Longest Path Optimization using Integer Programming

Hrvoje Abraham
hrvoje.abraham@avl.com

AVL-AST d.o.o., Zagreb, Croatia

November 10, 2022

The Problem

For some dictionary $\mathcal{D}$ find the longest chain of words without repetition with the previous word ending with two characters the next word begins.

house $\rightarrow$ semantic $\rightarrow$ iconic $\rightarrow$ icosahedral $\rightarrow$ alcohol $\rightarrow$ olala $\rightarrow$ lambda

Challenge formulation:

- $\mathcal{D} =$ Croatian dictionary containing 159.836 words
- Find the longest word-chain of at least 26.552 words.

The paper

SOLVING THE LONGEST WORD-CHAIN PROBLEM

Nobuo Inui, Yuji Shinano, Yuusuke Kounoike, Yoshiyuki Kotani
Tokyo University of Agriculture and Technology

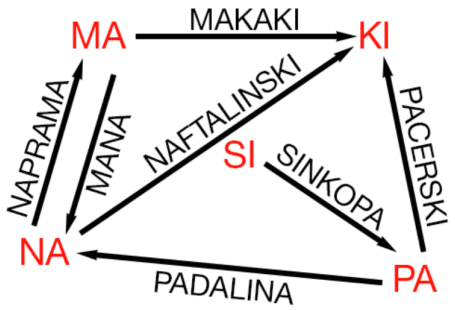
Keywords: The Longest Distance Problem, Word–Chain Game, Integer Programming, Heuristic Search, Linear Programming, Branch-and-Bound Method, Optimal Solution, Local-Maximum Solution

Abstract: The SHIRITORI game is a traditional Japanese word-chain game. This paper describes the definition of the longest SHIRITORI problem (a kind of the longest distance problem) as a problem of graph and the solution based on the integer problem (IP). This formulation requires the exponential order variables from the problem size. Against this issue, we propose a solution based on the LP-based branch-and-bound method, which solves the relaxation problems repeatedly. This method is able to calculate the longest SHIRITORI sequences for 130 thousand words dictionary within a second. In this paper, we compare the performances for the heuristic-local search and investigate the results for several conditions to explore the longest SHIRITORI problem.

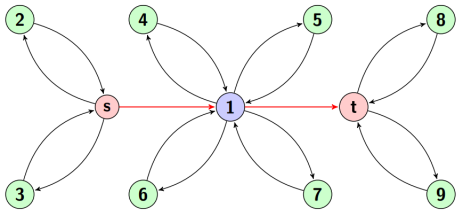
https://www.scitepress.org/papers/2004/11389/11389.pdf

Four Pillars of the Action Plan

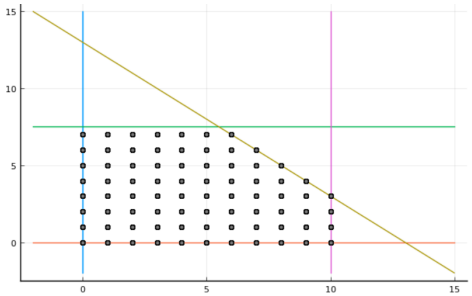
1. **Efficient formulation**
   Define suffix graph with words as edges and find the **longest path** containing the most edges.



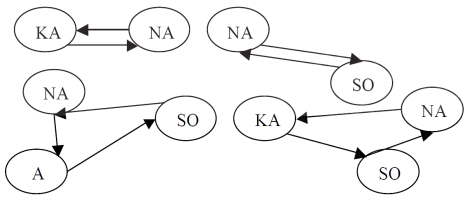
2. **Efficient technique**
   Find the **longest path** by extracting the largest connected semi-Eulerian subgraph using ILP.



3. **Implementation & Computation**
   Use some Integer Programming solver to find the best possible connections configuration.

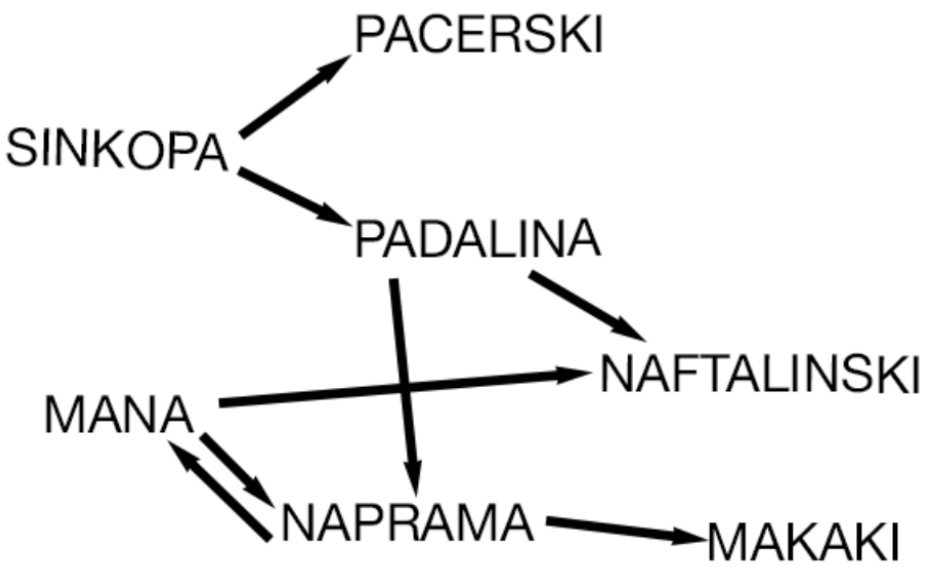


4. **Solution Extraction**
   Extract all cycles of the found semi-Eulerian subgraph and construct the **longest path**.



