Optimization of snow plowing operations in Skinnarila area, Lappeenranta

Carl Assmann, *Bundeswehr University*

Matthew Geleta, *University of Oxford*

Edyta Kabat, *Jagellionian University of Cracow*

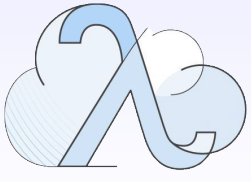
Albert Miguel Lopez, *Autonomous University of Barcelona*

Gandalf Saxe, *Technical University of Denmark*

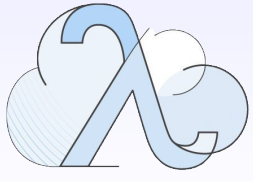
Sara Battiston, *University of Milan*

Instructor: Miika Tolonen, *Lappeenranta University of Technology*

ECMI Modelling Week, 15.07.2017

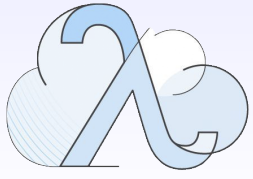


ECMI EUROPEAN CONSORTIUM FOR MATHEMATICS IN INDUSTRY



Motivation

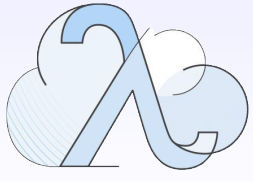
What is the shortest snow plowing route in Lappeenranta?



An instance of an old problem...

Chinese Postman Problem

To find the shortest closed path that visits every edge of a connected undirected graph.

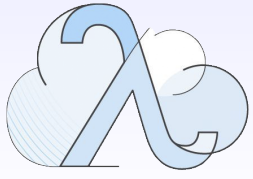


An instance of an old problem...

Chinese Postman Problem

To find the shortest closed path that visits every edge of a connected undirected graph.

- In our problem, **nodes** are intersections and **edges** are **road-segments**.

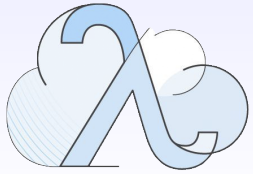


An instance of an old problem...

Chinese Postman Problem

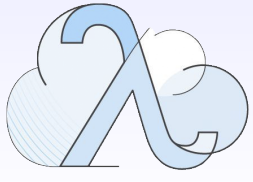
To find the shortest closed path that visits every edge of a connected undirected graph.

- In our problem, **nodes** are intersections and **edges** are **road-segments**.
- We want to find the route with the shortest total distance.



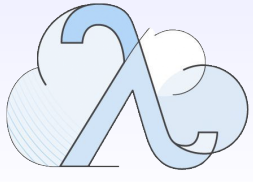
How to solve the CPP?

- If a graph G has an Eulerian circuit, that circuit solves CPP.



How to solve the CPP?

- If a graph G has an Eulerian circuit, that circuit solves CPP.
- In non-Eulerian graphs, the problem is to find the smallest number of edges to duplicate.

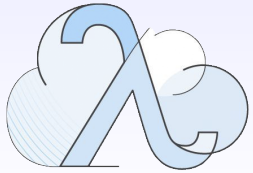


How to solve the CPP?

- If a graph G has an Eulerian circuit, that circuit solves CPP.
- In non-Eulerian graphs, the problem is to find the smallest number of edges to duplicate.

Algorithm:

- Is G Eulerian? Then find the Euler-circuit and stop.

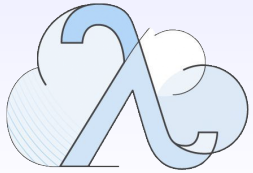


How to solve the CPP?

- If a graph G has an Eulerian circuit, that circuit solves CPP.
- In non-Eulerian graphs, the problem is to find the smallest number of edges to duplicate.

Algorithm:

- Is G Eulerian? Then find the Euler-circuit and stop.
- Otherwise, optimally augment G to get an Eulerian multigraph G'.

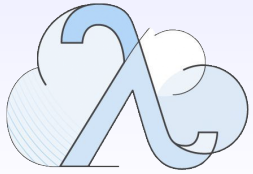


How to solve the CPP?

- If a graph G has an Eulerian circuit, that circuit solves CPP.
- In non-Eulerian graphs, the problem is to find the smallest number of edges to duplicate.

Algorithm:

- Is G Eulerian? Then find the Euler-circuit and stop.
- Otherwise, optimally augment G to get an Eulerian multigraph G'.
- Find an Euler-circuit in G'.



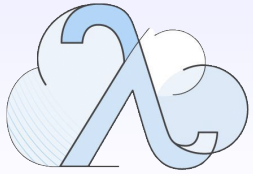
How to solve the CPP?

- If a graph G has an Eulerian circuit, that circuit solves CPP.
- In non-Eulerian graphs, the problem is to find the smallest number of edges to duplicate.

Algorithm:

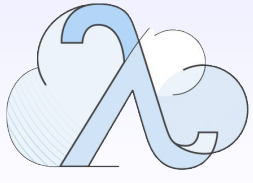
- Is G Eulerian? Then find the Euler-circuit and stop.
- Otherwise, optimally augment G to get an Eulerian multigraph G'.
- Find an Euler-circuit in G'.

The Lappeenranta road network is NOT Eulerian!



How to get Eulerian Multigraph G'?

