Lab Unit 6

Path, route and distribution planning as optimisation problems

Optimización de problemas de distribución y rutas

Contents Part 1

- Use of OptaPlanner

http://www.optaplanner.org/

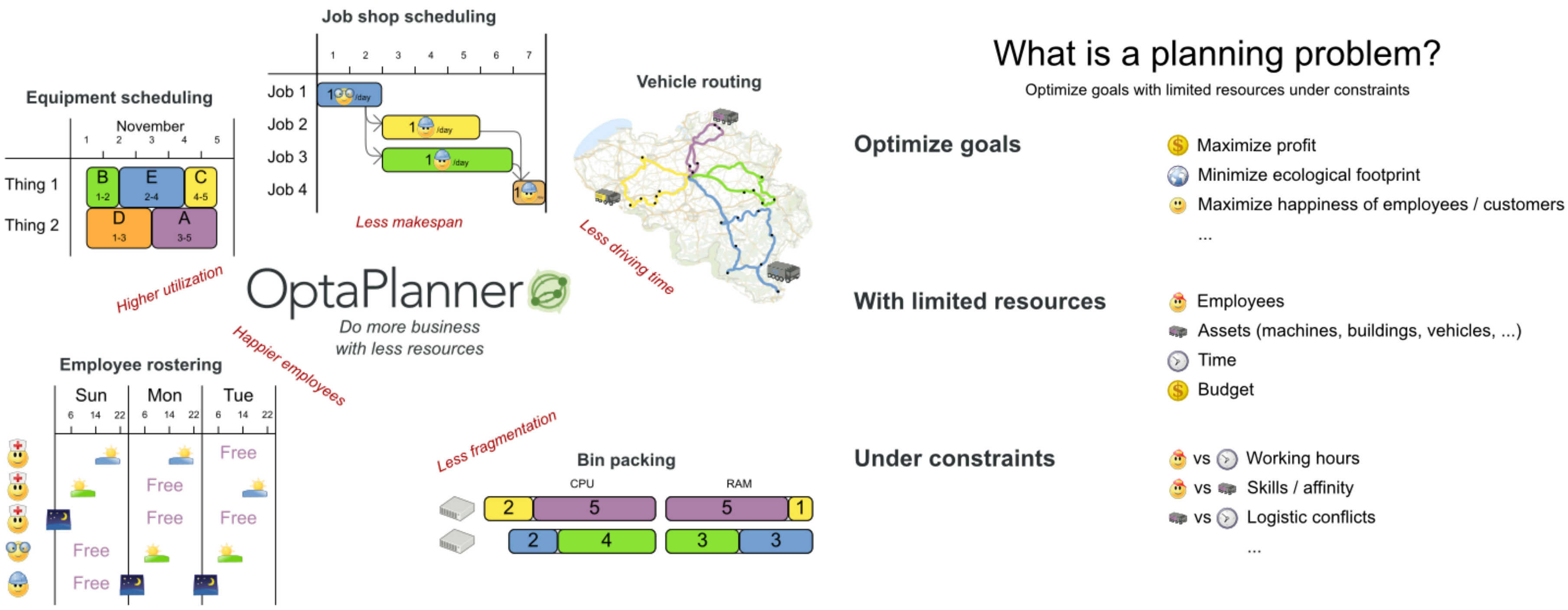
What’s Optaplanner?

- OptaPlanner is a **constraint satisfaction solver.** It optimises business resource planning
- OptaPlanner optimises planning problems to **do more business with less resources**
- OptaPlanner is:
  - a lightweight, embeddable planning engine
  - open source software
  - written in 100% pure Java
  - runs on any JVM

What's Optaplanner?

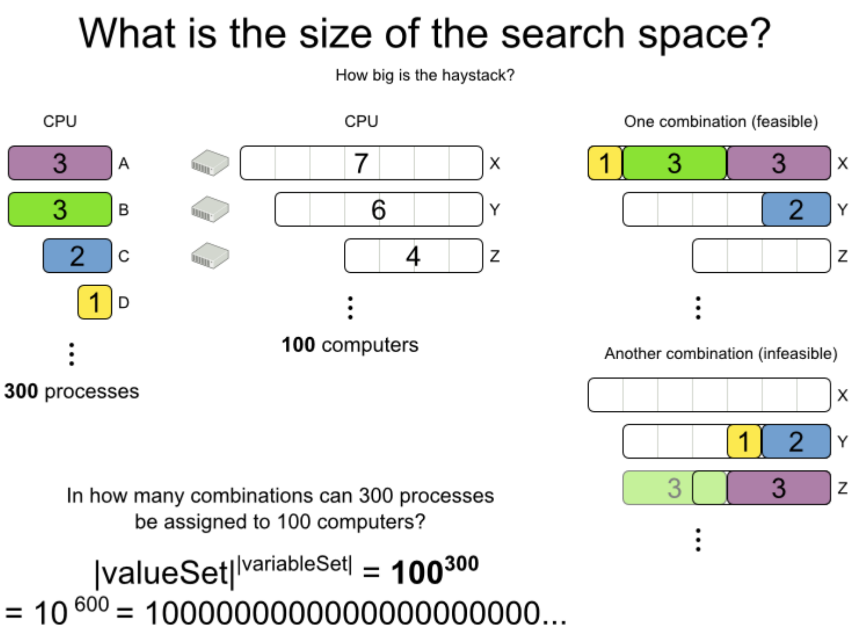
Use cases

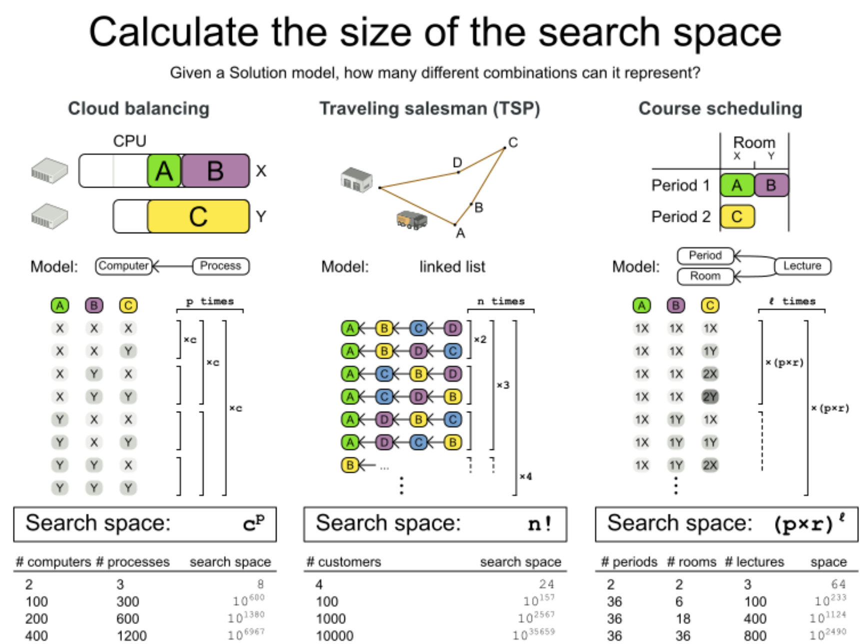
- Vehicle Routing, Employee Rostering, Job Scheduling, Bin Packing and more