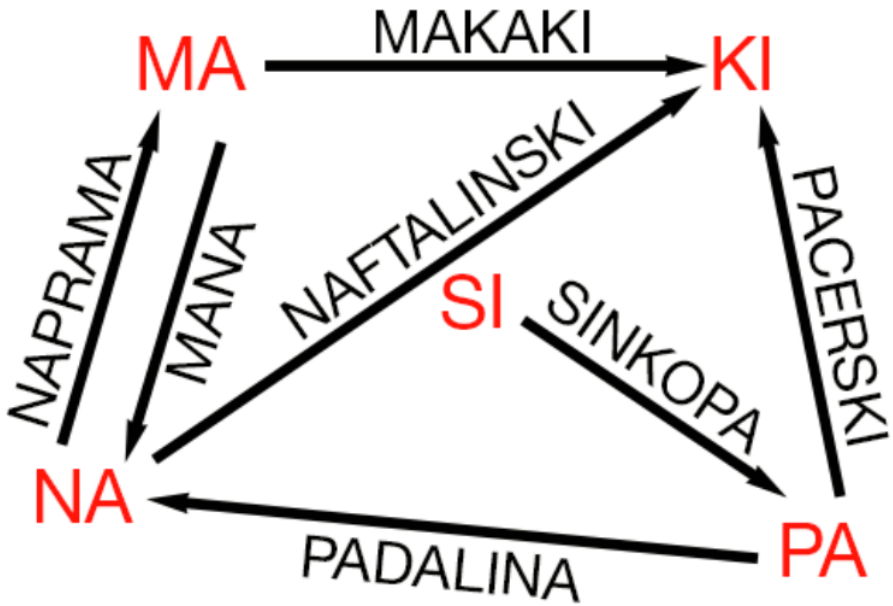
Efficient Formulation

Hamiltonian formulation



159.797 nodes
$\approx$ 2.500.000 edges
Longest Path = Most Nodes

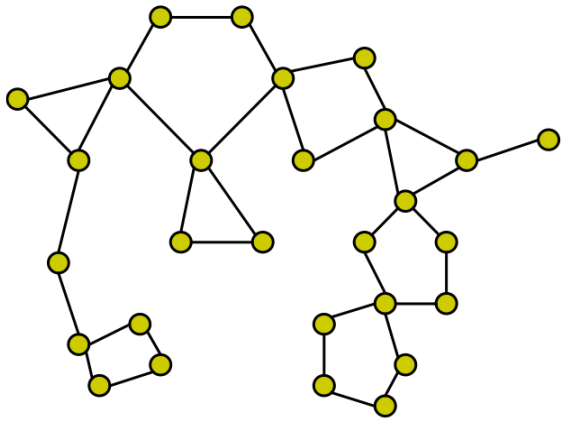
Eulerian formulation



509 nodes
159.797 edges
Longest Path = Most Edges

Longest Path Problem

cactus

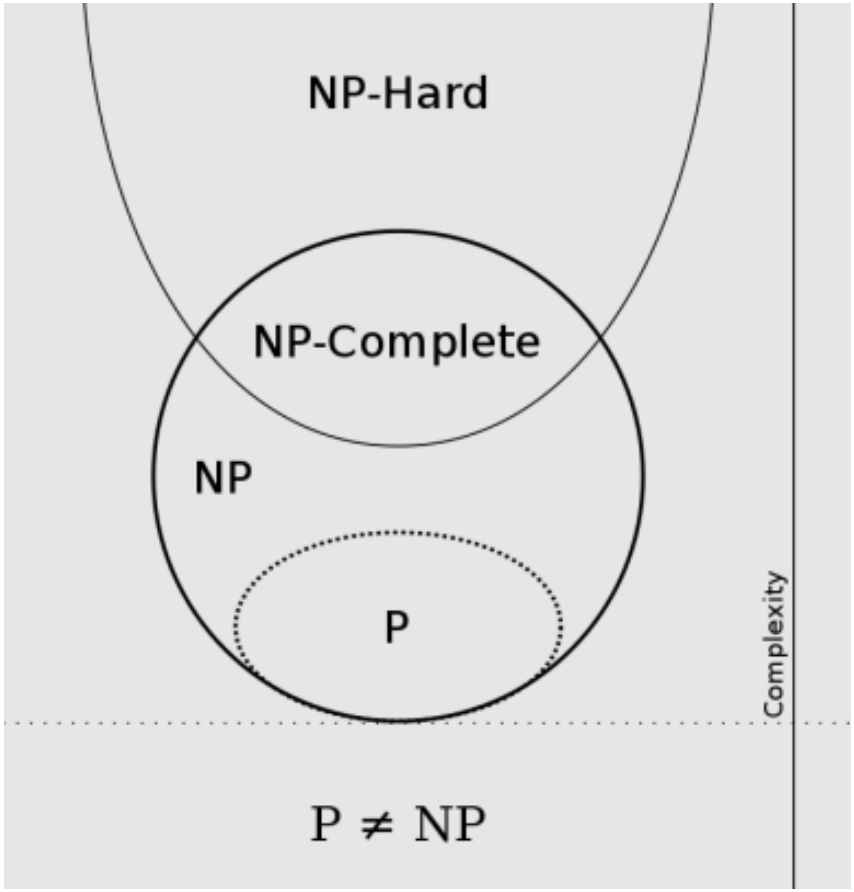


- DAG case is frequent, and solvable in $\mathcal{O}(N)$.
- Rectangular Grid Graph case solvable in $\mathcal{O}(N)$.
- Cactus Graph case solvable in $\mathcal{O}(N^2)$.
- Interval Graph case solvable in $\mathcal{O}(N^4)$.
- Ptolemaic Graph case solvable in $\mathcal{O}(N^5)$.

- **We consider a general case**, not acyclic or other special cases.
- General case is NP-complete, no polynomial-time algorithm known!
- General case rarely mentioned, hard to dig up any practical solution.

`https://en.wikipedia.org/wiki/Longest_path_problem`
`https://en.wikipedia.org/wiki/List_of_NP-complete_problems`

Longest Path Problem - NP-Complete



No **current** algorithm for solving any NP-complete problem has polynomial time-complexity (P), but no proof this must be so ($P \stackrel{?}{=} NP$).

The solution can be checked in P.

Longest Path Problem

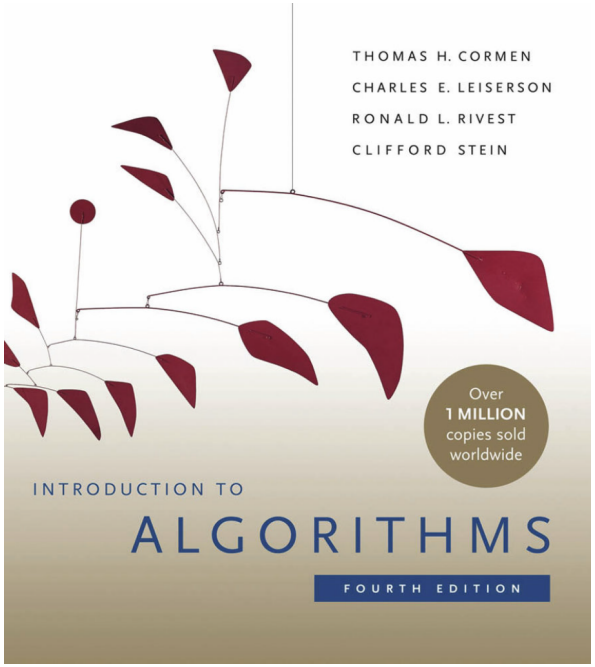
Some complexities for n-vertex m-edge Hamiltonian graphs:

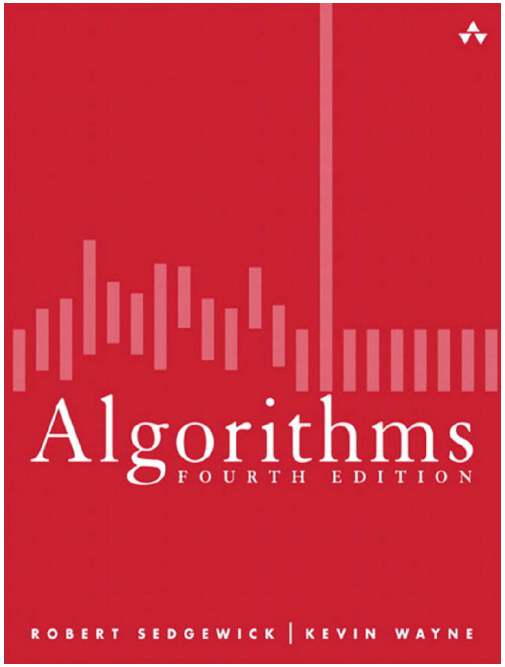
- Brute force - $\mathcal{O}(n!)$, $\mathcal{O}(n^n)$
- ? Quantum - $\mathcal{O}(1.823^m)$?
- Greedy approximation - $\mathcal{O}(n^2)$
- Held-Karp approx. (TSP) - $\mathcal{O}_{time}(n^2\,2^n)$, $\mathcal{O}_{space}(n\,2^n)$