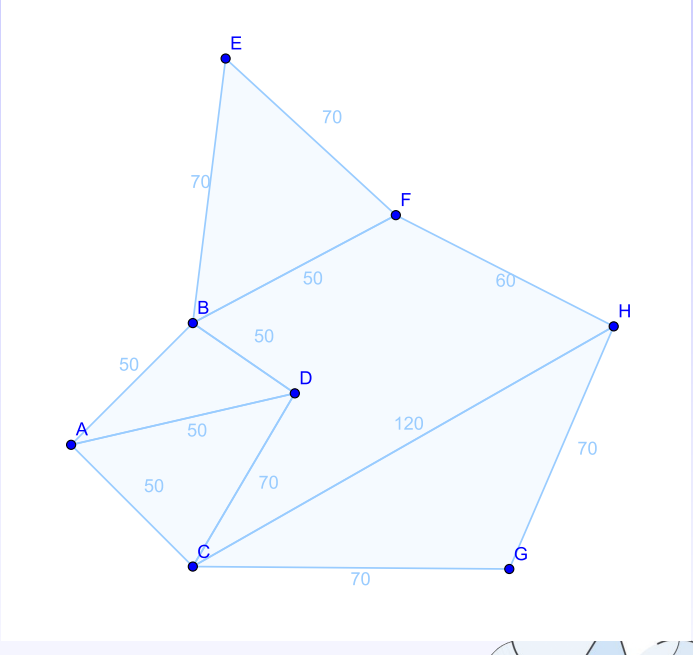
Naive idea:



How to get Eulerian Multigraph G'?

Naive idea:

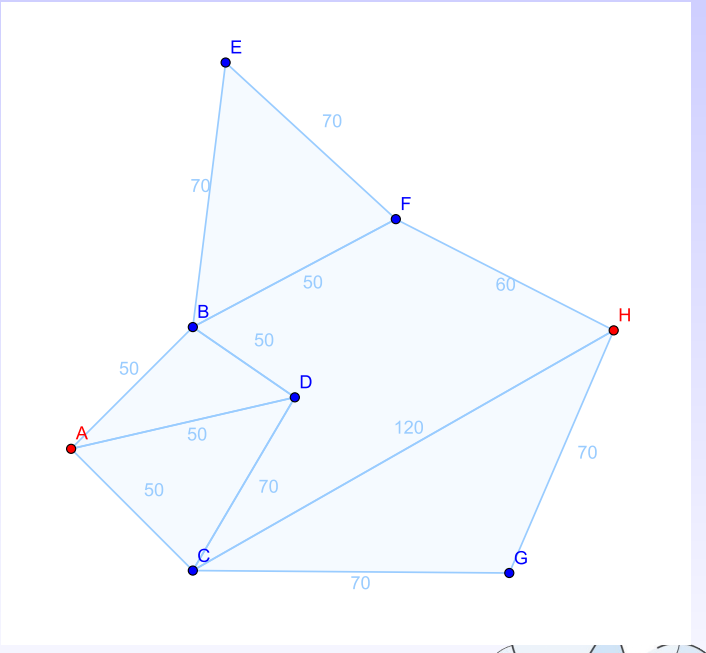
1. Consider a connected undirected graph G with weighted edges.



How to get Eulerian Multigraph G'?

Naive idea:

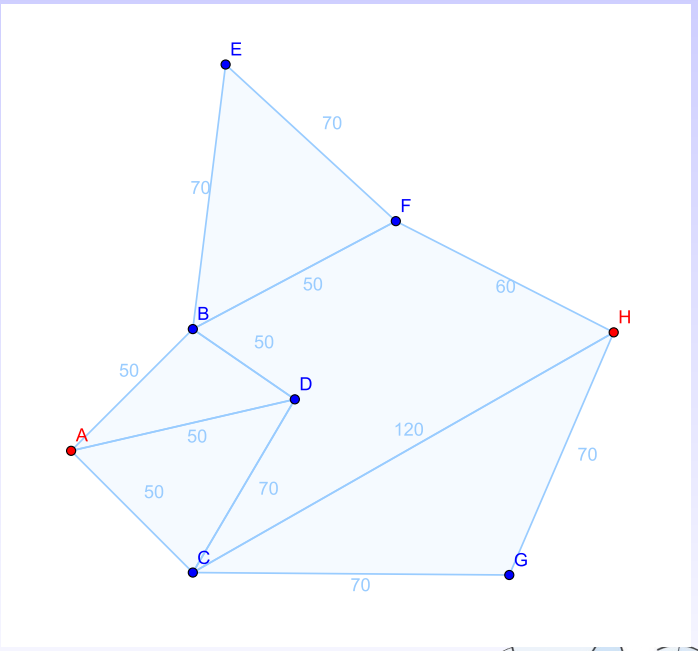
1. Consider a connected undirected graph G with weighted edges.
2. List all possible pairings among odd-degree vertices.



How to get Eulerian Multigraph G'?

Naive idea:

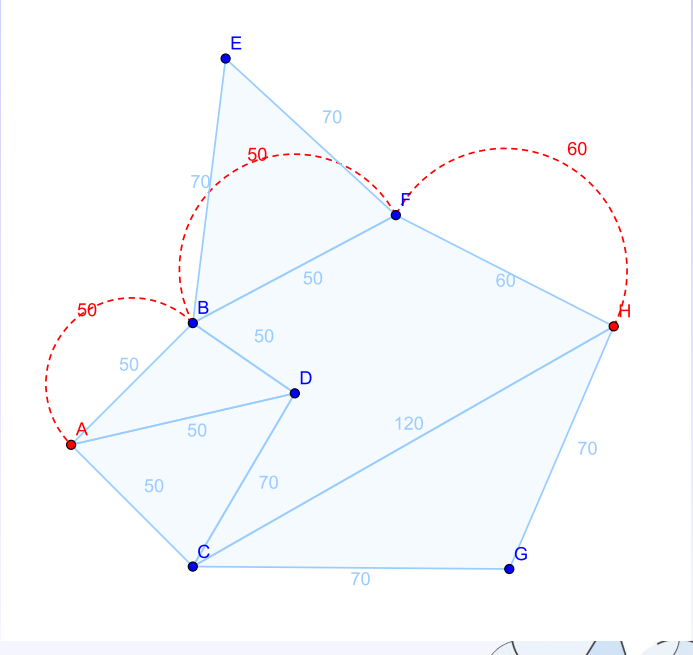
1. Consider a connected undirected graph G with weighted edges.
2. List all possible pairings among odd-degree vertices.
3. For each pairing, find the shortest path in G that connects the vertices (using Dijkstra's algorithm).



