Graph Path Optimization using Integer Programming

Hrvoje Abraham

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

October 28, 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.

Word-Chain Game

- Croatia - Kalodont, *Kaladont* - ? n-terokut ?
- Romania - Fazan
- England - Word Association Football
- India - Antakshari - a last letter song-chain game
- France - Jeu des Kyrielles
- China - jielong (接龍)
- Korea - ggeut-mar-it-gi (끝말잇기)
- Japan - Shiritori (しりとり) (*eng. taking the end)*

rajio (ラジオ) → onigiri (おにぎり) → risu (りす) → sumou (すもう) → udon (うどん)

A player who plays a word ending in the mora "N" (ん) loses the game, as no Japanese word begins with that character.

The paper

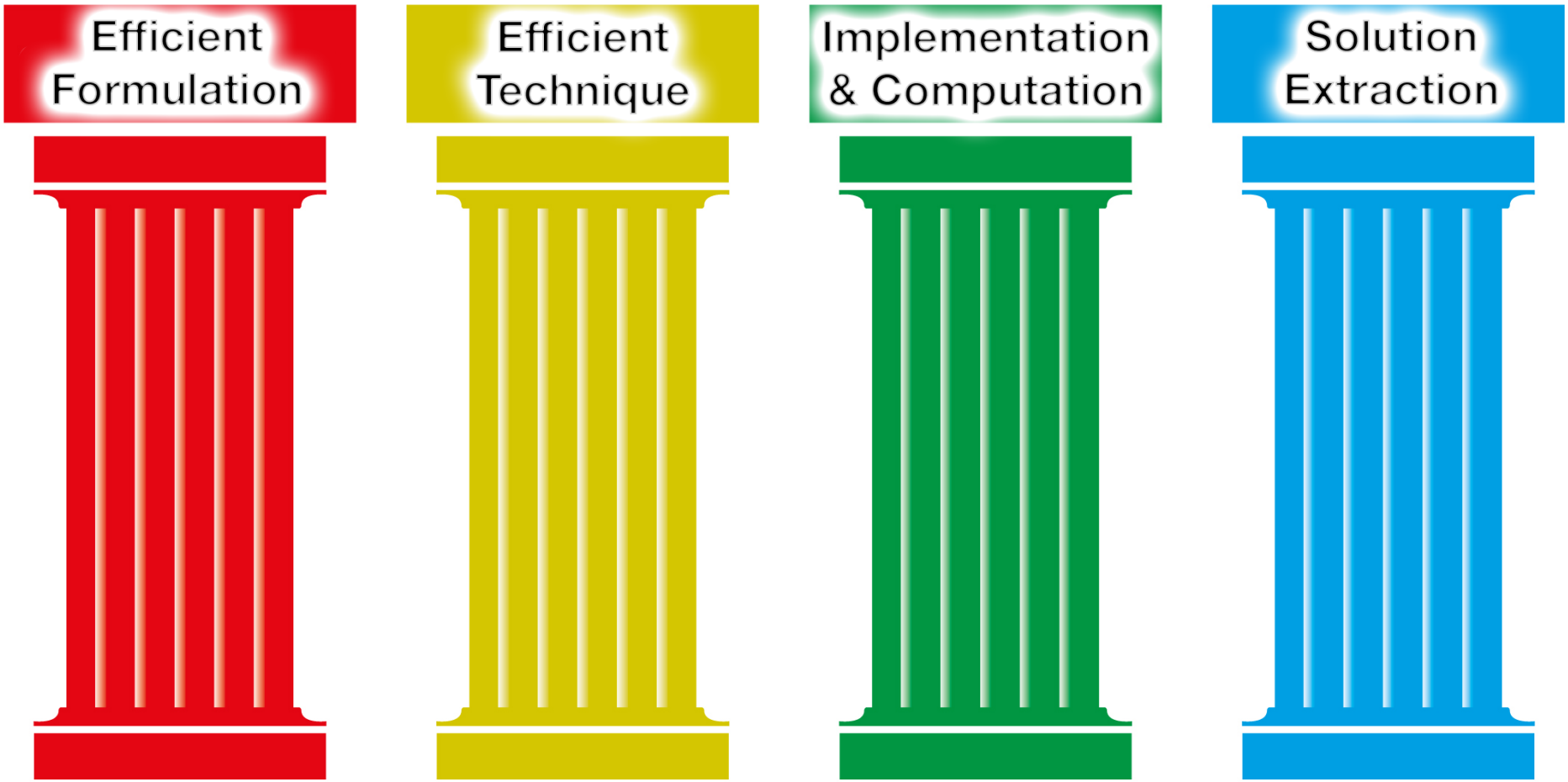
SOLVING THE LONGEST WORD-CHAIN PROBLEM

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.

Four Pillars of the Action Plan



Four Pillars of the Action Plan

1. Efficient formulation
   Define the words graph with (pre/suf)fixes as nodes, and words as edges and find the **longest path**. This is much better than words as nodes.

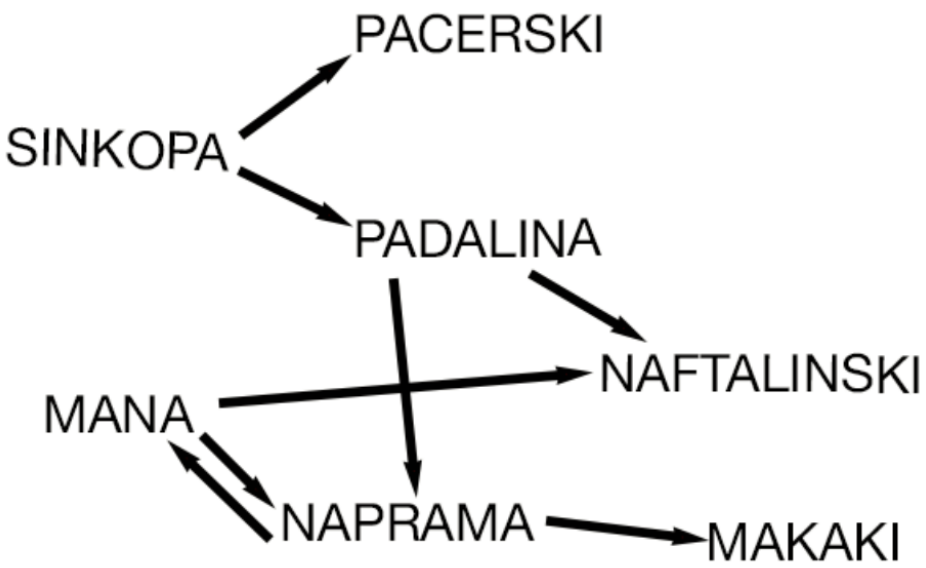
2. Efficient technique
   Find the **longest path** of the graph by extracting the largest semi-Eulerian subgraph using Integer Programming.

3. Implementation & Computation
   Use some of the available (free) Integer Programming or Mixed Programming solvers to solve for desired connections configuration.

4. Solution Extraction
   Identify the found semi-Eulerian subgraph, unwind all the cycles and construct the corresponding **longest path**.