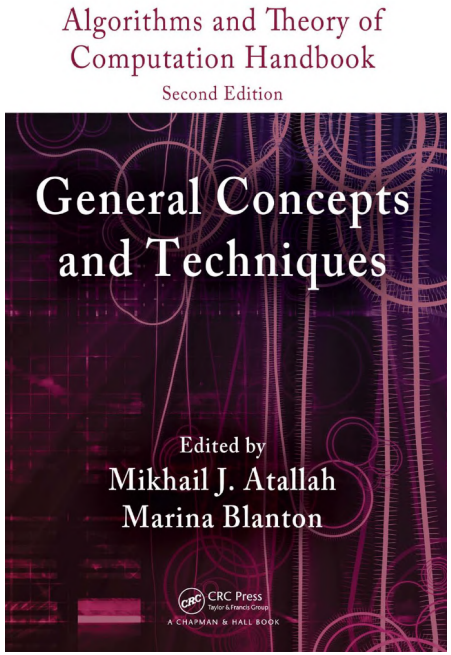
Formal boundaries for Hamiltonian graph:

- Hamiltonian graph easier than our general case.
- Polynomial time algorithm known for finding $\Omega(log^2 n/log\ log\ n)$ length path in any directed n-vertex Hamiltonian graph.
- For any $\epsilon < 1$ finding a path of length $n - n^\epsilon$ in an n-vertex Hamiltonian graph is NP-hard. - Karger, Motwani, Ramkumar (1997).

Longest Path in the Literature

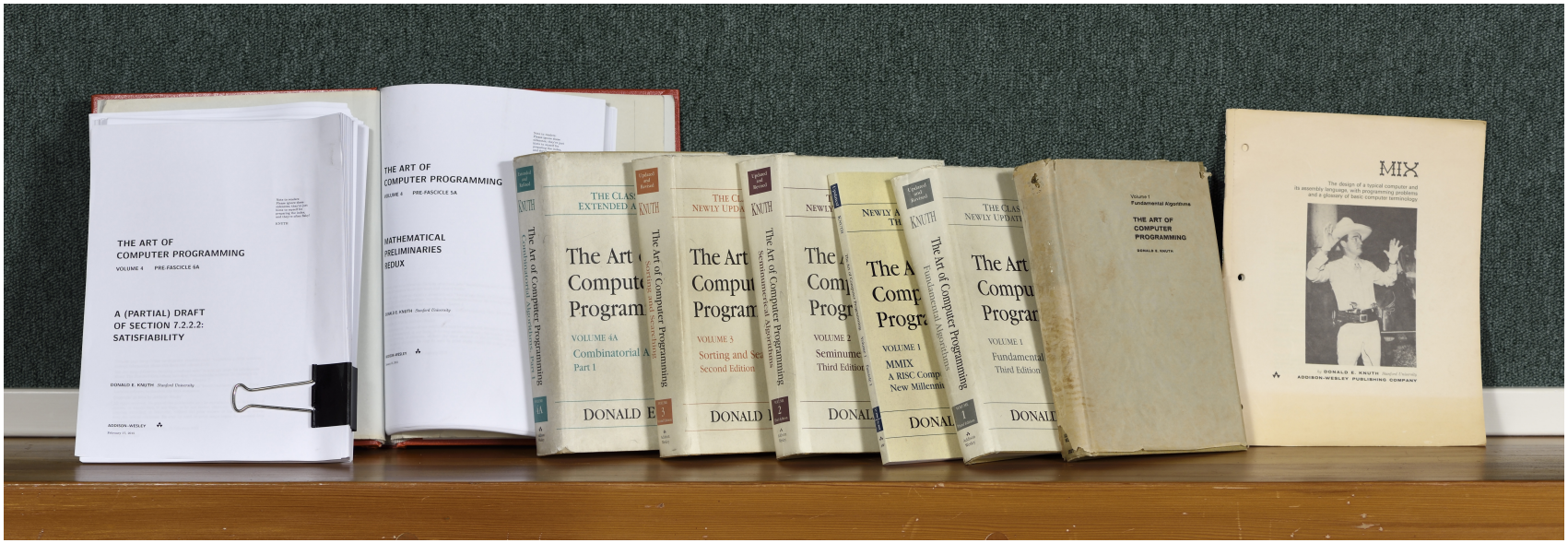






- Introducton to Algorithms NEW! 4th Edition, April 2022
- Some discussion and solutions for DAGs.
- Mention of general variant as an example of optimization problems.
- Noted as an example of a hard-to-approximate problem.
- Pseudo-code for a brute force solution.

Longest Path in the Literature - TAOCP



The Remainder of Volume 4

Present plans are for Volumes 4A and 4B to be the first in a series of several subvolumes 4A, 4B, 4C, ... entitled *Combinatorial Algorithms, Part 1, 2, 3, ...*. The remaining subvolumes, currently in preparation, will have the following general outline:

- 7.2.2.3. Constraint satisfaction
- 7.2.2.4. Hamiltonian paths and cycles
- 7.2.2.5. Cliques
- 7.2.2.6. Covers
- 7.2.2.7. Squares
- 7.2.2.8. A potpourri of puzzles
- 7.2.2.9. Estimating backtrack costs
- 7.2.3. Generating inequivalent patterns
- 7.3. Shortest paths
- 7.4. Graph algorithms
- 7.4.1. Components and traversal
- 7.4.1.1. Union-find algorithms
- 7.4.1.2. Depth-first search
- 7.4.1.3. Vertex and edge connectivity
- 7.4.2. Special classes of graphs
- 7.4.3. Expander graphs
- 7.4.4. Random graphs
- 7.5. Graphs and optimization
- 7.5.1. Bipartite matching
- 7.5.2. The assignment problem
- 7.5.3. Network flows
- 7.5.4. Optimum subtrees
- 7.5.5. Optimum matching
- 7.5.6. Optimum orderings
- 7.6. Independence theory
- 7.6.1. Independence structures
- 7.6.2. Efficient matroid algorithms
- 7.7. Discrete dynamic programming
- 7.8. Branch-and-bound techniques
- 7.9. Herculean tasks (aka NP-hard problems)
- 7.10. Near-optimization
- 8. Recursion

Linear Programming

- A **linear function to be maximized**

  e.g. $f(x_1, x_2) = c_1 x_1 + c_2 x_2$

- **Problem constraints** of the following form

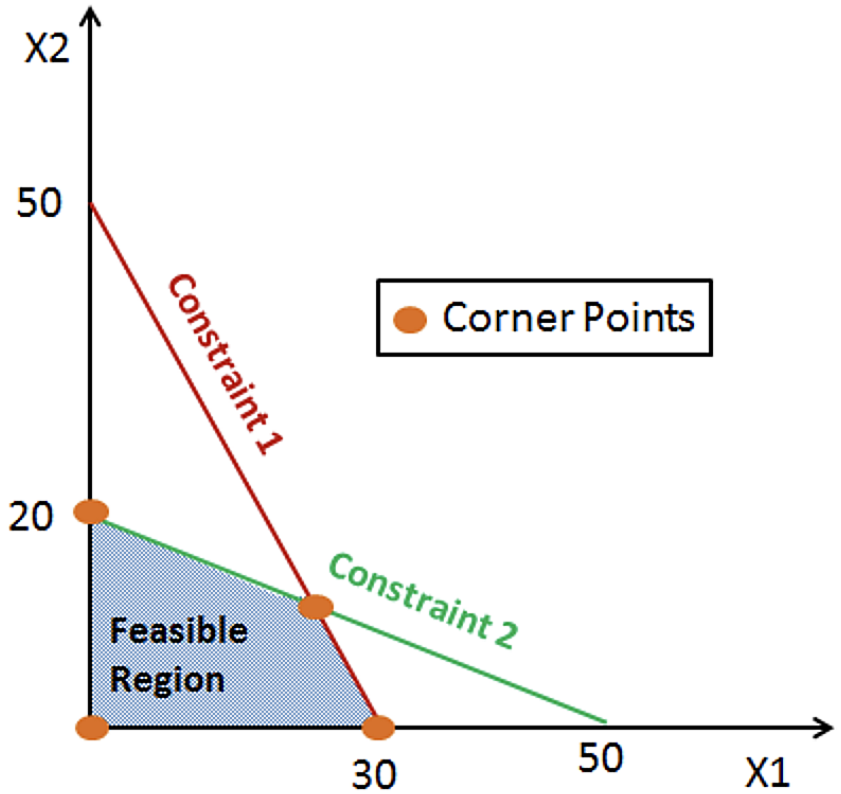
  e.g.