How to get Eulerian Multigraph G'?

Naive idea:

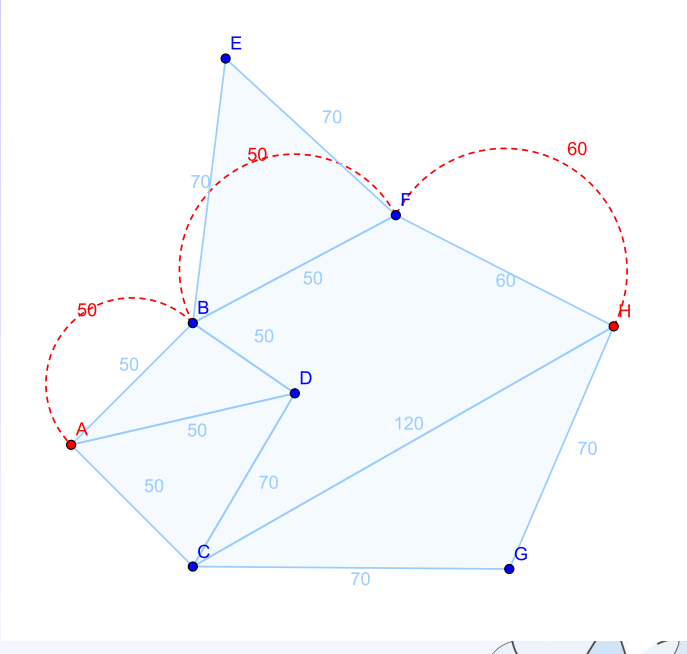
1. Consider a connected undirected graph G with weighted edges.
2. List all possible pairings among odd-degree vertices.
3. For each pairing, find the shortest path in G that connects the vertices (using Dijkstra's algorithm).
4. Choose the pairing such that the sum of the weights is minimum.



How to get Eulerian Multigraph G'?

Naive idea:

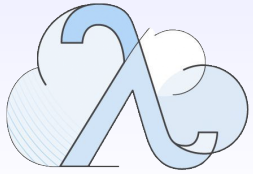
1. Consider a connected undirected graph G with weighted edges.
2. List all possible pairings among odd-degree vertices.
3. For each pairing, find the shortest path in G that connects the vertices (using Dijkstra's algorithm).
4. Choose the pairing such that the sum of the weights is minimum.
5. Add those edges to the original graph G'.



Python Implementation

How to find the edges that must be added to form G' while minimizing the extra distance?
Example: for two pairs of odd nodes A, B, C and D there are 3 possible combinations:

1. AB + CD
2. AC + BD
3. BC + AD



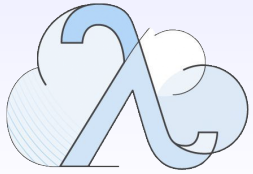
Python Implementation

How to find the edges that must be added to form G' while minimizing the extra distance?
Example: for two pairs of odd nodes A, B, C and D there are 3 possible combinations:

1. AB + CD
2. AC + BD
3. BC + AD

But for big networks the amount of combinations increases following a factorial rule. For N nodes:

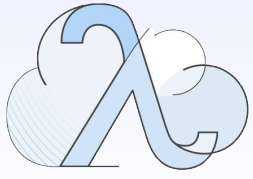
$$\text{Combinations} = \binom{N}{2}\binom{N-2}{2}\cdots\binom{4}{2} = N!\,!$$



Python Implementation: Blossom Algorithm

In 1973 Edmund and Johnson proposed the Blossom algorithm which solves this problem in polynomial time. For a map with 20 odd nodes:

$$\text{Combinations} = 20!\,! = 3,715,891,200$$



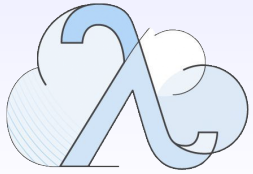
Python Implementation: Blossom Algorithm

In 1973 Edmund and Johnson proposed the Blossom algorithm which solves this problem in polynomial time. For a map with 20 odd nodes:

$$\text{Combinations} = 20!\,! = 3,715,891,200$$

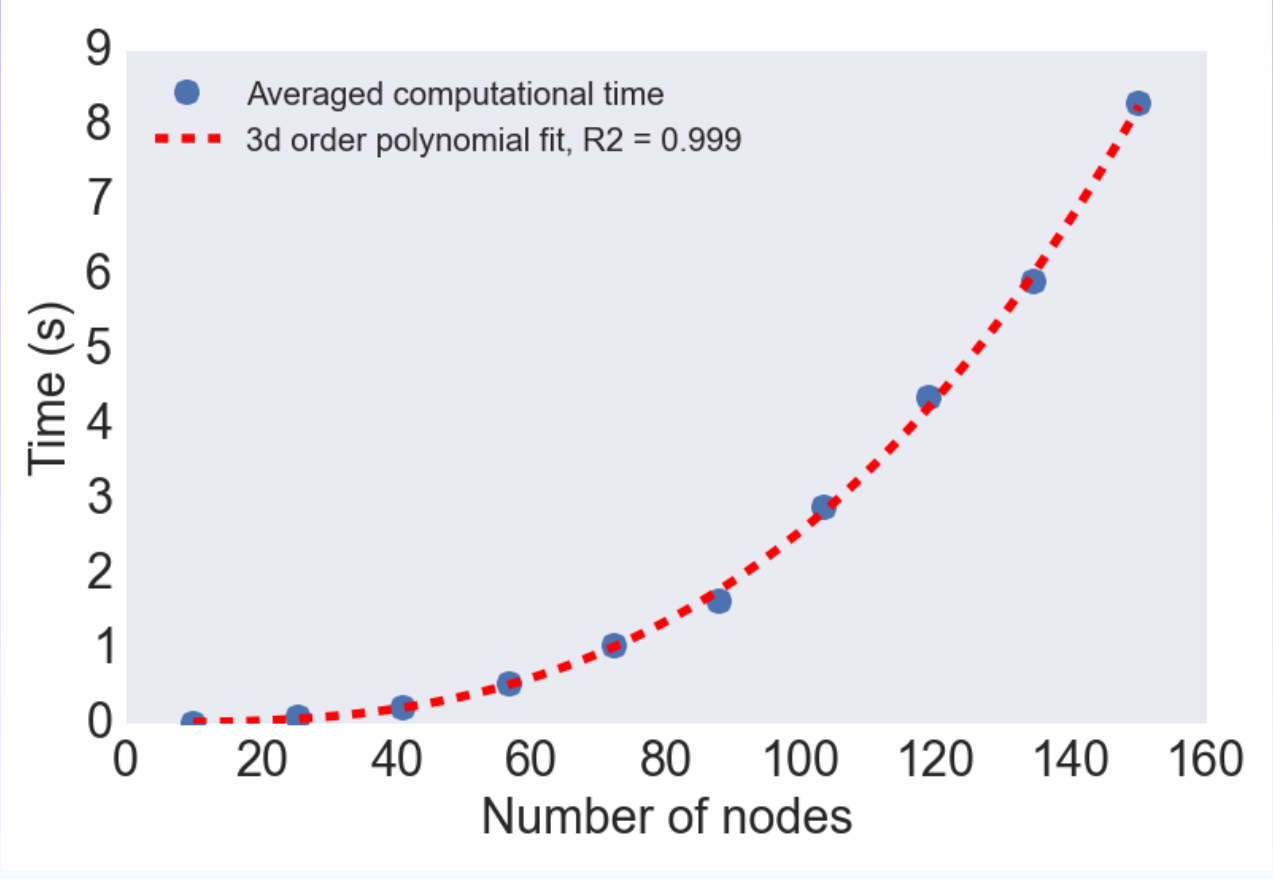
On a normal computer we get the following results:

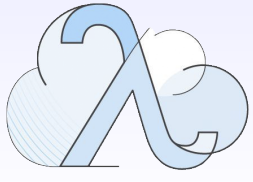
- Searching through all possibilities: possibly days of computational time.
- Blossom algorithm: 0.3 seconds.



Python Implementation: Scaling

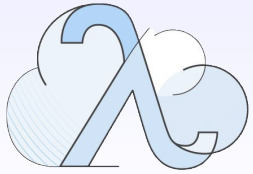
How does the computational time scale with graph size?





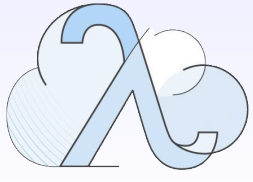
Python Implementation

- Pros
  - Scales well with size.
  - Optimality guarantee in finite time.
  - Can be used for graphs with one-way and two-way streets.
- Cons
  - Not flexible. Hard to add priorities, multiple machines, etc.



Dealing with the real problem

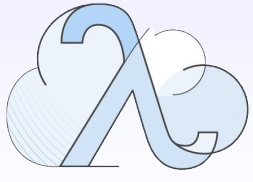
We were provided with the following data:



Dealing with the real problem

We were provided with the following data:

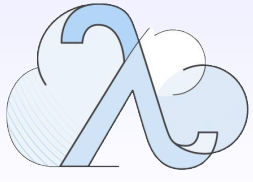
1. A Description of the task (in Finnish;)



Dealing with the real problem

We were provided with the following data:

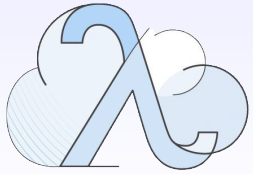
1. A Description of the task (in Finnish;)
2. A map of Lappeenranta;



Dealing with the real problem

We were provided with the following data:

1. A Description of the task (in Finnish;)
2. A map of Lappeenranta;
3. A list of street names and priorities (which streets should be plowed first, second etc.)



Dealing with the real problem

We were provided with the following data:

1. A Description of the task (in Finnish;)
2. A map of Lappeenranta;
3. A list of street names and priorities (which streets should be plowed first, second etc.)

LIUKKAUDEN TORJUNTA

Yleensä käsitellään kerrallaan koko pituudeltaan liukkauden esiintyessä. Väylät hiekoitetaan pääsääntöisesti traktorivetoisella hiekoitinlaitteella. Hiekoitus suoritetaan hiekoittimen levyisenä kaistana ja pyritään tekemään väylän keskelle.

TYÖN SUORITUSTAPA

Raitit aurataan joko heti täyteen leveyteen tai liikennöitävään leveyteen ja levitetään myöhemmin.

Työt arkisin aloitetaan tarvittaessa jo klo 3.00 - klo 7.00. Viikonloppuisin panostetaan tarpeen mukaan pääliikenneraittien kunnossapitoon.

A lot of Finnish words!