Use cases

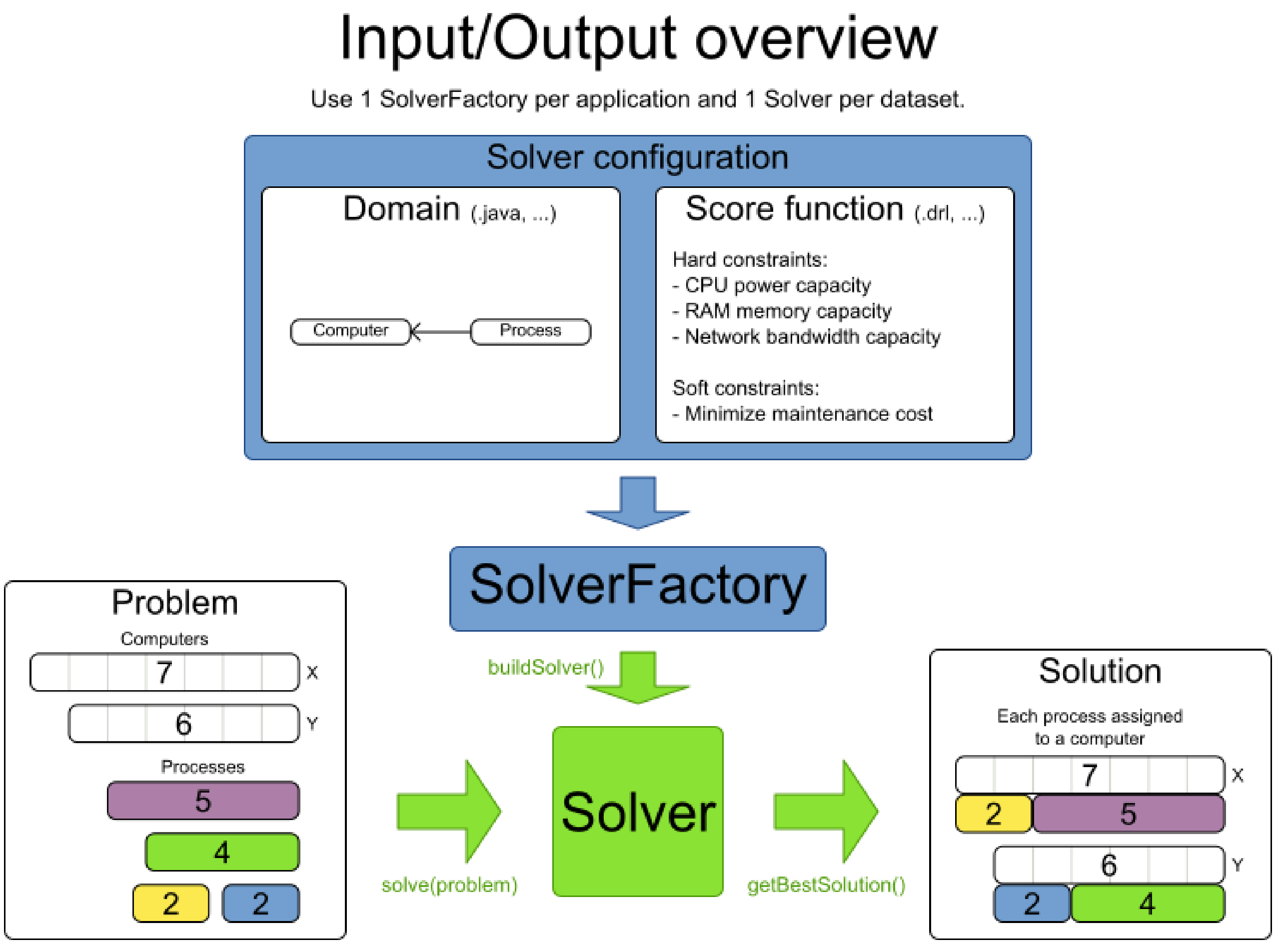
- Very useful in (combinatorial) optimization problems