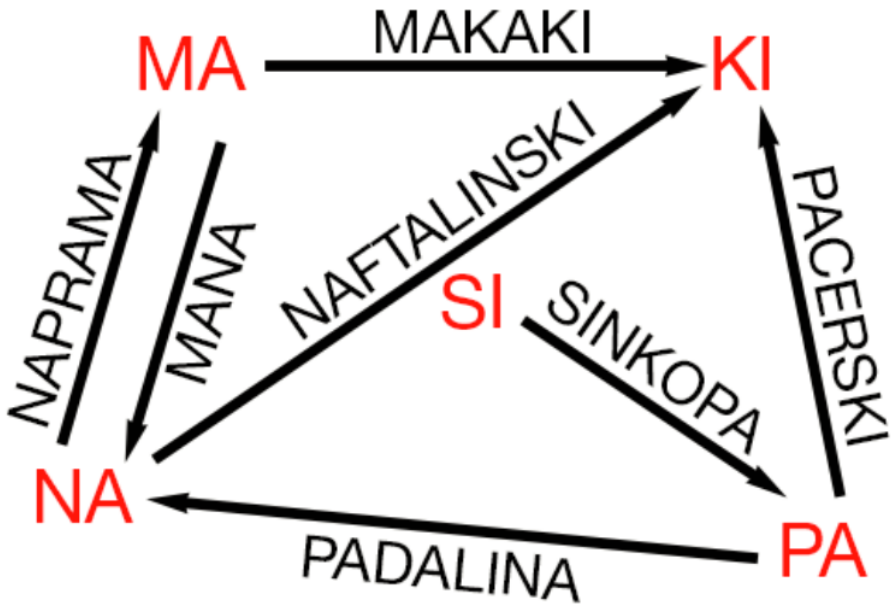
Efficient Formulation

Hamiltonian formulation



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

Eulerian formulation



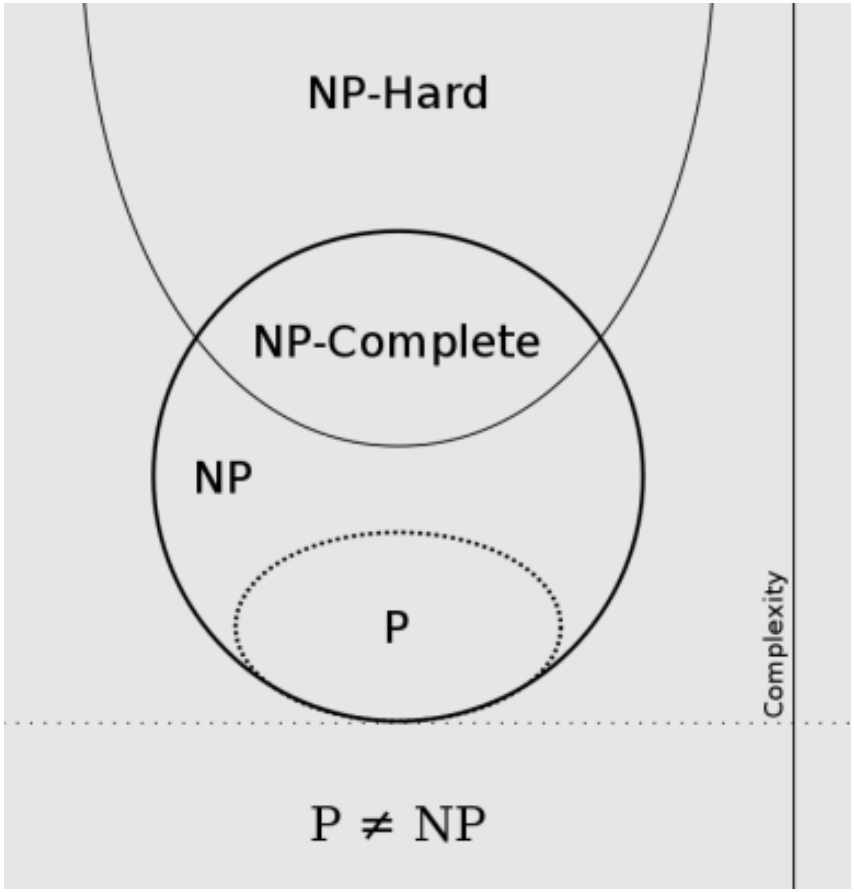
509 nodes
159.797 edges
Longest Path = Most Edges

Longest Path Problem

- Directed Acyclic Graph (DAG) case is frequent, and solvable in $\mathcal{O}(N)$.
- Rectangular Grid Graph case solvable in $\mathcal{O}(N)$.
- **We consider a general case**, not acyclic or other simple 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

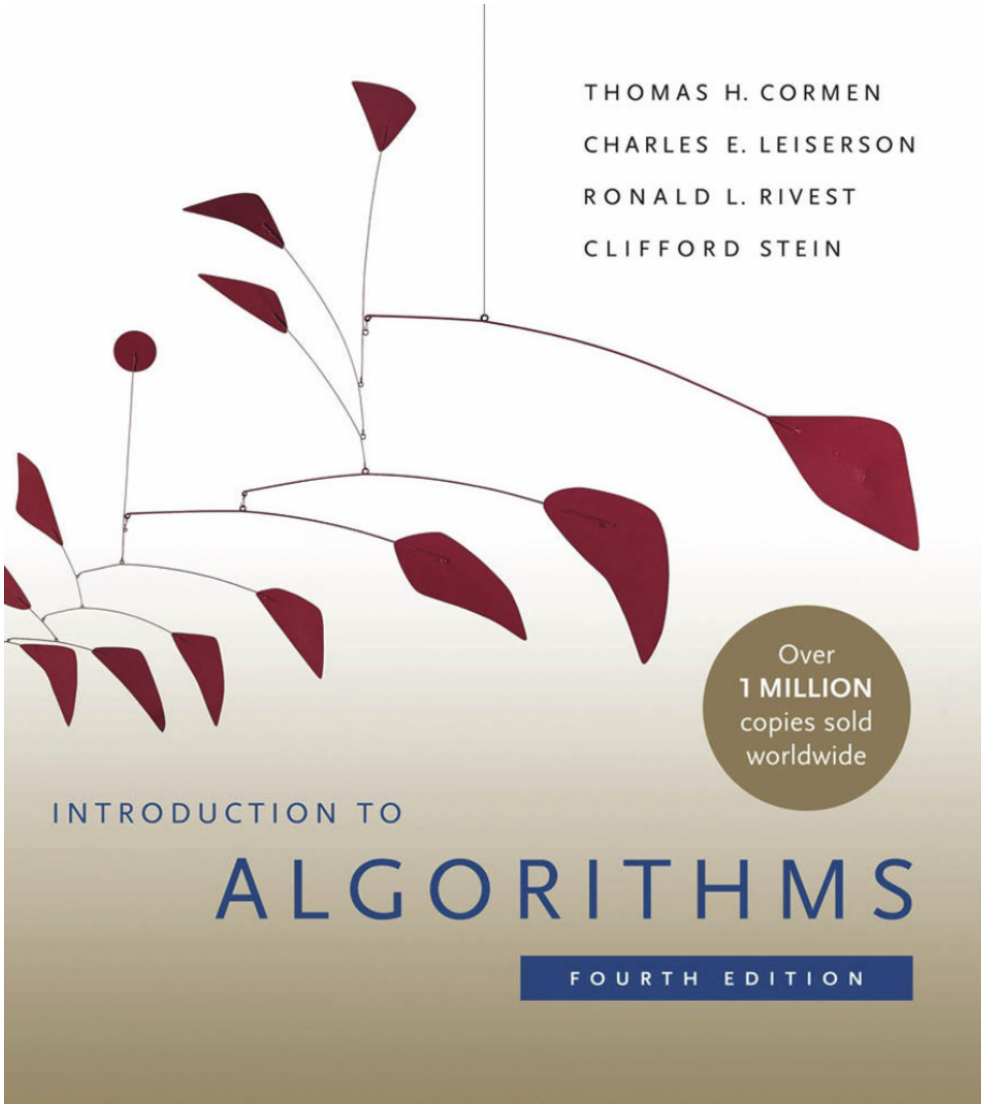
Some complexities for n-vertex and m-edge graphs:

- 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) - $\mathcal{O}_{time}(n^2\,2^n)$, $\mathcal{O}_{space}(n\,2^n)$
- Polynomial time algorithm known for finding $\Omega(log^2 n/log\ log\ n)$ length path in any directed n-vertex Hamiltonian graph.

Hard to approximate with formal guarantees:

