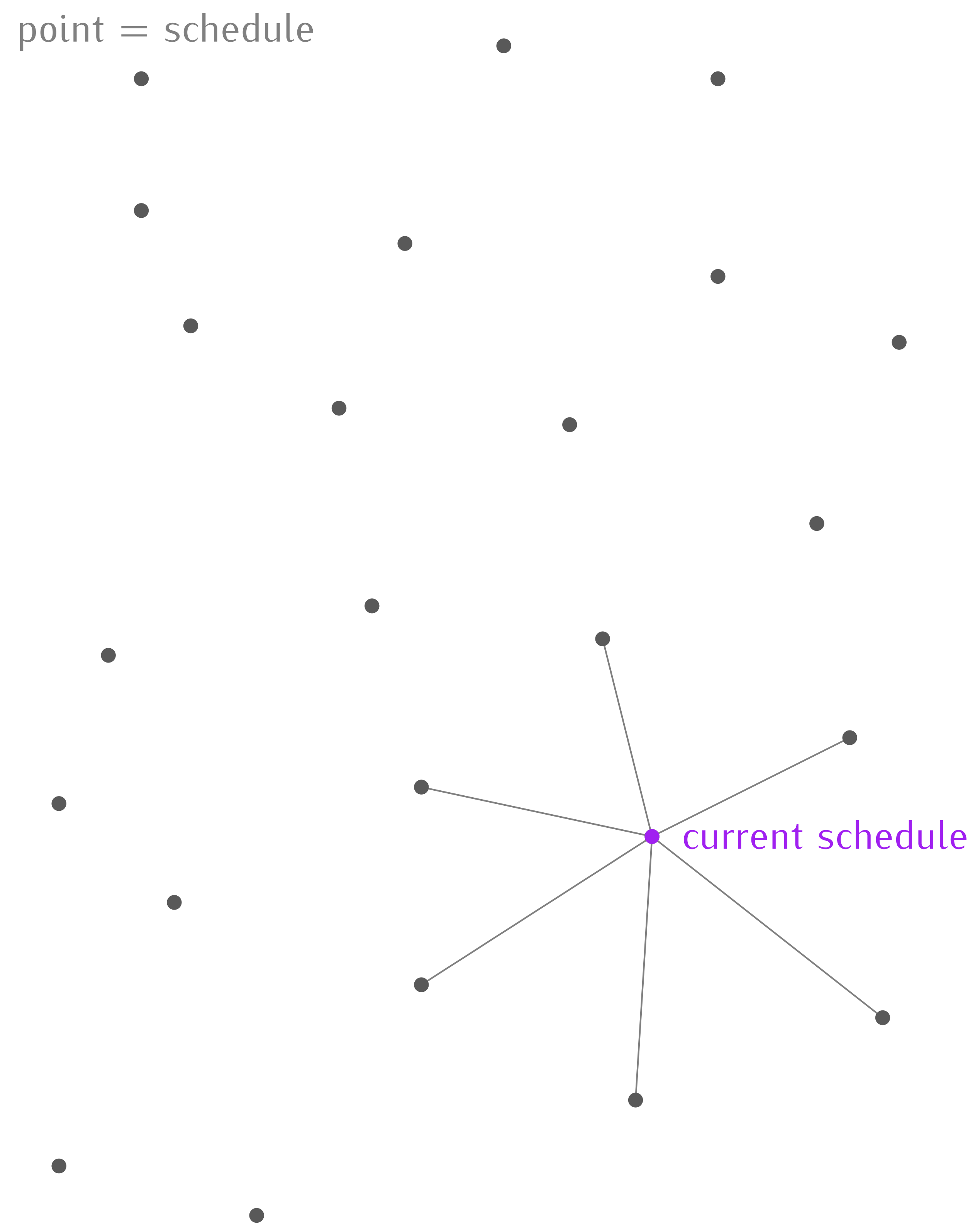
Local Search



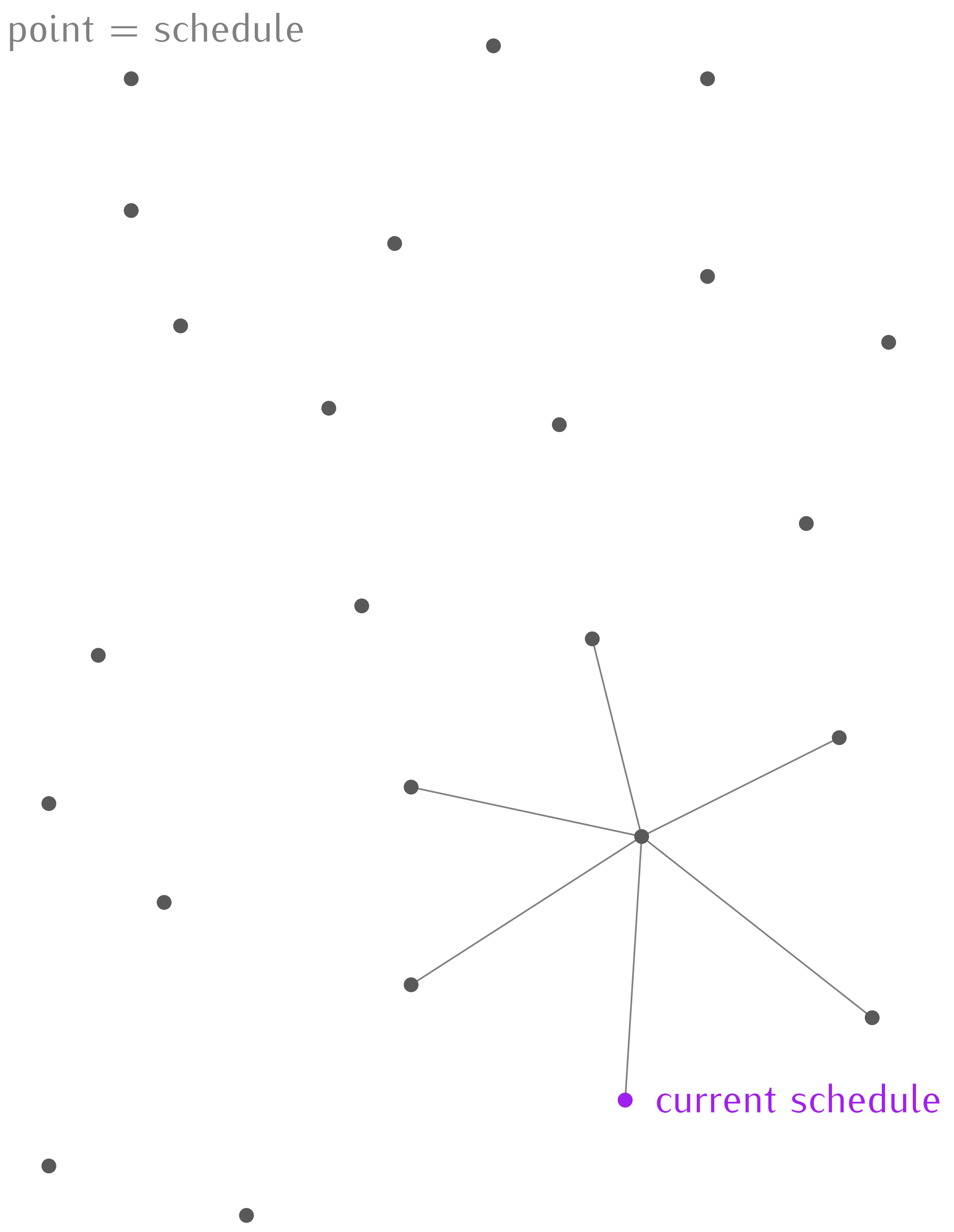
Local Search



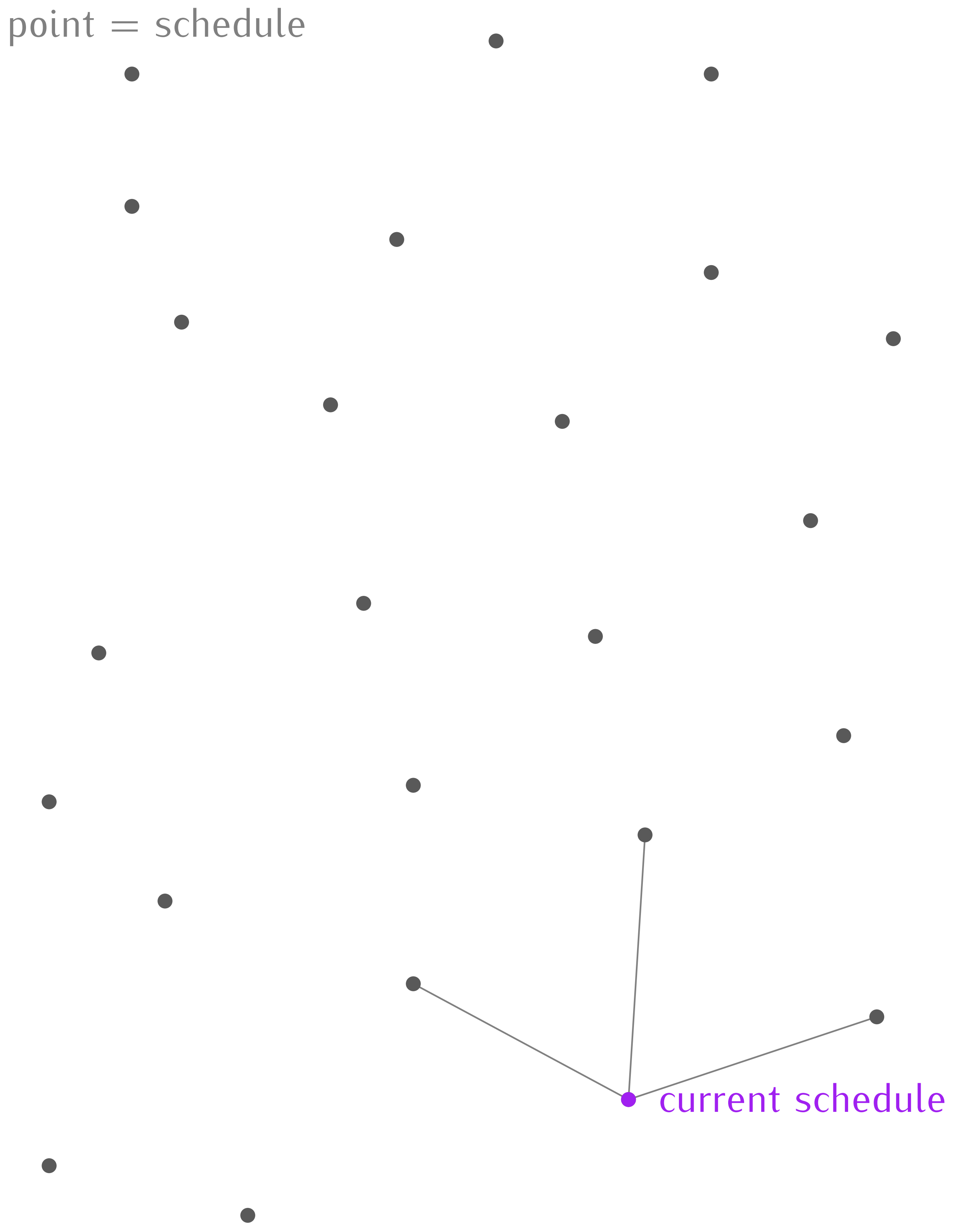
Local Search



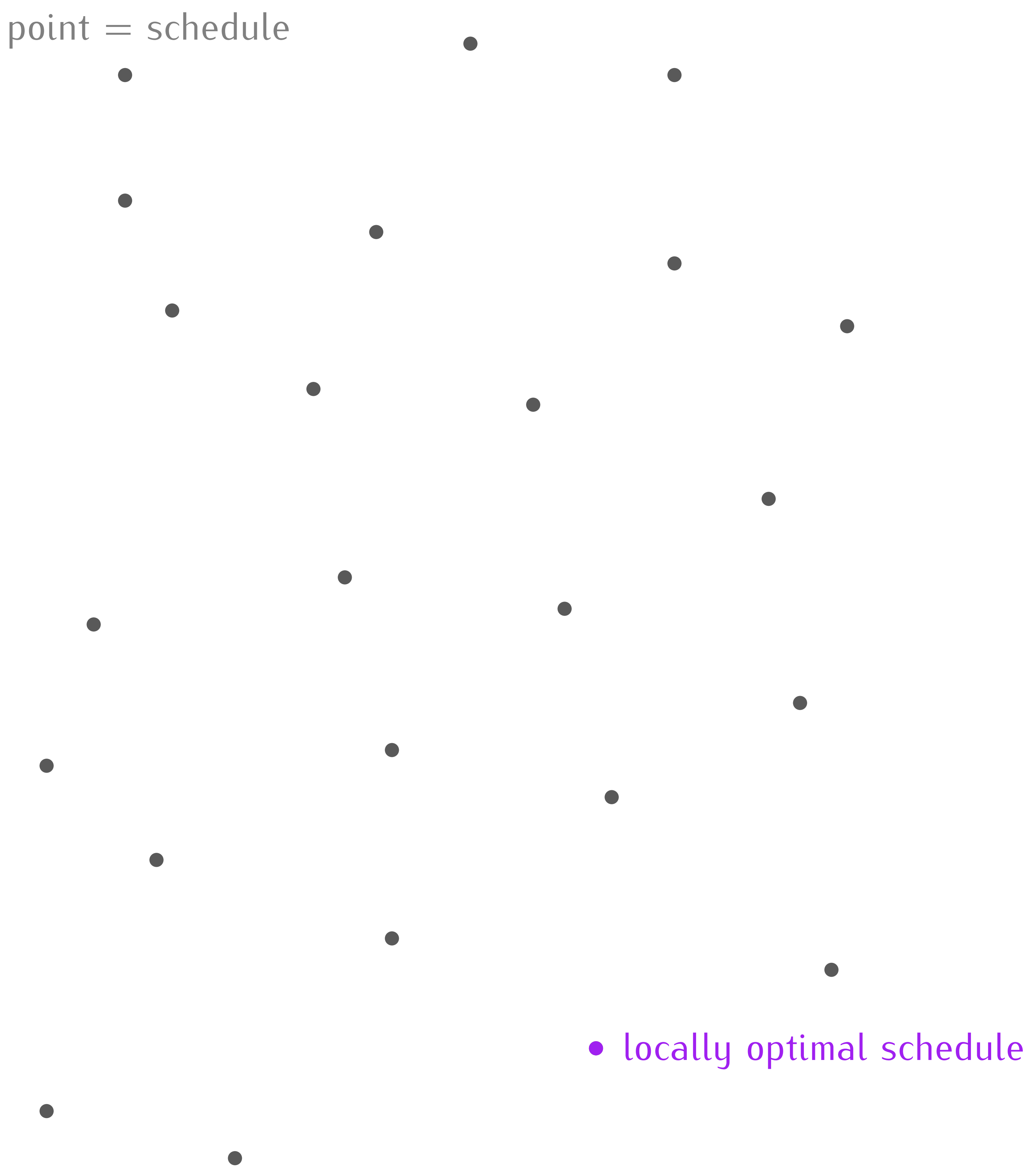
Local Search



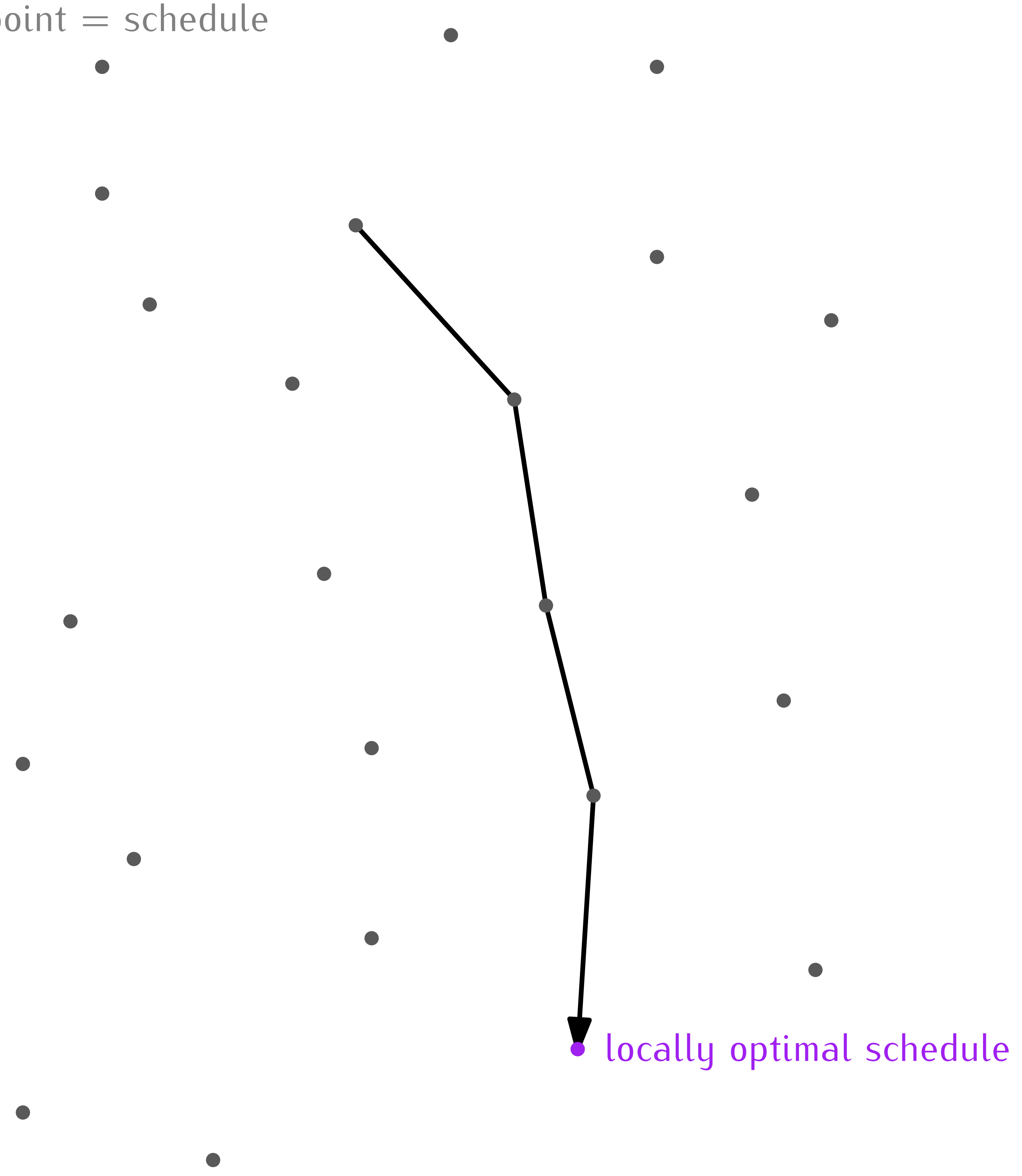
Local Search

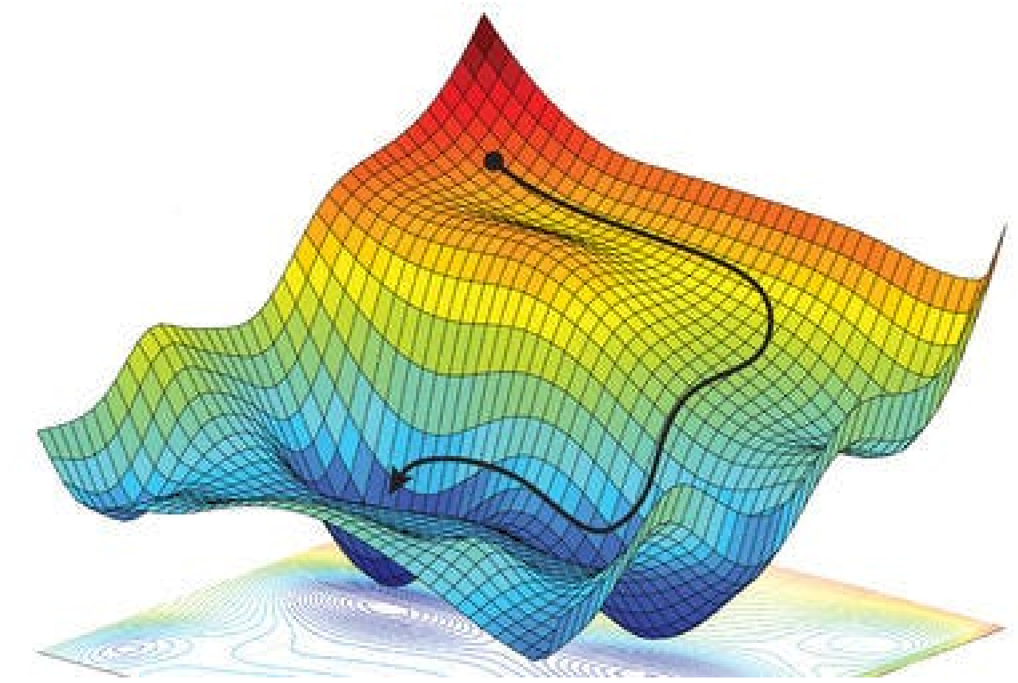


