Generic Dijkstra: correctness and tractability

Ireneusz Szcześniak
Department of Computer Science
Częstochowa University of Technology, Poland

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

NOMS 2023

Grant number 020/RID/2018/19 from the Polish Ministry of Education and Science

Introduction

The current networking setting:

- Network densification: networks keep growing
- Agility: quicker, nimbler, leaner, cheaper
- Reliability: fast network control and management
- A crucial component: routing algorithms
- Back to basics: routing of a single connection

Yet: routing a single connection should be fast.

Motivation: optical networks

- Objective: route a single connection
- Why not the Dijkstra algorithm?
- Telecom networks use discrete resources.
- We need a path:
    - of minimal cost,
    - meeting some constraints.
- Spectrum constraints of optical networks:
    - continuity,
    - contiguity.
- Is routing of a single connection tractable?
- Observation: heuristics in common use

Network resources

- Resources modeled by an integral interval.
- Optical networks: a frequency slot unit.
- Given $r_1 = [0, 1)$, $r_2 = [1, 3)$, $r_3 = [0, 2)$:
  - $r_3 \prec r_1$ because $r_3 \supset r_1$
  - $r_1 \parallel r_2$ because neither $\subset$, $\supset$, nor equality holds
  - $r_1 \parallel r_2$ and $r_2 \parallel r_3$ does not imply $r_1 \parallel r_3$ because $r_1 \subset r_3$
- We have to account for incomparability and intransitivity.

Generic Dijkstra

- Published in 2019 without a proof of correctness.
- Simulations: paths were found efficiently and correctly.
- That suggests the problem is tractable.
- Skeptics: maybe the simulations were wrong?
- Qualms:
  - Is the algorithm correct?
  - Is the problem tractable?

Contribution

- Generalization of the Bellman equation
- Spotted a shortcoming, offered a correction
- Proof of correctness
- Proof of tractability

Bellman equation - 1952

- The dynamic programming principle
- Used by Dijkstra: the edge relaxation (label updating)
- Assumption: labels (cost) are of linear ordering.
- Objective: find a single minimal label for vertex i from s.

$$
\begin{aligned}
f_s &= 0 \\
f_i &= \min_{e \in \mathrm{I}(v_i)} \{ f_{\mathrm{source}(e)} \oplus \mathrm{cost}(e) \} \qquad \text{if } i \neq s
\end{aligned}
$$

- Goal: pick the best label.
- Result: a shortest-path tree
- Observation: rather a stipulation than a solution.

Generic Bellman equation

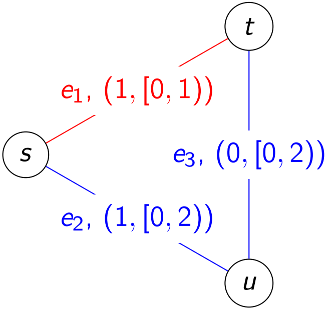
- We call it the generic dynamic programming principle.
- Used in generic Dijkstra
- Assumption: labels (cost and interval) are of partial ordering.
- Objective: find a set of incomparable labels for vertex i from s.

$$P_s = \{(0, \Omega)\}$$
$$P_i = \min\{ \bigcup_{e \in \mathrm{I}(v_i)} P_{\mathrm{source}(e)} \oplus e\} \qquad \text{if } i \neq s$$

- Goal: drop worse labels.
- Result: an efficient-path tree
- Observation: again, rather an assumption than a solution.

A shortcoming of generic Dijkstra

- Found while proving correctness – shows the power of math.
- Two paths from s to t: wrong (e_1), right ($e_2 + e_3$).
- We can get the wrong path even though this relation holds: $(1, [0, 2)) \prec (1, [0, 1))$
- Ordering labels by cost only is not enough.
- Ordering has to take intervals into account.



A correction to the shortcoming: the $<$ relation

We need two relations:

$\prec$ for comparing labels: which label is better?

$<$ for sorting labels: which label should be processed first?

We need $<$ for:

- sorting that requires a linear ordering (cannot use $\prec$),
- proving to show that labels are derived in expected order.

The properties of the $<$ relation for labels

Relation $<$ should be implied by $\prec$:

- $l_i \prec l_j \implies l_i < l_j$,
- better labels should be processed first,
- in line with the greedy strategy.

Relation $<$ (that was defined that way):

- establishes linear ordering (extends $\prec$),
- is transitive (labels derived in expected order).

Transitivity proven: $<$ is lexicographic.

Label terminology

A label can be:

- permanent - part of the efficient-path tree,
- tentative - an edge away from a permanent label,
- candidate - a tentative label for consideration.

Definition (Label efficiency)

Label l is efficient if there does not exist label l' such that $l' \prec l$.

Proposition

Relation $l \preceq l'$ *holds for* l' *derived from* l*.*

Intuition

Generic Dijkstra algorithm is correct for two reasons:

- the priority queue provides at the top an efficient label,
- derived labels cannot be better than a permanent label.

The proof

Theorem

The algorithm terminates with a complete set of efficient labels.

Proof.