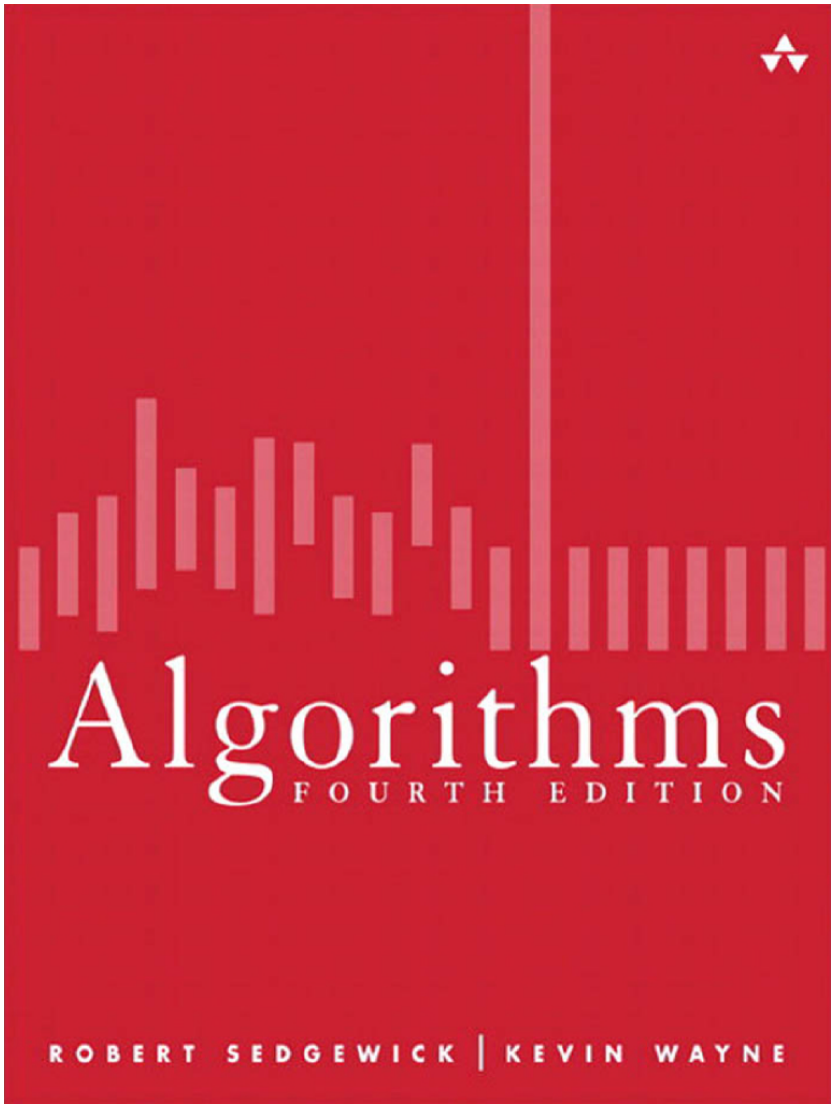
- For any $\epsilon < 1$ finding a path of length $n - n^\epsilon$ in an n-vertex Hamiltonian graph is NP-hard. - Karger, Motwani, Ramkumar (1997).

Longest Path in the Literature



- NEW! 4th Edition, April 2022
- some discussion for DAGs
- Fourth Edition news:

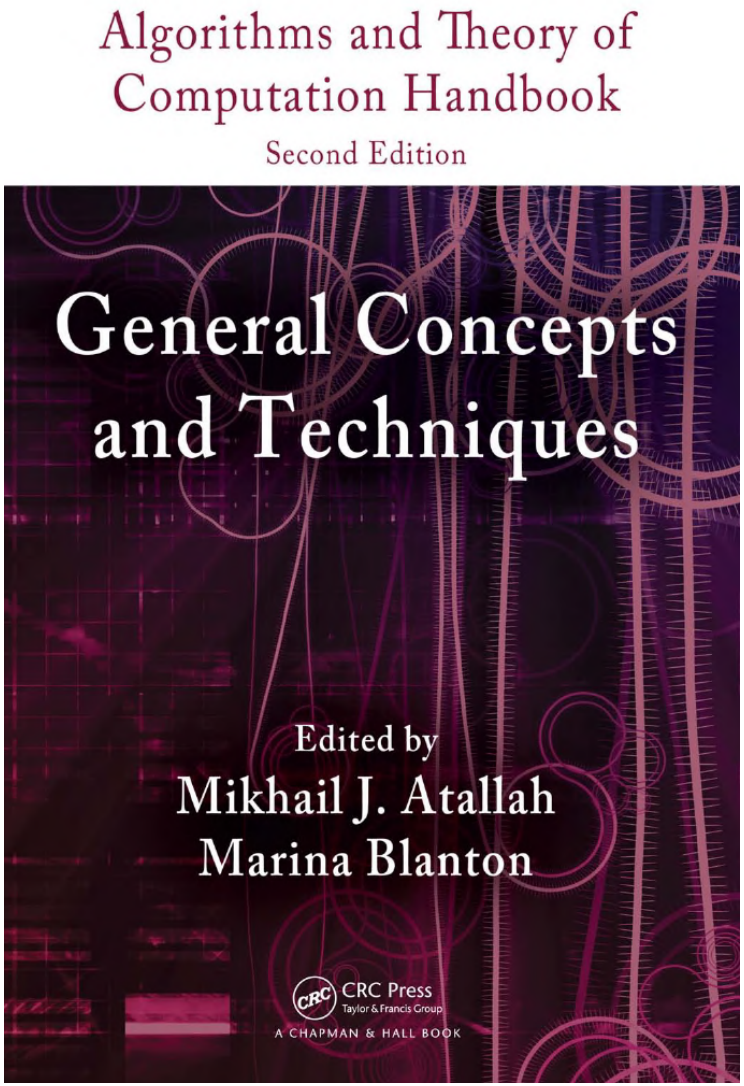
Longest Path in the Literature



- considered for DAGs
- mention of general variant as an example of optimization problems:

Other types of problems. The concept of search problems is one of many ways to characterize the set of problems that form the basis of the study of intractability. Other possibilities are *decision* problems (does a solution exist?) and *optimization* problems (what is the best solution)? For example, the longest-paths length problem on page 911 is an optimization problem, not a search problem (given a solution, we have no way to verify that it is a longest-path length). A search version of this problem is to *find* a simple path connecting all the vertices (this problem is known as the *Hamiltonian path problem*). A decision version of the problem is to ask whether *there exists* a simple path connecting all the vertices. Arbitrage, boolean satisfiability, and Hamiltonian path are search problems; to ask whether a solution exists to any of these problems is a decision problem; and shortest/longest paths, maxflow, and linear programming are all optimization problems. While not technically equivalent, search, decision, and optimization problems typically reduce to one another (see EXERCISE 6.58 and 6.59) and the main conclusions we draw apply to all three types of problems.

Longest Path in the Literature



7.4.8.4 Longest Path

In project scheduling, a DAG is used to model precedence constraints between tasks. A longest path in this graph is known as a critical path and its length is the least time that it takes to complete the project. The problem of computing the longest path in an arbitrary graph is NP-hard. However, longest paths in a DAG can be computed in linear time by using DFS. This method can be generalized to the case when vertices have weights denoting duration of tasks.

34.10 Hard-to-Approximate Problems

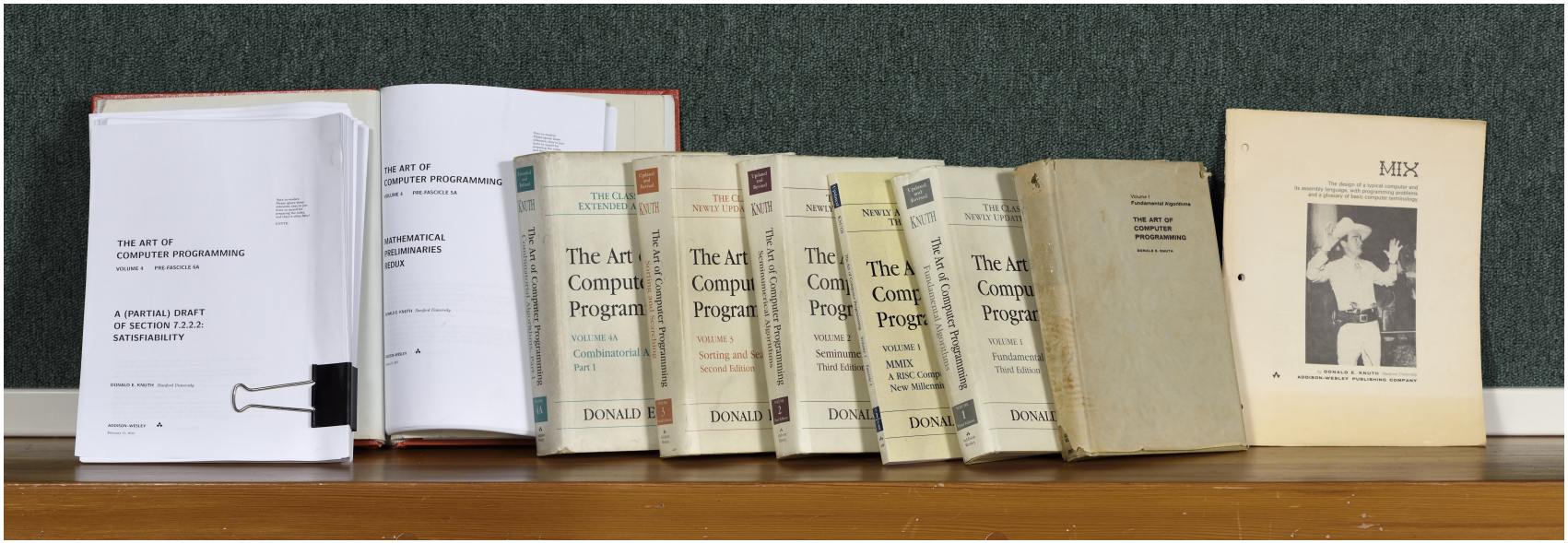
For some optimization problems, worst-case performance guarantees are unlikely to be possible: it is NP-hard to approximate these problems even if one is willing to accept very poor performance guarantees. Following are some examples [7, Sections 10.5 and 10.6].