Local Search

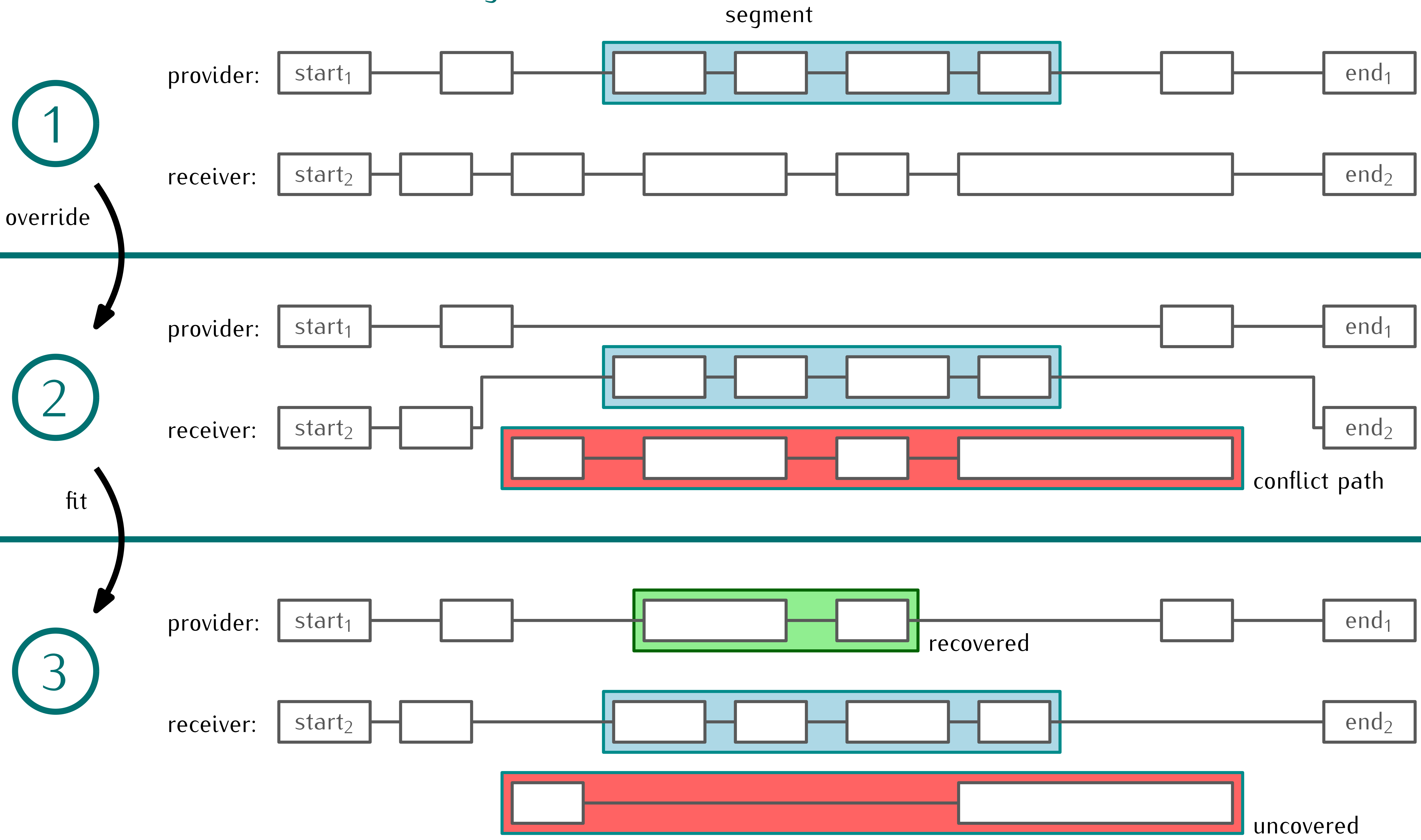


Local Search



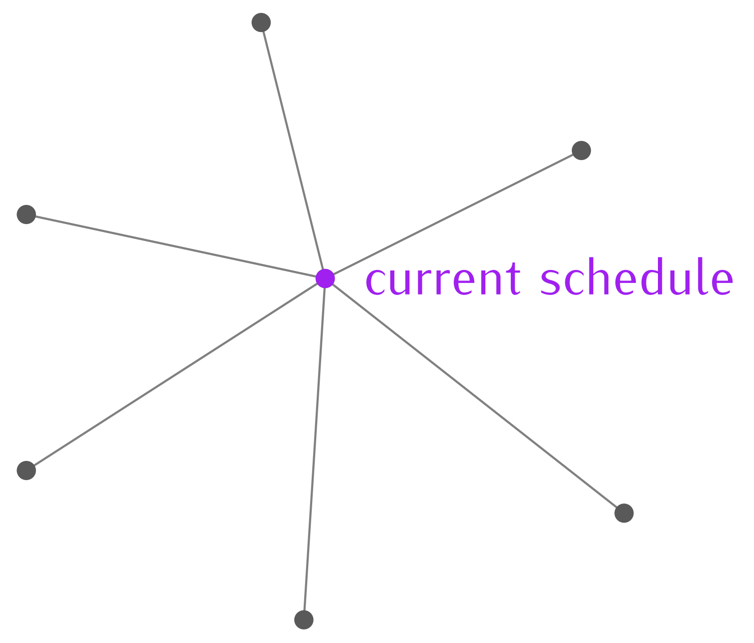


Local Modification: Path Exchange



Neighborhood

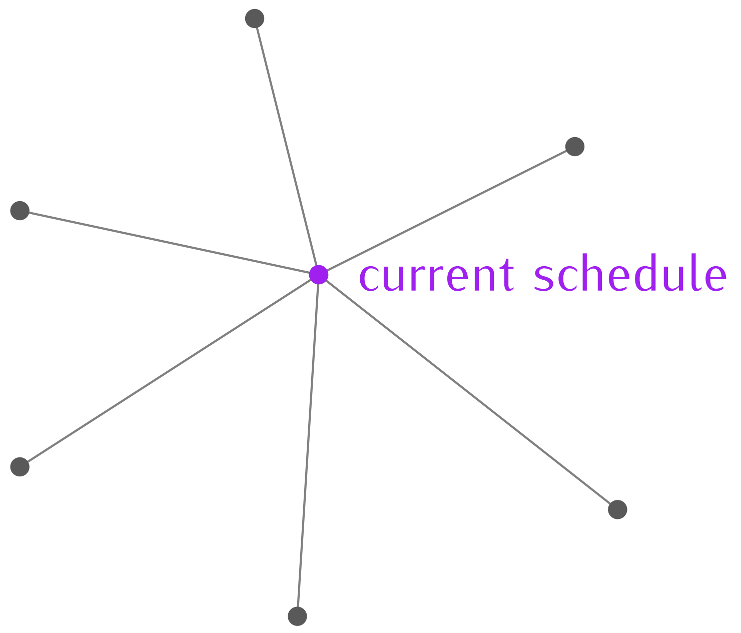
given the current schedule, consider the following modificaitons:



for each vehicle as **provider**:

Neighborhood

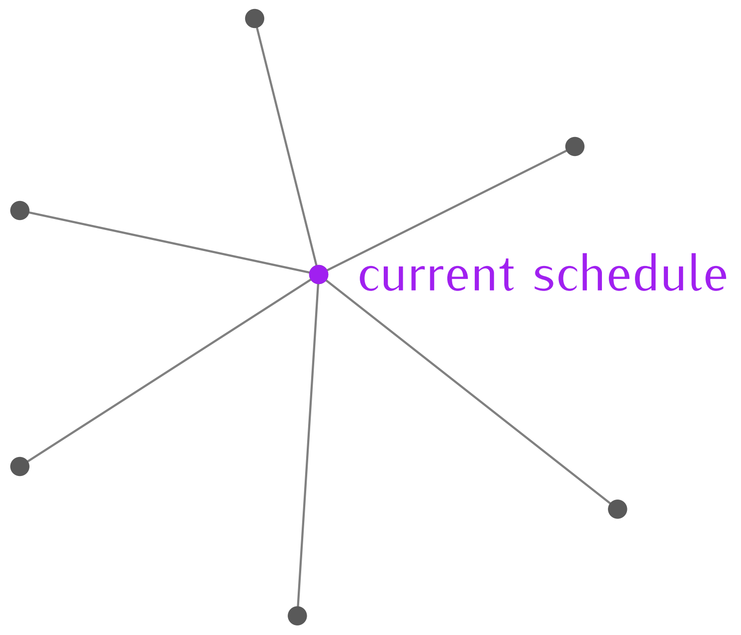
given the current schedule, consider the following modificaitons:



```
for each vehicle as provider:
    dead-head trips slower than service trips
    for each segment of provider's tour that can be removed
```

Neighborhood

given the current schedule, consider the following modificaitons:



for each vehicle as **provider**:

dead-head trips slower than service trips

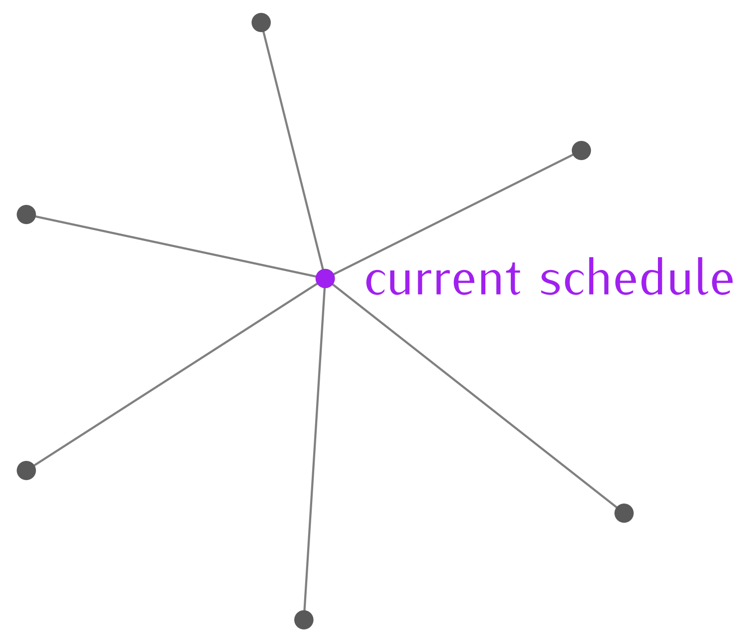
    for each segment of provider's tour that can be removed

        for each vehicle of the same type (that is not the provider) as **receiver**:

            (if segment is mainteance slot also different types are allwed)

Neighborhood

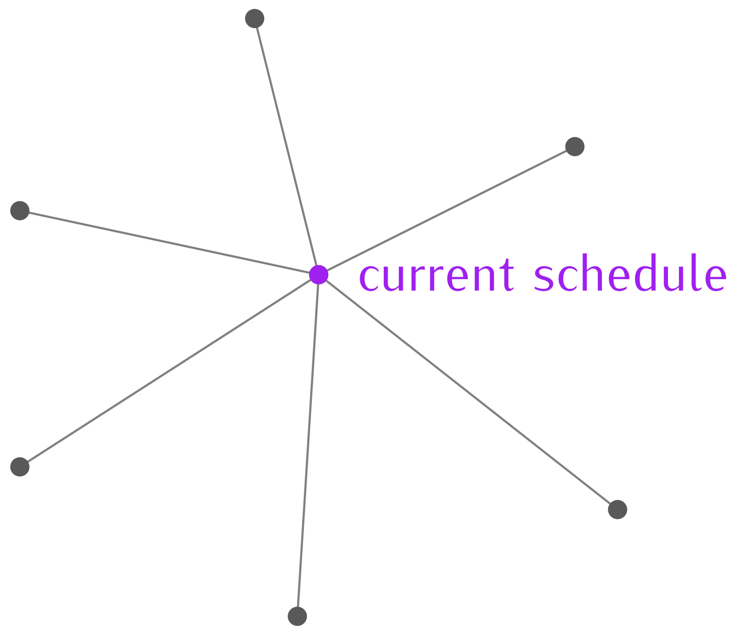
given the current schedule, consider the following modificaitons:



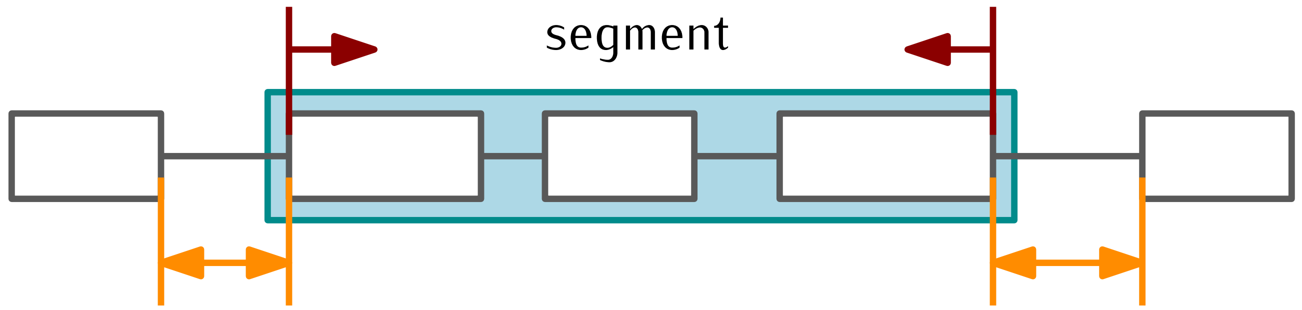
```
for each vehicle as provider:
    dead-head trips slower than service trips
    for each segment of provider's tour that can be removed
        for each vehicle of the same type (that is not the provider) as receiver:
            (if segment is mainteance slot also different types are allwed)
            PathExchange(segment, provider, receiver)
```

Neighborhood

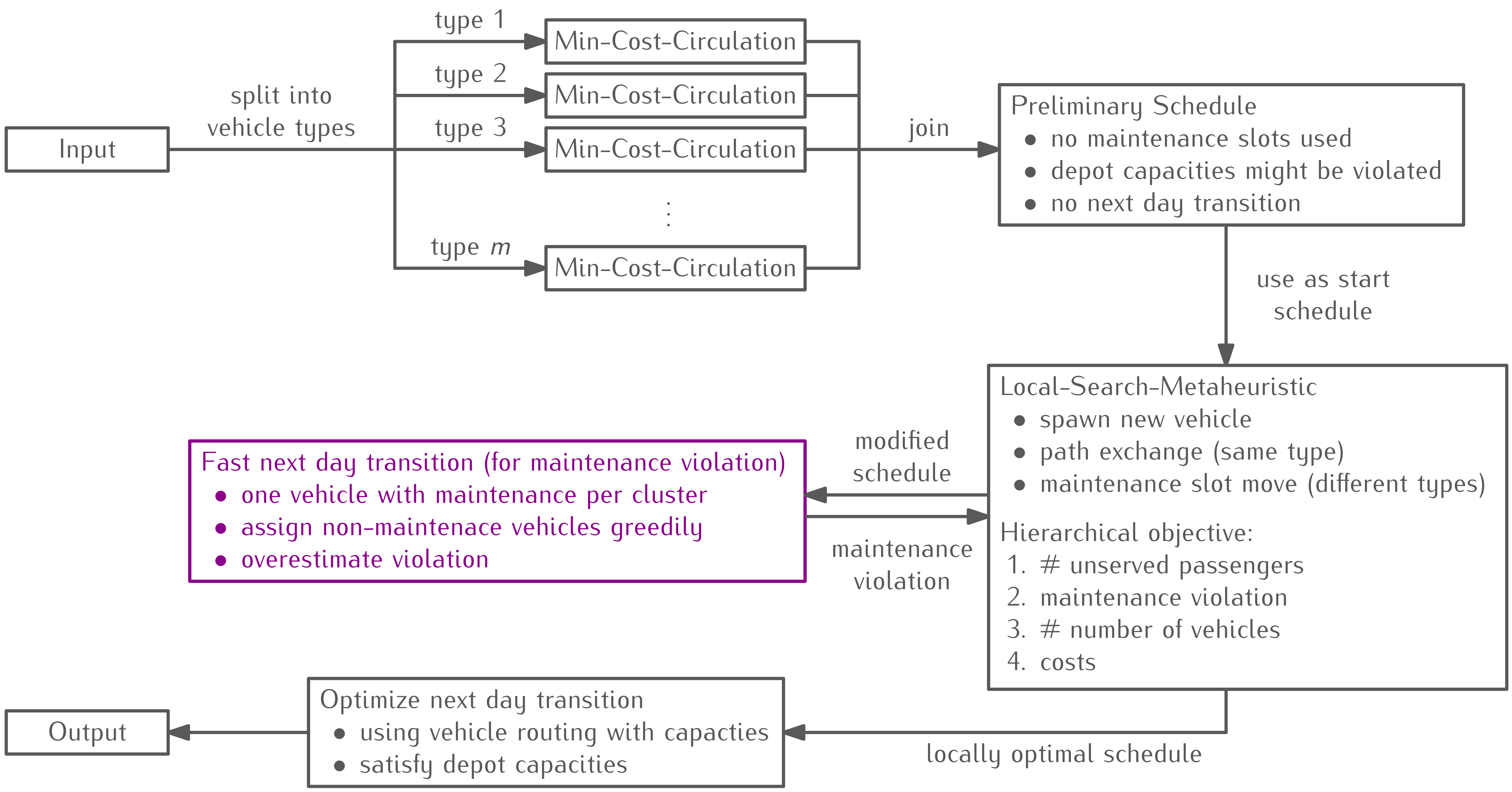
given the current schedule, consider the following modificaitons:



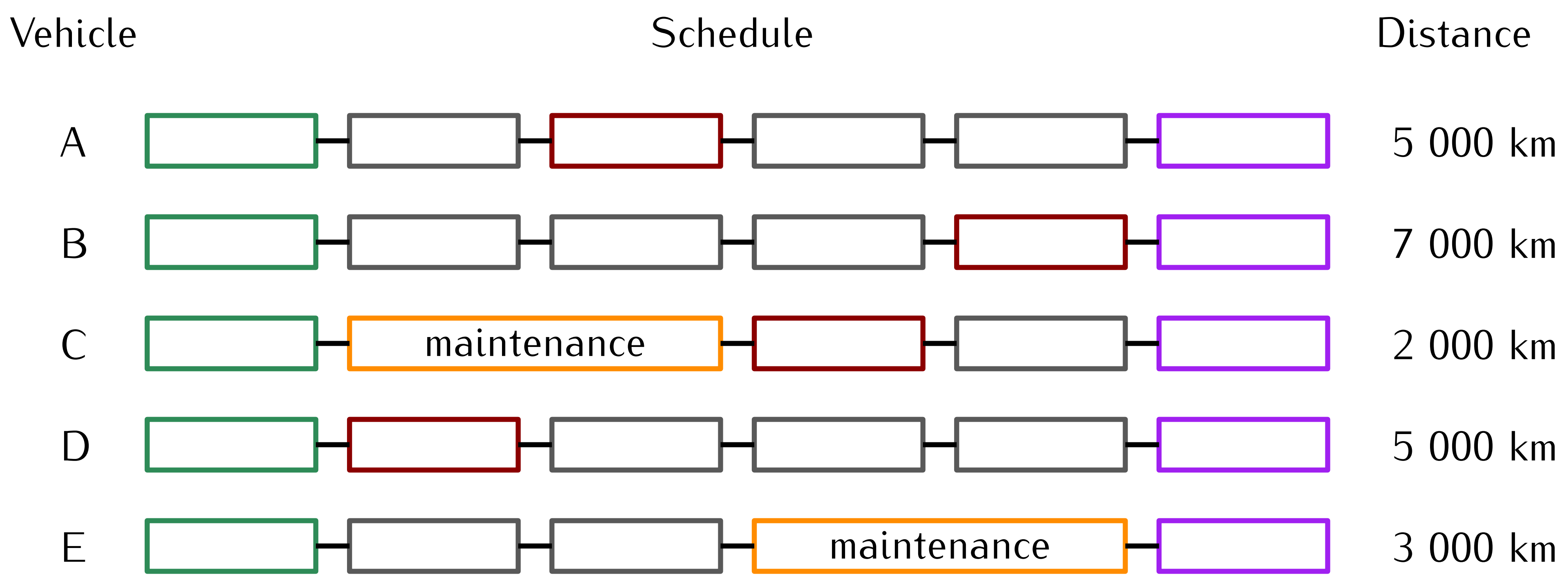
```
for each vehicle as provider:
    dead-head trips slower than service trips
    for each segment of provider's tour that can be removed
        • restrict length: start of first node to end of last node
        • restrict overhead time before and after
        for each vehicle of the same type (that is not the provider) as receiver:
            (if segment is mainteance slot also different types are allwed)

            PathExchange(segment, provider, receiver)
```