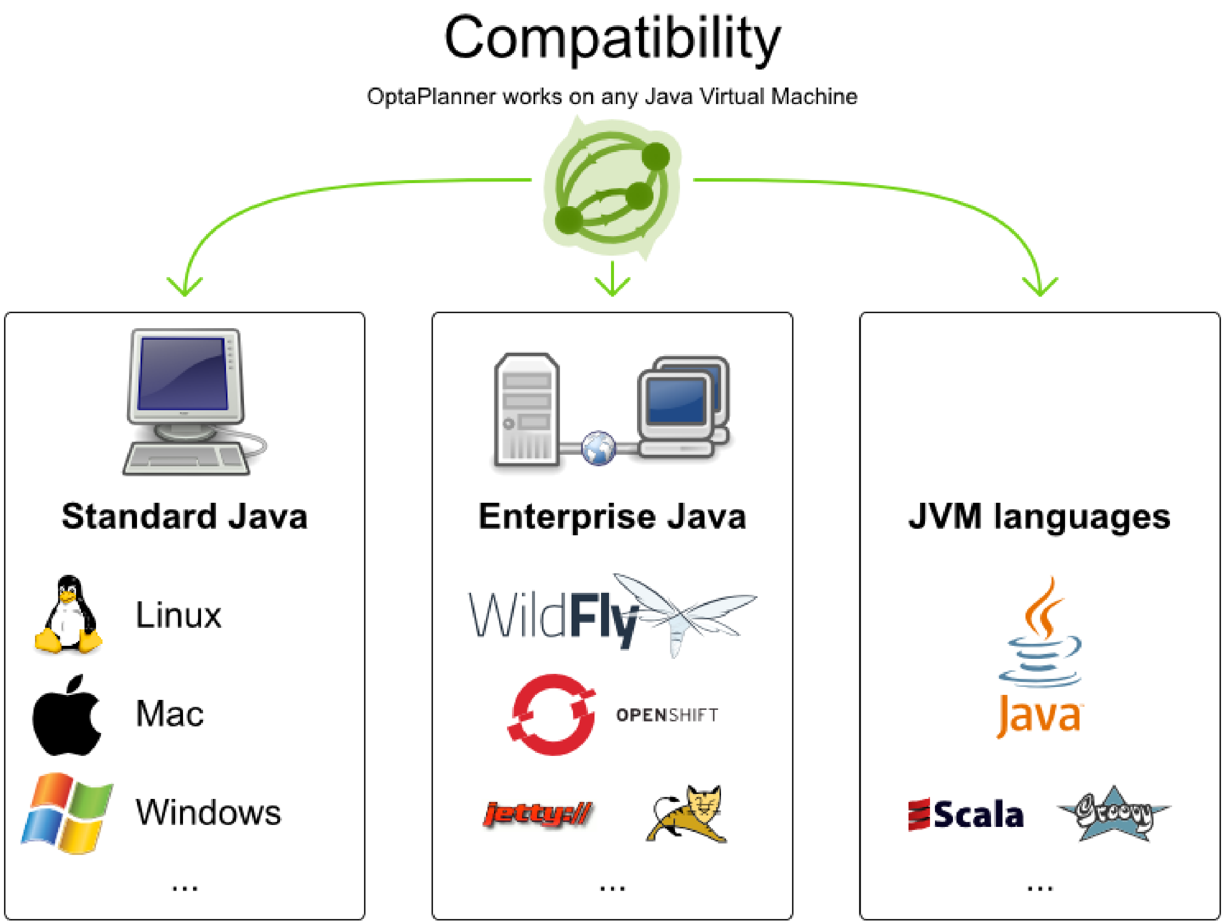


Usage

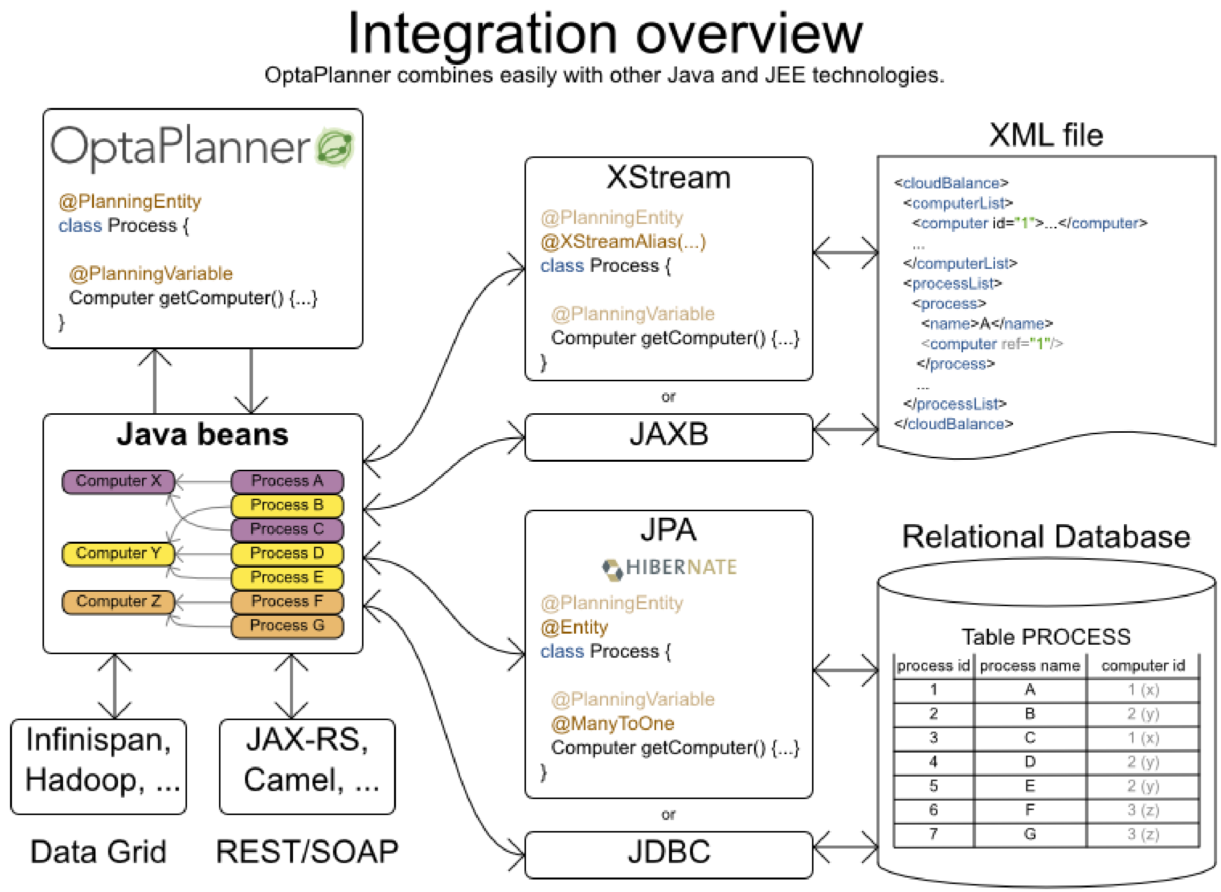
- UML class diagrams
- XML models and configurations
- Java classes
- Factory classes to solve the problems



Compatibility requirements



Integration

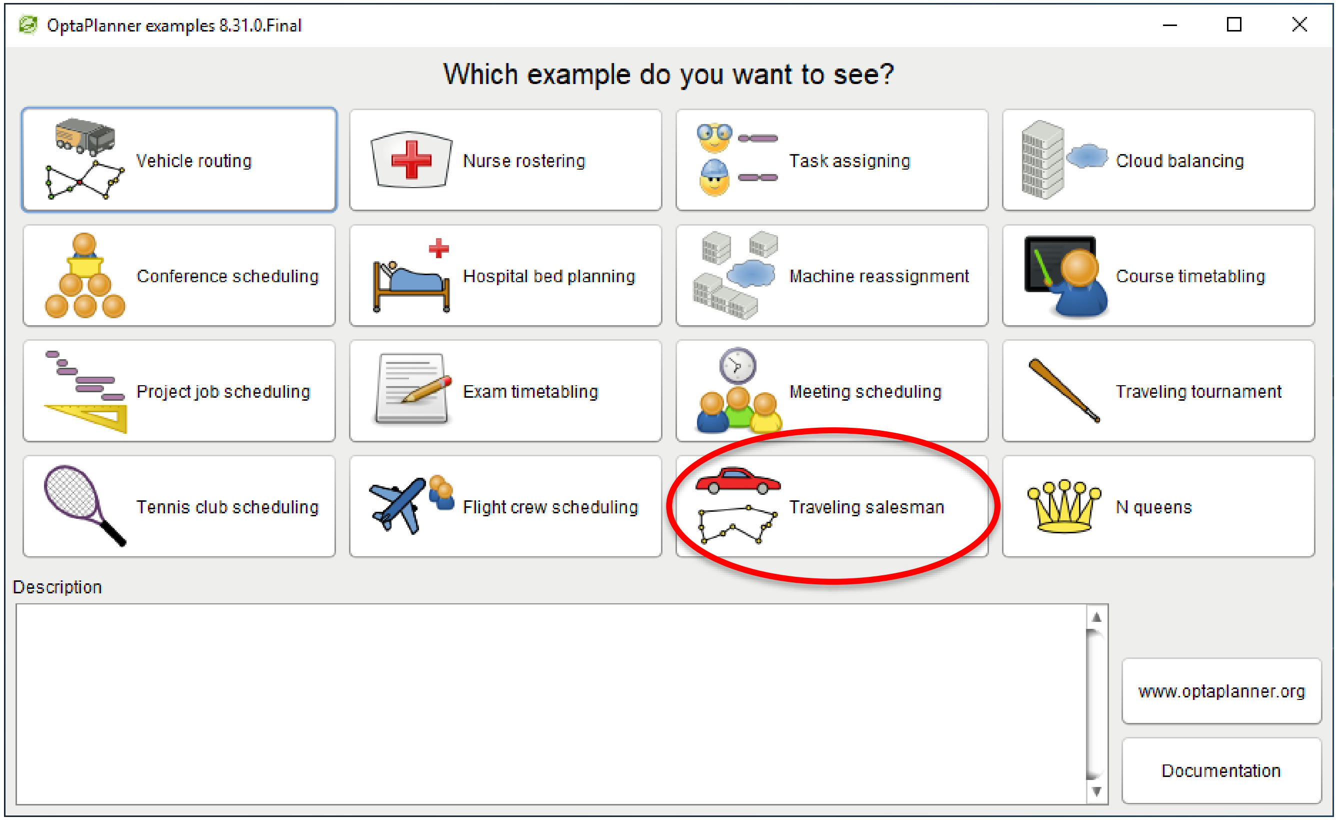


A lot of extra information

- User Guides and learning documentation
  - docs.jboss.org/optaplanner/release
  - www.optaplanner.org/learn/documentation.html
- Videos
  - www.optaplanner.org/learn/video.html