Maximum clique. Given a graph, find a largest set of vertices that are pairwise adjacent (see also [6]).

Minimum vertex coloring. Given a graph, color the vertices with a minimum number of colors so that adjacent vertices receive distinct colors.

Longest path. Given a graph, find a longest simple path.

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:

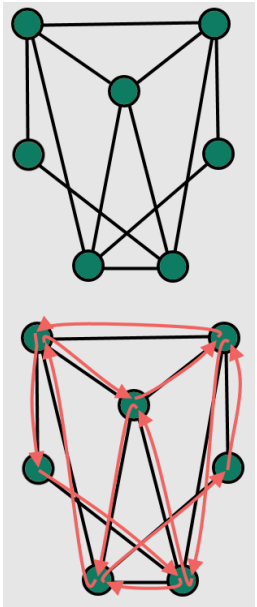
- 7.2.2.3. Constraint satisfaction
- 7.2.2.4. Hamiltonian paths and cycles
- 7.2.2.5. Cliques
- 7.2.2.6. Covers
- 7.2.2.7. Squares
- 7.2.2.8. A potpourri of puzzles
- 7.2.2.9. Estimating backtrack costs
- 7.2.3. Generating inequivalent patterns
- 7.3. Shortest paths
- 7.4. Graph algorithms
- 7.4.1. Components 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

Intermezzo 1

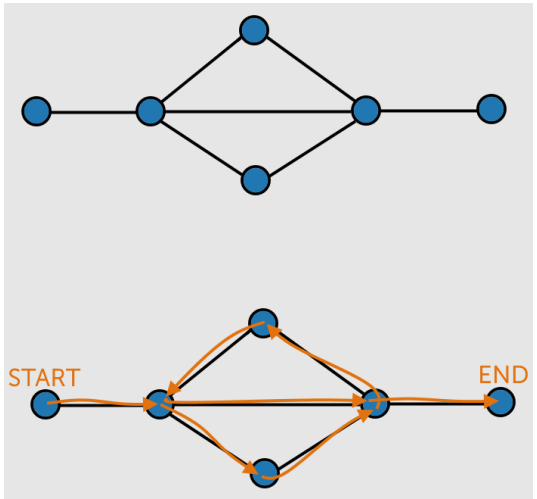
Questions, comments, reflections...

Semi-Eulerian Graph

- Eulerian graph
  If a graph has a **closed** trail (it starts and finishes at the same vertex) that uses every edge, it is called **Eulerian**.

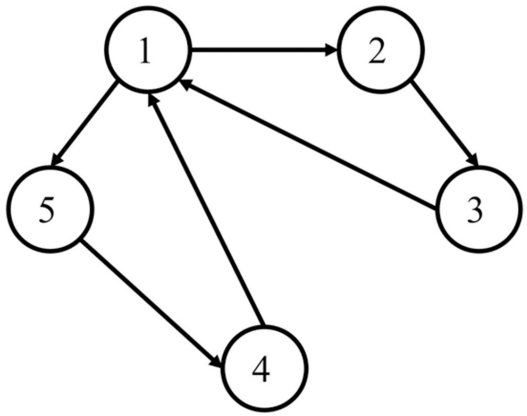


- Semi-Eulerian graph
  If a graph has an **open** trail (it starts and finishes at different vertices) that uses every edge, it is described as **semi-Eulerian**. It can also be called a semi-Eulerian trail.

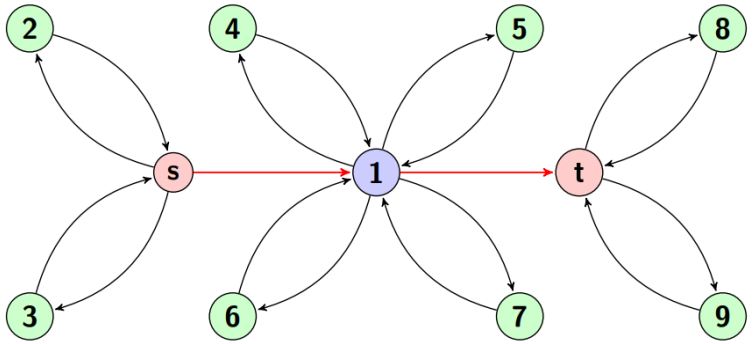


Directed semi-Eulerian Graph (Digraph)

- Eulerian digraph theorem:
  The directed graph G is an Euler graph, if and only if G is a connected graph, and the **in-degree** of all vertices is **equal** to the **out-degree**.



- Semi-Eulerian digraph 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.



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

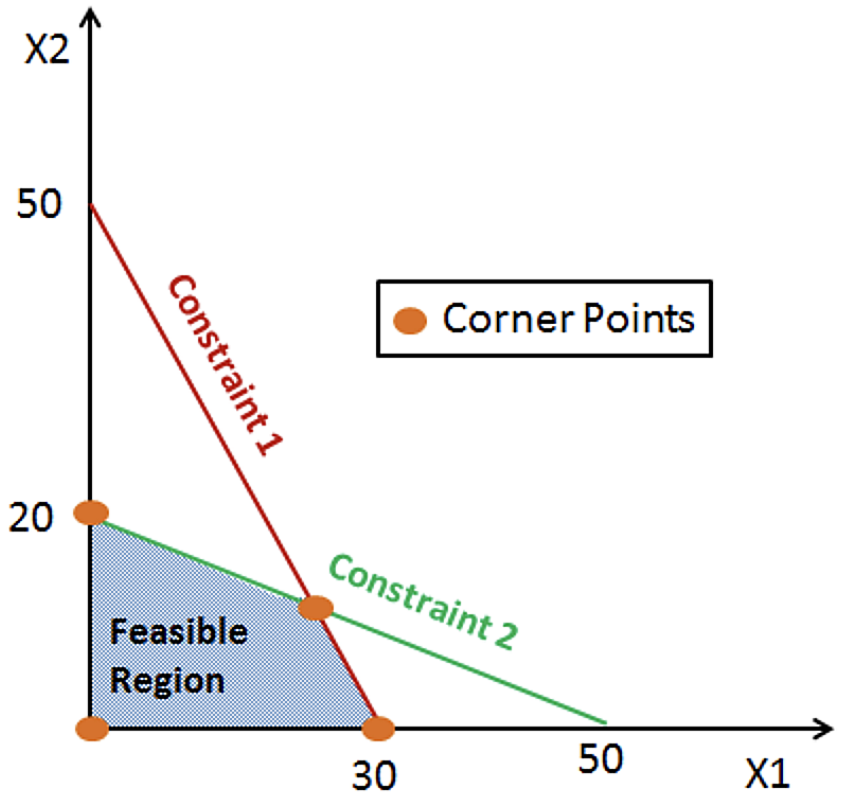
  e.g.

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

- **Non-negative variables**

  e.g.

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



The linear programming problem was first shown to be solvable in polynomial time by Leonid Khachiyan in 1979.

Linear Programming

George Dantzig wrote in "LINEAR PROGRAMMING":

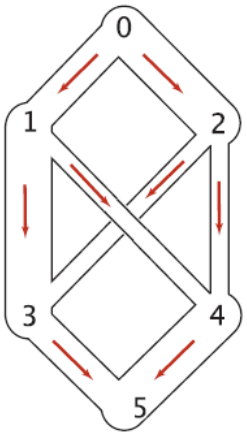
The military refer to their various plans or proposed schedules of training, logistical supply and deployment of combat units as a program. When I first analyzed the Air Force planning problem and saw that it could be formulated as a system of linear inequalities, I called my paper Programming in a Linear Structure. Note that the term 'program' was used for linear programs long before it was used as the set of instructions used by a computer. In the early days, these instructions were called codes. In the summer of 1948, Koopmans and I visited the Rand Corporation. One day we took a stroll along the Santa Monica beach. Koopmans said: "Why not shorten 'Programming in a Linear Structure' to 'Linear Programming'?"