Phase 2 - Algorithm Overview



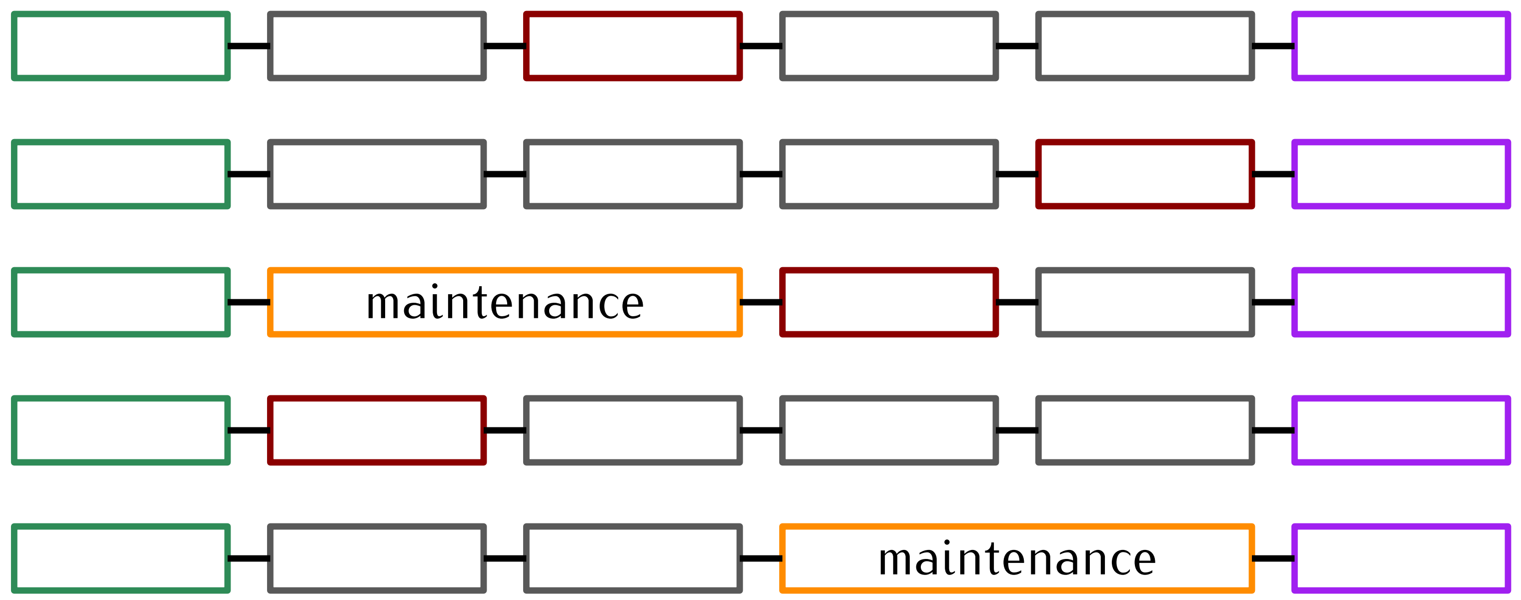
Next Day Transition (for maintenance regulations)



Next Day Transition (for maintenance regulations)

maintenance all 15 000 km

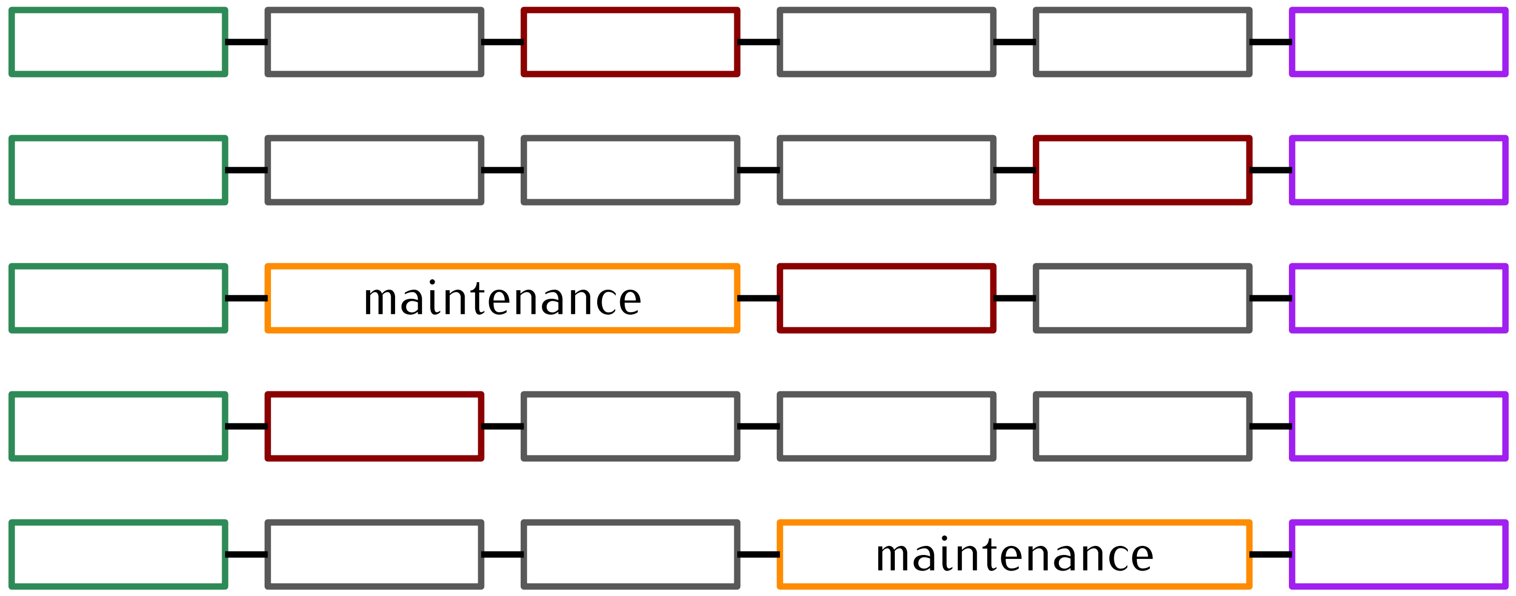
Schedule



Vehicle	Distance			Maintenance Counter
A	5 000 km		=	5 000 km
B	7 000 km		=	7 000 km
C	2 000 km	– 15 000 km	=	- 13 000 km
D	5 000 km		=	5 000 km
E	3 000 km	– 15 000 km	=	- 12 000 km

Next Day Transition (for maintenance regulations)

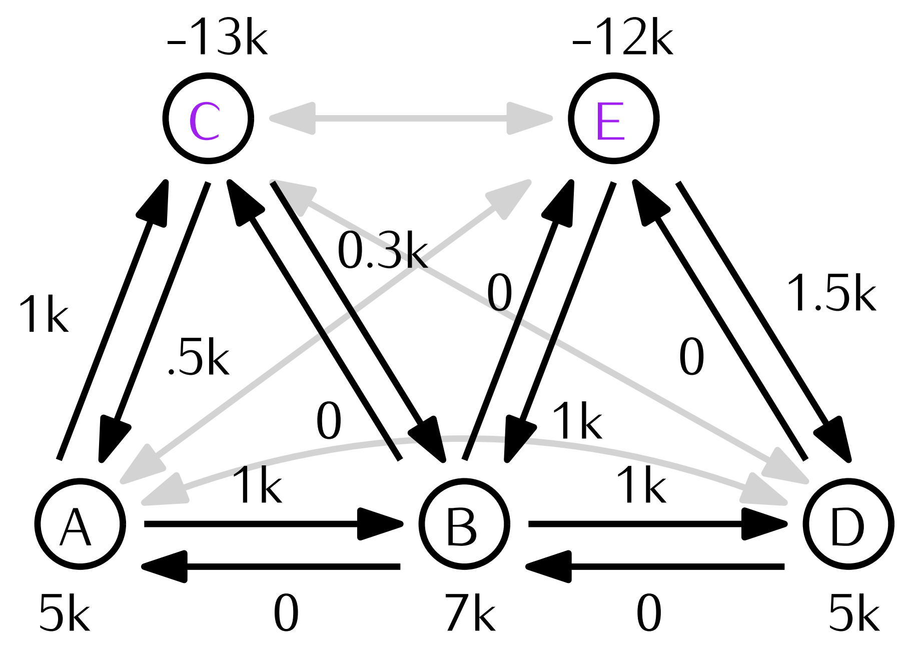
maintenance all 15 000 km



Vehicle	Schedule	Distance		Maintenance Counter
A		5 000 km		= 5 000 km
B		7 000 km		= 7 000 km
C	maintenance	2 000 km	– 15 000 km	= - 13 000 km
D		5 000 km		= 5 000 km
E	maintenance	3 000 km	– 15 000 km	= - 12 000 km

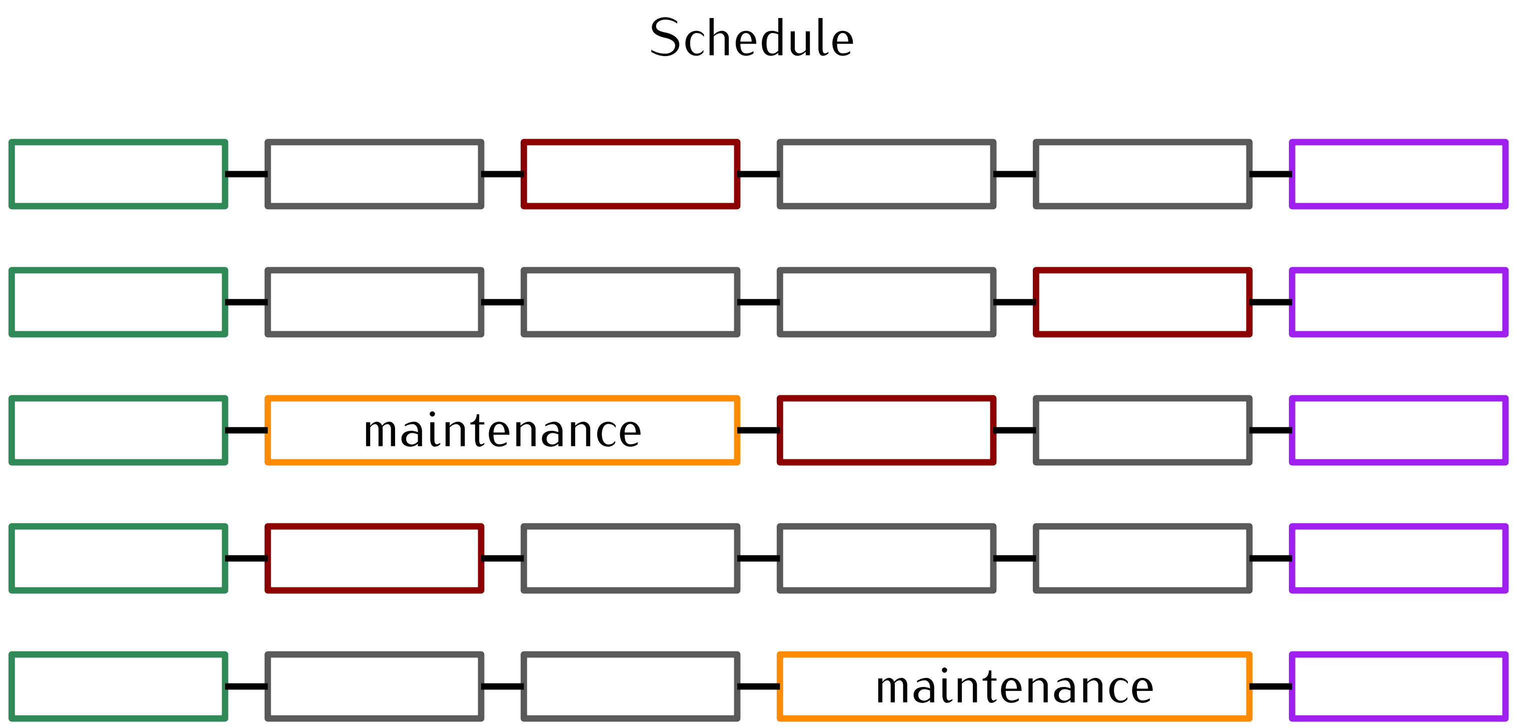
Model

- consider types individually
- fully connected directed graph
  - one node per vehicle labeled with maintenance counter
  - arcs (v_1, v_2) are labelled with dead-head-distance between end depot of v_1 to start depot of v_2



Next Day Transition (for maintenance regulations)

maintenance all 15 000 km



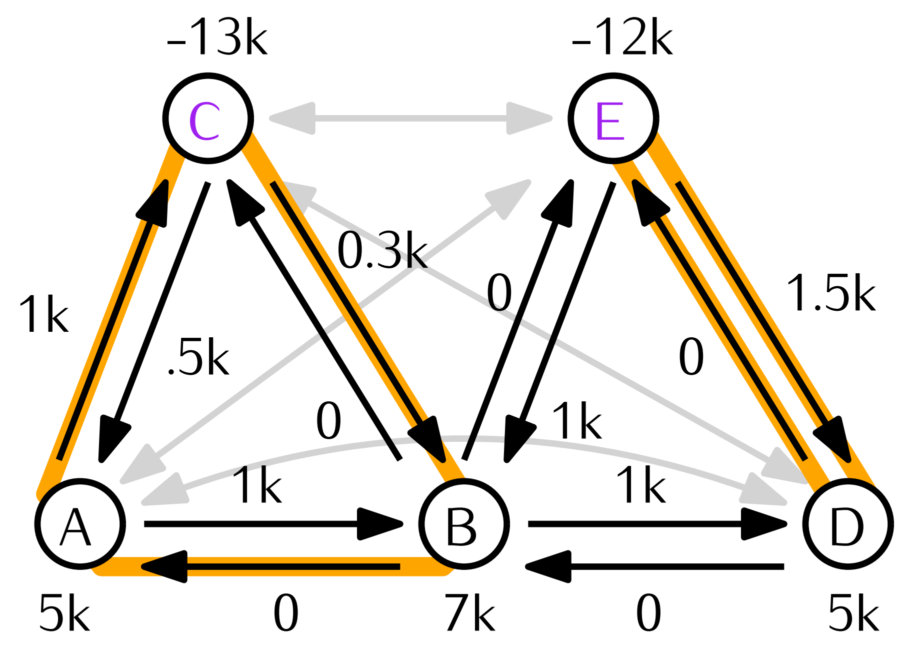
Vehicle	Distance			Maintenance Counter
A	5 000 km		=	5 000 km
B	7 000 km		=	7 000 km
C	2 000 km	– 15 000 km	=	- 13 000 km
D	5 000 km		=	5 000 km
E	3 000 km	– 15 000 km	=	- 12 000 km