Task 1. Testing OptaPlanner

[optaplanner-path]\runExamples.bat o runQuickstarts.bat

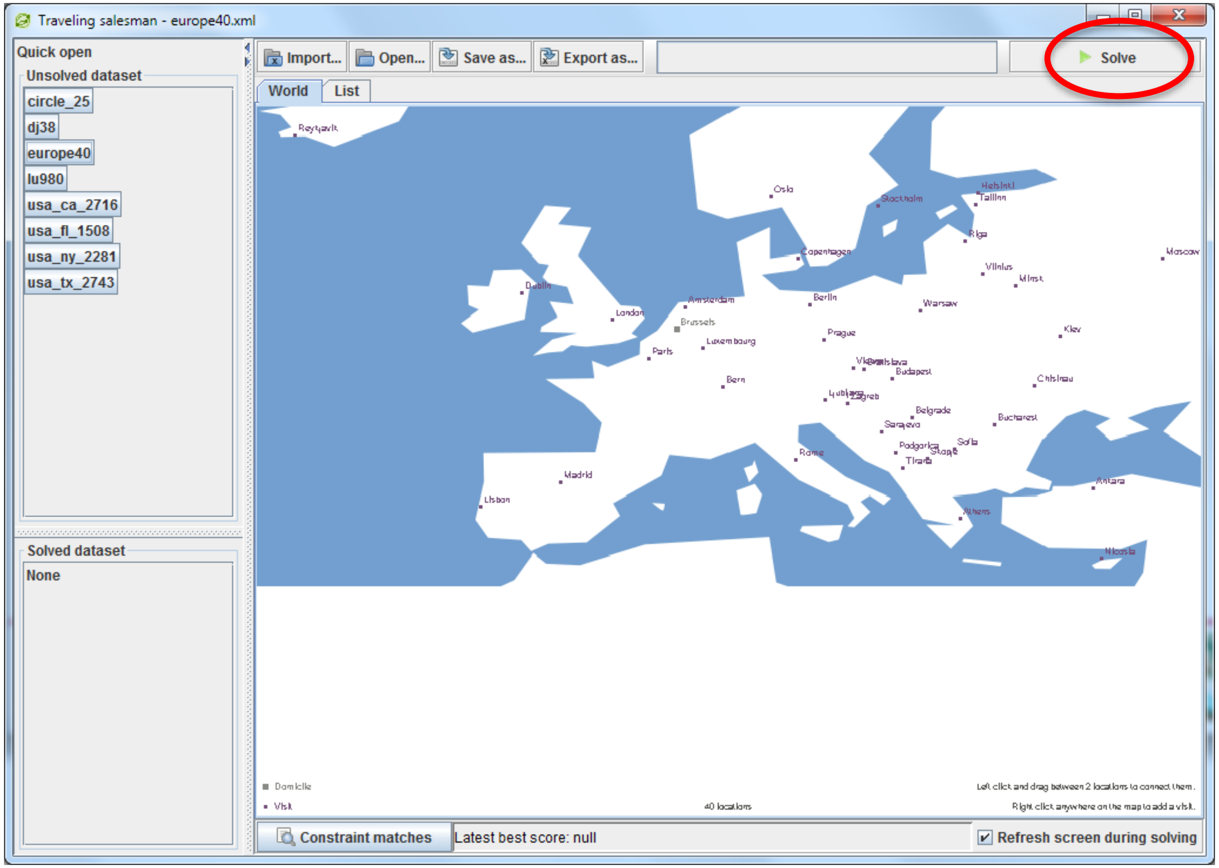


Task 1. Testing OptaPlanner

Exercise 1. Traveling Salesman Person (TSP)

Configuring **several examples**:

- graphically with the mouse in the GUI (much better than modifying the XML file: [optaplanner-path]\examples\sources\data\tsp\unsolved)
- under a MIP strategy (from the List tab)

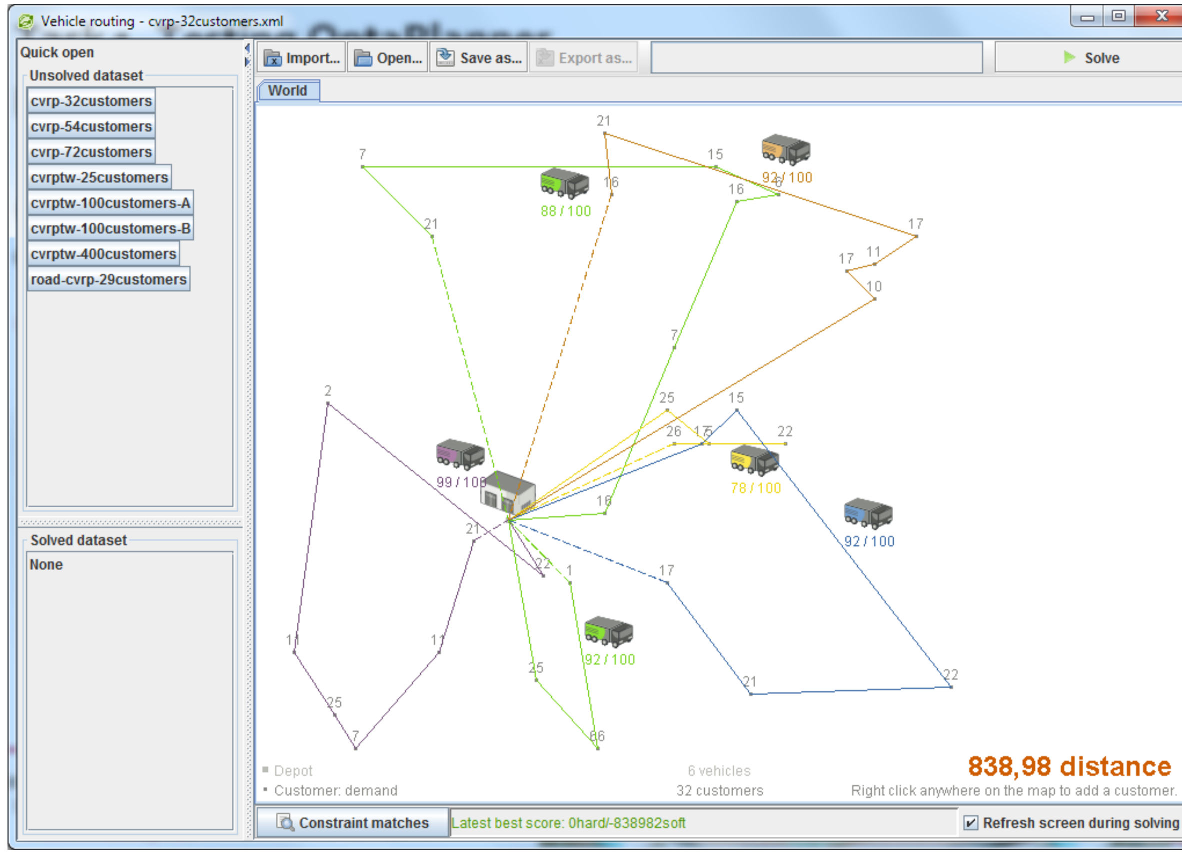


Task 1. Testing OptaPlanner

Exercise 2. Capacitated Vehicle Routing (CVRP)