E.g. TV program, training program, theatre program...

Linear Programming

maxflow problem

```
V
6
8        E
0 1  2.0
0 2  3.0
1 3  3.0
1 4  1.0
2 3  1.0
2 4  1.0
3 5  2.0
4 5  3.0
         capacities
```



LP formulation

Maximize $x_{35}+x_{45}$
subject to the constraints

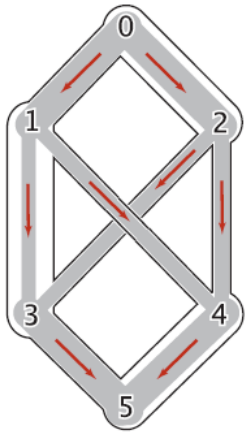
$$0 \le x_{01} \quad 2$$
$$0 \le x_{02} \quad 3$$
$$0 \le x_{13} \quad 3$$
$$0 \le x_{14} \quad 1$$
$$0 \le x_{23} \quad 1$$
$$0 \le x_{24} \quad 1$$
$$0 \le x_{35} \quad 2$$
$$0 \le x_{45} \quad 3$$
$$x_{01}=x_{13}+x_{14}$$
$$x_{02}=x_{23}+x_{24}$$
$$x_{13}+x_{23}=x_{35}$$
$$x_{14}+x_{24}=x_{45}$$

LP solution

$$x_{01}=2$$
$$x_{02}=2$$
$$x_{13}=1$$
$$x_{14}=1$$
$$x_{23}=1$$
$$x_{24}=1$$
$$x_{35}=2$$
$$x_{45}=2$$

maxflow solution

```
Max flow from 0 to 5
  0->2 3.0 2.0
  0->1 2.0 2.0
  1->4 1.0 1.0
  1->3 3.0 1.0
  2->3 1.0 1.0
  2->4 1.0 1.0
  3->5 2.0 2.0
  4->5 3.0 2.0
Max Flow value: 4.0
```



Example of reducing network flow to linear programming

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.

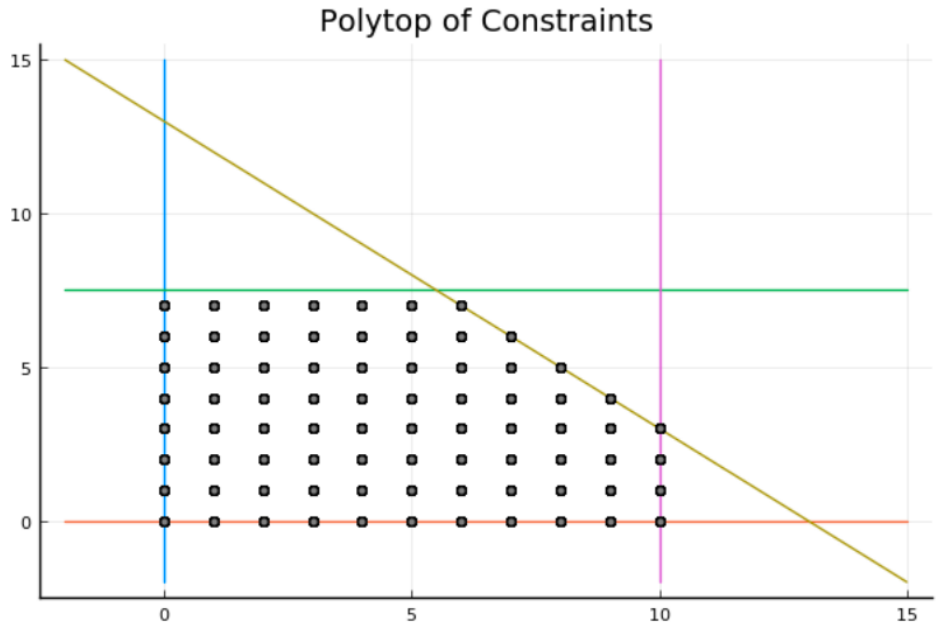
$$a_{11} x_1 + a_{12} x_2 \leq b_1$$
$$a_{21} x_1 + a_{22} x_2 \leq b_2$$
$$a_{31} x_1 + a_{32} x_2 \leq b_3$$

- **Non-negative variables**

  e.g.

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

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

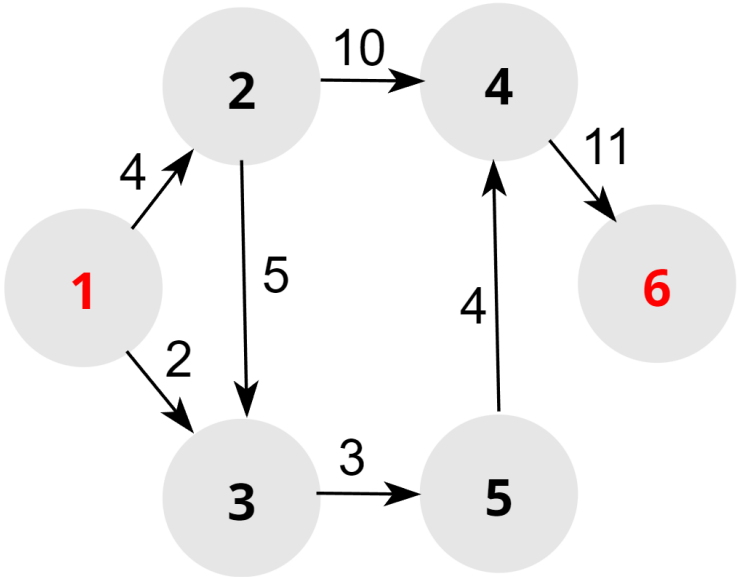


Integer programming is NP-complete. At present, all known algorithms for NP-complete problems require time that is superpolynomial in the input size.

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

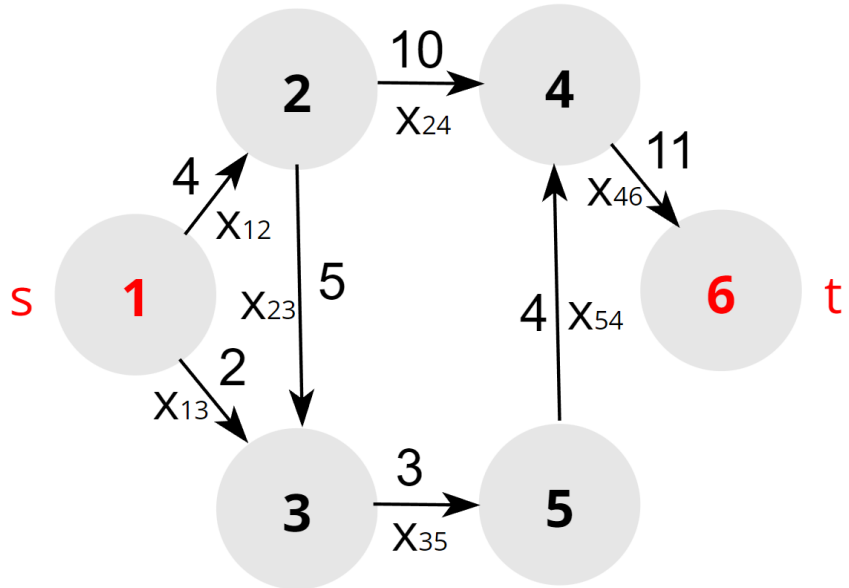
Shortest path problem formulated as a LP problem.

Objective:

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

s - source t - sink

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



With constraints:

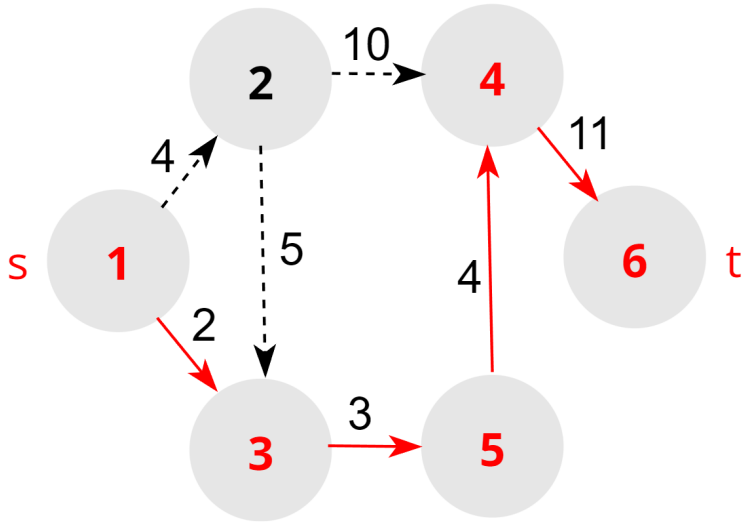
$$\sum_{u \in \{2,3\}} x_{su} = 1$$

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

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

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.