We prove by induction. The induction step corresponds to an iteration of the main loop. The induction hypotheses are:

1. P has efficient labels derived from efficient labels,
2. T has incomparable labels derived from efficient labels.

Basis. The hypotheses hold for the initial label.
Inductive step:

1. A popped label is efficient.
2. Maintained by relaxation using generic Bellman equations.

Termination. The queue eventually get empty.

□

Tractability

The complexity of the number of all labels produced, where Ω is the number of units, V is the number of vertexes:

$$O(L) = O(|\Omega|^2|V|)$$

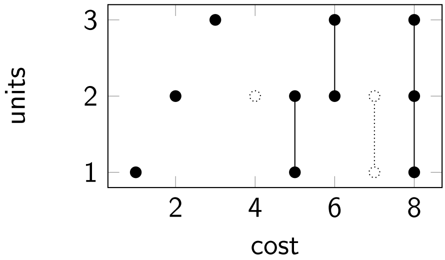


Figure: The maximal set of incomparable labels for three units.

Conclusion

- For tractable problems we want algorithms that are:
  - exact (correct),
  - efficient.
- Proven: feel safe to use generic Dijkstra!
- Personal reflexion of a humble programmer:
  - A simulator is a good start, but...
  - Who wants to read the simulator code?
  - Simulation results are hard to replicate.
  - In a math proof we offer a detailed reasoning.
- Further work:
  - Is the generic Dijkstra optimal? Can we do faster?
  - Optimal overall network performance with exact routing?

Algorithm 1 Generic Dijkstra

In: graph G, source vertex s

Out: an efficient-path tree

Here we concentrate on permanent labels.

$T_s = \{(0, \Omega)\}$ // The initial label.
while T is not empty **do**
  $l = \text{pop}(T)$
  $e = \text{edge}(l)$
  $v = \text{target}(e)$
  // Add l to the set of permanent labels for vertex v.
  $P_v = P_v \cup \{l\}$
  for all out edge e' of v in G **do**
    $\text{relax}(e', l)$
return P

Algorithm 2 relax

In: edge e', label l

Here we concentrate on tentative labels.

$c' = \text{cost}(l) \oplus \text{cost}(e')$
$v' = \text{target}(e')$
for all RI $I' \neq \emptyset$ in $\text{RI}(l) \cap \text{AU}(e')$ **do**
  $l' = (c', I')$
  // Can the candidate label l' become tentative?
  if $\nexists l_{v'} \in P_{v'} : l_{v'} \preceq l'$ **then**
    if $\nexists l_{v'} \in T_{v'} : l_{v'} \preceq l'$ **then**
      // Discard tentative labels $l_{v'}$ such that $l' \preceq l_{v'}$,
      // leave in $T_{v'}$ only labels incomparable with l'.
      $T_{v'} = T_{v'} - \{l_{v'} \in T_{v'} : l' \preceq l_{v'}\}$
      // Add l' to the set of tentative labels for vertex v'.
      $T_{v'} = T_{v'} \cup \{l'\}$

Table: Relations between RIs r_i and r_j.

	$\max(r_i) < \max(r_j)$		$\max(r_i) = \max(r_j)$		$\max(r_i) > \max(r_j)$	
$\min(r_i) < \min(r_j)$	$r_i \parallel r_j$	$r_i < r_j$	$r_i \supset r_j$	$r_i < r_j$	$r_i \supset r_j$	$r_i < r_j$
$\min(r_i) = \min(r_j)$	$r_i \subset r_j$	$r_i > r_j$	$r_i = r_j$		$r_i \supset r_j$	$r_i < r_j$
$\min(r_i) > \min(r_j)$	$r_i \subset r_j$	$r_i > r_j$	$r_i \subset r_j$	$r_i > r_j$	$r_i \parallel r_j$	$r_i > r_j$

Table: Relations between labels l_i and l_j.

	$\mathrm{RI}(l_i) \subset \mathrm{RI}(l_j)$		$\mathrm{RI}(l_i) = \mathrm{RI}(l_j)$		$\mathrm{RI}(l_i) \supset \mathrm{RI}(l_j)$		$\mathrm{RI}(l_i) \parallel \mathrm{RI}(l_j)$	
$\mathrm{cost}(l_i) < \mathrm{cost}(l_j)$	$l_i \parallel l_j$	$l_i < l_j$	$l_i \prec l_j$	$l_i < l_j$	$l_i \prec l_j$	$l_i < l_j$	$l_i \parallel l_j$	$l_i < l_j$
$\mathrm{cost}(l_i) = \mathrm{cost}(l_j)$	$l_i \succ l_j$	$l_i > l_j$	$l_i = l_j$		$l_i \prec l_j$	$l_i < l_j$	$l_i \parallel l_j$	$l_i < l_j$ if $\mathrm{RI}(l_i) < \mathrm{RI}(l_j)$ $l_i > l_j$ if $\mathrm{RI}(l_i) > \mathrm{RI}(l_j)$
$\mathrm{cost}(l_i) > \mathrm{cost}(l_j)$	$l_i \succ l_j$	$l_i > l_j$	$l_i \succ l_j$	$l_i > l_j$	$l_i \parallel l_j$	$l_i > l_j$	$l_i \parallel l_j$	$l_i > l_j$