Configuring **several examples**:

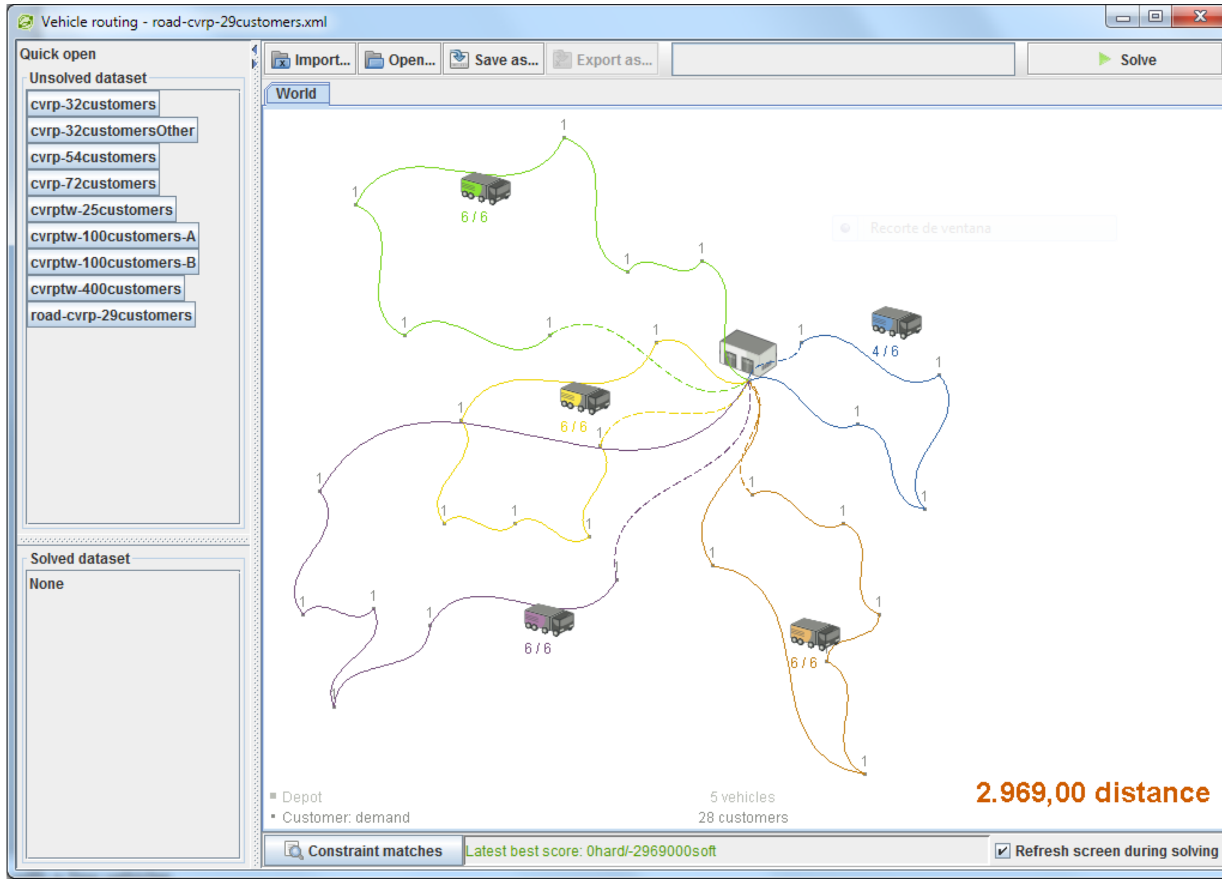
- graphically with the mouse in the GUI to add a customer (much better than modifying the XML file: [optaplanner-path]\examples\sources\data\vehiclerouting\unsolved)



Task 1. Testing OptaPlanner

Exercise 2bis. Capacitated Vehicle Routing

It also allows the user to define more realistic **roads**, rather than simply straight lines between two locations by means of more entries (see the .xml configuration file **road-cvrp-29customers**)

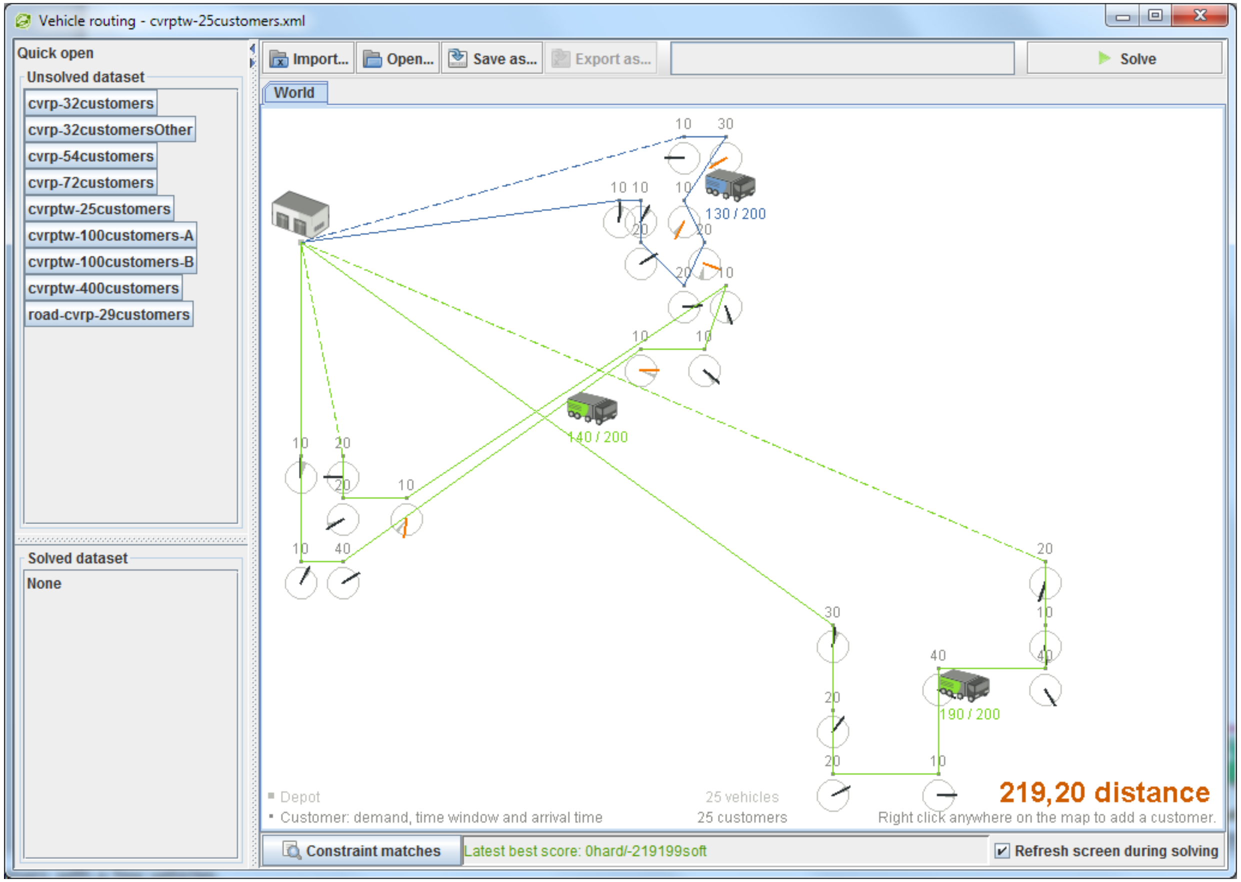


Task 1. Testing OptaPlanner

Exercise 3. Capacitated Vehicle Routing with Time Windows

Configuring **several examples**:

- graphically with the mouse in the GUI to add a customer (much better than modifying the XML file: [optaplanner-path]\examples\sources\data\vehiclerouting\unsolved)



Task 2. Testing OptaPlanner

Exercise 1. Scheduling and Timetable optimisation

Assign available spots to teams. Each team must play an almost equal number of times. Each team must play against each other team an almost equal number of times.

Multi-mode resource-constrained multi-project scheduling problem (MRCMPSP). Schedule all jobs in time and execution mode. Minimise project delays.

Curriculum course scheduling. Assign lectures to periods and rooms.

Examination timetabling. Assign exams to timeslots and rooms.

More info and examples in:

OptaPlanner playlist on YouTube

Some examples:

- Traveling Salesman Problem
- Vehicle Routing with Time Windows
- Tennis Club Scheduling
- Exam timetabling
- Course scheduling
- Project Job Scheduling
- Employee rostering

Contents Part 2

- Use of Planning techniques and PDDL (Planning Domain Definition Language)
- Modelling and solving a real-world logistics problem

PDDL (Planning Domain Definition Language) consists of **two** plain text **files** *http://cs-www.cs.yale.edu/homes/dvm*

- **Domain**, defines the predicates, functions and actions that can be planned, no matter the particular problem
- **Problem**, defines the initial state and the goals in terms of the predicates of the domain; it also includes the metric to be optimised
- 1 domain associated to many problems; 1 problem only to 1 domain

```
(define (domain <domain name>)
   <PDDL code for predicates>
   <PDDL code for first action>
   [...]
   <PDDL code for last action>
)
```

```
(define (problem <problem name>)
   (:domain <domain name>)
   <PDDL code for objects>
   <PDDL code for initial state>
   <PDDL code for goal specification>
)
```


PDDL domain in more detail

(define (domain <name>)

(:requirements <:req 1>… <:req n>) *; planning requirements*

(:types <subtype1>… <subtype n> – <type1> <typen>) *; types & subtypes to be used*

(:constants <cons1> … <consn>) *; constants to be used*

(:predicates <p1> <p2>… <pn>) *; predicates (true/false info) – for objets*

(:functions <f1> <f2>… <fn>) *; functions (fluents or numeric info) – for resources*

PDDL domain in more detail

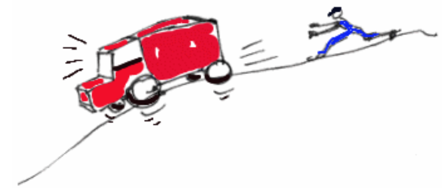
```
; example of one action
(:durative-action act1        ; action with duration
    :parameters (?par1 – <subtype1> ?par2 – <subtype2> ...) ; the needed parameters
    :duration <value>    ; duration of the action
    :condition (and (at start (<condition_1>))  ; "at start", "over all", "at end"
                    (over all (<condition_2>))
                    ...
                    (at end (<condition_n>)))
    :effect (and    (at start (<effect_1>))     ; "at start", "at end"
                    (at end (<effect_2>))
                    ...
                    (at end (not (<effect_n>))))
)
```


PDDL problem in more detail

(define (problem <name>)

(:domain <name >)	*; domain this problem belongs to*
(:objects <obj$_1$> - <type$_1$> ... <obj$_n$> - <type$_n$>)	*; objects and their types*
(:init	*; initial state*
(<predicate$_1$>) ... (<predicate$_i$>)	*; true/false predicates (propositional)*
(= <function$_1$> <value$_1$>) ... (= <function$_n$> <value$_n$>))	*; numeric info*
(:goal	*; goals*
(and ((<predicate$_1$>) ... (<predicate$_i$>)	*; propositional goals*
(<operator$_1$> <function$_1$> <value$_1$>) ...	
(<operator$_j$> <function$_j$> <value$_j$>)))	*; numeric goals*
(:metric minimize|maximize <expression>**))**	*; metric to min/max – plan quality*

Driverlog scenario (mono-modal) as a **basis**



This domain has **drivers** that can **walk** between **locations** and **trucks** that can **drive** between locations.

Walking requires traversal of **different paths from** those **used for driving**, and there is always one intermediate location on a footpath between two road junctions.

The **trucks** can be **loaded or unloaded** with **packages** (with or without a **driver** present) and the **objective** is to **transport packages** between locations, ending up with a subset of the packages, the trucks and the drivers **at specified destinations**.

The **metric** adds **costs** for walking and driving and problem instances required that the planner **optimise** some **linear combination** of them

LPG-td (http://zeus.ing.unibs.it/lpg/)

- Heuristic local search that uses planning graphs to calculate estimates
- Non-deterministic (random behaviour). To do: run it several times and get the median solution

Use (en Windows con cygwin1.dll):

./lpg-td –o domain-file.pddl –f problem-file.pddl –n 1 *;number of solutions*

e.g. -n 10 keeps searching until 10 solutions have been found (see .SOL files).

Be careful with this, because that number of solutions may not be found!

Other mutually exclusive options: -speed | -quality

LPG-td (http://zeus.ing.unibs.it/lpg/)

