Arc Routing Problems

Solving small-scale
vehicle routing problems
for waste collection

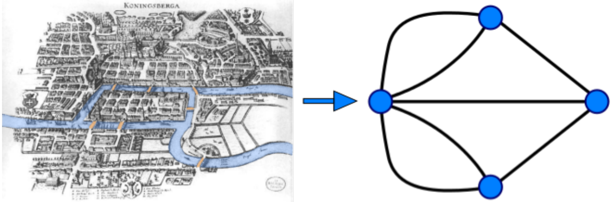
Summer Project by Katharina Limbeck
Supervised by Thu Dang

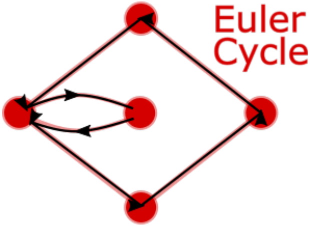
Structure

- Graph Theory - Euler Cycles
- Types of Arc Routing Problems
- Complexity Classes
- Exact solutions for the Chinese Postman Problem
- Integer Linear Programming and Optimisation
- Solutions in C#
- Further Areas of Interest

Königsberg Bridge Problem

Problem: Find a walk through the town that crosses each bridge exactly once.
Euler's Theorem: A connected undirected graph has an Euler cycle if and only if every vertex has even degree.

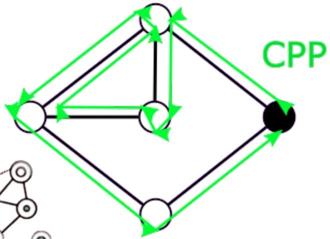




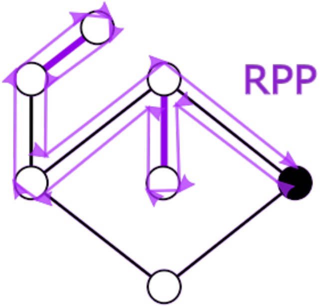
Arc Routing Problems

- find a set of cycles that cover all required edges $R \subseteq E$
- start and end at depot vertex
- minimize the total distance travelled

Chinese Postman Problem
- 1 cycle
- $R = E$



Rural Postman Probem
- 1 cycle
- $R \subset E$

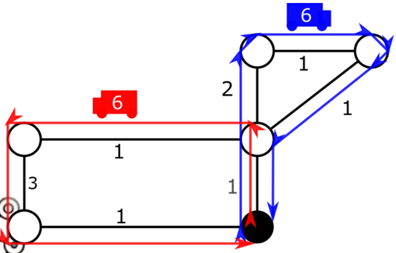


Capacitated Arc Routing

- find a number of vehicle tours
- each vehicle capacity *W*
- satisfy demand for each edge