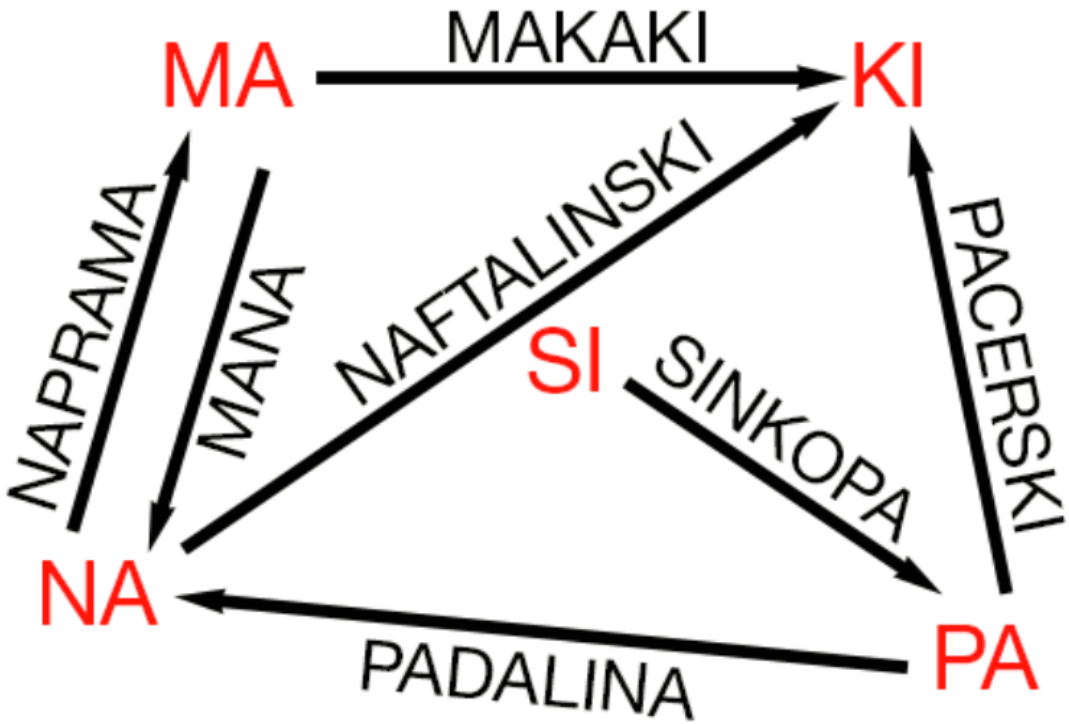
Don't do it this way in practice, use Dijkstra, Bellman-Ford, Goldberg for negative cycles, A^* if given start & end, contraction hierarchies for huge graphs (car navigation), etc...

Intermezzo 2

Questions, comments, reflections...

Integer Programming

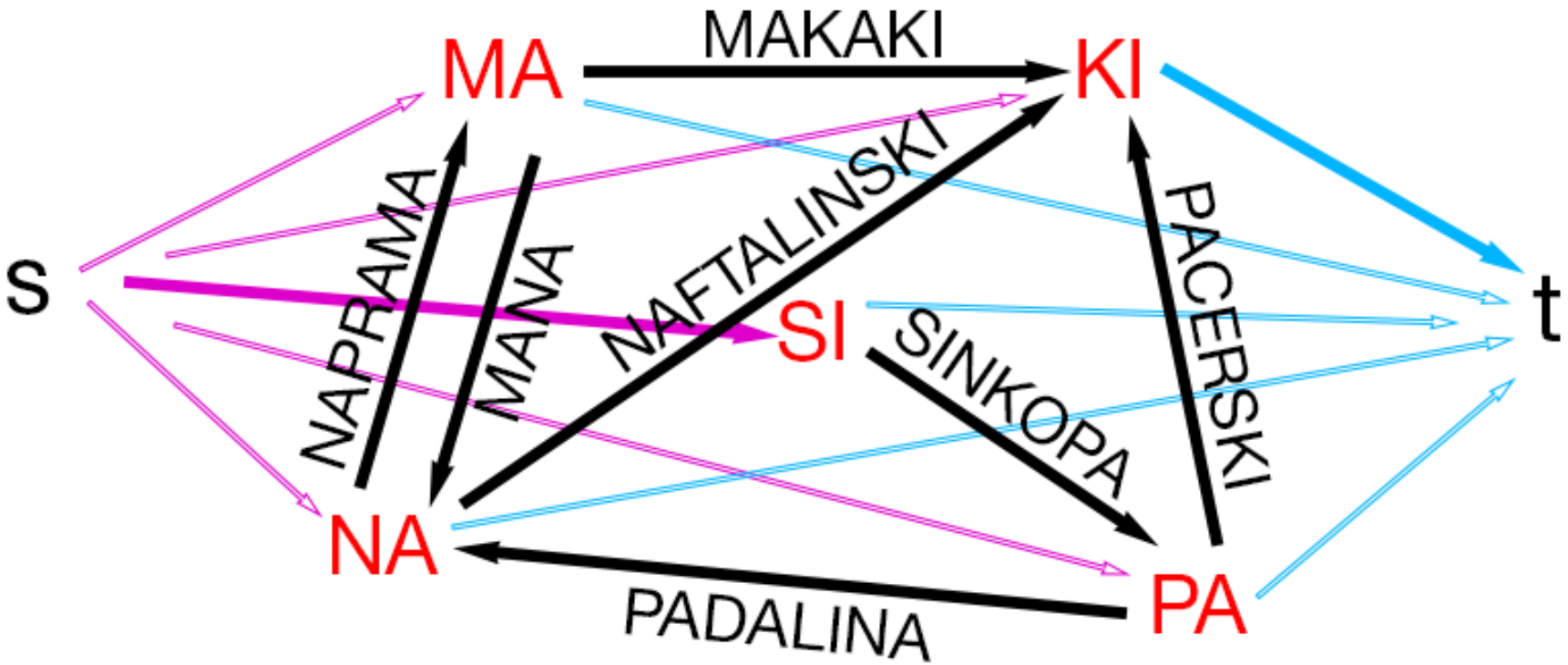
Let's now find the largest semi-Eulerian subgraph by using ILP.



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

Integer Programming

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.

Integer Programming

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

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

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

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

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

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

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

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

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

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

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

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

1) A graph is a connected graph.
2) In-degree and out-degree at every vertex but two are same. A vertex with one more out-degree becomes a beginning one and another vertex with one more in-degree becomes a terminating one (in our model, these vertices are s and t).

We formulate these conditions as an integer programming problem (P), but the condition 1) requires exponential order variables and constraints in this problem formulation.

All same-direction connections between some two nodes are bundled using f_{ij} coefficients. This way the problem is further reduced to "only" 20.468 unknowns.

Integer Programming

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

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

Subject to :

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

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

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

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

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

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

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

a) The graph of the solution of (RP_0) is a connected graph. In this case, the solution is also optimal for problem (P).

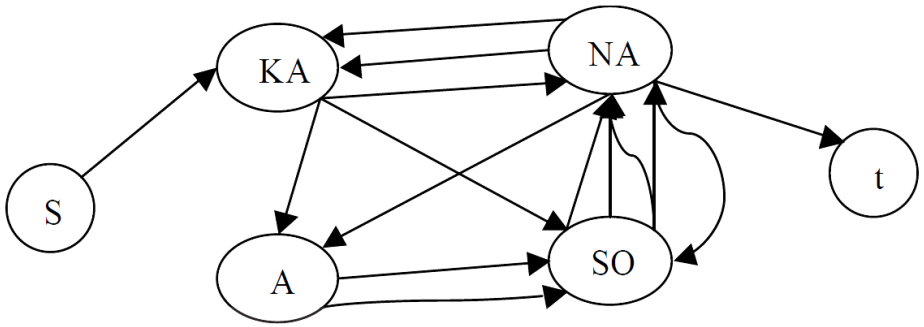
b) If the graph is disconnected, every connected component of the solution of (RP_0) is a semi-Eulerian graph or Eulerian graph, then two cases below is possible:

   i) The Eulerian path from s to t in the connected component gives an optimal solution of problem (P).