$$a_{11} x_1 + a_{12} x_2 \leq b_1$$
$$a_{21} x_1 + a_{22} x_2 \leq b_2$$
$$a_{31} x_1 + a_{32} x_2 \leq b_3$$

- **Non-negative variables**

  e.g.

$$x_1 \geq 0$$
$$x_2 \geq 0$$



Program(ming) as in TV program, training program, theatre program...

Linear programming solvable in polynomial time - Leonid Khachiyan, 1979

Integer Programming

- A **linear function to be maximized**

  e.g. $f(x_1, x_2) = c_1 x_1 + c_2 x_2$

- **Problem constraints** of the following form

  e.g.

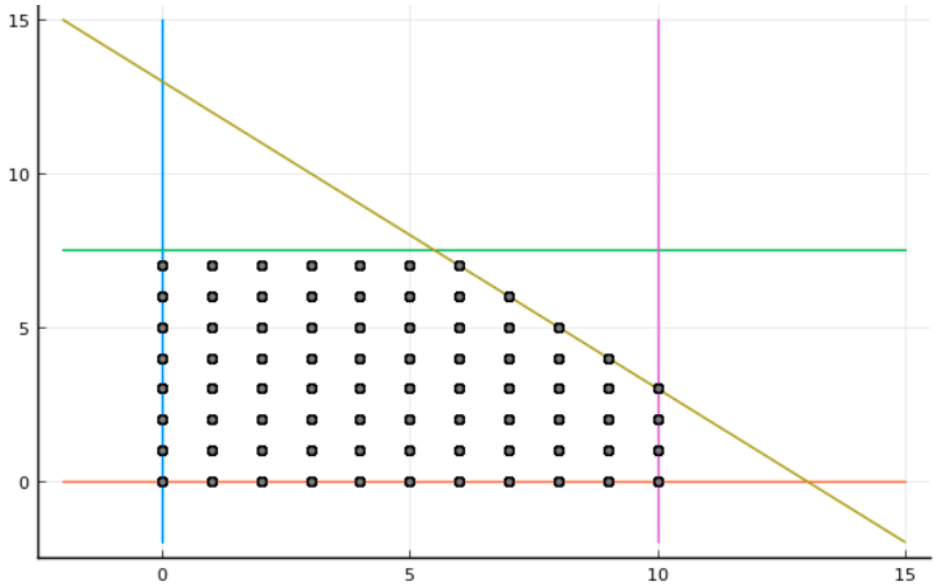
$$\begin{aligned} a_{11} x_1 + a_{12} x_2 &\leq b_1 \\ a_{21} x_1 + a_{22} x_2 &\leq b_2 \\ a_{31} x_1 + a_{32} x_2 &\leq b_3 \end{aligned}$$

- **Non-negative variables**

  e.g.

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

$$x_1, x_2 \in \mathbb{Z}$$

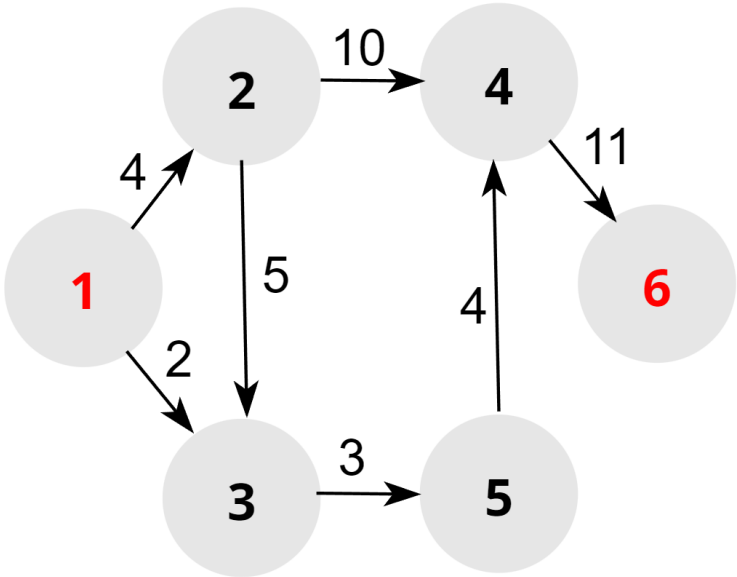


Integer programming is NP-complete. At present, all known algorithms for NP-complete problems require time that is superpolynomial in the input size.

Integer Programming

Find the shortest path for the given graph, start & end:

vertices $V = \{1, 2, 3, 4, 5, 6\}$
weights $q_{uv} \in \{q_{12}, q_{13} \dots\}$



Integer Programming

Shortest path problem formulated as a LP problem.

Objective:

$$minimize \sum_{u,v \in V} q_{uv} x_{uv}$$

s - source t - sink

x_{uv} - 0/1 for connection off/on

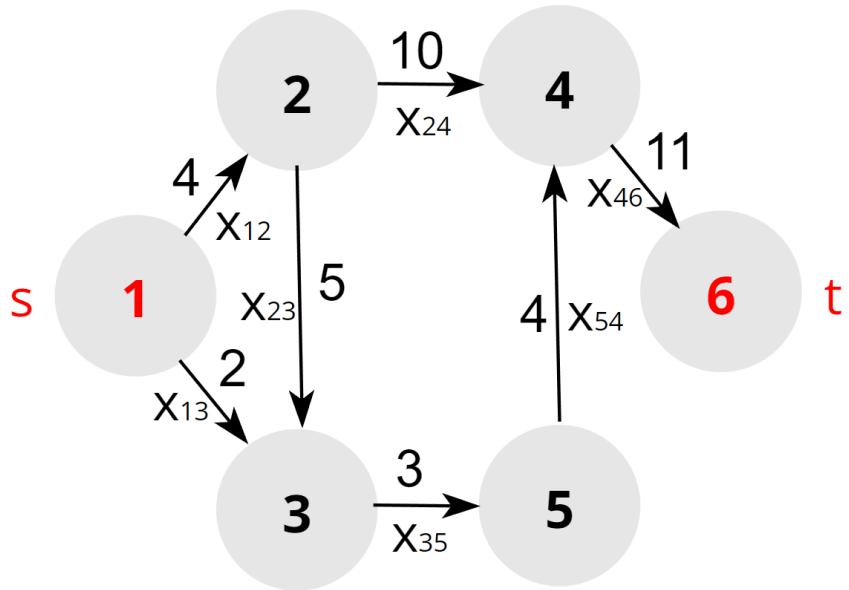
Constraints:

$$\sum_{u \in \{2,3\}} x_{su} = 1$$

$$\sum_{u,w \in V} x_{uv} = \sum_{u,w \in V} x_{vw}, \quad \forall v \in V \setminus \{s,t\}$$