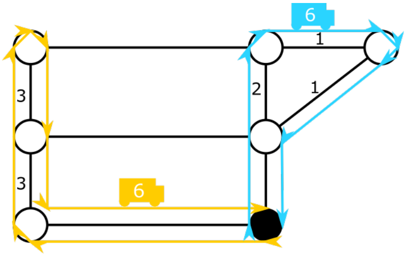
Capacitated CPP

- edge demand > 0

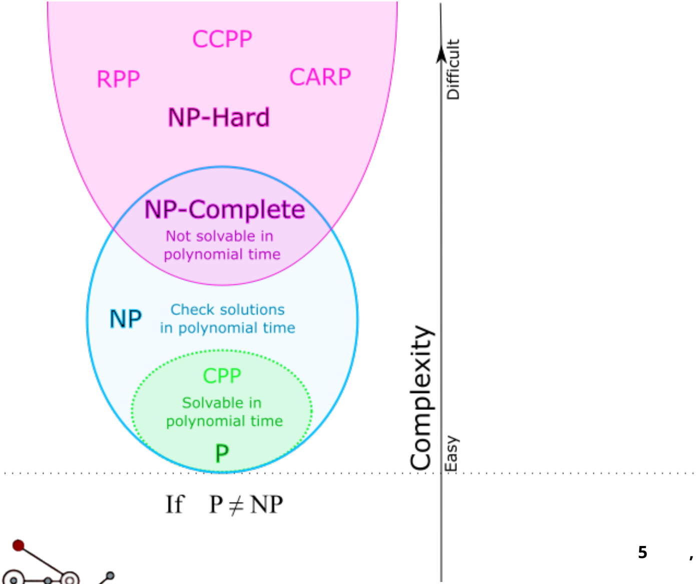


CARP

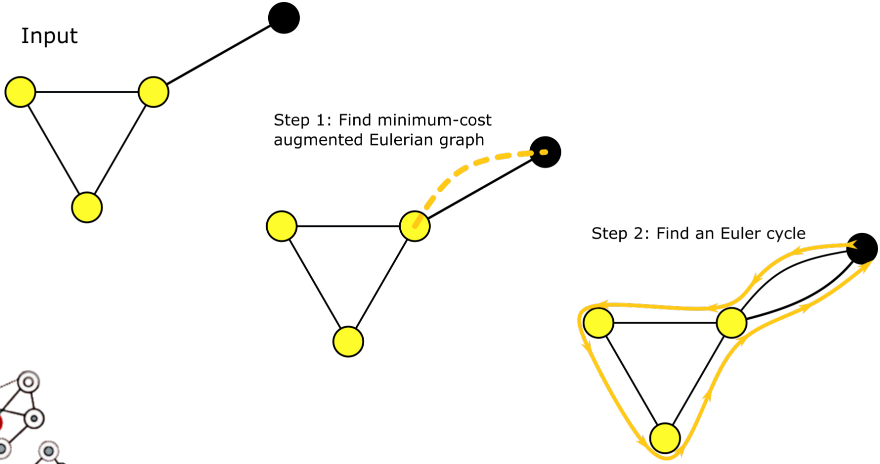
- edge demand ≥ 0



Complexity


CCPP
RPP
CARP
NP-Hard
NP-Complete
Not solvable in polynomial time
NP
Check solutions in polynomial time
CPP
Solvable in polynomial time
P
Difficult
Complexity
Easy
If $P \neq NP$

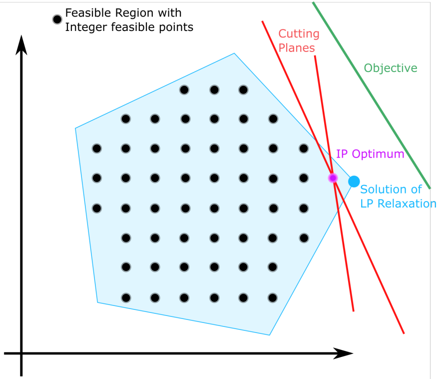
Exact Solutions to the CPP



Integer Linear Programming

Integer Linear Programming:
- optimize objective function
- requirements as linear relationships
- some variables restricted to be integer
- NP-complete

Branch and Cut Algorithm:
- solve without integer constraints
- cutting plane algorithm
- branch and bound into multiple sub-problems



Step 1: Find the minimum-cost augmentation:

Minimize

$$\sum_{(v_i,v_j)\in E} c_{ij}x_{ij} \tag{1}$$

subject to

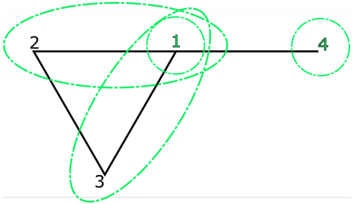
$$\sum_{(v_i,v_j)\in A(S)} x_{ij} \geq 1 \qquad (S \subset V, \text{S odd}) \tag{2}$$

$$x_{ij} \geq 0 \qquad ((v_i, v_j) \in E) \tag{3}$$

$$x_{ij} \text{ integer} \qquad ((v_i, v_j) \in E) \tag{4}$$

- x_{ij}**, the number of times the edge** (v_i, v_j) **is used in the solution**
  so the number of copies of each edge added to augment the graph
- $A(S)$**, all edges between the vertex set** $S \subset V$ **and its complement**
  $A(S) = \{(v_i, v_j); v_i \in S, v_j \in V \setminus S \text{ or } v_j \in S, v_i \in V \setminus S\}$

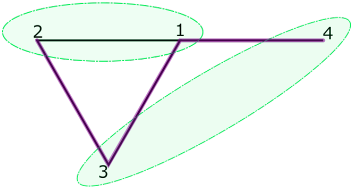
Some of the proper odd vertex subsets S:



Apply the constraint from equation (2) to all of them.
Here is an example choosing one possible subset:

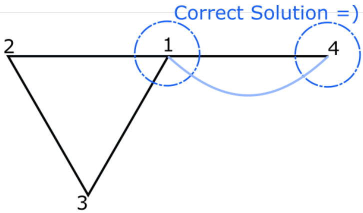
S = {2, 1}
A(S) = {(1,4); (1,3); (2,3)}

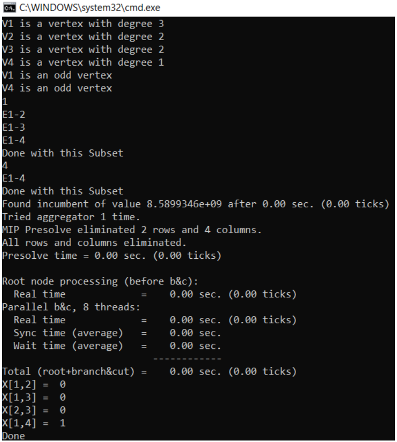
$$x_{13} + x_{14} + x_{23} \geq 1$$



Constraint $\sum_{(v_i,v_j)\in A(S)} x_{ij} \geq 1$ for

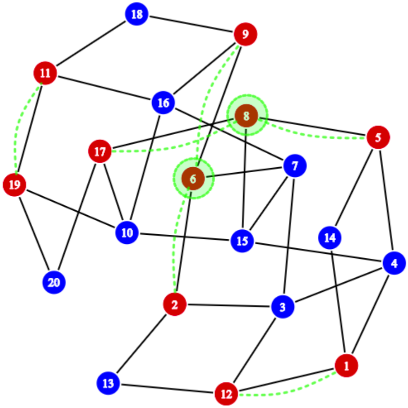
- all proper odd subsets S with 1 element
- works well for very small examples





```
C:\WINDOWS\system32\cmd.exe
V1 is a vertex with degree 3
V2 is a vertex with degree 2
V3 is a vertex with degree 2
V4 is a vertex with degree 1
V1 is an odd vertex
V4 is an odd vertex
1
E1-2
E1-3
E1-4
Done with this Subset
4
E1-4
Done with this Subset
Found incumbent of value 8.5899346e+09 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
MIP Presolve eliminated 2 rows and 4 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec. (0.00 ticks)

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.00 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.00 ticks)
X[1,2] =  0
X[1,3] =  0
X[2,3] =  0
X[1,4] =  1
Done
```

But this algorithm doesn’t give an optimal solution for more complicated examples like this one.

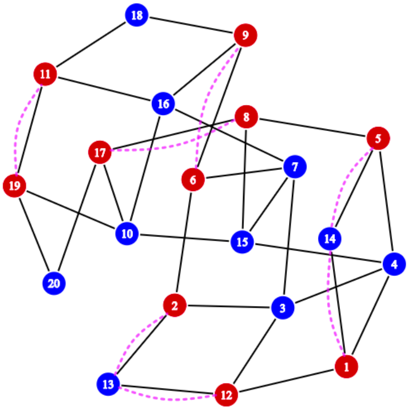


C:\WINDOWS\system32\cmd.exe

```
Found incumbent of value 6.6571993e+10 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
MIP Presolve eliminated 10 rows and 29 columns.
All rows and columns eliminated.
Presolve time = 0.02 sec. (0.02 ticks)

Root node processing (before b&c):
  Real time             =    0.02 sec. (0.02 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.02 sec. (0.02 ticks)
X[1,12] =  1
X[1,14] =  0
X[12,13] =  0
X[13,2] =  0
X[3,12] =  0
X[1,4] =  0
X[5,14] =  0
X[2,3] =  0
X[3,4] =  0
X[5,4] =  0
X[5,8] =  1
X[4,15] =  0
X[3,7] =  0
X[6,2] =  1
X[6,7] =  0
X[7,15] =  0
X[8,15] =  0
X[8,17] =  1
X[10,15] =  0
X[7,16] =  0
X[6,9] =  1
X[9,16] =  0
X[10,16] =  0
X[10,17] =  0
X[17,20] =  0
X[10,19] =  0
X[11,16] =  0
X[9,18] =  0
X[11,18] =  0
X[11,19] =  1
X[19,20] =  0
Done
```

Step 1: Creating an algorithm to solve constraint (2) for

- all proper odd subsets with 1 element
- all possible combinations of *k* elements of the vertex set ($2 \leq k \leq n-2$) where the combination is an odd subset