AURAUSURAKKA
LÄNSIALUE
Katulista

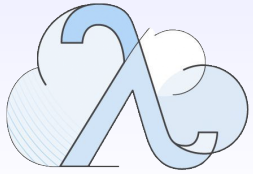
Lappeenrannan kaupunki
Tekninen toimi, Kadut ja ympär
25.5.2016

KADUT	pituus m 58 574
I LK	**2 168**
LAVOLANTIE	1 277
MUNTERONKATU	198
RUOHOLAMMENKATU	560
SKINNARILANKATU	133
II LK	**10 059**
HIETAKALLIONKATU	164
HOPEAMÄENRAITTI	166
JUPITERINKATU	201
JÄRVINIITUNKATU	296
KAARTINKATU	110
KALLIOPELLONKATU	264
KARJAKATU	209
KIERTOKATU	116

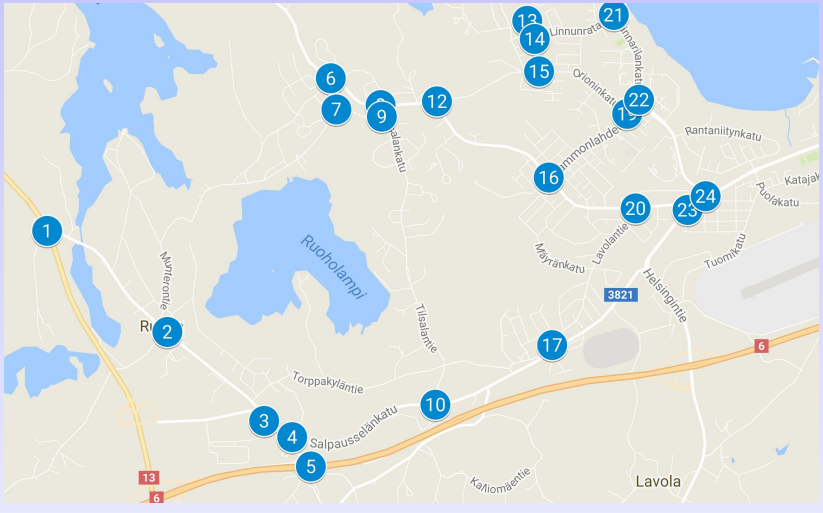
Dealing with the real problem

We were provided with the following data:

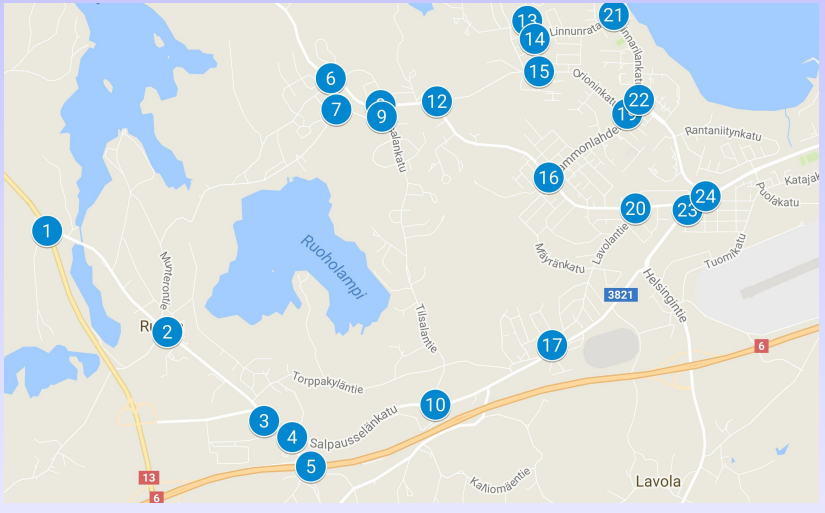
1. A Description of the task (in Finnish;)
2. A map of Lappeenranta;
3. A list of street names and priorities (which streets should be plowed first, second etc.)
4. We also used Google maps;)



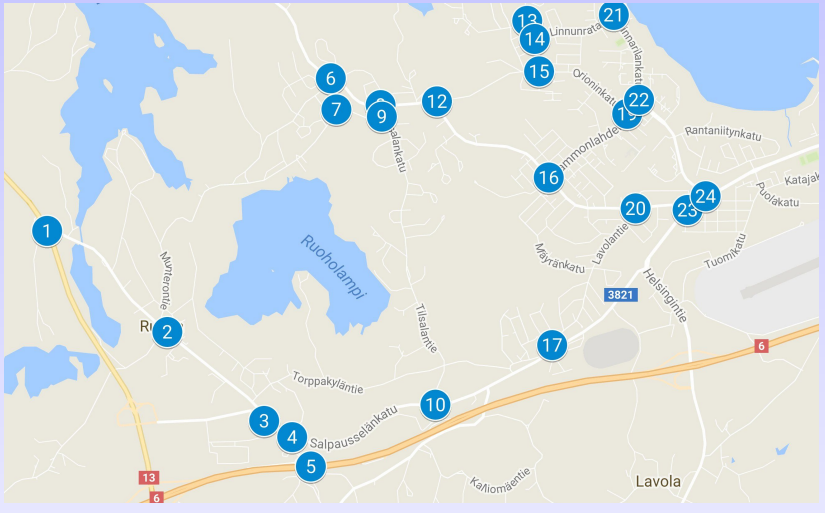
- We focused on streets with the highest priority.