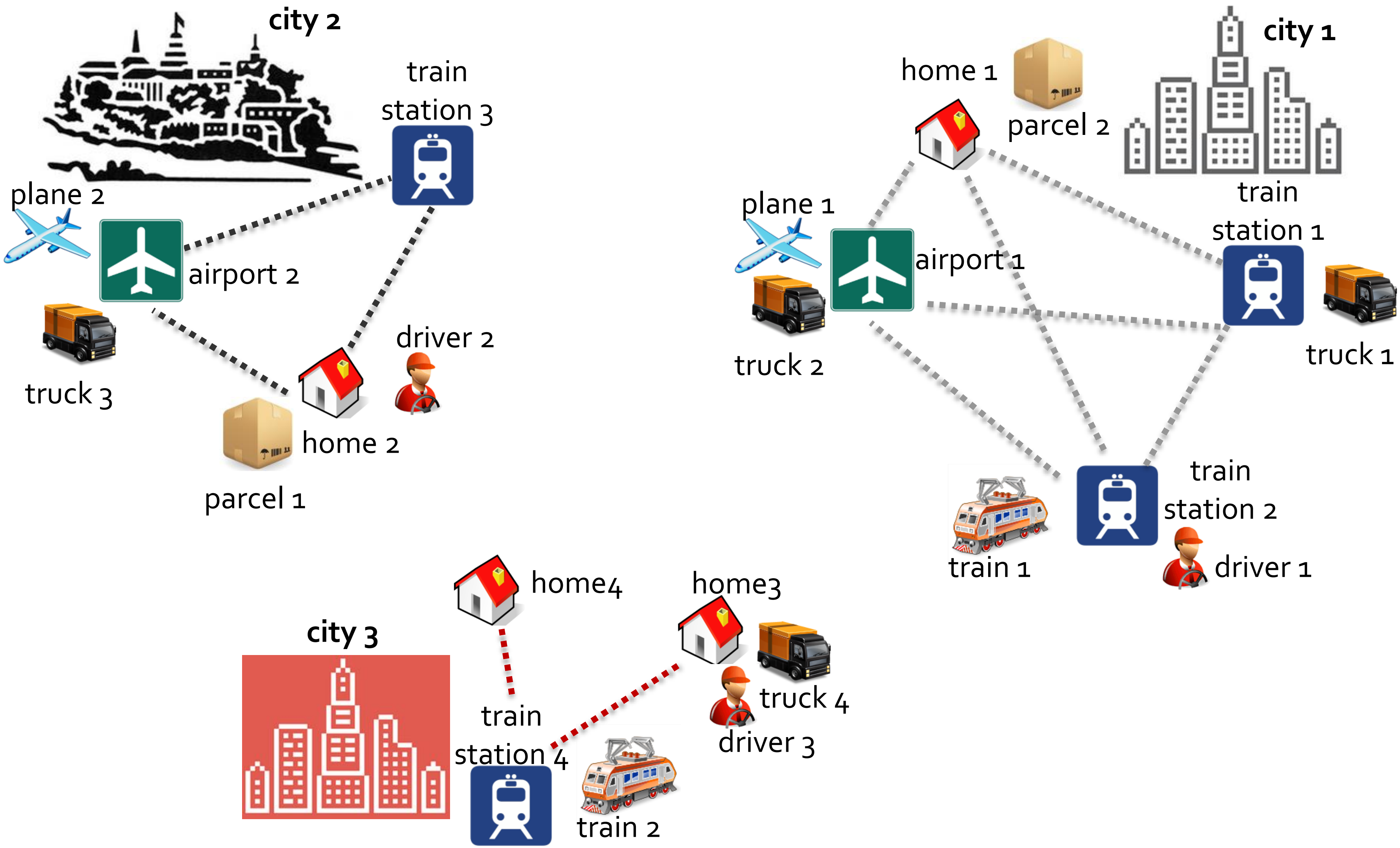
An example of execution:

timestamp: action1 [duration, cost]
timestamp: action2 [duration, cost]
...

```
Plan computed:
   Time: (ACTION) [action Duration; action Cost]
 0.0000: (WALK DRIVER1 S2 P1-2) [D:79.0000; C:0.1000]
 79.0000: (WALK DRIVER1 P1-2 S1) [D:29.0000; C:0.1000]
 108.0000: (WALK DRIVER1 S1 P1-0) [D:43.0000; C:0.1000]
 151.0000: (WALK DRIVER1 P1-0 S0) [D:80.0000; C:0.1000]
 231.0000: (BOARD-TRUCK DRIVER1 TRUCK1 S0) [D:1.0000; C:0.1000]
 232.0000: (DRIVE-TRUCK TRUCK1 S0 S1 DRIVER1) [D:70.0000; C:0.1000]
 302.0000: (DISEMBARK-TRUCK DRIVER1 TRUCK1 S1) [D:1.0000; C:0.1000]


Solution number: 1
Total time:      0.00
Search time:     0.00
Actions:         7
Execution cost:  0.70
Duration:        303.000
Plan quality:    303.000
     Plan file:       plan_pfile1_1.SOL
```

actions of the plan
makespan
quality, according to the metric


city 2
train station 3
plane 2
airport 2
truck 3
driver 2
home 2
parcel 1
home 1
parcel 2
city 1
plane 1
airport 1
train station 1
truck 2
truck 1
train station 2
train 1
driver 1
home4
home3
city 3
truck 4
driver 3
train station 4
train 2

Some additional constraints to model in the actions

- Trucks can only move between locations of the same city; trucks need a driver that can also walk between locations of the same city
- Trucks have capacity constraints on the weight/volume or type of parcel to be transported
- Trains/planes only move between train stations/airports (no matter the city)

- Problem **goal**: transport parcel1 and parcel2 to home4 in city3

Some additional constraints to model in the actions

- **Define the PDDL domain+problem files (with durative actions and metric)**

- Additional (valuable) **extensions:**
  - Paths between locations within a city
  - Numeric cost & capacities in trains/planes
  - Use of additional resources, e.g. pilot, crane to (un)load, extra staff, etc.
  - Different types of parcels and delivery deadlines, etc.

Contents Part 3

- Use of Geographical Information Systems (GIS) to improve our models

There are many possibilities to get **real data to improve our models**

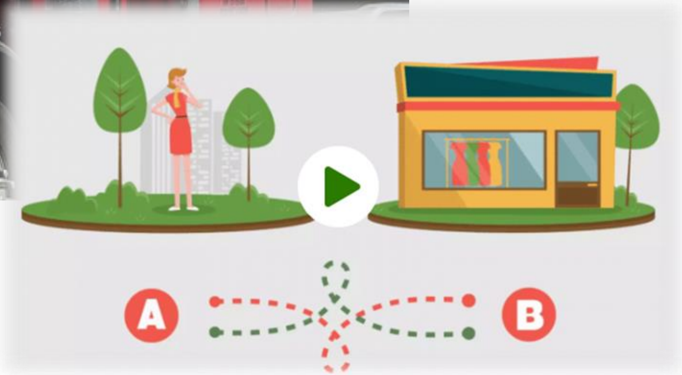
- Duration, vehicle routing, different paths, directions and geocoding (conversions to/from coordinates), etc.
- (Real time) information of roads

Use of **APIs** for retrieving information from **on-line spatial and geographic databases**, such as GraphHopper, Google Maps, OpenStreetMap, Bing Maps, FIWARE lab, etc.

Example 1. GraphHopper

https://graphhopper.com

graphhopper



Geocoding API

Get coordinates of an address/place (and reverse)

https://graphhopper.com/api/1/**geocode**?

q=universitat%20politecnica%20valencia&

locale=de&debug=true&key=[YOUR_KEY]

In JSON and XML


```
{
    "hits": [
        {
            "osm_id": 29701959,
            "osm_type": "W",
            "extent": [
                -0.1667901,
                38.9970655,
                -0.1653129,
                38.9955392
            ],
            "country": "Spanien",
            "osm_key": "amenity",
            "city": "Gandia",
            "street": "Polígono Universidad",
            "osm_value": "university",
            "postcode": "46730",
            "name": "Universitat Politècnica de València",
            "state": "Valencianische Gemeinschaft",
            "point": {
                "lng": -0.16604533062880328,
                "lat": 38.9964155
            }
        },
        {
            "osm_id": 3373439347,
            "osm_type": "N",
            "country": "Spanien",
            "osm_key": "railway",
            "city": "Valencia",
            "street": "Avinguda dels Tarongers",
            "osm_value": "tram_stop",
            "postcode": "46021",
            "name": "Universitat Politècnica",
            "state": "Valencianische Gemeinschaft",
            "point": {
                "lng": -0.3500157,
                "lat": 39.4812005
            }
        }
    ],
    "locale": "de"
}
```