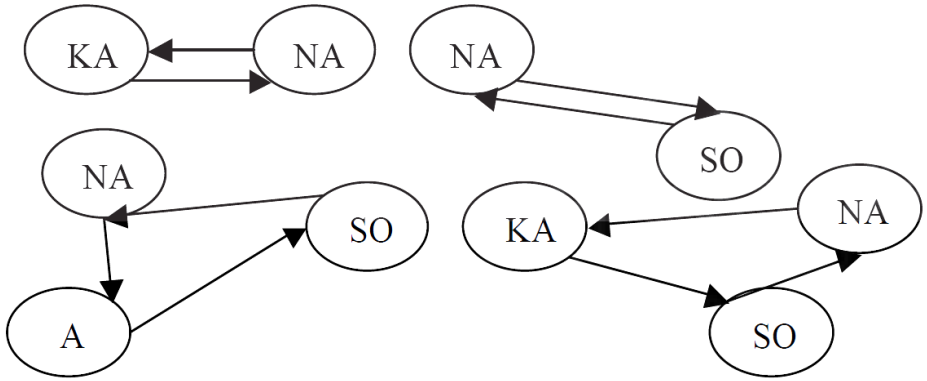
   ii) The path does not give an optimal solution of problem (P).

Solution Extraction

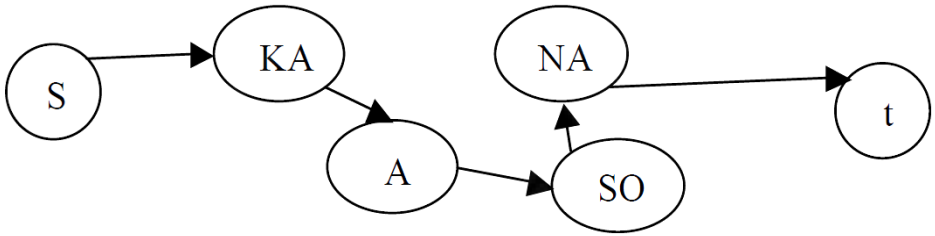
1. Found semi-Eulerian subgraph:



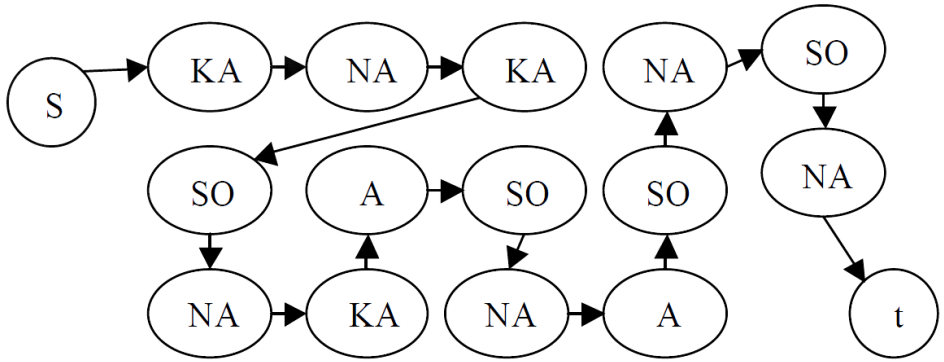
2. Extract all the cycles:



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



4. Break & merge cycles into path:



(I)LP solvers

Free:

- Gekko
- GLOP
- CLP
- glpk - GNU, used in the paper
- Julia - free solvers wrapped

Proprietary:

- AMPL
- Maple
- MATLAB
- Mathematica
- SCIP

Implementation & computation

Words imported and the graph constructed using *Julia*.

ILP system defined, solved & extracted with *Wolfram Mathematica*.

Solving the ILP system with 20.468 unknowns took only 3 seconds! How is this possible if the problem is NP-complete and all algorithms are at least of superpolynomial complexity?!

Identifying all the cycles is linear per cycle, i.e. $O((E+V)(c+1))$ where c is a number of cycles in the graph, e.g. by Johnson's or Tarjan's algorithm.

Found solution contains 26.552 words.

NOTE - Show the text file with the solution, and Mathematica notebook.

Wolfram Mathematica

Free try available at `https://www.wolframcloud.com/`.

LinearOptimization method documented at
`https://reference.wolfram.com/language/ref/LinearOptimization.html`.

- The option Method ->*method* may be used to specify the method to use. Available methods include:

Automatic	choose the method automatically
"Simplex"	simplex method
"RevisedSimplex"	revised simplex method
"InteriorPoint"	interior point method (machine precision only)
"CLP"	COIN library linear programming (machine precision only)
"MOSEK"	commercial MOSEK linear optimization solver
"Gurobi"	commercial Gurobi linear optimization solver
"Xpress"	commercial Xpress linear optimization solver

```
sol = x /. LinearOptimization[objective,
    {eulerConstraint, countConstraint, superSConstraint, superTConstraint},
    x ∈ Vectors[{Length[arcdata]}, Integers]]
```

Finalizing the Connectivity Topic

Connectivity was ignored for simplification. Iterative procedure forces for more connections. Paper claims this is optimal, but without proof.

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

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

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

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

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

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

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

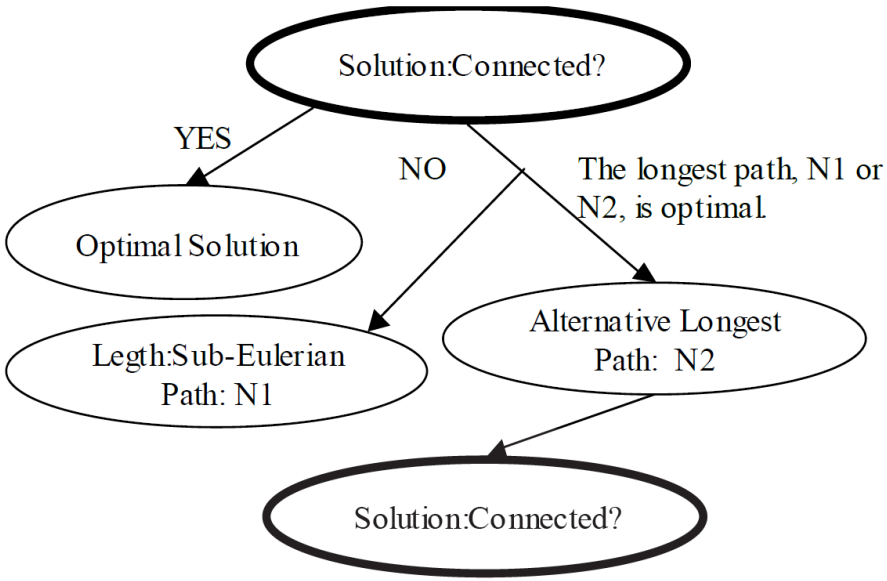


Figure 2: LP-Based Branch-and-Bound Method.

Seems that further iterations don't improve our specific solution.