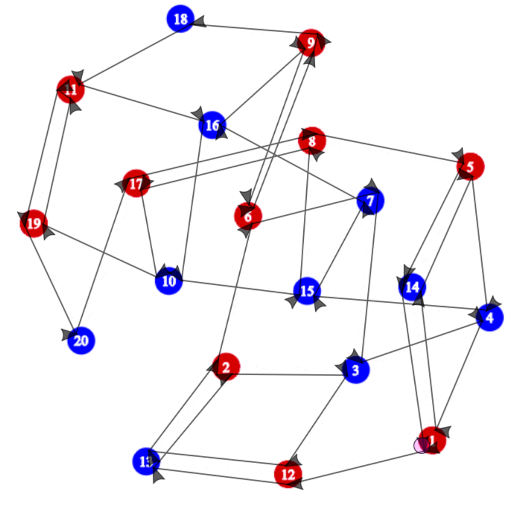


C:\WINDOWS\system32\cmd.exe

```
We creat a constraint for a set with vertices V12, V13, V2, V3, V4, V5, V8, V15,
Found incumbent of value 6.6571993e+10 after 0.02 sec. (28.25 ticks)
Tried aggregator 1 time.
MIP Presolve eliminated 262134 rows and 0 columns.
Reduced MIP has 262144 rows, 31 columns, and 4063232 nonzeros.
Reduced MIP has 31 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 2.03 sec. (1399.17 ticks)
Probing time = 0.13 sec. (29.17 ticks)
Tried aggregator 1 time.
Reduced MIP has 262144 rows, 31 columns, and 4063232 nonzeros.
Reduced MIP has 31 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 3.47 sec. (1916.18 ticks)
Probing time = 0.19 sec. (29.84 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time = 3.69 sec. (1457.12 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                      6.65720e+10        0.0000             100.00%
*     0+    0                           7.0000        0.0000             100.00%
      0     0        cutoff             7.0000        7.0000       34    0.00%
      0     0        cutoff             7.0000        7.0000       34    0.00%
Elapsed time = 11.14 sec. (6023.31 ticks, tree = 0.01 MB, solutions = 1)

Root node processing (before b&c):
  Real time             =   11.17 sec. (6023.31 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =   11.17 sec. (6023.31 ticks)
X[1,12] =  0
X[1,14] =  1
X[12,13] =  1
X[13,2] =  1
X[3,12] =  0
X[1,4] =  0
X[5,14] =  1
X[2,3] =  0
X[3,4] =  0
X[5,4] =  0
X[5,8] =  0
X[4,15] =  0
X[3,7] =  0
X[6,2] =  0
X[6,7] =  0
X[7,15] =  0
X[8,15] =  0
X[8,17] =  1
X[10,15] =  0
X[7,16] =  0
X[6,9] =  1
X[9,16] =  0
X[10,16] =  0
X[10,17] =  0
X[17,20] =  0
X[10,19] =  0
X[11,16] =  0
X[9,18] =  0
X[11,18] =  0
X[11,19] =  1
X[19,20] =  0
Done
```

Step 2: Find an Euler Cycle in the augmented graph using Fleury's Algorithm

1. Start at an arbitrary vertex v_i traverse an edge (v_i, v_j) that is not a bridge and erase edge (v_i, v_j)
2. Set $v_i := v_j$ and repeat step 1 starting from v_j or stop if all edges have been deleted.




```
The Euler Cycle includes:
E[1,12]
E[12,13]
E[13,2]
E[2,3]
E[3,12]
E[12,13]
E[13,2]
E[2,6]
E[6,7]
E[7,3]
E[3,4]
E[4,1]
E[1,14]
E[14,5]
E[5,4]
E[4,15]
E[15,7]
E[7,16]
E[16,9]
E[9,6]
E[6,9]
E[9,18]
E[18,11]
E[11,16]
E[16,10]
E[10,15]
E[15,8]
E[8,17]
E[17,10]
E[10,19]
E[19,11]
E[11,19]
E[19,20]
E[20,17]
E[17,8]
E[8,5]
E[5,14]
E[14,1]
```

Other Areas of Interest

- Consider other types arc routing problems
  - mixed or directed graphs
  - capacitated problems
  - extend constraints to practical problems
- Solving large-scale instances quickly and reliably
  - Trade-off between quality of solution and running time
  - Consider best ways to relax constraints
  - Find good heuristic algorithms and upper bounds
  - Find good lower bounds on the optimal solution

References

1. A. Corberan G. Laporte (2014) Arc Routing: Problems, Methods, and Applications. Philadelphia, PA: SIAM.
2. B.L. Golden R.T. Wong (1981) Capacitated arc routing problems. Networks, 11, 305-15.
3. D. Ahr (2004) Contributions to Multiple Postmen Problems. PhD dissertation, Department of Computer Science, Heidelberg University.
4. Rafael Martinelli (2013) Improved bounds for large scale capacitated arc routing problem. Computers Operations Research, 40, 2145–2160.
5. H. A. Eiselt, Michel Gendreau, Gilbert Laporte, (1995) Arc Routing Problems, Part I: The Chinese Postman Problem. Operations Research 43(2):231-242.
6. H. A. Eiselt, Michel Gendreau, Gilbert Laporte, (1995) Arc Routing Problems, Part II: The Rural Postman Problem. Operations Research 43(3):399-414.
7. Claude Tadonki (2013) High Performance Computing as a Combination of Machines and Methods and Programming. Paris XI; Ecole Nationale Superieure des Mines de Paris, Universite Paris Sud.