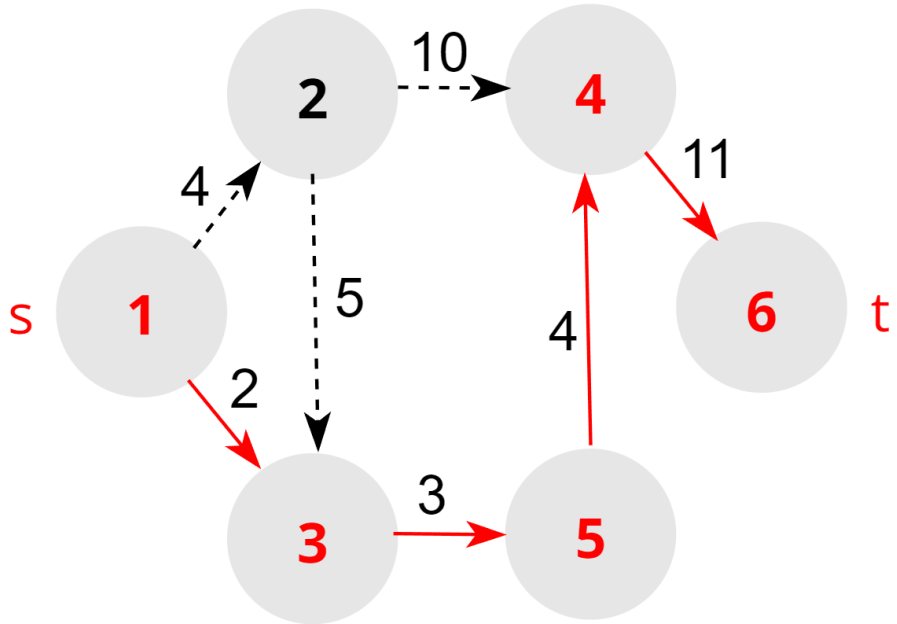
$$\sum_{u \in \{4,5\}} x_{ut} = 1$$

$$x_{ij} \geq 0$$



Integer Programming

$$x_{12} = 0$$
$$x_{13} = 1$$
$$x_{23} = 0$$
$$x_{24} = 0$$
$$x_{35} = 1$$
$$x_{54} = 1$$
$$x_{46} = 1$$



Works even with general LP, not necessary to use Integer variant! Nice educational example for ILP nevertheless.

Don't do it this way in practice, use Dijkstra, Bellman-Ford, Goldberg for negative cycles, contraction hierarchies for big graphs (navigation)...

Integer Programming

In [1]:
```
using JuMP
using HiGHS
```

In [2]:
```
model = Model(HiGHS.Optimizer);
```

In [3]:
```
@variable(model, x12>=0); @variable(model, x13>=0); @variable(model, x23>=0);
@variable(model, x24>=0); @variable(model, x35>=0); @variable(model, x54>=0);
@variable(model, x46>=0);
```

In [4]:
```
@objective(model, Min, 4x12 + 2x13 + 5x23 + 10x24 + 3x35 + 4x54 + 11x46);
```

In [5]:
```
@constraint(model, c1, x12 + x13 == 1.0); @constraint(model, c2, x12 == x23 + x24);
@constraint(model, c3, x13 + x23 == x35); @constraint(model, c4, x24 + x54 == x46);
@constraint(model, c5, x35 == x54);       @constraint(model, c6, x46 == 1.0);
```

In [6]:
```
optimize!(model);
```

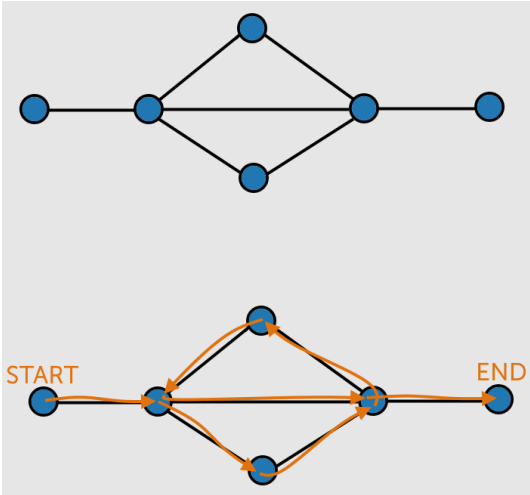
```
Running HiGHS 1.3.0 [date: 1970-01-01, git hash: e5004072b-dirty]
Copyright (c) 2022 ERGO-Code under MIT licence terms
Presolving model
3 rows, 4 cols, 8 nonzeros
1 rows, 3 cols, 3 nonzeros
0 rows, 0 cols, 0 nonzeros
Presolve : Reductions: rows 0(-6); columns 0(-7); elements 0(-14) - Reduced to empty
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Objective value     :  2.0000000000e+01
HiGHS run time      :          0.00
```

In [7]:
```
value(x12), value(x13), value(x23), value(x24), value(x35), value(x54), value(x46)
```

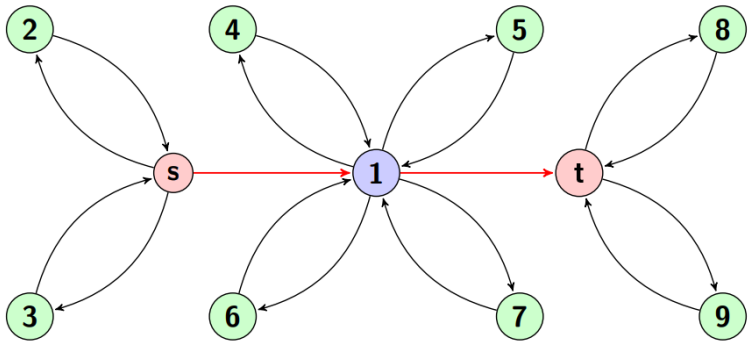
Out[7]:
```
(0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0)
```

Semi-Eulerian Graph

- Definition
  If a graph has an **open** trail (it starts and finishes at different vertices) that uses every edge, it is described as **semi-Eulerian**. It can also be called a semi-Eulerian trail.

