On The Discovery of The Near-Linear Time Shortest Path Algorithm on Graphs with Negative Weights

Hrvoje Abraham
hrvoje.abraham@avl.com

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

June 16, 2023

Agenda

- Introduction
  - The Team, The Paper, The Problem
- Ingredients
  - The Ideas, The Techniques, The Theorems
  - Dijkstra, Bellman–Ford, SCC, LDD, Price Function...
- Discovery
  - Combining all the elements into a solution
  - Presenting the recipe, not the final dish/implementation

The Team

Aaron Bernstein

Department of Computer Science
Rutgers University
New Brunswick, NJ, USA

approximation theory, computational complexity, directed graphs, data structures, graph theory

Danupon Nanongkai

MPI-INF, University of Copenhagen and KTH

computational complexity, approximation theory, graph theory, directed graphs, randomised algorithms

Christian Wulff-Nilsen

Department of Computer Science
University of Copenhagen
Copenhagen, Denmark

computational complexity, approximation theory, directed graphs, data structures

The Media

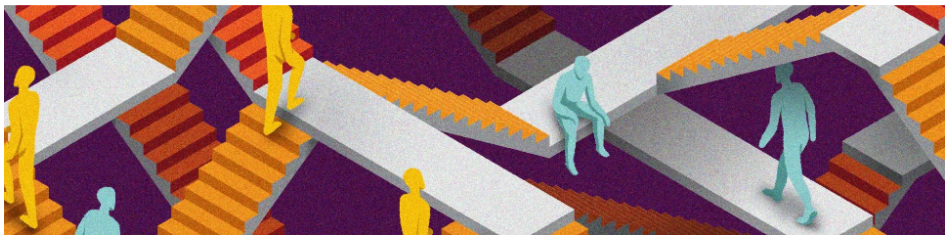
Quanta magazine — Physics Mathematics Biology Computer Science Topics Archive

GRAPH THEORY

Finally, a Fast Algorithm for Shortest Paths on Negative Graphs

6

Researchers can now find the shortest route through a network nearly as fast as theoretically possible, even when some steps can cancel out others.



"I just couldn't believe such a simple algorithm exists," said Maximilian Probst Gutenberg, a computer scientist at the Swiss Federal Institute of Technology Zurich. "All of it has been there for 40 years. It just took someone to be really clever and determined to make it all work."

ScienceDaily

Your source for the latest research news

Science News *from research organizations*

Researcher lauded for superb solution of algorithmic riddle from the 1950s

Date: November 14, 2022

Source: University of Copenhagen - Faculty of Science

Summary: Solving the riddle can reduce electric car battery consumption and make life tougher for currency speculators in the future. The discovery has just won the award for best research article and was honored at the field's most prestigious conference in the United States.

Share:

RELATED TOPICS

Matter & Energy
- Engineering
- Physics
- Energy Technology
- Telecommunications

Computers & Math
- Computer Programming

FULL STORY

For more than half a century, researchers around the world have been struggling with an algorithmic problem known as "the single source shortest path problem." The problem is essentially about how to devise a mathematical recipe that best finds the shortest route between a node and all other nodes in a network, where there may be connections with negative weights.

The Paper

https://arxiv.org/abs/2203.03456

Cornell University

arXiv > cs > arXiv:2203.03456

Search... Help | Advanc

Computer Science > Data Structures and Algorithms

[Submitted on 7 Mar 2022 (v1), last revised 30 Oct 2022 (this version, v4)]

Negative-Weight Single-Source Shortest Paths in Near-linear Time

Aaron Bernstein, Danupon Nanongkai, Christian Wulff-Nilsen

We present a randomized algorithm that computes single-source shortest paths (SSSP) in $O(m \log^8(n) \log W)$ time when edge weights are integral and can be negative. This essentially resolves the classic negative-weight SSSP problem. The previous bounds are $\tilde{O}((m+n^{1.5}) \log W)$ [BLNPSSSW FOCS'20] and $m^{4/3+o(1)} \log W$ [AMV FOCS'20]. Near-linear time algorithms were known previously only for the special case of planar directed graphs [Fakcharoenphol and Rao FOCS'01].
In contrast to all recent developments that rely on sophisticated continuous optimization methods and dynamic algorithms, our algorithm is simple: it requires only a simple graph decomposition and elementary combinatorial tools. In fact, ours is the first combinatorial algorithm for negative-weight SSSP to break through the classic $\tilde{O}(m\sqrt{n} \log W)$ bound from over three decades ago [Gabow and Tarjan SICOMP'89].

Comments: Simplified algorithm for Low-Diameter Decomposition and minor corrections
Subjects: **Data Structures and Algorithms (cs.DS)**
Cite as: arXiv:2203.03456 **[cs.DS]**
(or arXiv:2203.03456v4 **[cs.DS]** for this version)
https://doi.org/10.48550/arXiv.2203.03456

Submission history

From: Danupon Nanongkai [view email]
[v1] Mon, 7 Mar 2022 15:15:09 UTC (51 KB)
[v2] Tue, 5 Apr 2022 08:34:33 UTC (48 KB)
[v3] Sun, 8 May 2022 12:50:44 UTC (44 KB)
[v4] Sun, 30 Oct 2022 05:54:26 UTC (57 KB)

https://ieeexplore.ieee.org/document/9996870

Conferences > 2022 IEEE 63rd Annual Symposi...

Negative-Weight Single-Source Shortest Paths in Near-linear Time

Abstract:
We present a randomized algorithm that computes single-source shortest paths (SSSP) in $O\left(m \log^8(n) \log W\right)$ time when edge weights are integral and can be negative. [1] This essentially resolves the classic negative-weight SSSP problem. The previous bounds are $\tilde{O}\left((m+n^{1.5}) \log W\right)$ [BLNPSSSW FOCS'20] and $m^{4/3+o(1)} \log W$ [AMV FOCS'20]. Near-linear time algorithms were known previously only for the special case of planar directed graphs [Fakcharoenphol and Rao FOCS'01]. In contrast to all recent developments that rely on sophisticated continuous optimization methods and dynamic algorithms, our algorithm is simple: it requires only a simple graph decomposition and elementary combinatorial tools. In fact, ours is the first combinatorial algorithm for negative-weight SSSP to break through the classic $O(m\sqrt{n} \log W)$ bound from over three decades ago [Gabow and Tarjan SICOMP'89].