Model

- consider types individually
- fully connected directed graph
  - one node per vehicle labeled with maintenance counter
  - arcs (v_1, v_2) are labelled with dead-head-distance between end depot of v_1 to start depot of v_2

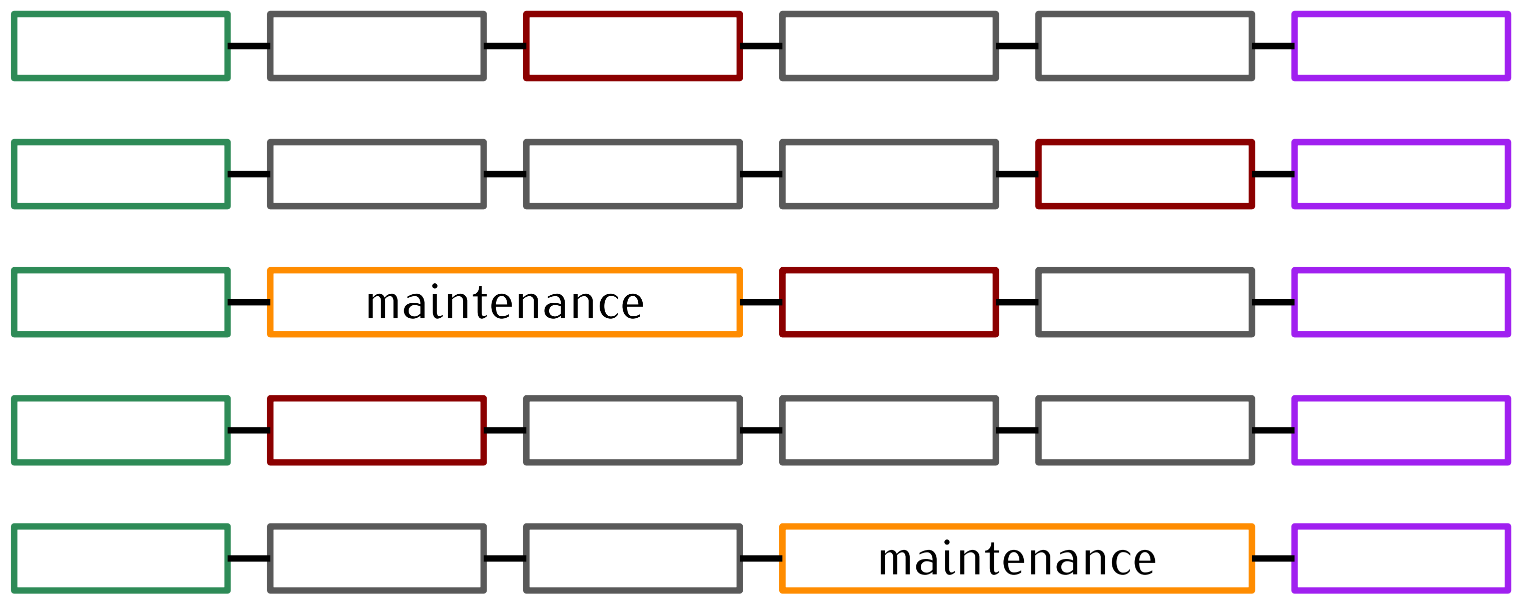
Next Day Transition

- a set of disjoint cycles
- all nodes are covered



Next Day Transition (for maintenance regulations)

maintenance all 15 000 km



Vehicle	Schedule	Distance			Maintenance Counter
A		5 000 km		=	5 000 km
B		7 000 km		=	7 000 km
C	maintenance	2 000 km	– 15 000 km	=	- 13 000 km
D		5 000 km		=	5 000 km
E	maintenance	3 000 km	– 15 000 km	=	- 12 000 km

Model

- consider types individually
- fully connected directed graph
  - one node per vehicle labeled with maintenance counter
  - arcs (v_1, v_2) are labelled with dead-head-distance between end depot of v_1 to start depot of v_2

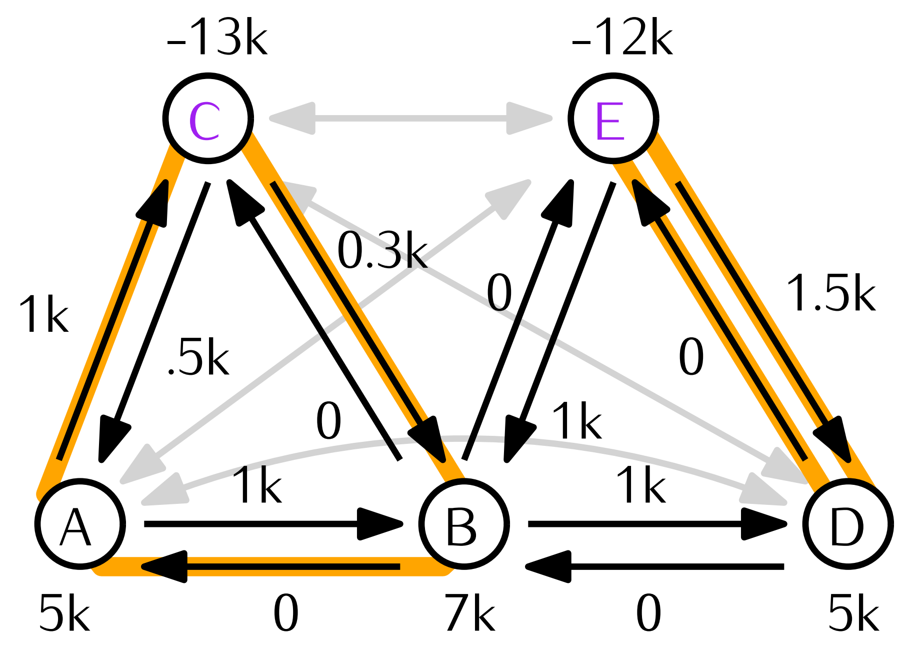
Next Day Transition

- a set of disjoint cycles
- all nodes are covered

Objective:

- minimize maintenance violation: $\max_{C \in \mathcal{C}} \ (\max\{\, d(C), 0 \,\})$

  where $d(C)$ = sum over all node and arc labels of cycle C



Fast next day transition

Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

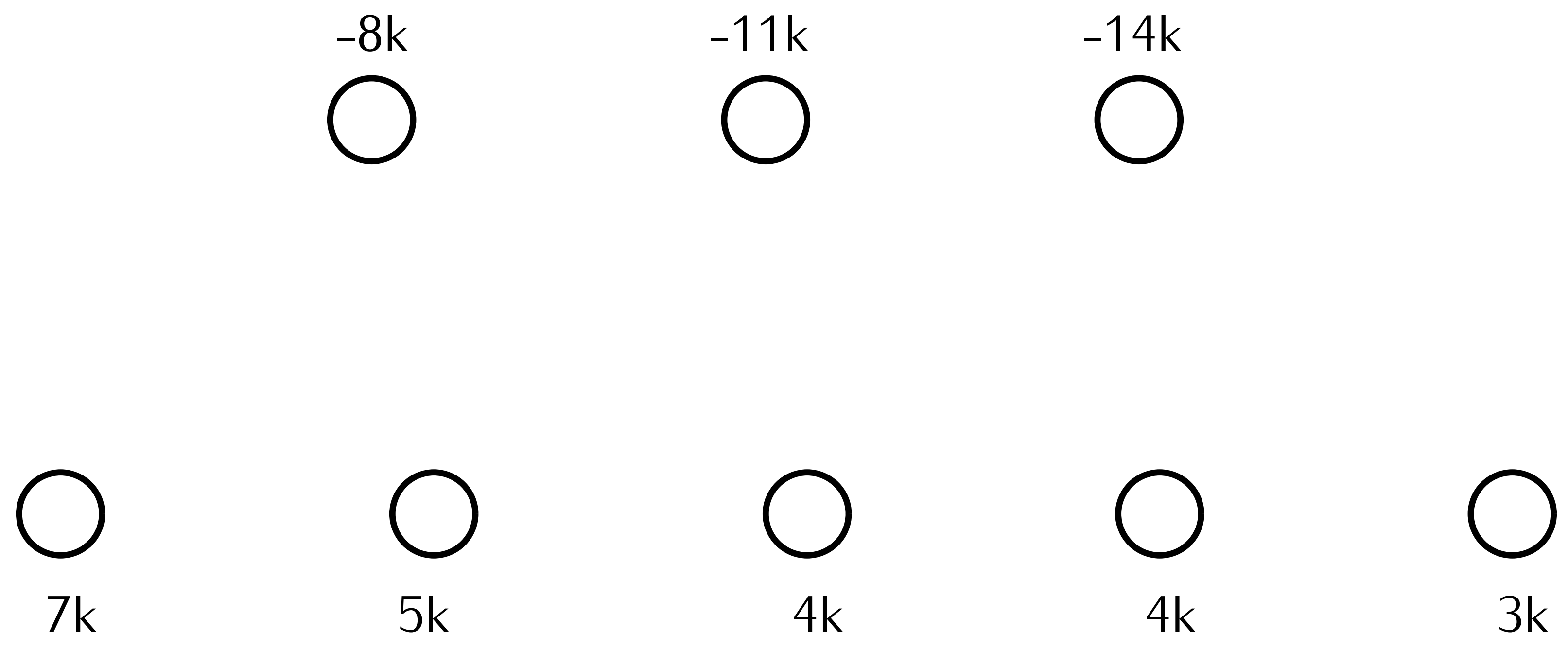
Fast next day transition

Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

maintenance vehicles:
(decreasing)

non-maintenance vehicles:
(decreasing)

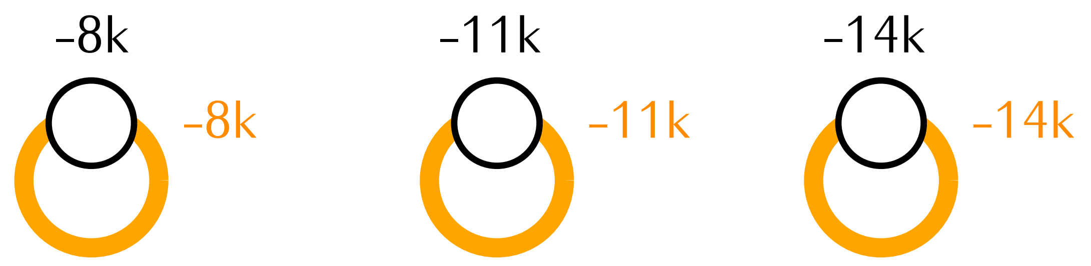


Fast next day transition

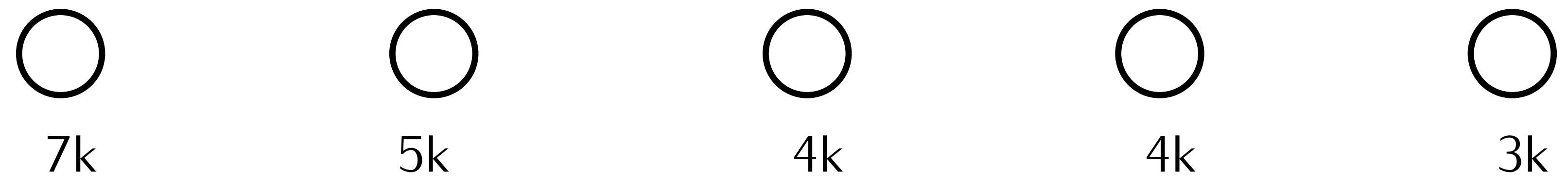
Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

maintenance vehicles:
(decreasing)



non-maintenance vehicles:
(decreasing)



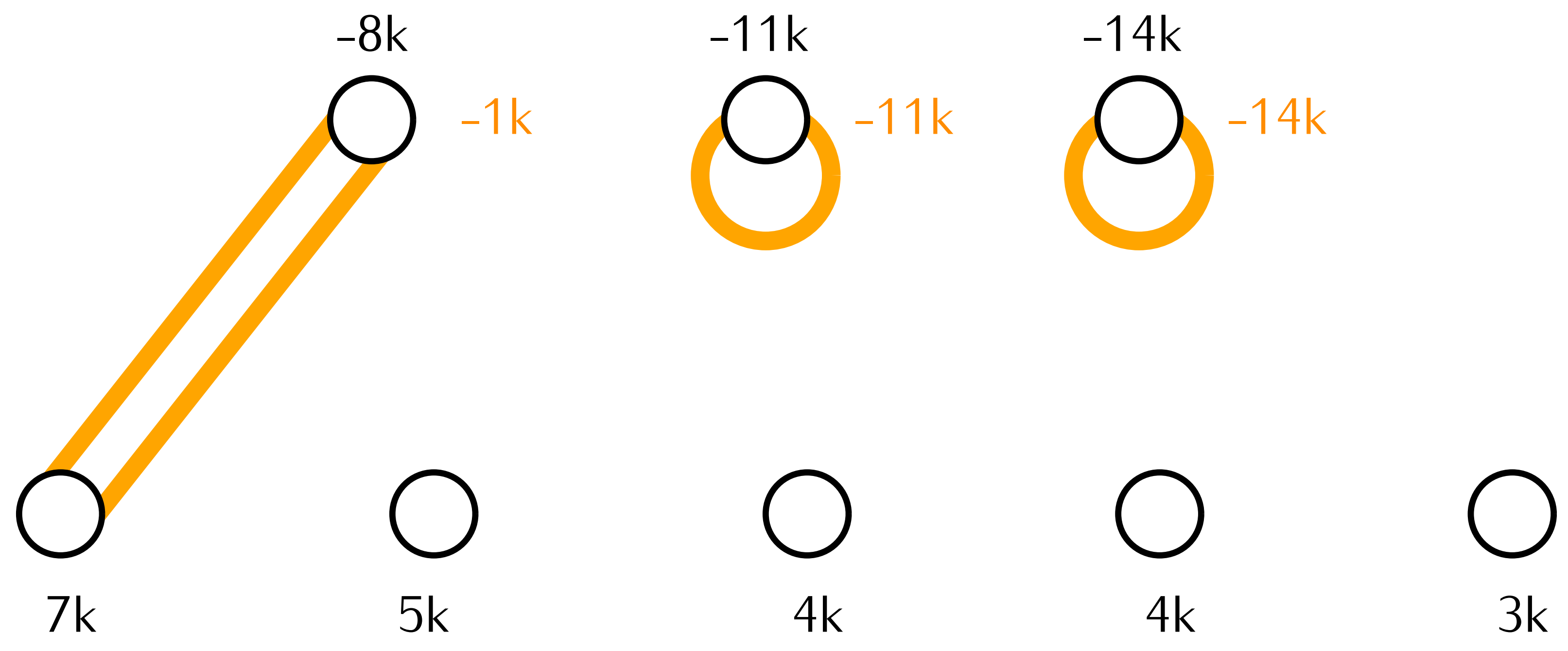
Fast next day transition

Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

maintenance vehicles:
(decreasing)

non-maintenance vehicles:
(decreasing)