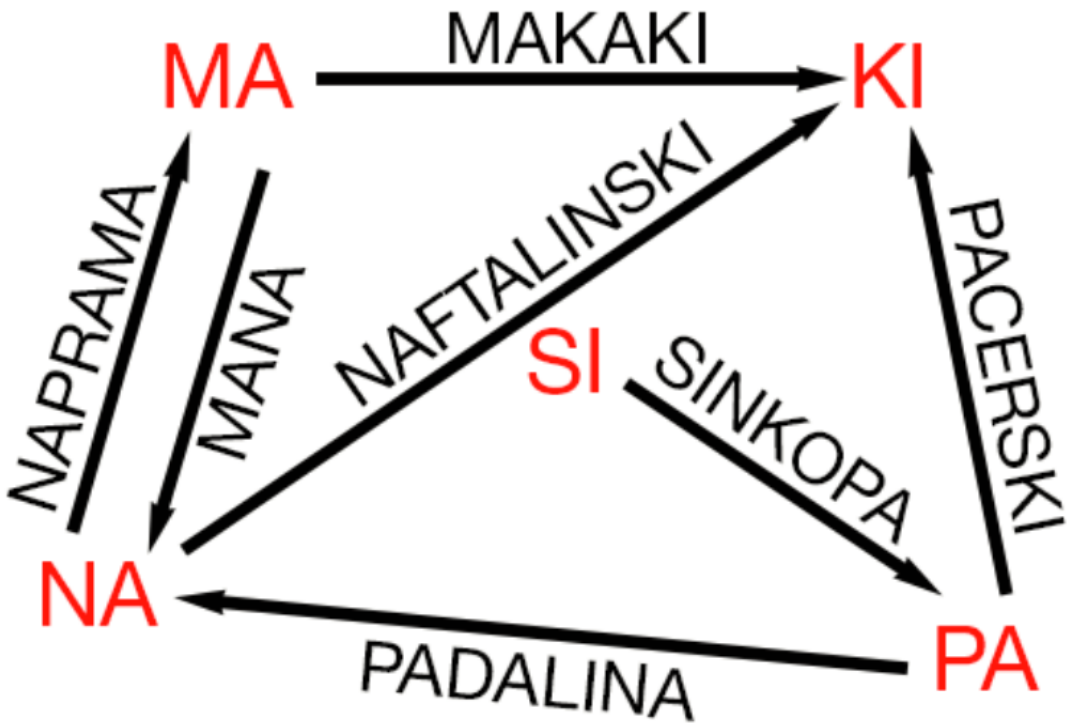


- Theorem
  The directed graph G is a semi-Eulerian graph, if and only if G is a connected graph, the in-degree of the **source** vertex s is less than the out-degree by 1, and the in-degree of the **sink** vertex t is greater than the out-degree by 1, and the in-degree of all other vertices is **equal** to the out-degree degree.



Semi-Eulerian Graph

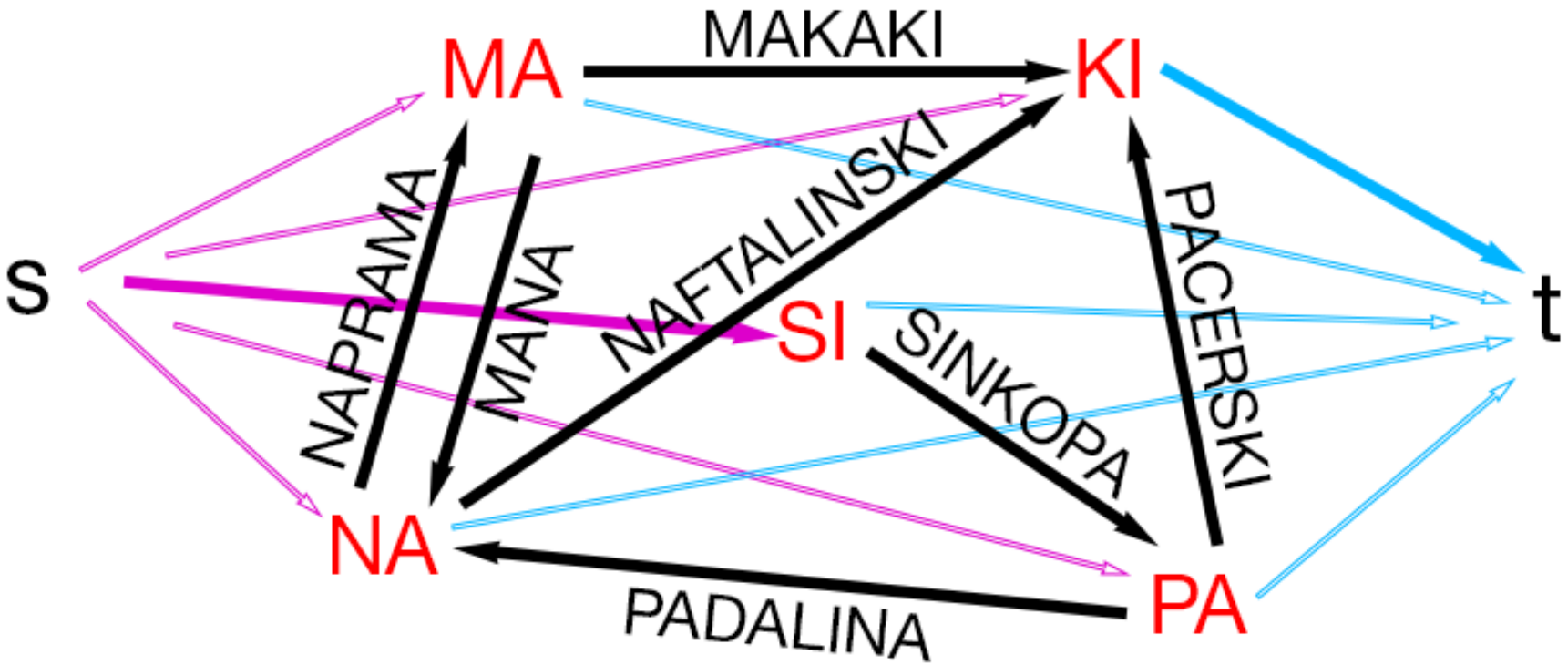
Let's now find the largest semi-Eulerian subgraph by using ILP.



Note we don't know where to start and end!

Semi-Eulerian Graph

Manually introduce new **initially non-existing** super-source s with exactly one *out* and super-sink t with exactly one *in* connection via any suffix.



s & t will "pick" the best start and end or the longest path.

Semi-Eulerian Graph

$$(\mathrm{P})\ \text{Maximize}\ z_0 = \sum_{i\in V\cup\{s\}} \sum_{j\in V\cup\{t\}} x_{ij}$$

Subject to :

$$\sum_{i\in V} x_{si} = 1$$

$$\sum_{i\in V} x_{ij} - \sum_{i\in V} x_{ji} = 0 \ \forall j \in V$$

$$\sum_{i\in V} x_{it} = 1$$

Connectivity

$$\sum_{k=1}^{2^{|V\cup\{s,t\}|}-2} (1 - y_k) = 1$$

$$\sum_{\substack{i\in Sk \\ j\in V\cup\{s,t\}\setminus S_k}} x_{ij} \geq y_k - g_k, \forall S_k$$

$$g_k + \sum_{\substack{i\in S_k \\ j\in S_k}} x_{ij} \geq 1, \forall S_k$$

$$g_k \sum_{\substack{i\in S_k \\ j\in S_k}} f_{ij} + \sum_{\substack{i\in S_k \\ j\in S_k}} x_{ij} \leq \sum_{\substack{i\in S_k \\ j\in S_k}} f_{ij}, \forall S_k$$

$$0 \leq x_{ij} \leq f_{ij}, \forall i \in V, \forall j \in V$$

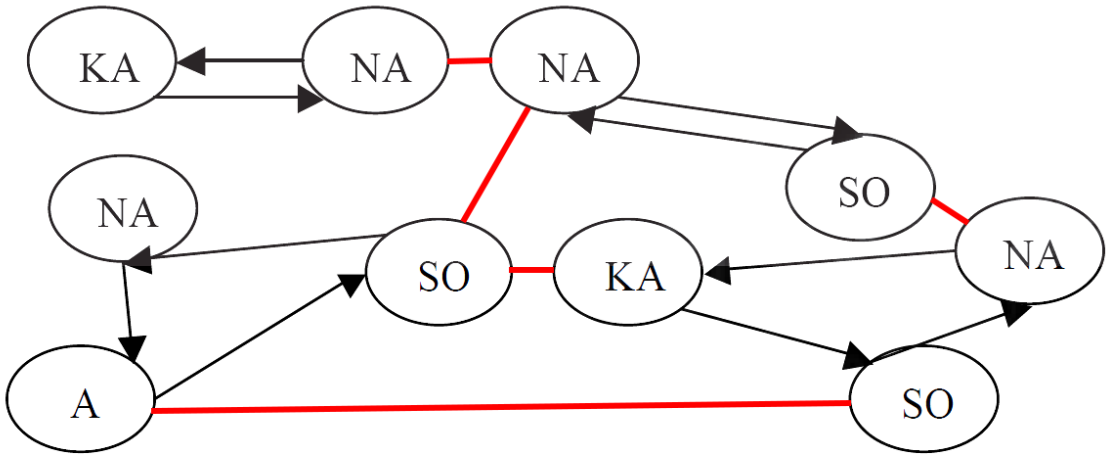
$$0 \leq x_{sj} \leq 1, \ \forall j \in V$$

$$0 \leq x_{it} \leq 1, \forall i \in V$$

$$x_{ij} \in \mathbf{Z}, \forall i \in V \cup \{s\}, \forall j \in V \cup \{t\}$$

$$y_k \in \{0,1\}, \forall k$$

$$g_k \in \{0,1\}, \forall k$$



All same-direction connections between some two nodes are bundled using f_{ij} coefficients. This way the problem is further reduced from 159k to "only" 20k unknowns.