Published in: 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)

Date of Conference: 31 October 2022 - 03 November 2022
INSPEC Accession Number: 22448549
Date Added to IEEE *Xplore*: 28 December 2022
DOI: 10.1109/FOCS54457.2022.00063
ISBN Information:
Publisher: IEEE
ISSN Information:
Conference Location: Denver, CO, USA

FOCS 2022: 63rd IEEE Symposium on Foundations of Computer Science

October 31—November 3, 2022, Hilton Denver City Center, Denver, Colorado, USA

Home

CONFERENCE DETAILS
Registration
Program
Call for Papers
Accepted Papers
Important Dates
PC / Organizers

Paper Awards

Best Paper Award co-winners

- **Negative-Weight Single-Source Shortest Paths in Near-Linear Time**
  Aaron Bernstein, Danupon Nanongkai and Christian Wulff-Nilsen
- **Maximum Flow and Minimum-Cost Flow in Almost-Linear Time**
  Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg and Sushant Sachdeva

The Paper

About 30 pages of serious business without illustrations:

Definition 2.4 ($\eta_G(v)$, $P_G(v)$)**.** For any graph $G = (V, E, w)$, let G_s and s be as in Definition 2.3. Define

$$\eta_G(v) := \begin{cases} \infty & \text{if } \mathrm{dist}_{G_s}(s, v) = -\infty \\ \min\{|E^{neg}(G) \bigcap P| : P \text{ is a shortest } sv\text{-path in } G_s\}; & \text{otherwise.} \end{cases}$$

Let $\eta(G) = \max_{v \in V} \eta_G(v)$. When $\mathrm{dist}_G(s, v) \neq -\infty$, let $P_G(v)$ be a shortest sv-path on G_s such that

$$|E^{neg}(G) \bigcap P_G(v)| = \eta_G(v). \tag{1}$$

When the context is clear, we drop the subscripts.

2.1 Price Functions and Equivalenc

Our algorithm heavily relies on price function

Definition 2.5 (Price Function)**.** Consider a g
$\mathbb{Z}$ is the set of integers. Then, we define w_ϕ to
and we define $G_\phi = (V, E, w_\phi)$. We will refer t

Definition 2.6 (Graph Equivalence)**.** We sa
are *equivalent* if **(1)** any shortest path in G i
contains a negative-weight cycle if and only if

Lemma 2.7 ([Joh77])**.** *Consider any graph G
V we have $\mathrm{dist}_{G_\phi}(u, v) = \mathrm{dist}_G(u, v) + \phi(u) -$
a result, G and G_ϕ are equivalent. Finally, if
for some positive c, then G and G' are equiva*

The overall goal of our algorithm will be to
in G_ϕ are non-negative (assuming no negative
lemma below, originally used by Johnson, wil

Lemma 2.8 ([Joh77])**.** *Let $G = (V, E)$ be a
be the dummy source in G_s. Let $\phi(v) = \mathrm{dist}_G$
non-negative. (The lemma follows trivially fr*

Phase 3's runtime. Now we are ready to prove Theorem 4.2, i.e. the runtime bound of $\mathrm{ElimNeg}(G^B_{\phi_2})$ in Phase 3 when G contains no negative-weight cycle. Recall (Lemma 3.3 and definition 2.4) that the runtime of $\mathrm{ElimNeg}(G^B_{\phi_2})$ is

$$O\left(\left(m + \sum_{v \in V} \eta_{G^B_{\phi_2}}(v)\right) \log m\right)$$

Fix any $v \in V$. Note that, regardless of the value of ϕ_2, $P_{G^B}(v)$ is a shortest sv-path in $(G^B_{\phi_2})_s$ (because $(G^B_{\phi_2})_s$ and G^B_s are equivalent and $P_{G^B}(v)$ is a shortest sv-path in G^B_s by definition). Thus,

$$\begin{aligned} \eta_{G^B_{\phi_2}}(v) &= \min\{|P \cap E^{neg}(G^B_{\phi_2})| : P \text{ is a shortest } sv\text{-path in } (G^B_{\phi_2})_s\} && \text{(Definition 2.4)} \\ &\leq |P_{G^B}(v) \cap E^{neg}(G^B_{\phi_2})| && \text{by the above.} \end{aligned}$$

By Lemma 4.7, all negative-weight edges in $G^B_{\phi_2}$ are in E^{rem}, i.e. $E^{neg}(G^B_{\phi_2}) \subseteq E^{rem}$; so,

$$\eta_{G^B_{\phi_2}}(v) \leq |P_{G^B}(v) \cap E^{rem}|.$$

By Lemma 4.4 and the fact that $\eta(G^B) \leq \Delta$ (input requirement Item 1b in Theorem 3.5),[14]

$$E\left[\eta_{G^B_{\phi_2}}(v)\right] \leq E\left[|P_{G^B}(v) \cap E^{rem}|\right] = O(\log^2 m). \tag{5}$$

Thus, the expected runtime of $\mathrm{ElimNeg}(G^B_{\phi_2})$ is

$$O\left(\left(m + E\left[\sum_{v \in V} \eta_{G^B_{\phi_2}}(v)\right]\right) \log m\right) = O\left(m \log^3 m\right).$$

$$E[P_{G^B}(v) \cap E^{rem}] = O\left(\frac{w^B_{\geq 0}(P_{G^B}(v)) \cdot (\log n)^2}{B\Delta/2} + |(P_{G^B}(v))| \cdot n^{-10}\right)$$
$$\overset{(2)}{=} O\left(\frac{2\eta_{G^B}(v) \cdot (\log n)^2}{\Delta} + n^{-9}\right)$$

which is $O(\log^2 n)$ when $\eta(G^B) \leq \Delta$. □

Phase 1: Make edges inside the SCCs $G^B[V_i]$ non-negative. We argue that $\mathrm{ScaleDown}(H, \Delta/2, B)$ is called with an input that satisfies its input requirements (Theorem 3.5). The most important requirement is $\eta(H^B) \leq \Delta/2$ (Item 1b) which we prove below (other requirements are trivially satisfied). Recall that we set $d := \Delta/2$ in Line 3.

then $\eta(H^B) \leq d = \Delta/2$.