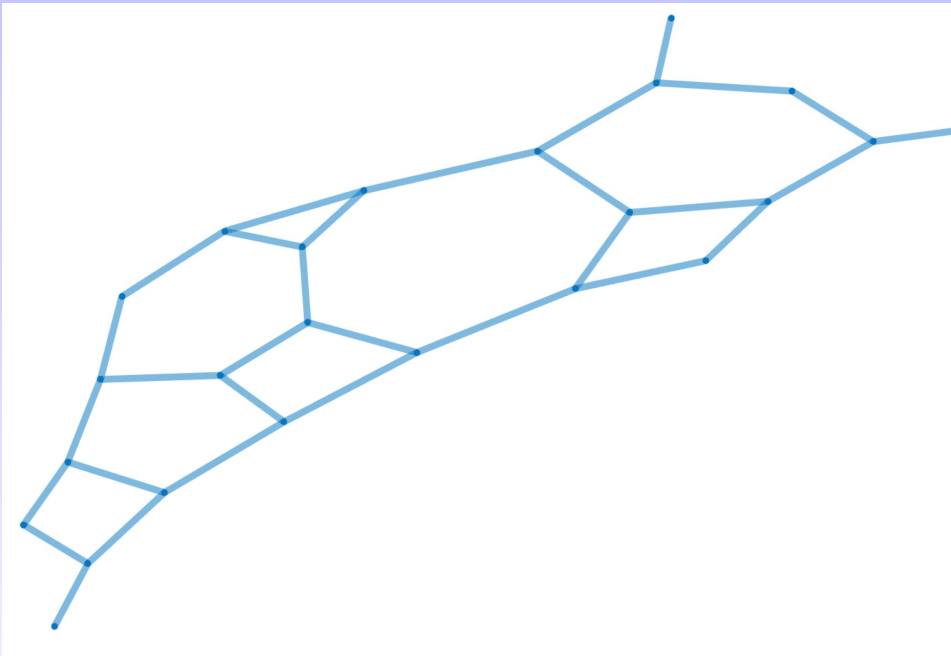
- We focused on streets with the highest priority.
- To construct the Lappeenranta road network, we obtained the geographical coordinates of intersections.



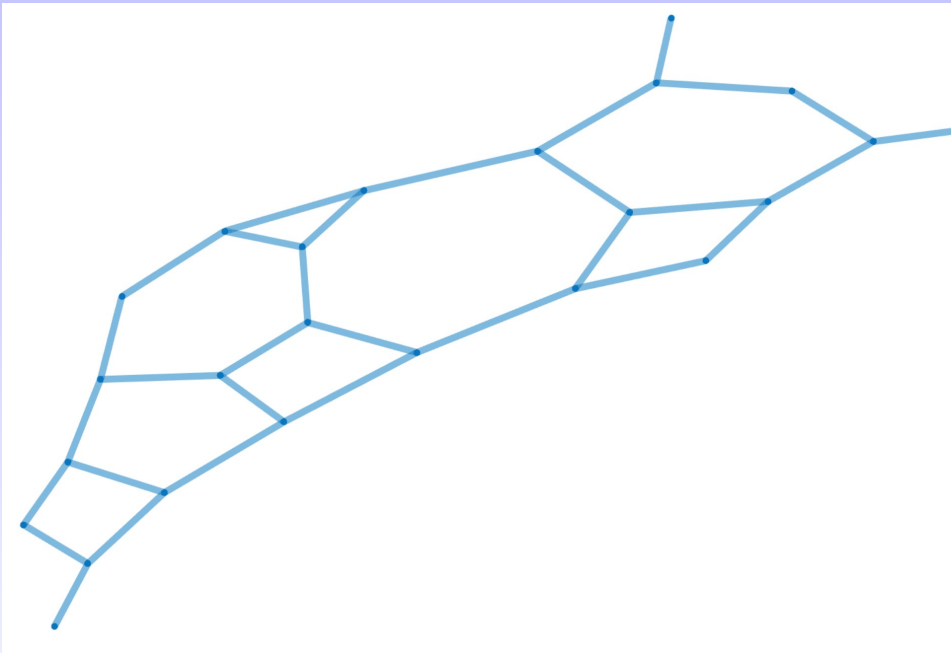
- We focused on streets with the highest priority.
- To construct the Lappeenranta road network, we obtained the geographical coordinates of intersections.
- The intersections are the nodes.



- We focused on streets with the highest priority.
- To construct the Lappeenranta road network, we obtained the geographical coordinates of intersections.
- The intersections are the nodes.
- The edges are road segments.

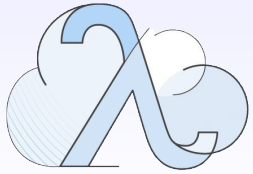


- We focused on streets with the highest priority.
- To construct the Lappeenranta road network, we obtained the geographical coordinates of intersections.
- The intersections are the nodes.
- The edges are road segments.
- The distances are the lengths of the segments.

