Adapted and Constrained Dijkstra for Elastic Optical Networks

Ireneusz Szcześniak
Department of Communications
AGH University of Science and Technology
Poland

Bożena Woźna-Szcześniak
Institute of Mathematics and Computer Science
Jan Długosz University
Poland

ONDM 2016

Grant number DEC-2013/08/S/ST7/00576 from the Polish National Science Centre

Introduction

- WDM networks evolving into elastic optical networks (EONs).
- Fundamental problem: routing of a single demand.
- Routing and wavelength assignment (RWA) is NP-complete.
- RWA has the spectrum continuity constraint.
- Routing and spectrum assignment (RSA) is NP-complete.
- RSA has the spectrum continuity and contiguity constraints.
- Existing solutions:
  - heuristic algorithms: practical but suboptimal,
  - ILP formulations: optimal but impractical.

Contribution

Novel algorithm: adapted and constrained Dijkstra.

Our algorithm solves optimally and practically:

- the constrained RSA problem in the EONs,
- the constrained RWA problem in WDM networks.

Performance comparison with two heuristics:

- routing with the edge-disjoint shortest paths,
- routing with the Yen K-shortest paths.

The high-quality code using the Boost Graph Library (BGL) is freely-available under the General Public License (GPL).

Optimally and practically? Really?

- Yes, but we are constraining the RSA and RWA problems.
- Constriction: the limit on the path length.
- Our algorithm solves the constrained RSA and RWA problems:
  - OPTIMALLY: based on the optimal Dijkstra algorithm,
  - PRACTICALLY: returns results for large problems.
- However, we offer no proof.

Problem statement

Given:

- directed multigraph G,
- the cost $e.c$ of edge e,
- slices available $e.SSC$ on edge e,
- maximal path cost (length) m,
- demand d with n slices.

Sought:

- shortest path p for demand d,
- largest set of slices for demand d.

Algorithm adaptation and constriction

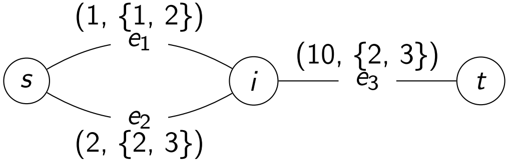
Adaptation was shaped by the need to:

- revisit nodes, but avoid loops,
- purge worse labels.

Constriction limits the path length. It's a typical Dijkstra constriction, during edge relaxation.

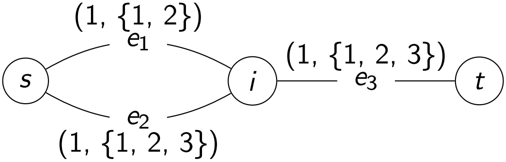
Revisit nodes, but avoid loops

Find a shortest path from node s to node t with 2 slices.
Edge label: (cost, {set of available slices})



Purge worse labels

Find a shortest path from node s to node t with 2 slices.
Edge label: (cost, {set of available slices})



What specifically we adapted

	Dijkstra	our algorithm
label	cost, preceding node	cost, preceding edge, maximal set of slices
node has	a single label	a set of labels
label comparison	cost only	cost and slices

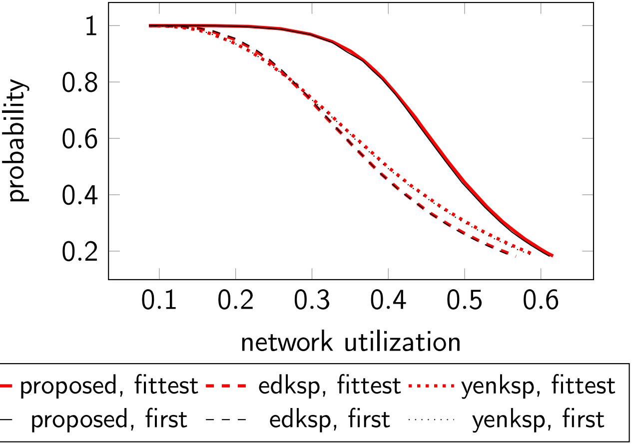
Label l_1 is better than or equal to label l_2, denoted by $l_1 \leq l_2$, if $\mathrm{cost}(l_1) \leq \mathrm{cost}(l_2)$ and $SSC(l_1) \supseteq SSC(l_2)$.