$_B(v) \setminus s$; i.e. P is obtained by removing s from a
gative weights in H^B_s. Let u be the first vertex in

,

$\eta_{H^B}(v)$, and

o u and v in H^B_s have weight zero. Then,

$_{H^B}(P) - |E^{neg}(H^B) \cap P| \cdot B$
$_{H^B}(v) \cdot B \overset{(c)}{\leq} -\eta_{H^B}(v) \cdot B.$ (3)

thus, by Lemma 4.3:

$u) \leq dB.$ (4)

$\mathrm{dist}_G(u, v) + \mathrm{dist}_G(v, u) \geq 0$ and thus $\eta_{H^B}(v) \leq$
his holds for every $v \in V$, Lemma 4.5 follows. □

uaranteed to output ϕ_1 as follows.

The Paper

Abstract

We present a randomized algorithm that computes single-source shortest paths (SSSP) in $O(m \log^8(n) \log W)$ time when edge weights are integral and can be negative.[1] This essentially resolves the classic negative-weight SSSP problem. The previous bounds are $\tilde{O}((m+n^{1.5}) \log W)$ [BLNPSSSW FOCS'20] and $m^{4/3+o(1)} \log W$ [AMV FOCS'20]. Near-linear time algorithms were known previously only for the special case of planar directed graphs [Fakcharoenphol and Rao FOCS'01].

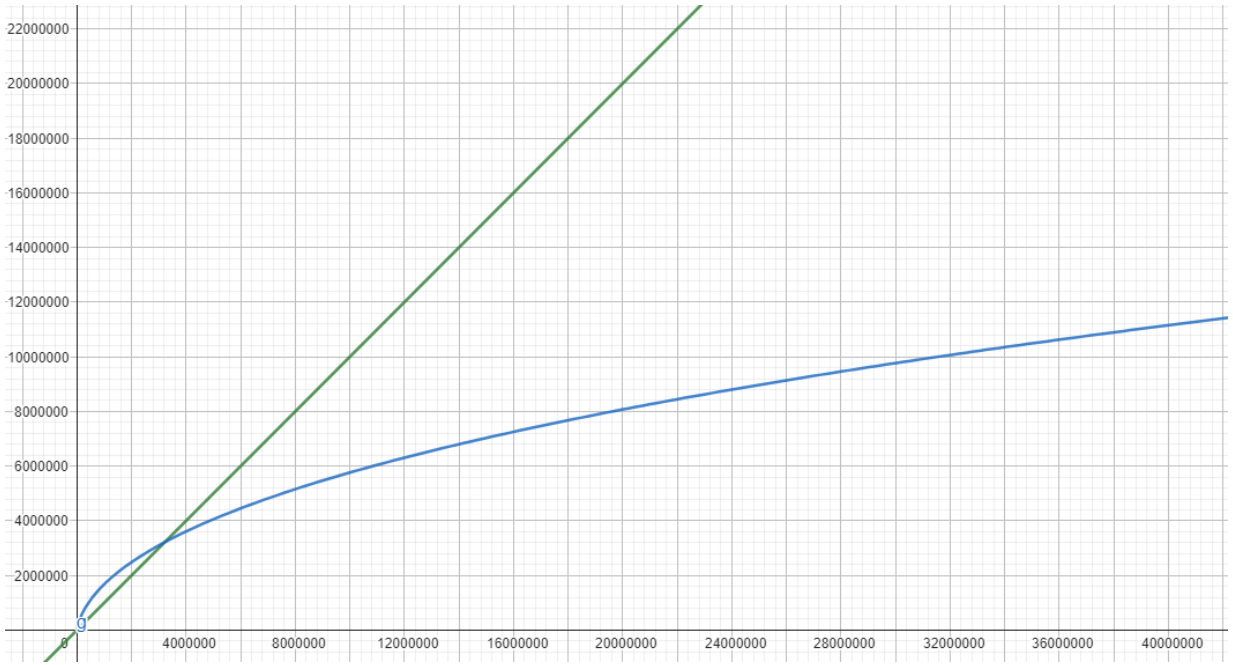
In contrast to all recent developments that rely on sophisticated continuous optimization methods and dynamic algorithms, our algorithm is simple: it requires only a simple graph decomposition and elementary combinatorial tools. In fact, ours is the first combinatorial algorithm for negative-weight SSSP to break through the classic $\tilde{O}(m\sqrt{n} \log W)$ bound from over three decades ago [Gabow and Tarjan SICOMP'89].

Near-Linear

Near-linear if complexity is below $\mathcal{O}(n^{1+\epsilon})$ for all $\epsilon > 0$.

Logarithmic function is slow, so this includes $\mathcal{O}(n \log^k n)$ for all $k > 0$.

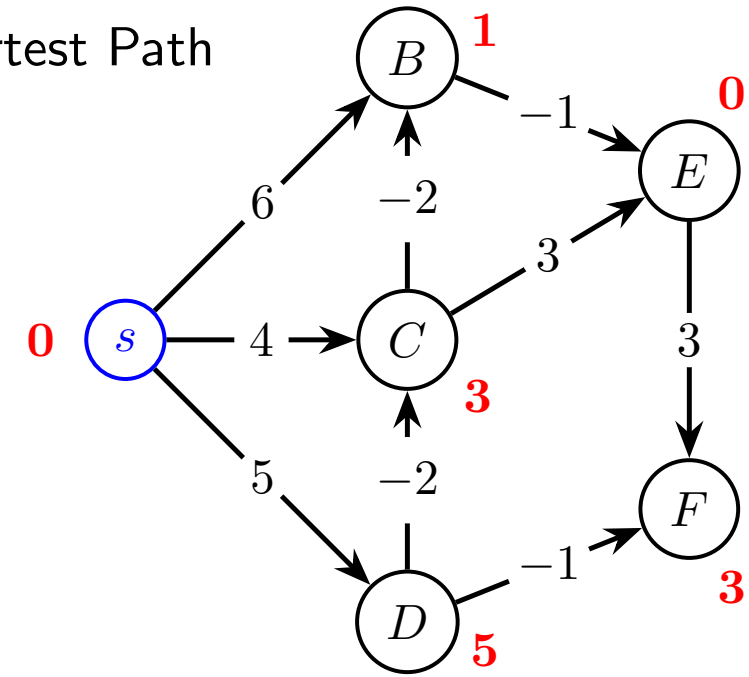
x vs $\log^8 x$



$\mathcal{O}(n \log^k n)$ is sometimes written as $\widetilde{\mathcal{O}}(n)$ with log factors being ignored.