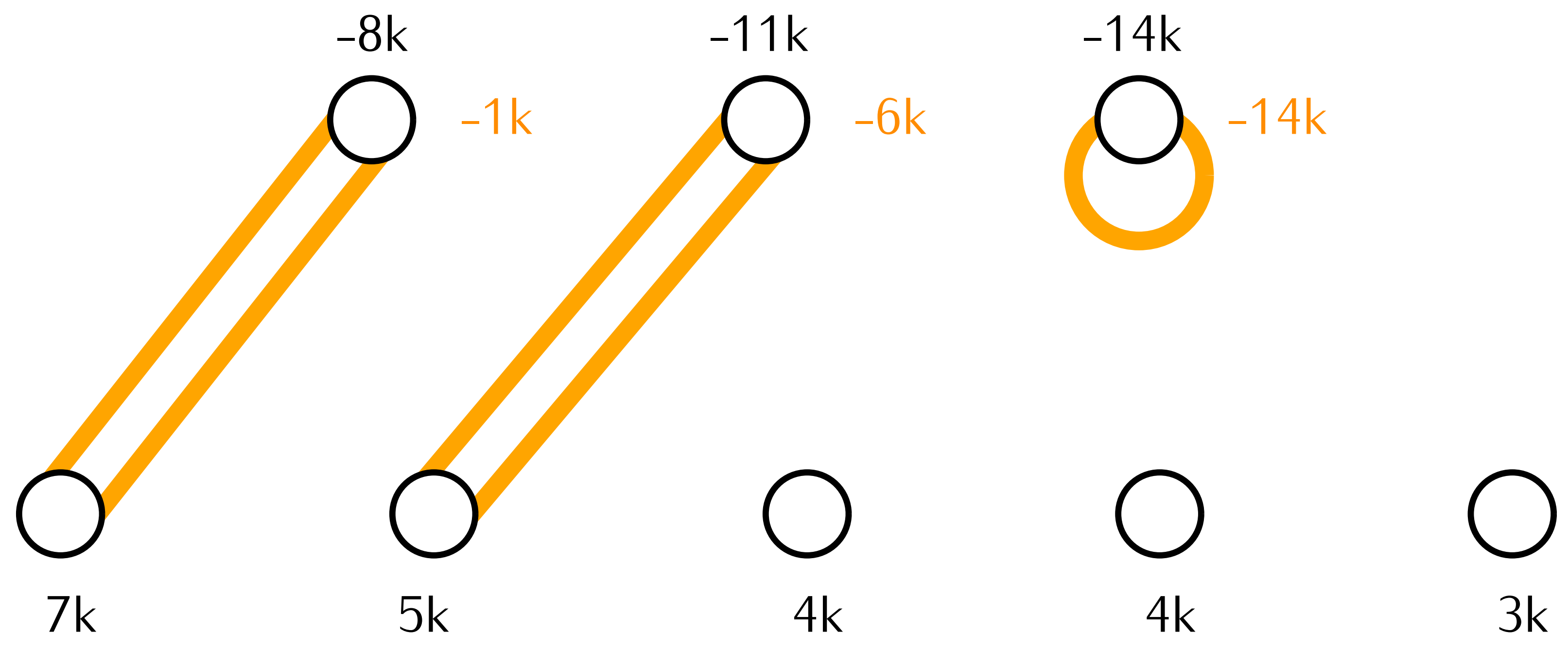
Fast next day transition

Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

maintenance vehicles:
(decreasing)

non-maintenance vehicles:
(decreasing)



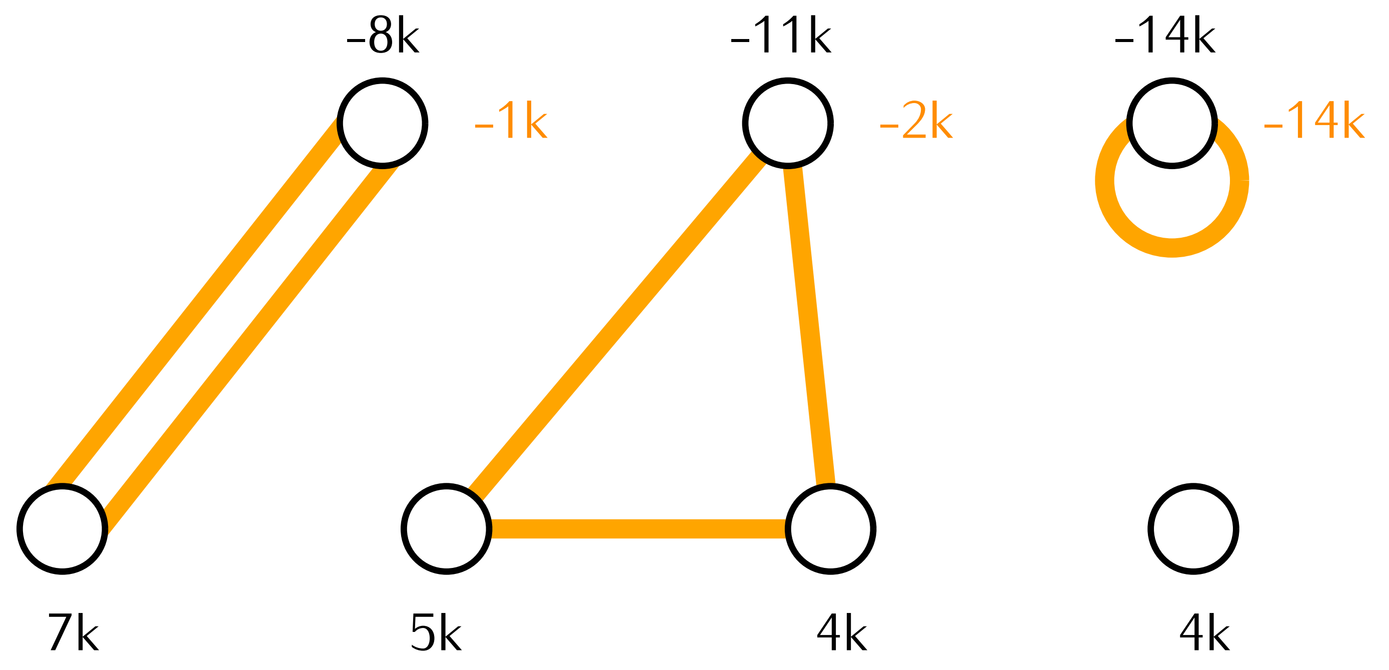
Fast next day transition

Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

maintenance vehicles:
(decreasing)

non-maintenance vehicles:
(decreasing)



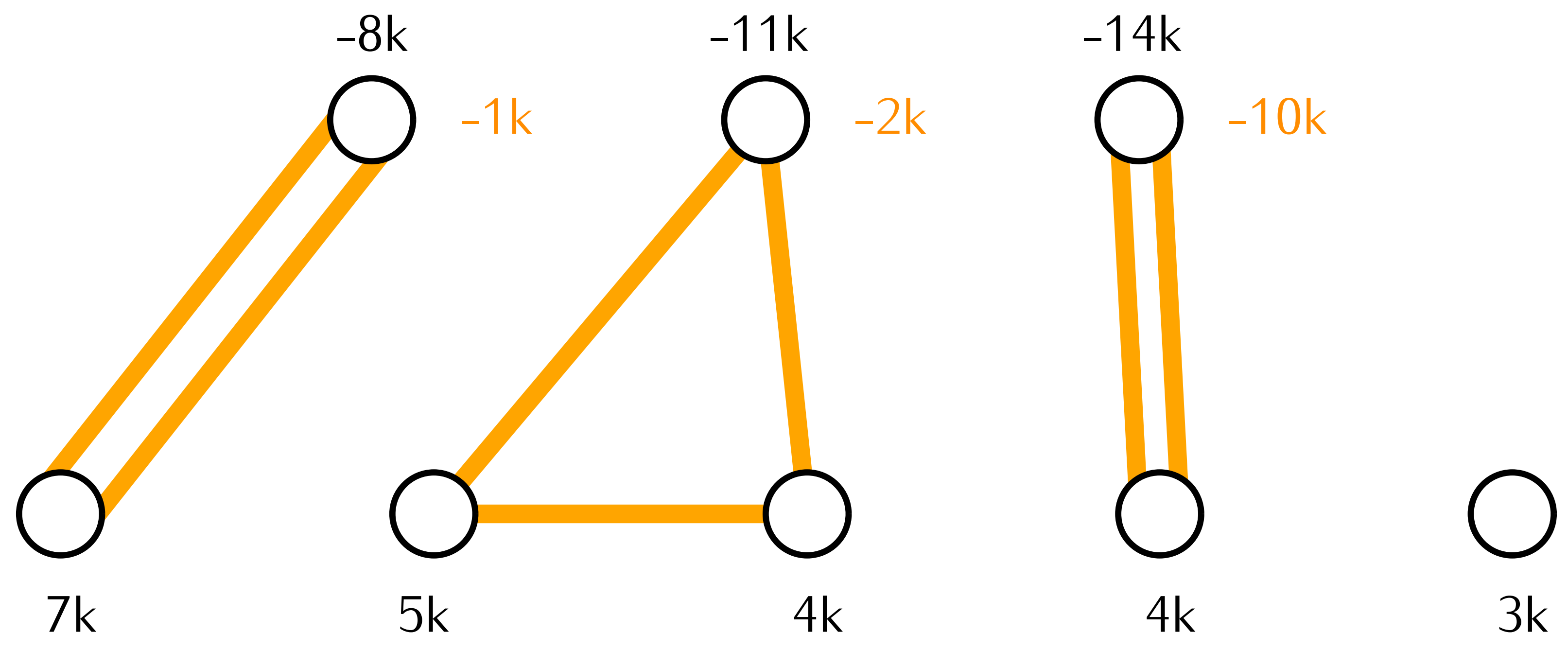
Fast next day transition

Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

maintenance vehicles:
(decreasing)

non-maintenance vehicles:
(decreasing)



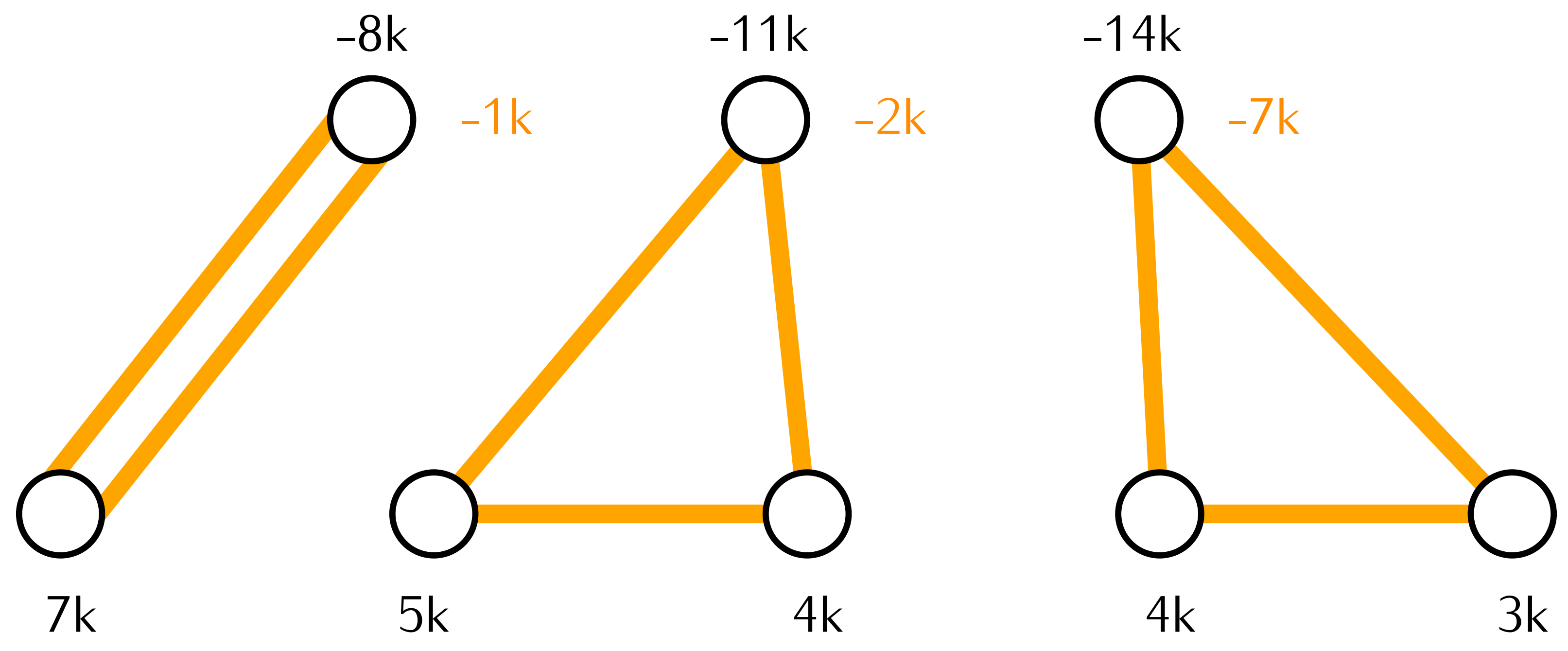
Fast next day transition

Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

maintenance vehicles:
(decreasing)

non-maintenance vehicles:
(decreasing)