Matrix API

graphhopper

Calculates many-to-many distances, times or routes efficiently

https://graphhopper.com/api/1/**matrix**?

point=49.932707%2C11.588051&

point=50.241935%2C10.747375&

point=50.118817%2C11.983337&

type=json&**vehicle=car**&debug=true&

out_array=**weights**&out_array=**times**&

out_array=**distances**&key=[YOUR_KEY]

name	description	Restrictions	Icon	Real life image
car	Car mode	car access		
motorcycle	Motor bike avoiding motorways	motorcycle access		
small_truck	Small truck like a Mercedes Sprinter, Ford Transit or Iveco Daily	height=2.7m, width=2+0.4m, length=5.5m, weight=2080+1400 kg		
bus	overland bus, no psv	height=3.54m, width=2.55+0.5m, length=10.34m, weight=9600 + 6000 kg		
truck	Truck like a MAN or Mercedes-Benz Actros	height=3.7m, width=2.6+0.5m, length=12m, weight=13000 + 13000 kg, hgv=yes, 3 Axes		
foot	Pedestrian or walking	foot access		

Matrix API

graphhopper

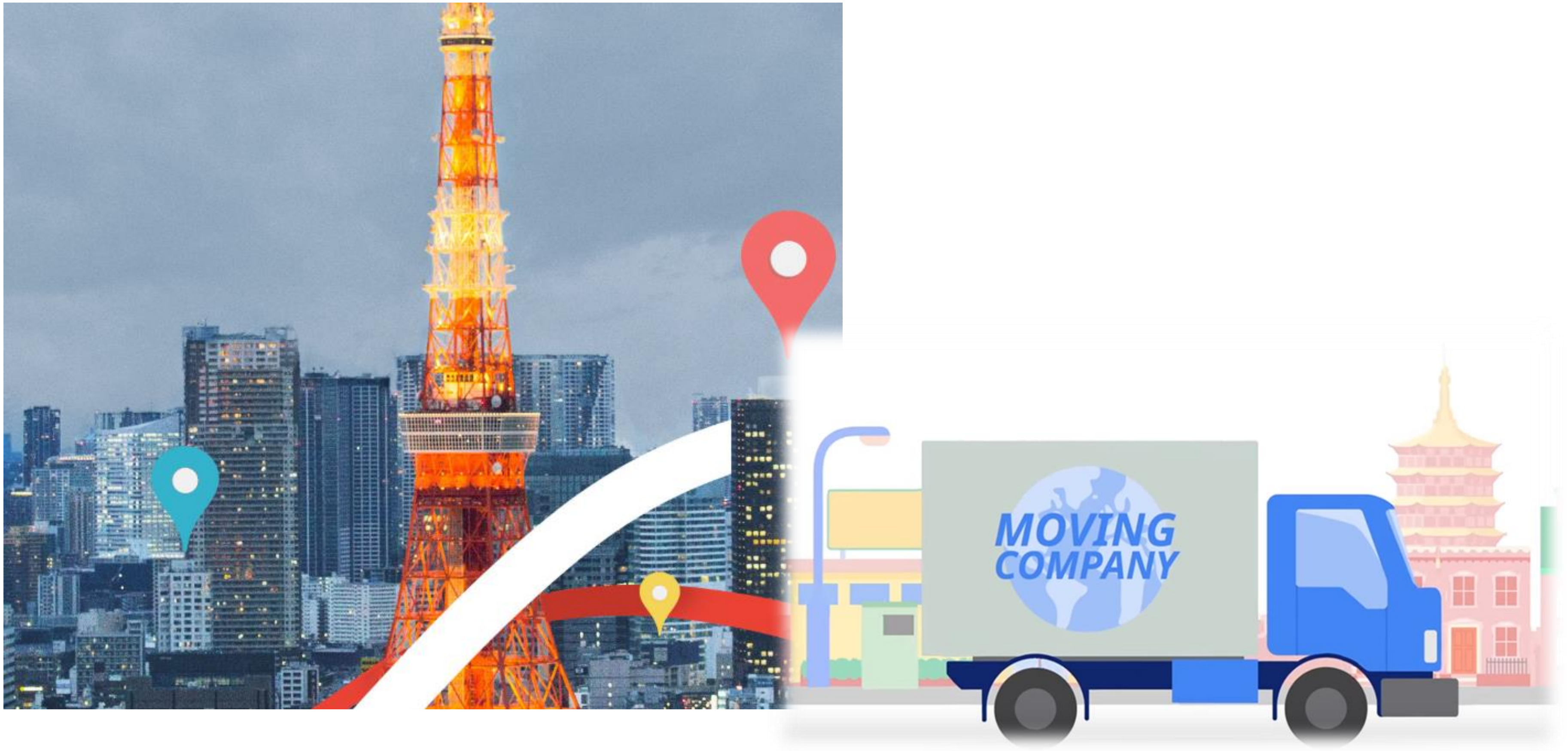
Calculates many-to-many distances, times or routes efficiently


```
{
  "distances" : [ [ 0, 104642, 48844 ], [ 104071, 0, 127780 ], [ 48958, 128132, 0 ] ],
  "times" : [ [ 0, 4199, 2816 ], [ 4086, 0, 5905 ], [ 2837, 6011, 0 ] ],
  "weights" : [ [ 0.0, 4198.993, 2815.981 ], [ 4086.019, 0.0, 5904.992 ], [ 2837.495, 6010.913, 0.0 ] ],
  "info" : {
    "copyrights" : [ "GraphHopper", "OpenStreetMap contributors" ]
  }
}
```


E.g. for distances matrix

0	104642	48844
104071	0	127780
48958	128132	0

https://developers.google.com/maps



Geocoding and elevation APIs

https://maps.googleapis.com/maps/api/geocode/xml?
address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&
key=[YOUR_KEY]


```xml
<GeocodeResponse>
 <status>OK</status>
 <result>
  <type>street_address</type>
  <formatted_address>1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA</form
  <address_component>
   <long_name>1600</long_name>
   <short_name>1600</short_name>
   <type>street_number</type>
  </address_component>
  <address_component>
   <long_name>Amphitheatre Pkwy</long_name>
   <short_name>Amphitheatre Pkwy</short_name>
   <type>route</type>
  </address_component>
  <address_component>
   <long_name>Mountain View</long_name>
   <short_name>Mountain View</short_name>
   <type>locality</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>San Jose</long_name>
   <short_name>San Jose</short_name>
   <type>administrative_area_level_3</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>Santa Clara</long_name>
   <short_name>Santa Clara</short_name>
```


...

In JSON and XML


```json
{
   "results" : [
      {
         "elevation" : 1608.637939453125,
         "location" : {
            "lat" : 39.73915360,
            "lng" : -104.98470340
         },
         "resolution" : 4.771975994110107
      }
   ],
   "status" : "OK"
}
```


Web Services

- Google Maps Directions API
- Google Maps Distance Matrix API [4]
- Google Maps Elevation API
- Google Maps Geocoding API
- Google Maps Geolocation API
- Google Maps Roads API
- Google Maps Time Zone API
- Google Places API Web Service