The Problem — Types

- NAPSP, NSSSP, NSSSS — Shortest Paths on Non-Negative Weights
- MSSD — Multiple Sources Single–destination Shortest Path
- SSSP — Single Source Shortest Path (to All Sinks)
- SSSS — Single Source Single Sink Shortest Path
- APSP — All Pairs Shortest Path



The Problem — Specific Considerations

- We don't solve *a simple path* problem, e.g. paths with no repeated vertices. It would mean for algorithm to "actively" avoid repetitions. This problem is NP-complete, as demonstrated via reduction to *the longest path* problem by inverting weights' signs for all-negative graph.
- As repetition is allowed, negative cycle would result in an "infinitely short path" by cycling around it. Algorithm detects it and stops.
- Directed case only. Undirected would result in every edge being used back&forth to immediately create an infinite cycle.
- Integer weights only, due algorithmic needs of the procedure. Enough to be considered an "SSSP problem solution". $log(max\_neg\_Weight)$

The Problem — History

- Classic
  - n^4 — Shimbel 1955 via matrix multiplication
  - $n^2 m$ — Ford 1956
  - $n\,m$ — Bellman 1958, Moore 1959
- Combinatorial Improvements
  - $m\,n^{3/4}\log n$ — Gabow 1983
  - $m\,n^{1/2}\log n$ — Gabow 1989
  - $m\,n^{1/2}$ — Goldberg 1993
- Continuous Optimization and Dynamic Data Structures
  - $m^{10/7}$ — Cohen, Mandry, Sankowski, Vladu 2017
  - $m^{4/3}$ — Axiotis, Madry, Vladu 2020
  - $m + n^{3/2}$ — Brand, Lee, Nanogkai, Peng, Saranurak, Sidford, Song, Wang 2020

New result $\mathcal{O}(m \log^8 n)$.

The Ingredients

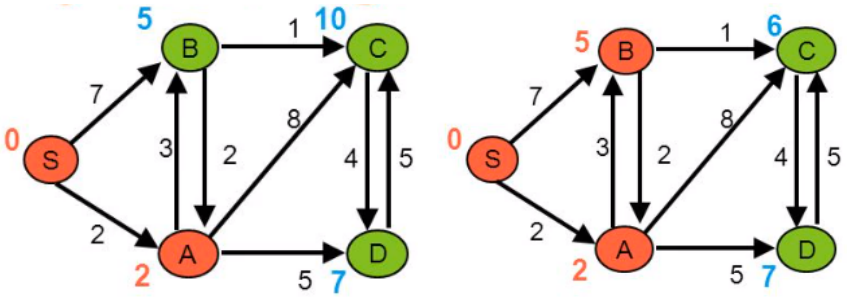
- Dijkstra — 1959 (Dijkstra, Johnson 1972, Fredman&Tarjan 1989)
- Bellman–Ford — 1955 (Shimbel, Ford, Bellman, Moore)
- Strongly Connected Components — 1971 (Munro)
- Low Diameter Decomposition — 1989 (Awerbuch)
- Price Function — 1977 (Johnson)

> "I just couldn't believe such a simple algorithm exists," said Maximilian Probst Gutenberg, a computer scientist at the Swiss Federal Institute of Technology Zurich. "All of it has been there for 40 years. It just took someone to be really clever and determined to make it all work."

Comment based on the fact the techniques are "classic".

Dijkstra $\mathcal{O}(m + n \log n)$

```
1  function Dijkstra(Graph, source):
2
3      for each vertex v in Graph.Vertices:
4          dist[v] ← INFINITY
5          prev[v] ← UNDEFINED
6          add v to Q
7      dist[source] ← 0
8
9      while Q is not empty:
10         u ← vertex in Q with min dist[u]
11         remove u from Q
12
13         for each neighbor v of u still in Q:
14             alt ← dist[u] + Graph.Edges(u, v)
15             if alt < dist[v]:
16                 dist[v] ← alt
17                 prev[v] ← u
18
19     return dist[], prev[]
```



It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors.

With Fibonacci heap $\mathcal{O}(m + n \log n)$
With binary heap $\mathcal{O}((m+n) \log n)$

Dijkstra & heaps

Michael Fredman, Robert Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM, 34(3):596–615, 1987

Procedure	Binary heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

We can in fact achieve a running time of $O(V \lg V + E)$ by implementing the min-priority queue with a Fibonacci heap (see Chapter 19). The amortized cost of each of the $|V|$ EXTRACT-MIN operations is $O(\lg V)$, and each DECREASE-KEY call, of which there are at most $|E|$, takes only $O(1)$ amortized time. Historically, the development of Fibonacci heaps was motivated by the observation that Dijkstra's algorithm typically makes many more DECREASE-KEY calls than EXTRACT-MIN calls, so that any method of reducing the amortized time of each DECREASE-KEY operation to $o(\lg V)$ without increasing the amortized time of EXTRACT-MIN would yield an asymptotically faster implementation than with binary heaps.