Computation

In first step we ignore the connectivity conditions and solve for the simplified problem without them:

$$(\mathrm{RP}_0)\ \text{Maximize}\ z_0 = \sum_{i \in V \cup \{s\}} \sum_{j \in V \cup \{t\}} x_{ij}$$

$$\text{Subject to:}$$

$$\sum_{i \in V} x_{si} = 1$$

$$\sum_{i \in V} x_{ij} - \sum_{i \in V} x_{ji} = 0 \ \forall j \in V$$

$$\sum_{i \in V} x_{it} = 1$$

$$0 \le x_{ij} \le f_{ij}, \forall i \in V, \forall j \in V$$

$$0 \le x_{sj} \le 1, \ \forall j \in V$$

$$0 \le x_{it} \le 1, \forall i \in V$$

$$x_{ij} \in \mathbf{Z}, \forall i \in V \cup \{s\}, \forall j \in V \cup \{t\}$$

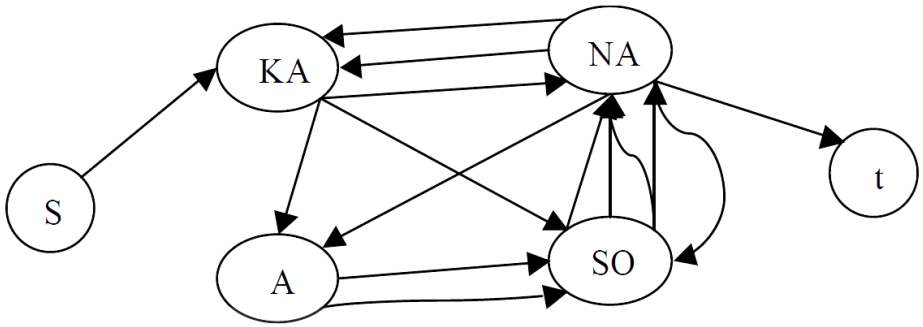
The ILP problem defined & solved using *Wolfram Mathematica*.

Our semi-Eulerian subgraph is the largest connected component of this solution.

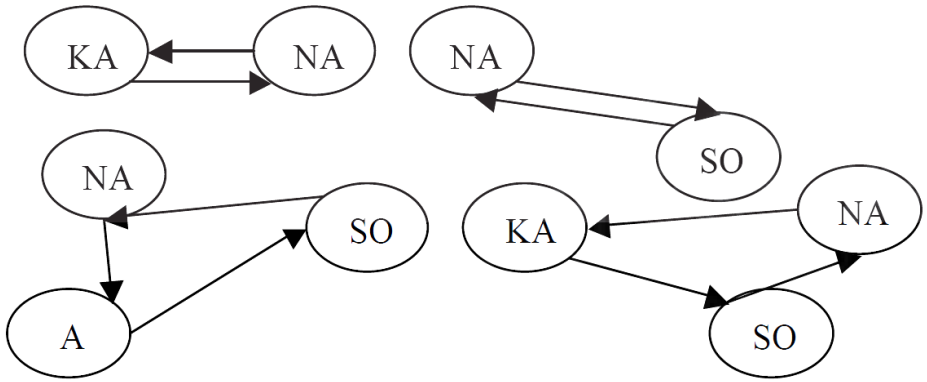
Solving ILP with 20.468 unknowns took only 3 seconds! How if it's NP-complete and all algorithms are superpolynomial?!

Solution Extraction

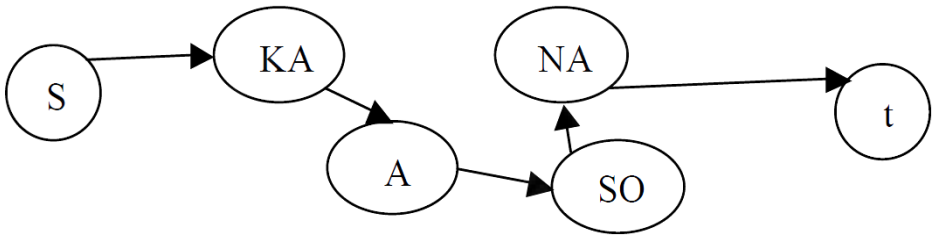
1. Found semi-Eulerian subgraph:



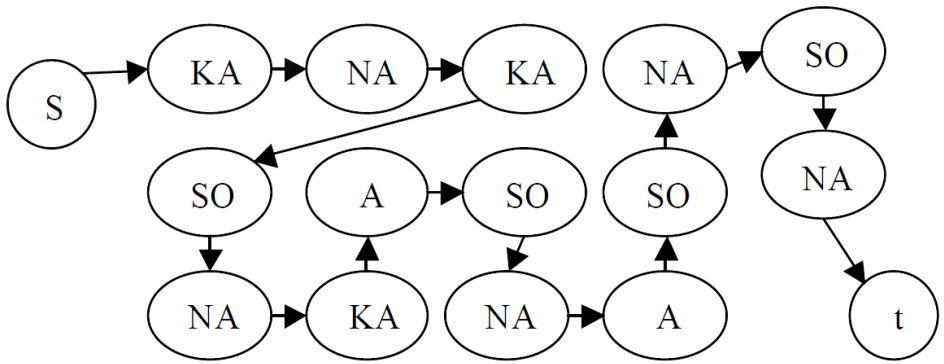
2. Extract cycles in $\mathcal{O}((E+V)(c+1))$:



3. Get cycle-free *s*-*t* path, property of sE graph, i.e. we didn't get a tree:



4. Merge cycles, 26.552 words solution:



Finalizing the Connectivity Topic

Iterative procedure forces for more connections. Paper claims this is optimal, but without proof. **Didn't improve the first iteration RP_0.**

$$(RP_k)\ \text{Maximize}\ z_0 = \sum_{i \in V \cup \{s\}} \sum_{j \in V \cup \{t\}} x_{ij}$$