Finalizing the Connectivity Topic

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

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

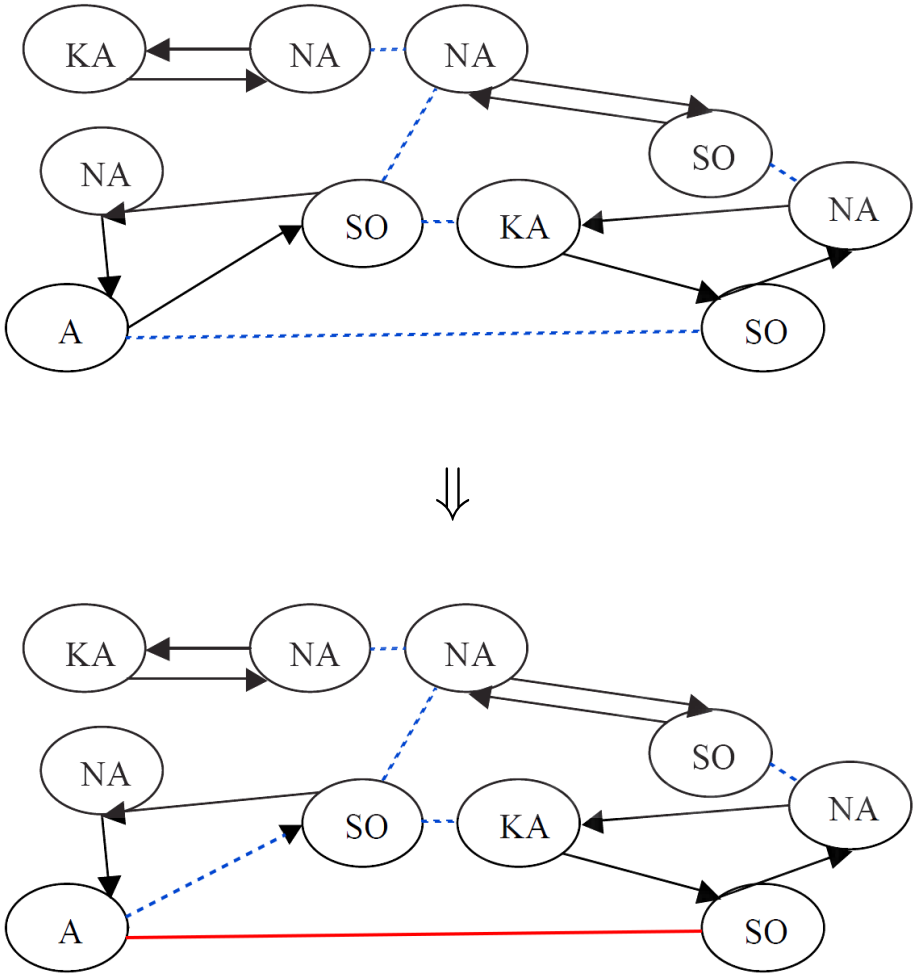
$$\sum_{\substack{i \in V_l^* \\ j \in V \setminus V_l^*}} x_{ij} \geq 1, l = 0,1,\cdots,k-1$$

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

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

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

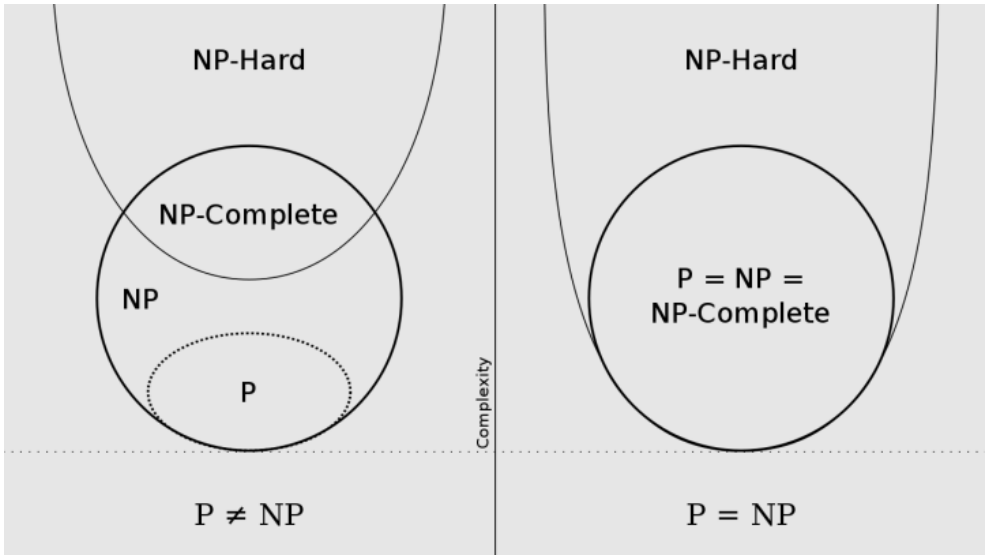
$$x_{ij} \in \mathbf{Z}, \forall i \in V \cup \{s\}, \forall j \in V \cup \{t\}$$



Open for Discussion

- Optimization, not search problem - can't check optimality in P
- Is the suspiciously fast ILP solution optimal?
  - Or this is an average case, so only the worst is superpolynomial?
- Is the iterative connectivity procedure optimal as per the paper?
  - Benefit by strongly connected graph - large connected solution in 1 iter.
- What is the worst case for this entire procedure?

The problem is NP-complete, solving it polynomial time would imply $P = NP$, still no proof, although "widely" believed it is not possible



Open for Discussion

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

On the other hand, even if a problem is shown to be NP-complete, and even if $P \neq NP$, there may still be effective approaches to tackling the problem in practice. There are algorithms for many NP-complete problems, such as the knapsack problem, the traveling salesman problem and the Boolean satisfiability problem, that can solve to optimality many real-world instances in reasonable time. 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.

Other (I)LP Examples

Strontium atomic clock system

$$
\begin{aligned}
f_r\, n_1 + f_0 &= f_1, & f_1^{min} \le f_1 \le f_1^{max} \\
f_r\, n_2 + f_0 &= f_2, & f_2^{min} \le f_2 \le f_2^{max} \\
f_r\, n_3 + f_0 &= f_3, & f_3^{min} \le f_3 \le f_3^{max} \\
f_r\, n_4 + f_0 &= f_4, & f_4^{min} \le f_4 \le f_4^{max}
\end{aligned}
$$

$$
\begin{aligned}
f_1^{min} &= f_1^r - 2\,\gamma_1, & f_1^{max} &= f_1^r - \gamma_1/2 \\
f_2^{min} &= f_2^r - \gamma_2, & f_2^{max} &= f_2^r + \gamma_2 \\
f_3^{min} &= f_3^r - 2\,\gamma_3, & f_3^{max} &= f_3^r - \gamma_3/2 \\
f_4^{min} &= f_4^r - \gamma_4, & f_4^{max} &= f_4^r + \gamma_4
\end{aligned}
$$

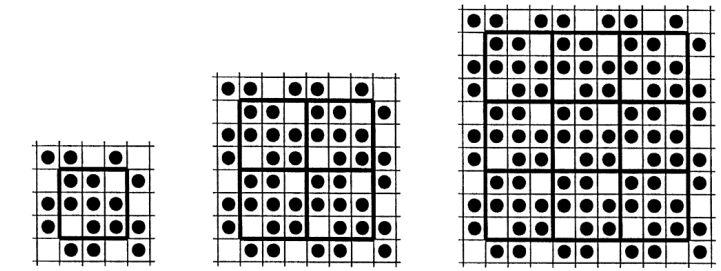
$$
\begin{aligned}
f_1^r &= 325\,251\,612\,993\,750.8, & \gamma_1 &= 32\,000\,000 \\
f_2^r &= 441\,331\,162\,793\,613.4, & \gamma_2 &= 1\,760\,000 \\
f_3^r &= 434\,828\,994\,151\,297.2, & \gamma_3 &= 7\,600 \\
f_4^r &= 423\,913\,370\,880\,042.4, & \gamma_4 &= 8\,960\,000
\end{aligned}
$$

$$f_r, f_0 \in \mathbb{R},\ f_r^{min} \le f_r \le f_r^{max},\ 0 \le f_0 < f_r$$

$$f_r^{min} = 100\,000\,000,\ f_r^{max} = 300\,000\,000$$

$$n_1, n_2, n_3, n_4 \in \mathbb{N}$$

Conway's "Game of Life"



$d = 17/25 = 0.68$
CPU time ≈ 2.6 seconds

$d = 46/64 \approx 0.7188$
CPU time ≈ 1.4 minutes

$d = 89/121 \approx 0.7355$
CPU time ≈ 5.0 hours

E.g. find min. stable density, or max. density at some generation:

$$\text{maximize} \sum_{e \in R} x_e$$

$$2x_e - \sum_{f \in N(e)} x_f \leqslant 0,$$

$$2x_e + 2\sum_{f \in S} x_f - \sum_{f \in N(e)-S} x_f \leqslant 8,$$

$$\forall S \subseteq N(e)\colon |S| = 4,$$

$$2x_e + \sum_{f \in N(e)} x_f \leqslant 8.$$

Summary

1. Efficient formulation
   Define the words graph with (pre/suf)fixes as nodes, and words as edges and find the **longest path**. This is much better than words as nodes.

2. Efficient technique
   Find the **longest path** of the graph by extracting the largest semi-Eulerian subgraph using Integer Programming.

3. Implementation & Computation
   Use some of the available (free) Integer Programming or Mixed Programming solvers to solve for desired connections configuration.

4. Solution Extraction
   Identify the found semi-Eulerian subgraph, unwind all the cycles and construct the corresponding **longest path**.

Thank You So Much!

Questions, comments, reflections...