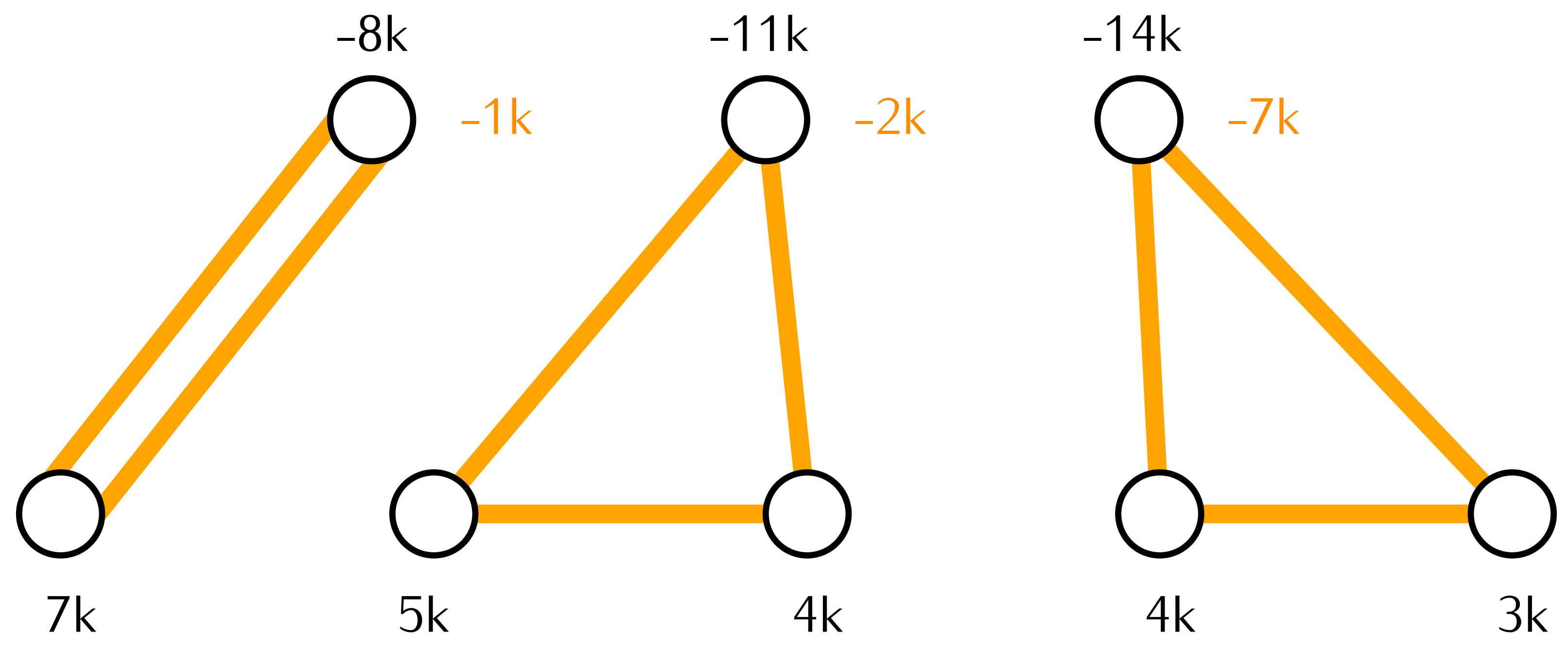
Fast next day transition

Algorithm

1. put all maintenance vehicles in a cycle (and sort them)
2. for each non-maintenance vehicle (in decreasing order):
   (a) put vehicle into the first fitting cycle (or the last if none are fitting)
   (b) resort cycles

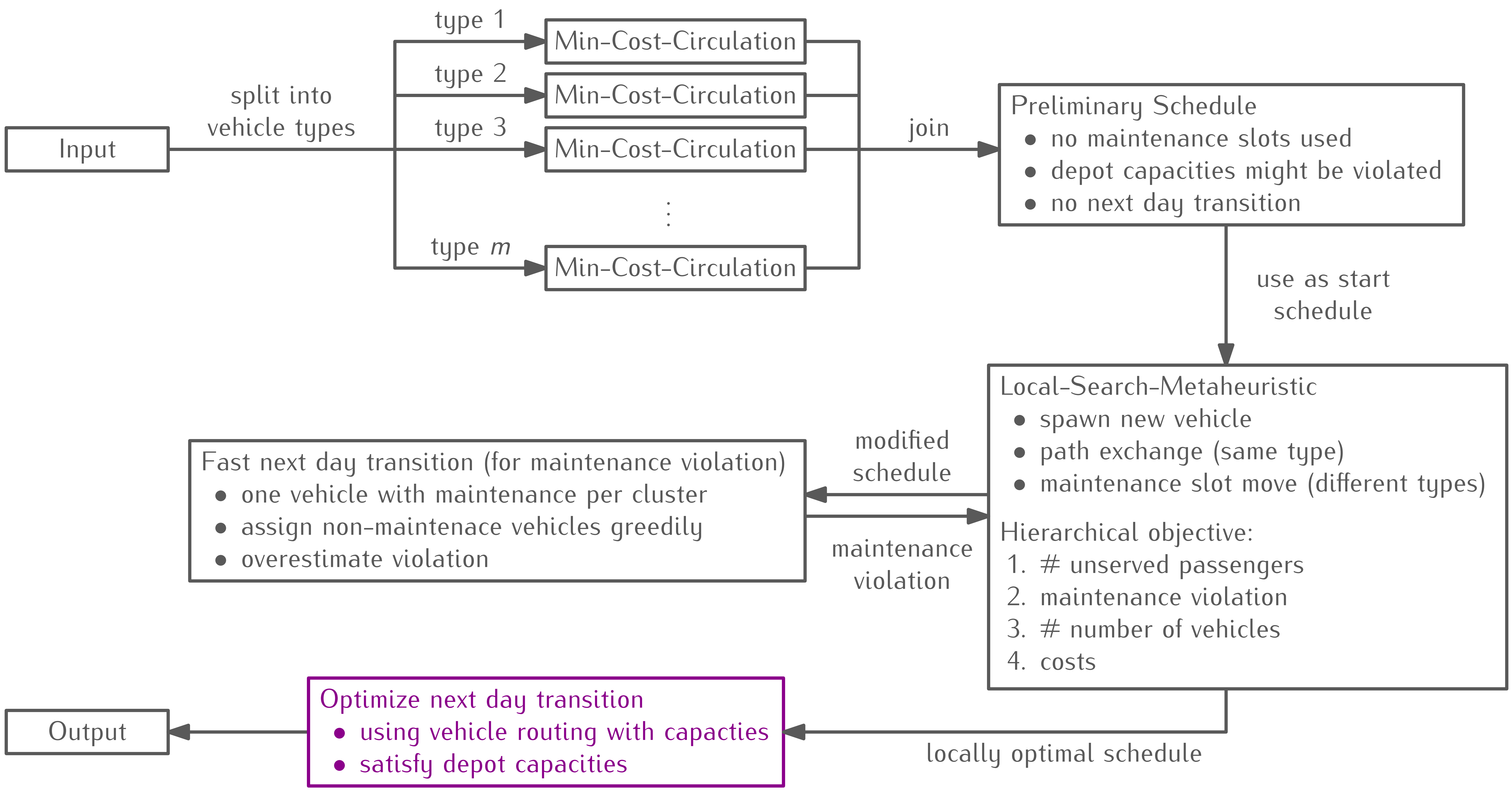
maintenance vehicles:
(decreasing)

non-maintenance vehicles:
(decreasing)



(dead-head trips between the depots are ignored for the slides)

Phase 2 - Algorithm Overview



Preliminary Schedule
- no maintenance slots used
- depot capacities might be violated
- no next day transition

Local-Search-Metaheuristic
- spawn new vehicle
- path exchange (same type)
- maintenance slot move (different types)

Hierarchical objective:
1. # unserved passengers
2. maintenance violation
3. # number of vehicles
4. costs

Fast next day transition (for maintenance violation)
- one vehicle with maintenance per cluster
- assign non-maintenace vehicles greedily
- overestimate violation

Optimize next day transition
- using vehicle routing with capacties
- satisfy depot capacities

Optimize Next Day Transition

Local Search (again):

- still: only one maintenance vehicle per cycle
- exchange vehicles between two cycles
- reoptimize cycles with 3-opt TSP solver

Optimize Next Day Transition

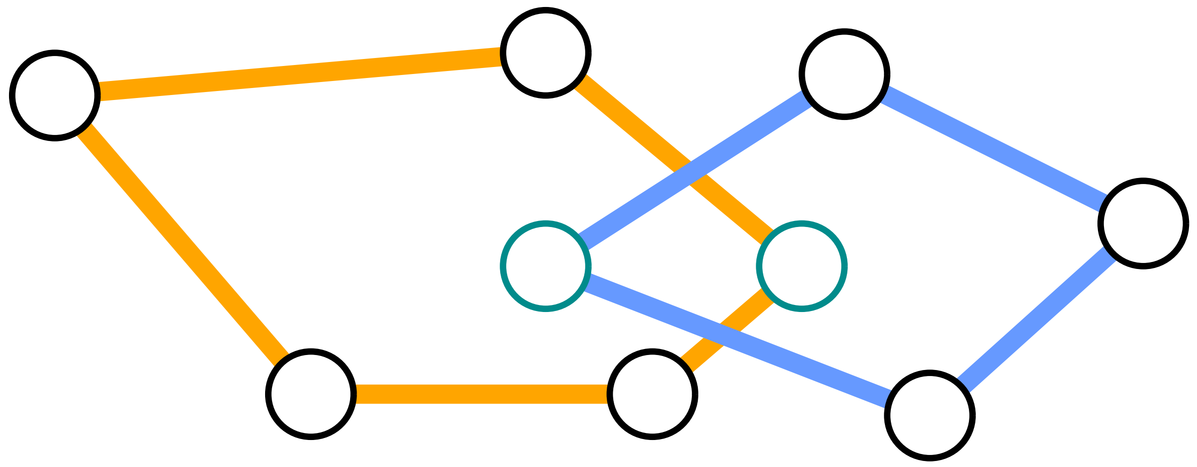
Local Search (again):

- still: only one maintenance vehicle per cycle
- exchange vehicles between two cycles
- reoptimize cycles with 3-opt TSP solver

Neighborhood:

```
for each cycle1:
    for each other cycle2:
        for each vehicle1 of cycle1
            for each vehicle2 of cycle2
                VehicleExchange(cycle1, vehicle1, cycle2, vehicle2)
                reoptimize cycle with a 3-opt-TSP-solver
                (another local search)
```

Optimize Next Day Transition



Local Search (again):

- still: only one maintenance vehicle per cycle
- exchange vehicles between two cycles
- reoptimize cycles with 3-opt TSP solver

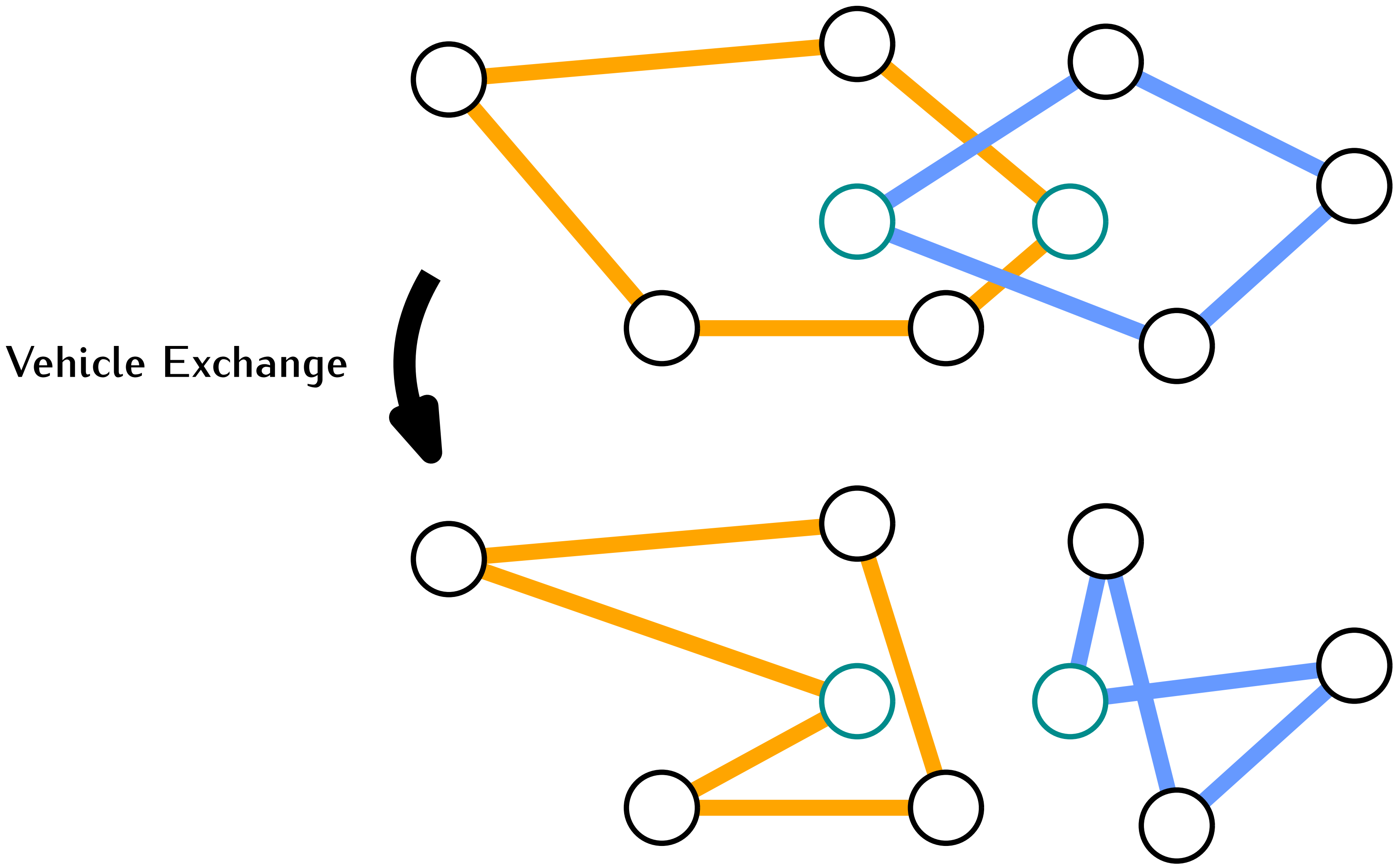
Neighborhood:

```
for each cycle1:
    for each other cycle2:
        for each vehicle1 of cycle1
            for each vehicle2 of cycle2
                VehicleExchange(cycle1, vehicle1, cycle2, vehicle2)
                reoptimize cycle with a 3-opt-TSP-solver
                (another local search)
```