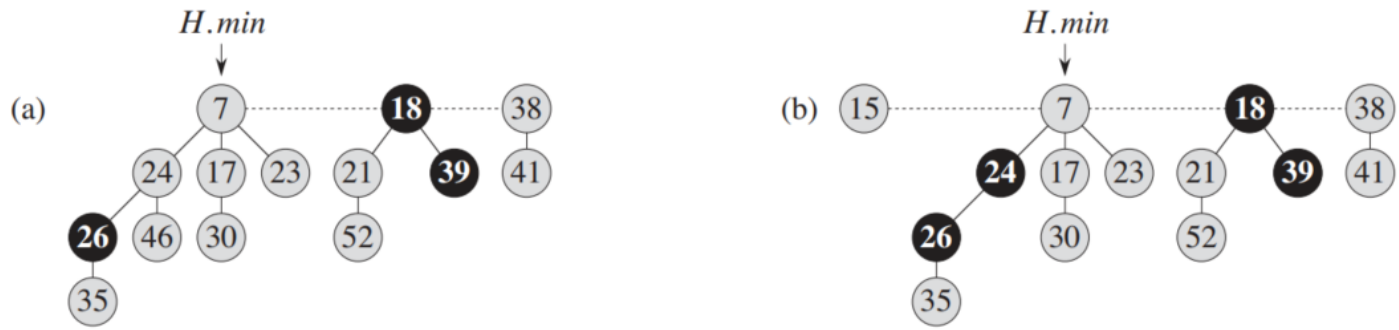
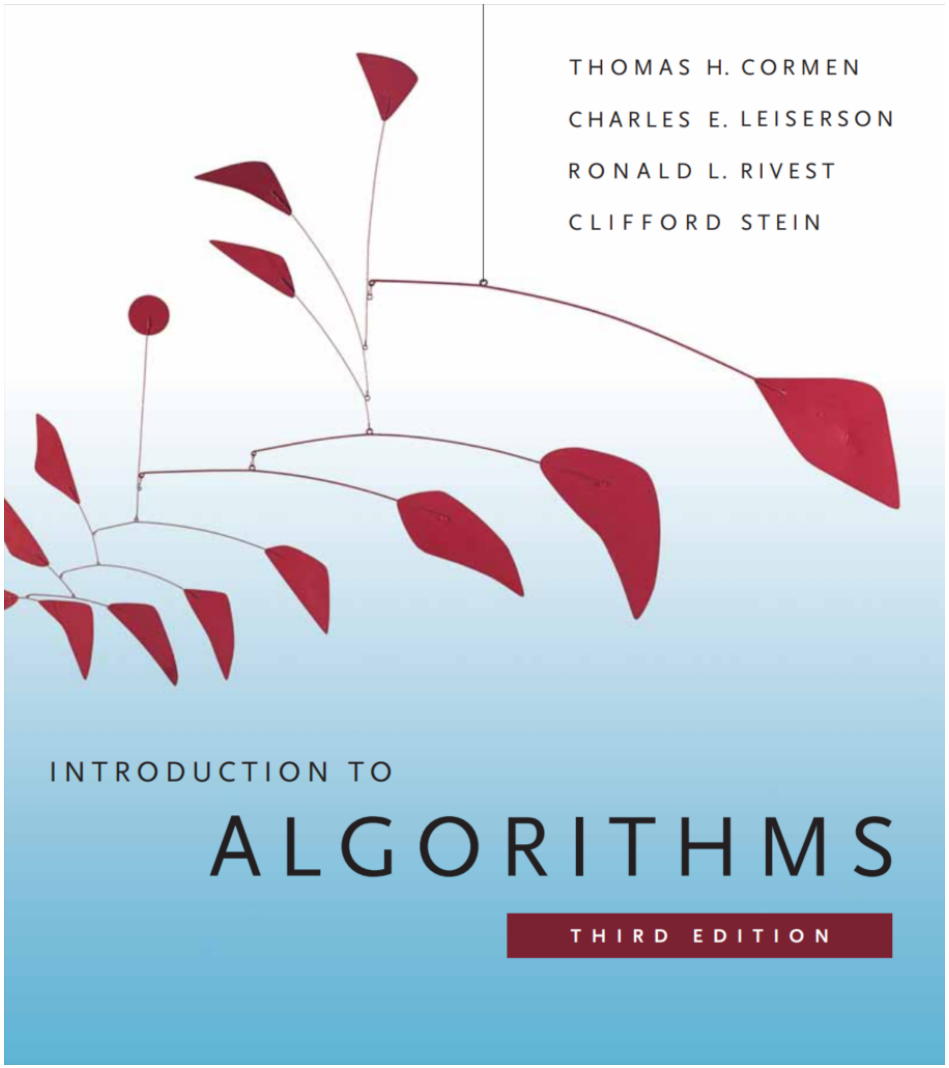
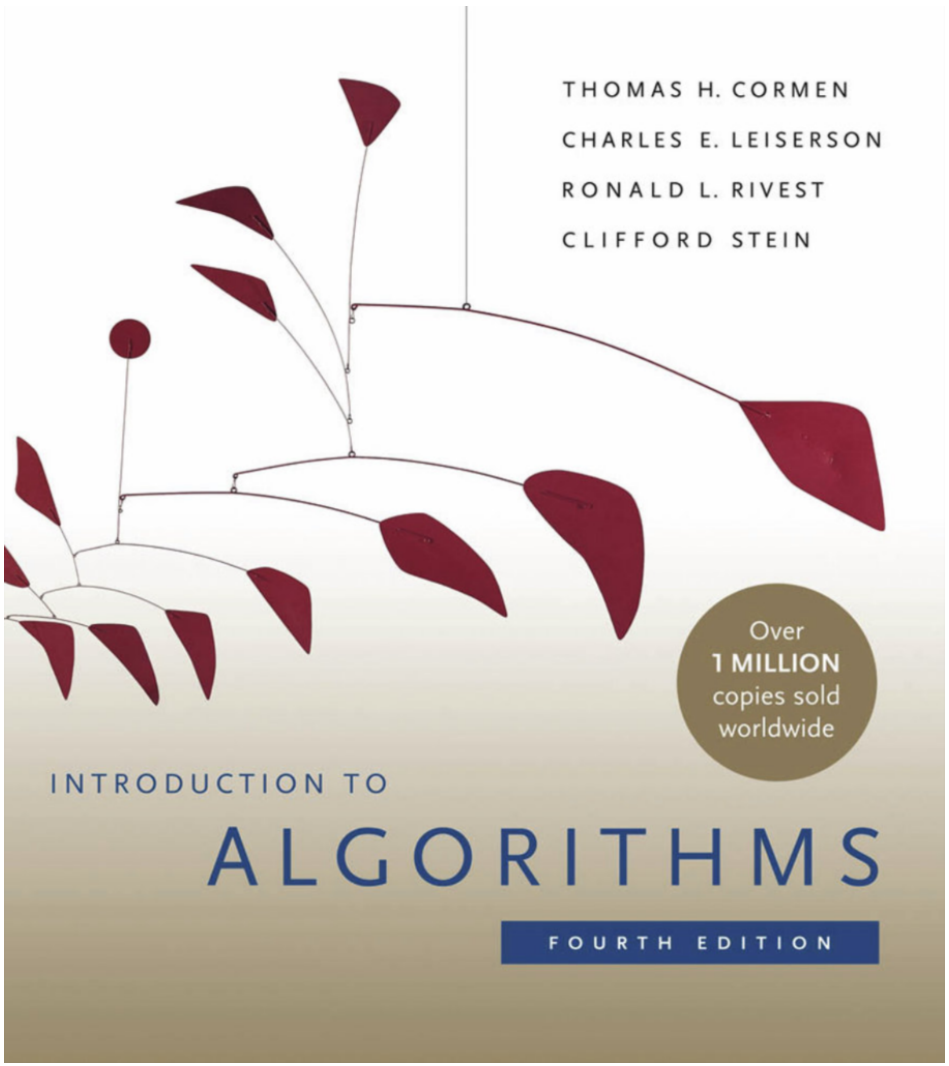