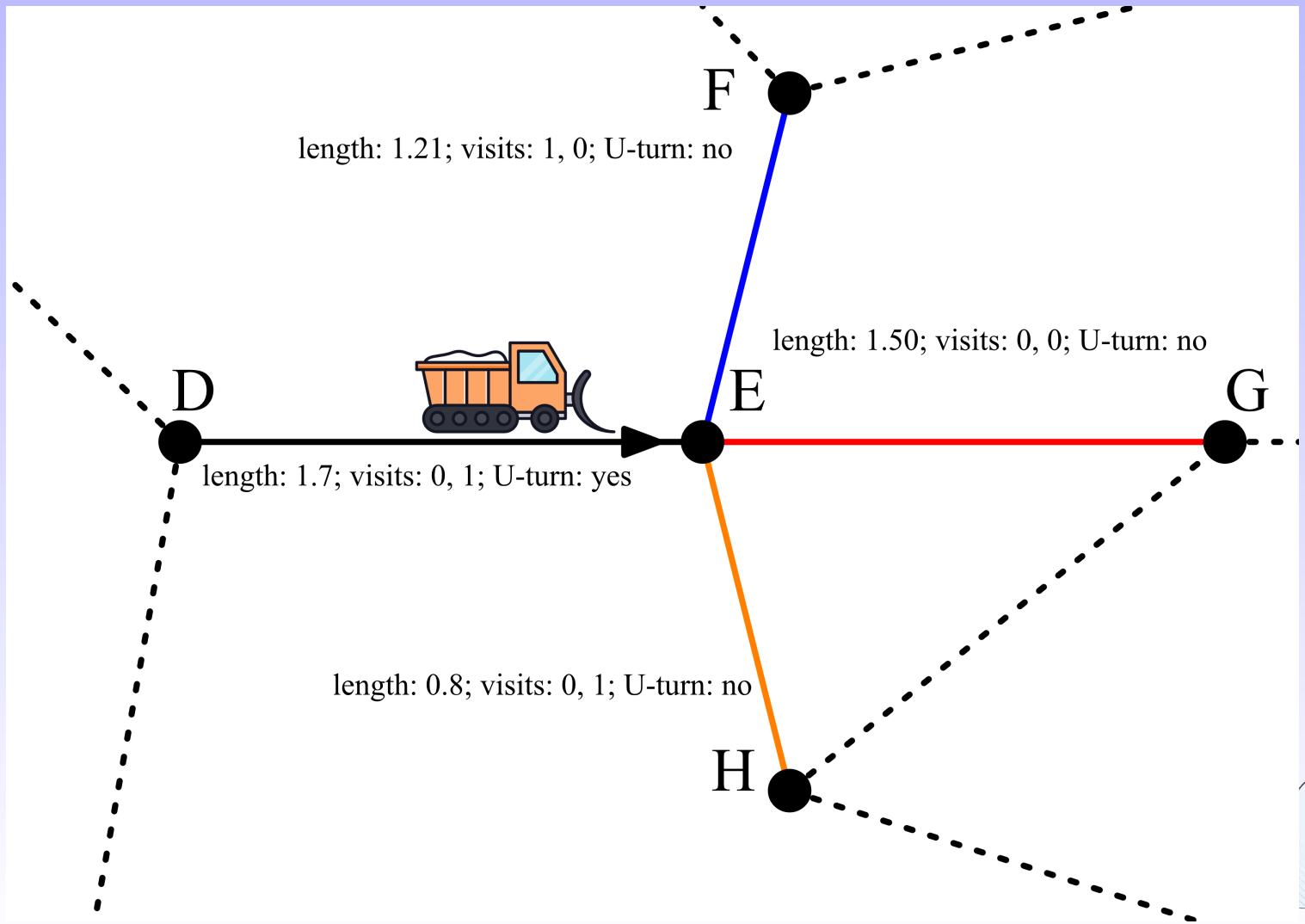


Penalty Scout Algorithm

- Heuristic methods for finding a path in an undirected connected graph.
- The idea is to create a walk along the graph and calculate the fitness.
- It uses a penalty term whenever a possible path choice has to be made.
- It allows us to compare different generated paths.

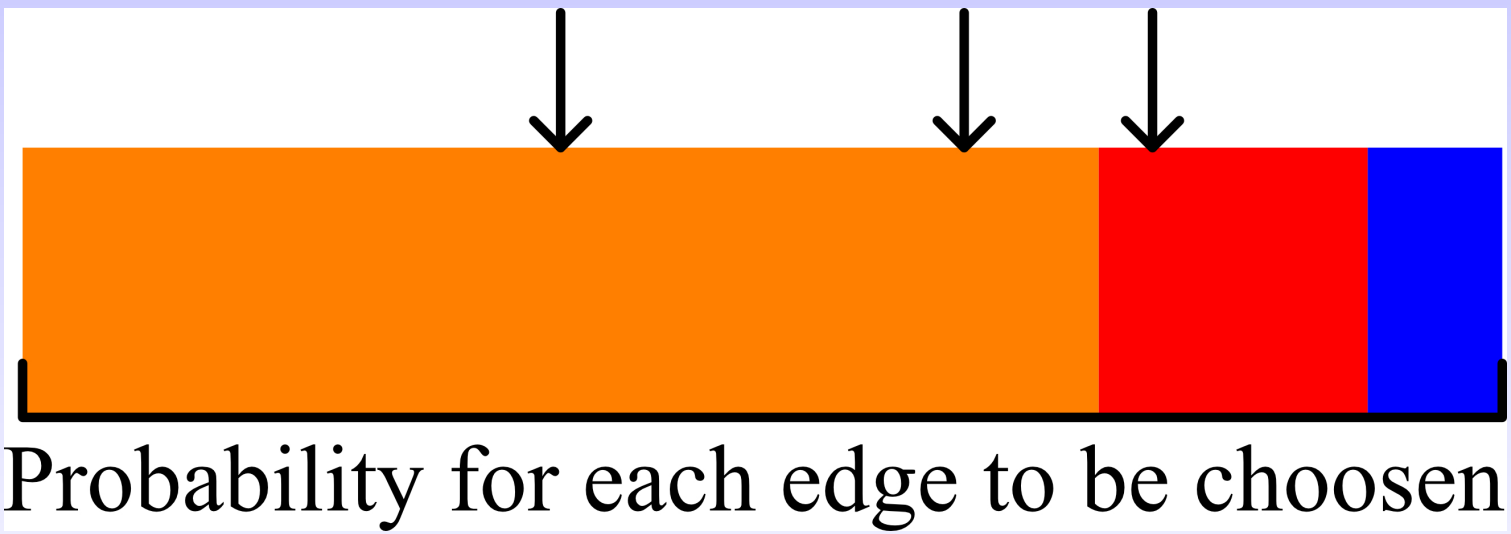


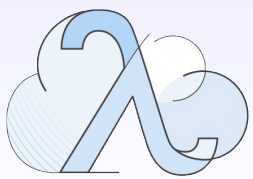
A pictorial example



A pictorial example (cont'd)

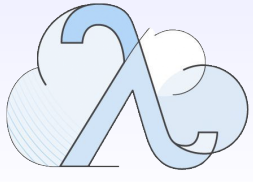
- Sort these points by penalty (the lower the penalty, the better)