Optimize Next Day Transition

Local Search (again):

- still: only one maintenance vehicle per cycle
- exchange vehicles between two cycles
- reoptimize cycles with 3-opt TSP solver



Neighborhood:

```
for each cycle1:
    for each other cycle2:
        for each vehicle1 of cycle1
            for each vehicle2 of cycle2
                VehicleExchange(cycle1, vehicle1, cycle2, vehicle2)
                reoptimize cycle with a 3-opt-TSP-solver
                (another local search)
```

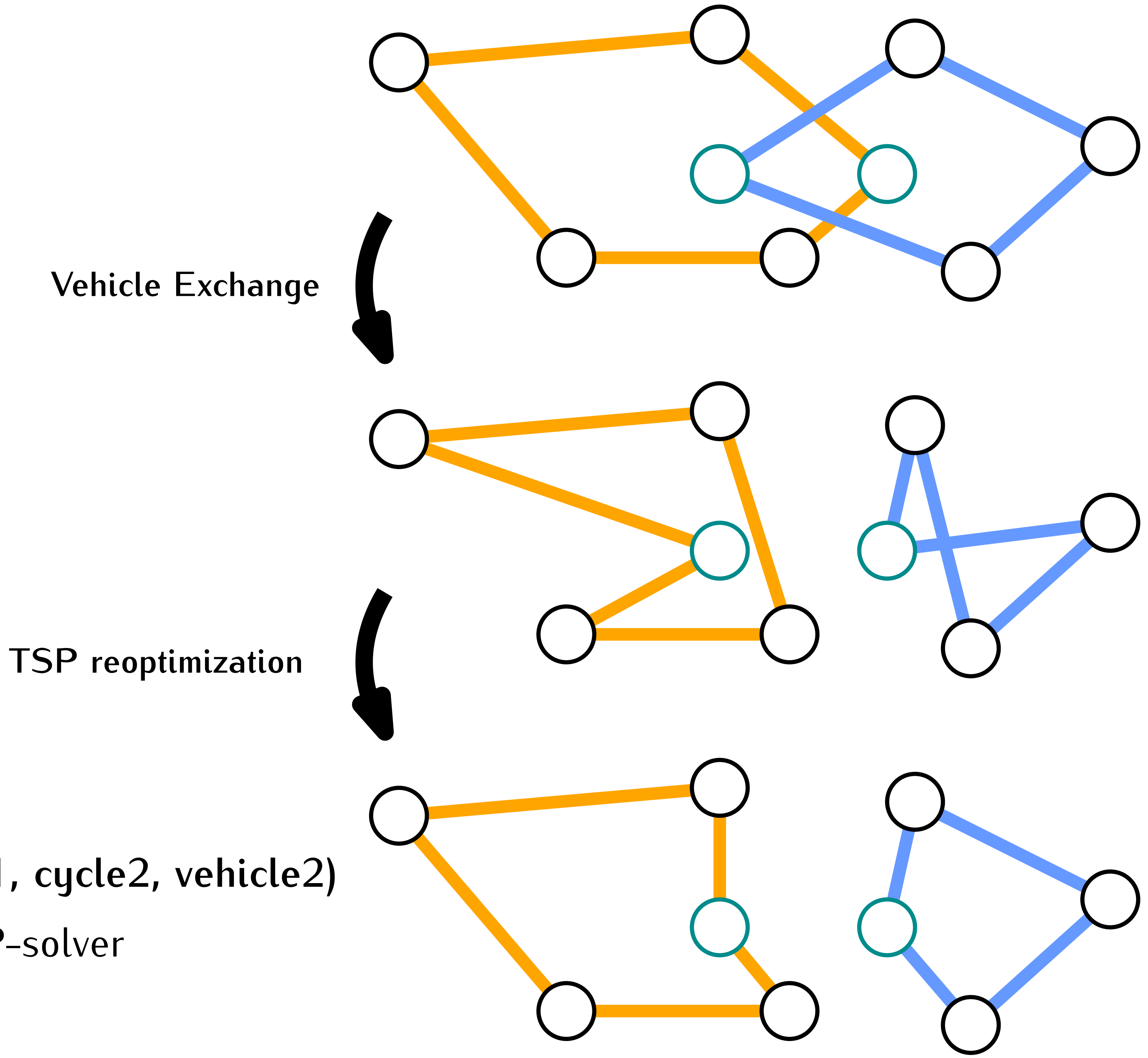
Optimize Next Day Transition

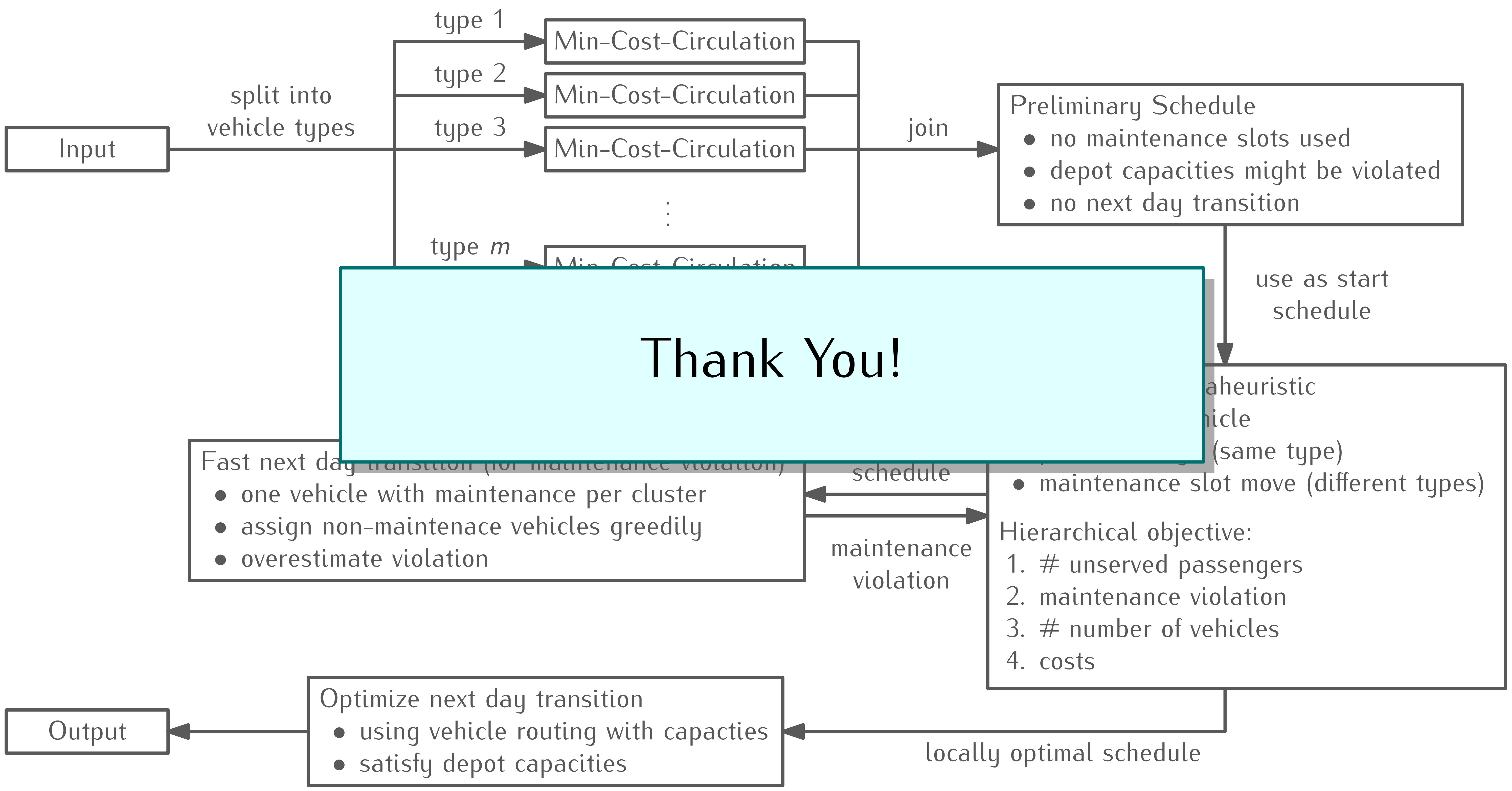
Local Search (again):

- still: only one maintenance vehicle per cycle
- exchange vehicles between two cycles
- reoptimize cycles with 3-opt TSP solver

Neighborhood:

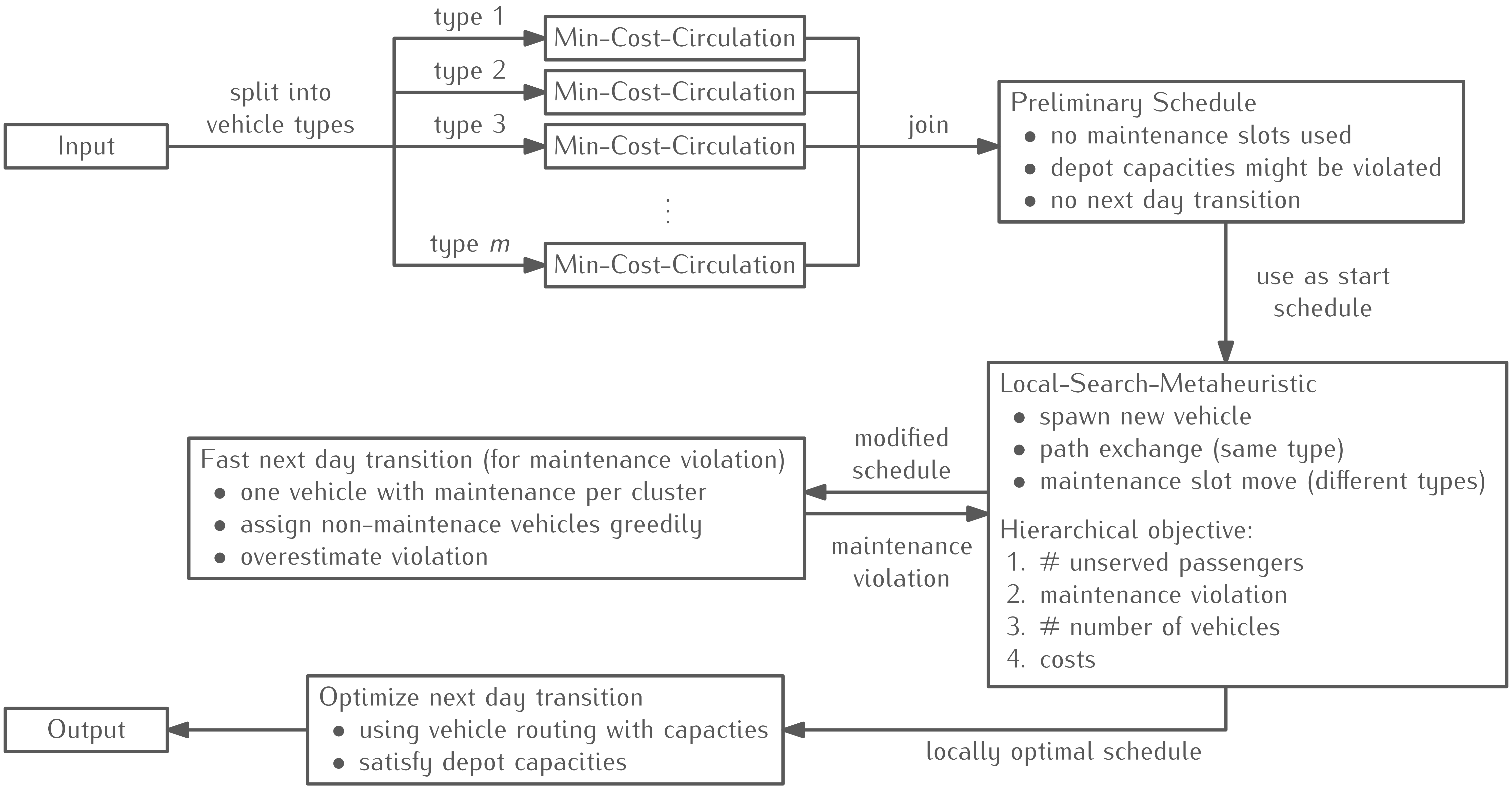
```
for each cycle1:
    for each other cycle2:
        for each vehicle1 of cycle1
            for each vehicle2 of cycle2
                VehicleExchange(cycle1, vehicle1, cycle2, vehicle2)
                reoptimize cycle with a 3-opt-TSP-solver
                (another local search)
```





Input
split into vehicle types
type 1
type 2
type 3
type m
Min-Cost-Circulation
Min-Cost-Circulation
Min-Cost-Circulation
Min-Cost-Circulation
join
Preliminary Schedule
no maintenance slots used
depot capacities might be violated
no next day transition
use as start schedule
aheuristic
icle
(same type)
maintenance slot move (different types)
Hierarchical objective:
1. # unserved passengers
2. maintenance violation
3. # number of vehicles
4. costs
schedule
maintenance violation
Fast next day transition (for maintenance violation)
one vehicle with maintenance per cluster
assign non-maintenace vehicles greedily
overestimate violation
locally optimal schedule
Optimize next day transition
using vehicle routing with capacties
satisfy depot capacities
Output

Thank You!


Input
split into vehicle types
type 1
type 2
type 3
type m
Min-Cost-Circulation
Min-Cost-Circulation
Min-Cost-Circulation
⋮
Min-Cost-Circulation
join
Preliminary Schedule
● no maintenance slots used
● depot capacities might be violated
● no next day transition
use as start schedule
Local-Search-Metaheuristic
● spawn new vehicle
● path exchange (same type)
● maintenance slot move (different types)
Hierarchical objective:
1. # unserved passengers
2. maintenance violation
3. # number of vehicles
4. costs
modified schedule
maintenance violation
Fast next day transition (for maintenance violation)
● one vehicle with maintenance per cluster
● assign non-maintenace vehicles greedily
● overestimate violation
locally optimal schedule
Optimize next day transition
● using vehicle routing with capacties
● satisfy depot capacities
Output