Figure 19.5 Two calls of FIB-HEAP-DECREASE-KEY. **(a)** The initial Fibonacci heap. **(b)** The node with key 46 has its key decreased to 15. The node becomes a root, and its parent (with key 24), which had previously been unmarked, becomes marked. **(c)–(e)** The node with key 35 has its key

Dijkstra & heaps



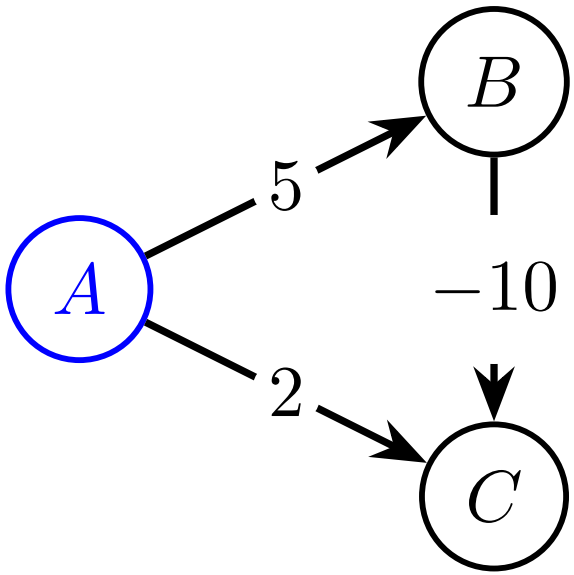
2009, 1292 pgs, Fibonacci heap ch.



2022, 1312 pgs, no Fibonacci heaps

Dijkstra & negative edges

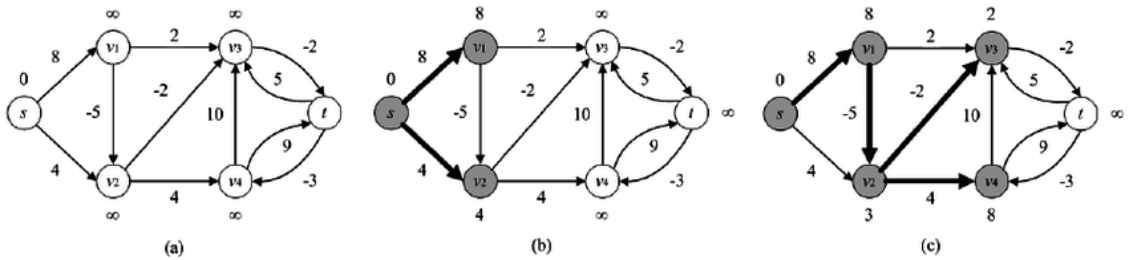
Dijkstra doesn't work for negative weights.



$$A = 0 \; B = \infty \; C = \infty \; \mathbf{Q} = [A, B, C]$$
$$A = 0 \; B = 5 \; C = 2 \; \mathbf{Q} = [B, C]$$
$$A = 0 \; B = 5 \; C = 2 \; \mathbf{Q} = [B]$$
$$A = 0 \; B = 5 \; C = 2 \; \mathbf{Q} = []$$

$$A \longrightarrow B \longrightarrow C = -5$$

Bellman–Ford $\mathcal{O}(m\,n)$




```
function BellmanFord(list vertices, list edges, vertex source) is
    distance := list of size n
    predecessor := list of size n

    // Step 1: initialize graph
    for each vertex v in vertices do
        distance[v] := inf     // Initialize vertices to infinity
        predecessor[v] := null // And having a null predecessor

    distance[source] := 0      // The distance from the source to itself

    // Step 2: relax edges repeatedly
    repeat |V|-1 times:
         for each edge (u, v) with weight w in edges do
             if distance[u] + w < distance[v] then
                 distance[v] := distance[u] + w
                 predecessor[v] := u

    // Step 3: check for negative-weight cycles
    for each edge (u, v) with weight w in edges do
        if distance[u] + w < distance[v] then
            // Step 4: find a negative-weight cycle
            negativeloop := [v, u]
            repeat |V|-1 times:
                u := negativeloop[0]
                for each edge (u, v) with weight w in edges do
                    if distance[u] + w < distance[v] then
                        negativeloop := concatenate([v], negativeloop)
            find a cycle in negativeloop, let it be ncycle
            // use any cycle detection algorithm here
            error "Graph contains a negative-weight cycle", ncycle

    return distance, predecessor
```

It initializes the distance to the source to 0 and all other nodes to infinity.