A node has a set of labels, where no label is better or equal to some other label.

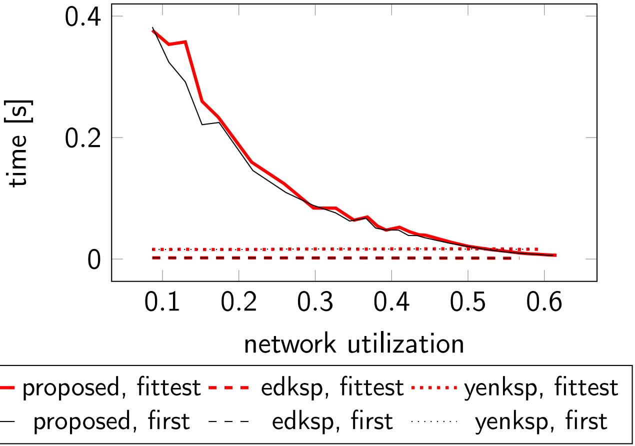
Evaluation setting

- We compared the performance of our algorithm to two heuristics:
  - routing along the edge-disjoint paths,
  - routing along the Yen $K = 10$ shortest paths.
- 50 Gabriel graphs model simulation topologies.
- Each graph has 100 nodes and 400 slices per edge.
- Spectrum selection: first or fittest.
- Connection arrivals: Poisson with mean $10 \leq \lambda \leq 1000/\text{day}$.
- Connection holding time: Poisson with mean 10 days.
- The requested number of slices: Poisson with mean 10.
- A simulation run lasts 100 simulated days.
- 8100 simulation runs, 1% relative standard error.

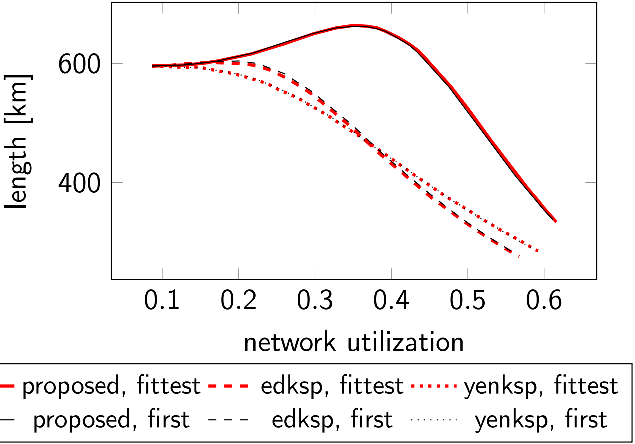
The probability of establishing a connection



The time of shortest path search



The length of an established connection



Conclusions

- The RSA and RWA problems tamed: constrained and solved.
- We reckon the constrained RSA and RWA problems tractable.
- The simulations show the algorithm is doing quite well.
- But we offer no proof.

The algorithm

In: $G = (V = \{v_i\}, E = \{e_i\}), W(e_i), S(e_i), m, d = (s, t, n)$
Out: $p = (e_1, \ldots, e_i, \ldots, e_l)$, $\Sigma = \{\sigma_i\}$

```
L_s = {(0, e_∅, Ω)}
push (0, e_∅) to Q
while Q is not empty do
    q = (c, e) = pop(Q)
    v = e.target
    if v == t then
        break the while loop
    end if
    SSSC = {l.SSC : l ∈ L_v and l.c == c and l.e == e}
    for all S ∈ SSSC do
        for all e' ∈ outgoing edges of v do
            S' = S ∩ S(e')
            c' = c + W(e')
            if c' ≤ m and S' can support d then
                v' = e'.target
                l' = (c', e', S')
                if ∄l ∈ L_v' : l ≤ l' then
                    L_v' = L_v' \ {l : l ∈ L_v' and l' ≤ l}
                    L_v' = L_v' ∪ {l'}
                    push (c', e') to Q
                end if
            end if
        end for
    end for
end while
return (p, Σ) = trace(L, s, t)
```

The number of slices of an established connection