Subject to :

$$\sum_{i \in V} x_{si} = 1$$

$$\sum_{i \in V} x_{ij} - \sum_{i \in V} x_{ji} = 0 \quad \forall j \in V$$

$$\sum_{i \in V} x_{it} = 1$$

Connectivity step

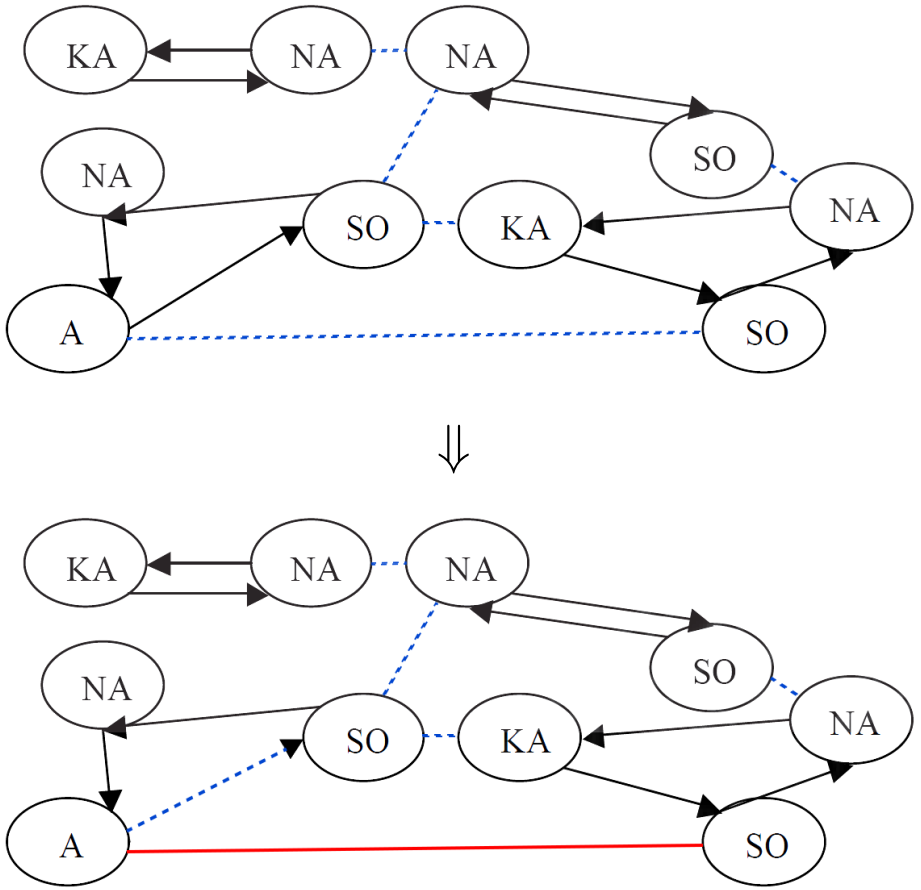
$$\sum_{\substack{i \in V_l^* \\ j \in V \setminus V_l^*}} x_{ij} \geq 1, l = 0,1,\cdots,k-1$$

$$0 \leq x_{ij} \leq f_{ij}, \forall i \in V, \forall j \in V$$

$$0 \leq x_{sj} \leq 1, \quad \forall j \in V$$

$$0 \leq x_{it} \leq 1, \forall i \in V$$

$$x_{ij} \in \mathbf{Z}, \forall i \in V \cup \{s\}, \forall j \in V \cup \{t\}$$

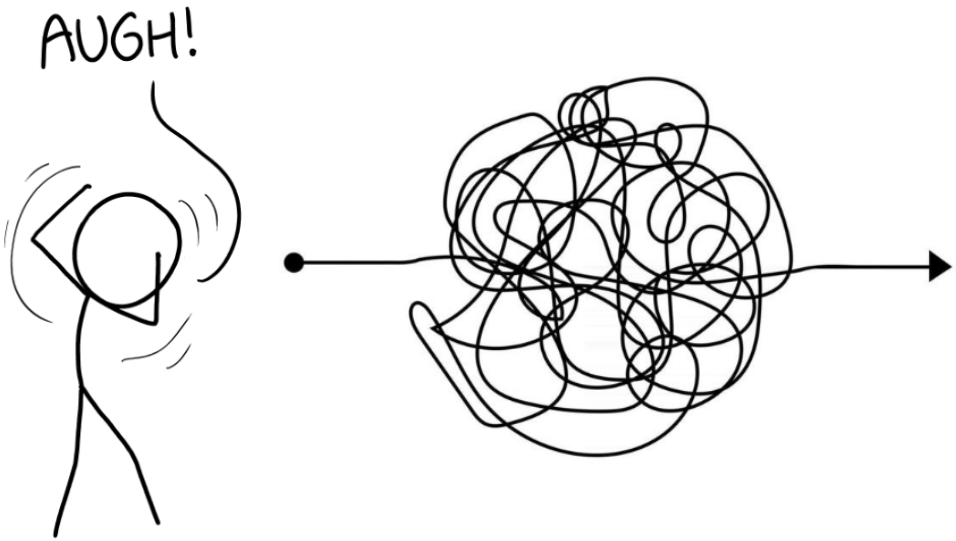


Finalizing the Connectivity Topic

We used the strong connectivity against it.

Calculation was *much* simpler due to ignoring the connectivity, and the solution got "connected by itself"!

The Problem



Judo Solution

So maybe our case isn't that general after all.

Open for Discussion

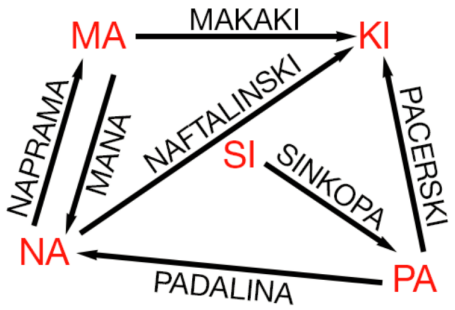
- What is the complexity of the procedure?
- What is the worst case?
- Is the suspiciously fast ILP solution optimal?
- Is the iterative connectivity procedure optimal as per the paper?

`https://en.wikipedia.org/wiki/P_versus_NP_problem`

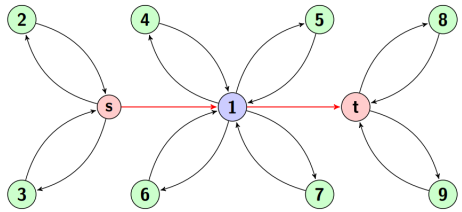
The empirical average-case complexity (time vs. problem size) of such algorithms can be surprisingly low. An example is the simplex algorithm in linear programming, which works surprisingly well in practice; despite having exponential worst-case time complexity, it runs on par with the best known polynomial-time algorithms.

Summary

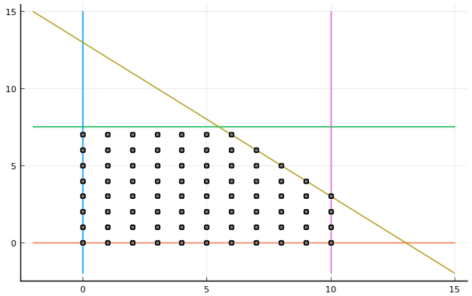
1. **Efficient formulation**
   Define suffix graph with words as edges and find the **longest path** containing the most edges.



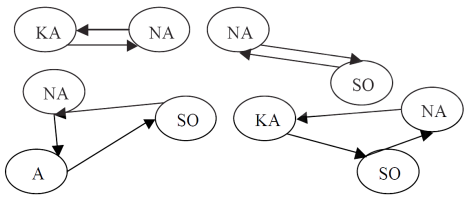
2. **Efficient technique**
   Find the **longest path** by extracting the largest connected semi-Eulerian subgraph using ILP.



3. **Implementation & Computation**
   Use some Integer Programming solver to find the best possible connections configuration.



4. **Solution Extraction**
   Extract all cycles of the found semi-Eulerian subgraph and construct the **longest path**.



Thank You So Much!

Questions, comments, reflections...