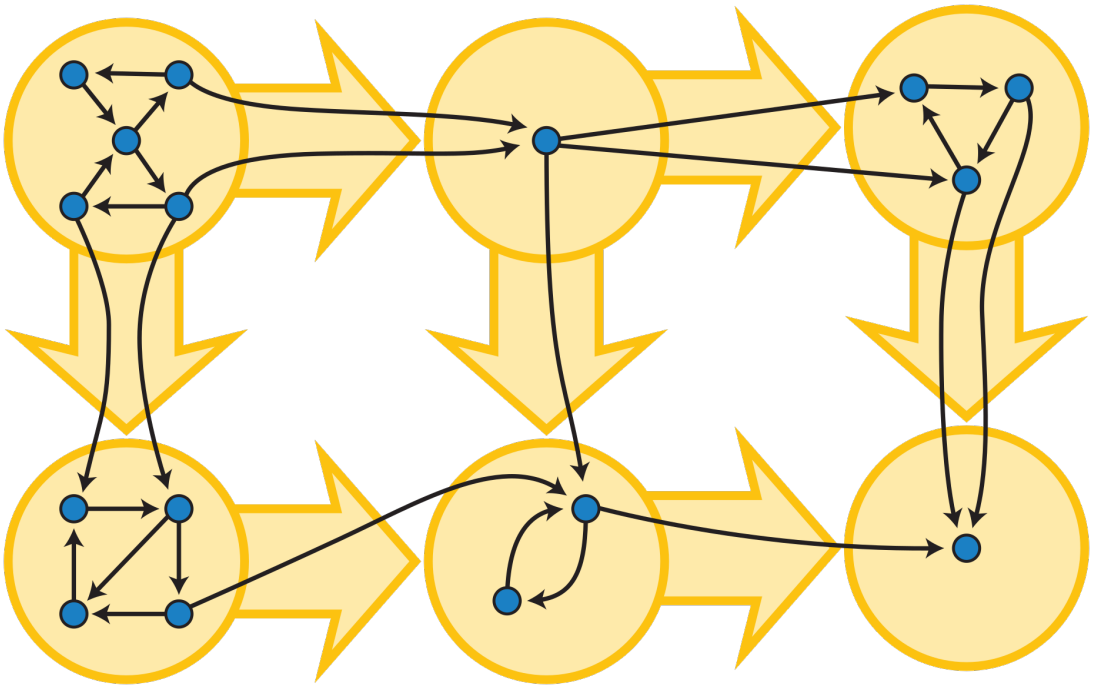
Then for all edges, if distance to the destination can be shortened by taking the edge, the distance is updated to the new lower value.

The core of the algorithm is a loop that scans $n-1$ times all m edges.

$\mathcal{O}(m\,\gamma)$ if shortest path has γ edges

Strongly Connected Components (SCC)

If we can reach every vertex of a component from every other vertex in that component then it is called a Strongly Connected Component (SCC).

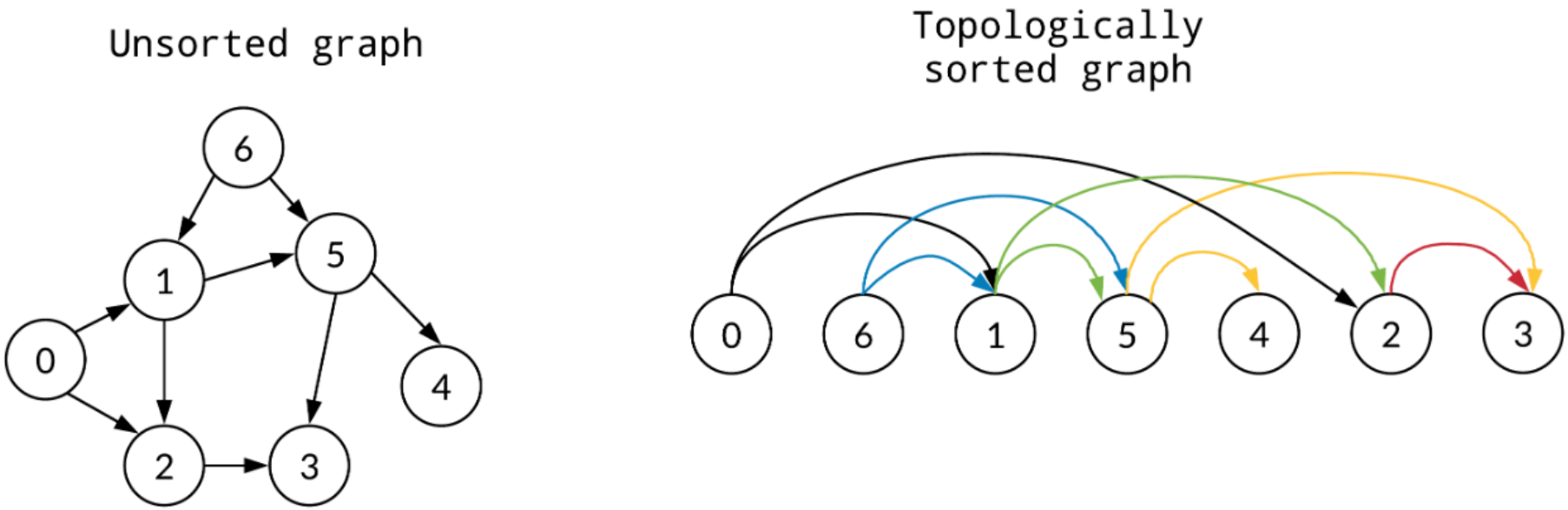


All SCCs found in $\mathcal{O}(m+n)$ time via Tarjan's or Kosaraju's algorithm.

With SCCs contracted resulting graph is directed acyclic graph (DAG).

Strongly Connected Components (SCC)

A topological sort is an ordering of nodes for a directed acyclic graph (DAG) such that for every directed edge from vertex u to vertex v, u comes before v in the ordering.