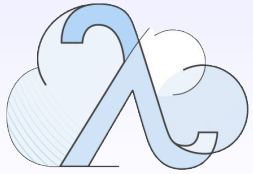
Results: Penalty Scout Algorithm

- One-way graph (24.2355 km total length)
  - One vehicle
    - 30.5275 km found within 1e6 attempts (30 minutes)
    - 35.0465 km found within 1e3 attempts (2 seconds)
  - Two vehicles
    - 41.1874 km found within 1e5 attempts (3 minutes)
  - Three vehicles
    - 58.8465 km found within 1e5 attempts (3 minutes)



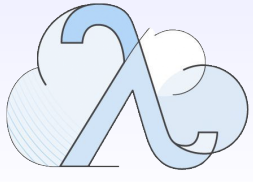
Results: Penalty Scout Algorithm

- Two-way graph (48.471 km total length)
  - One vehicle
    - 53.005 km found within 1e6 attempts (60 minutes)
    - 53.627 km found within 1e4 attempts (40 seconds)
  - Two vehicles
    - 60.245 km found within 1e5 attempts (6 minutes)
  - Three vehicles
    - 89.137 km found within 1e5 attempts (9 minutes)



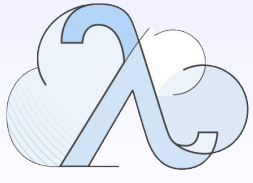
Solution using Penalty Scout Algorithm

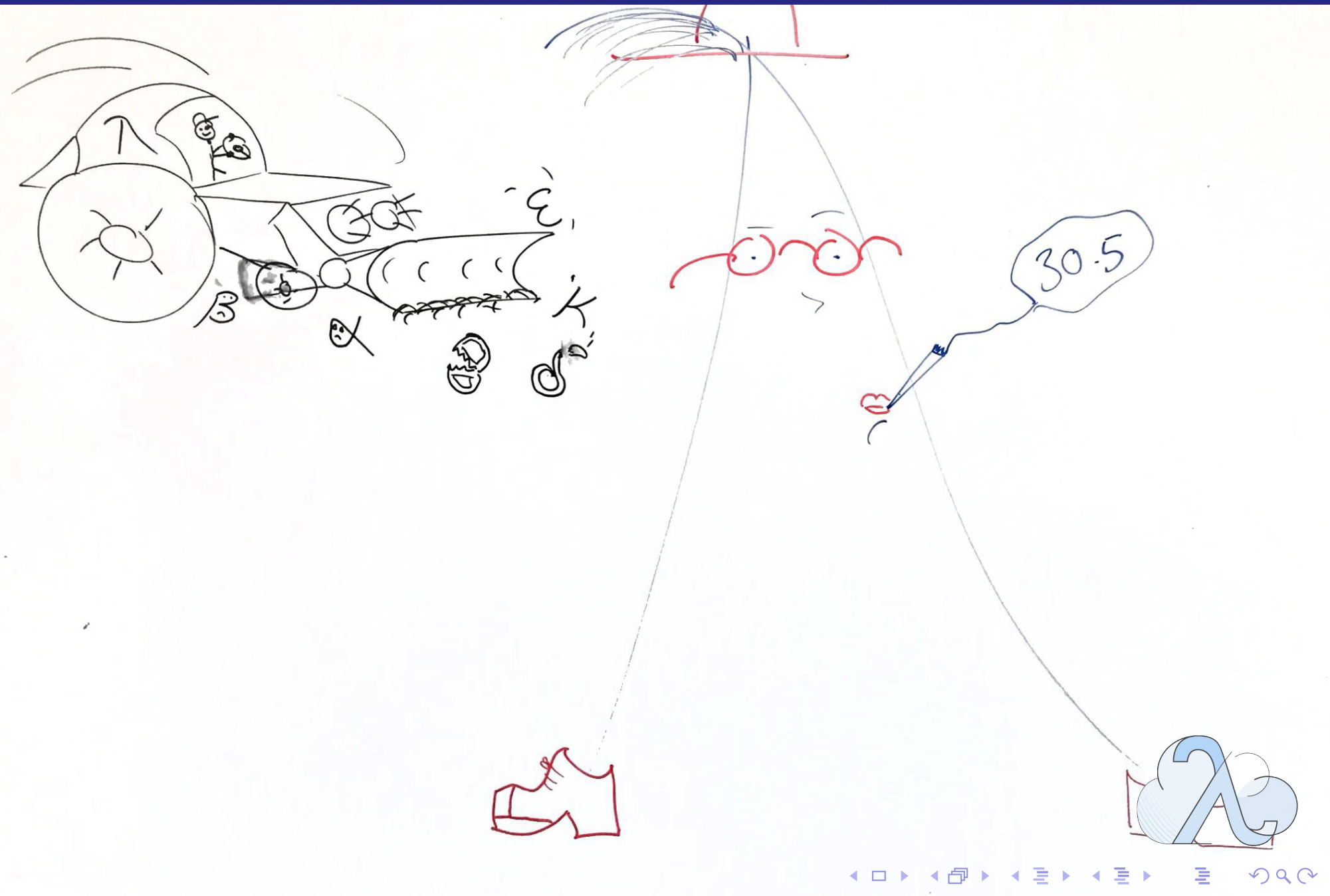
- Pros
  - Fast implementation
  - Terminating
  - Easy to set priorities of different parameters
  - Simple fitness calculation (length/time)
  - Fast expandable
  - Multiple vehicles considerable
- Cons
  - Decreasing compilation speed for more complicated graphs (exponential, but still acceptable)
  - No optimality guarantee (but usually performs well)



Results: Exact Solution to the CPP

The animation says it best...




30.5