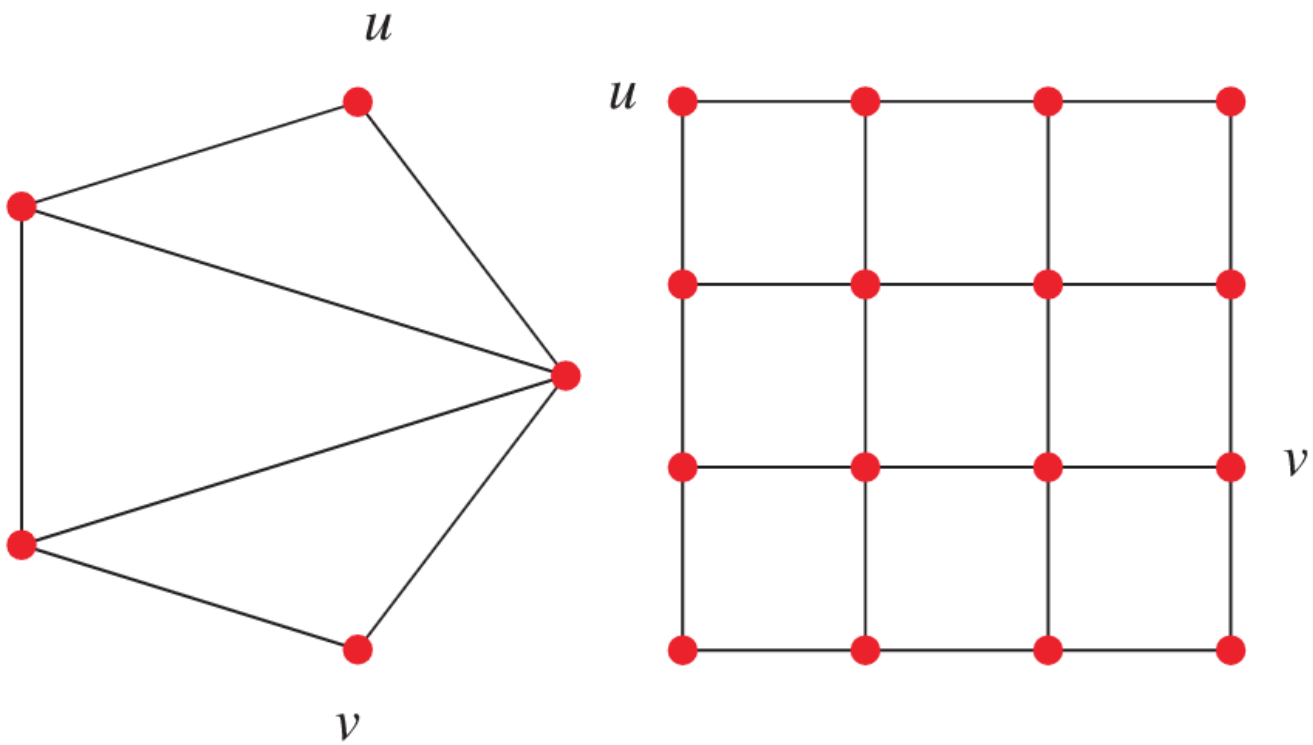


So SCCs can be topologically sorted — we will benefit by ordering them so that all their connecting edges point into the same direction.

Low Diameter Decomposition (LDD)

The distance between two vertices in a graph is the length of a shortest path connecting them.

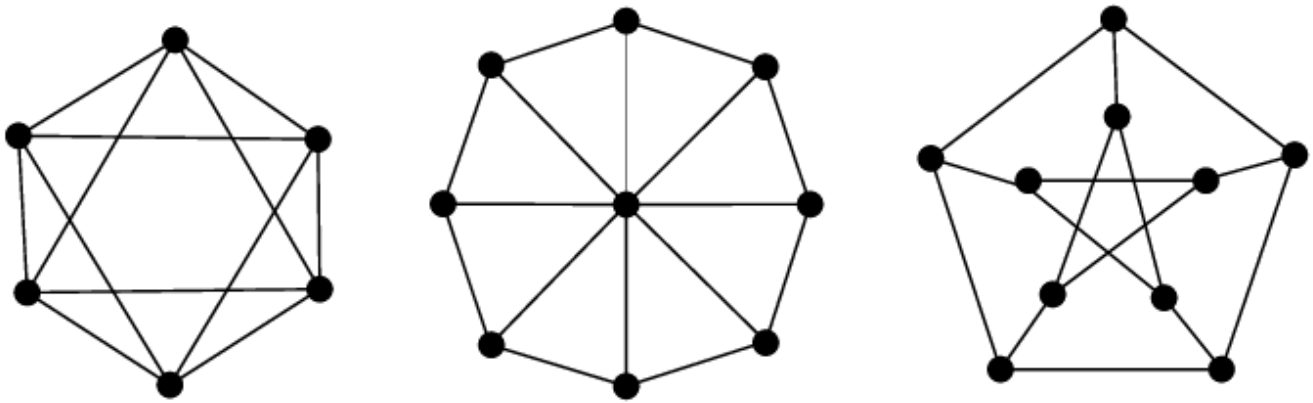
$d(u, v) = 2$ $d(u, v) = 5$



Low Diameter Decomposition (LDD)

The *diameter* D is the largest distance d between any two nodes.

$$D = 2$$

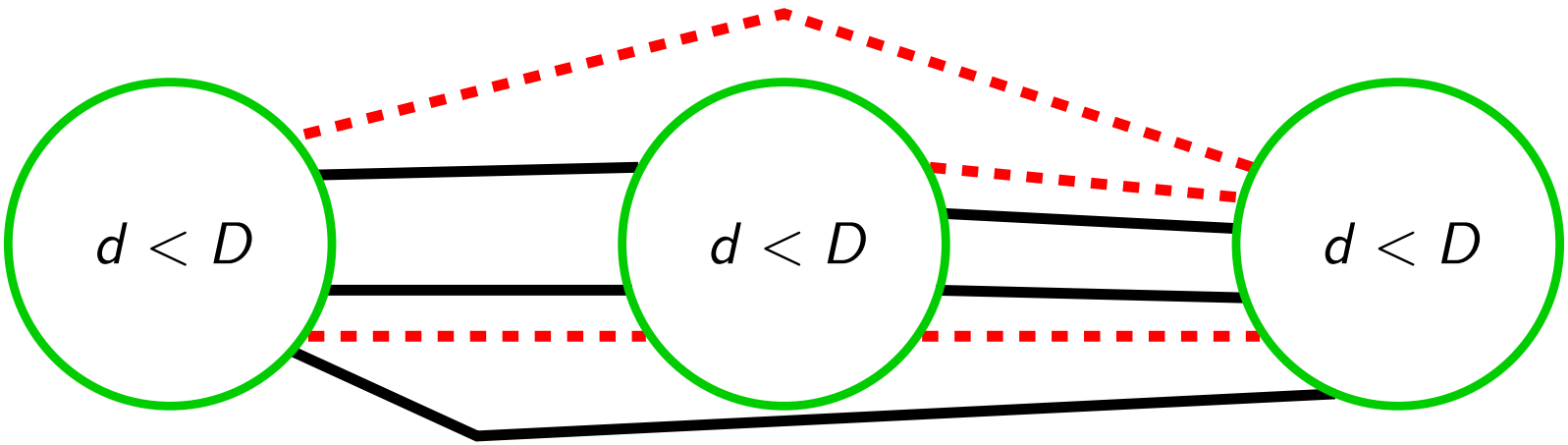


Low Diameter Decomposition (LDD)

Every n-vertex m-edge graph has a decomposition into a small number of blocks having (small) diameter D with a *low edge–cutting probability*

$$Pr[e] = \mathcal{O}\left(\frac{w(e)\, log^2 n}{D}\right).$$

There are randomized algorithms producing it in time $\tilde{\mathcal{O}}(m)$.

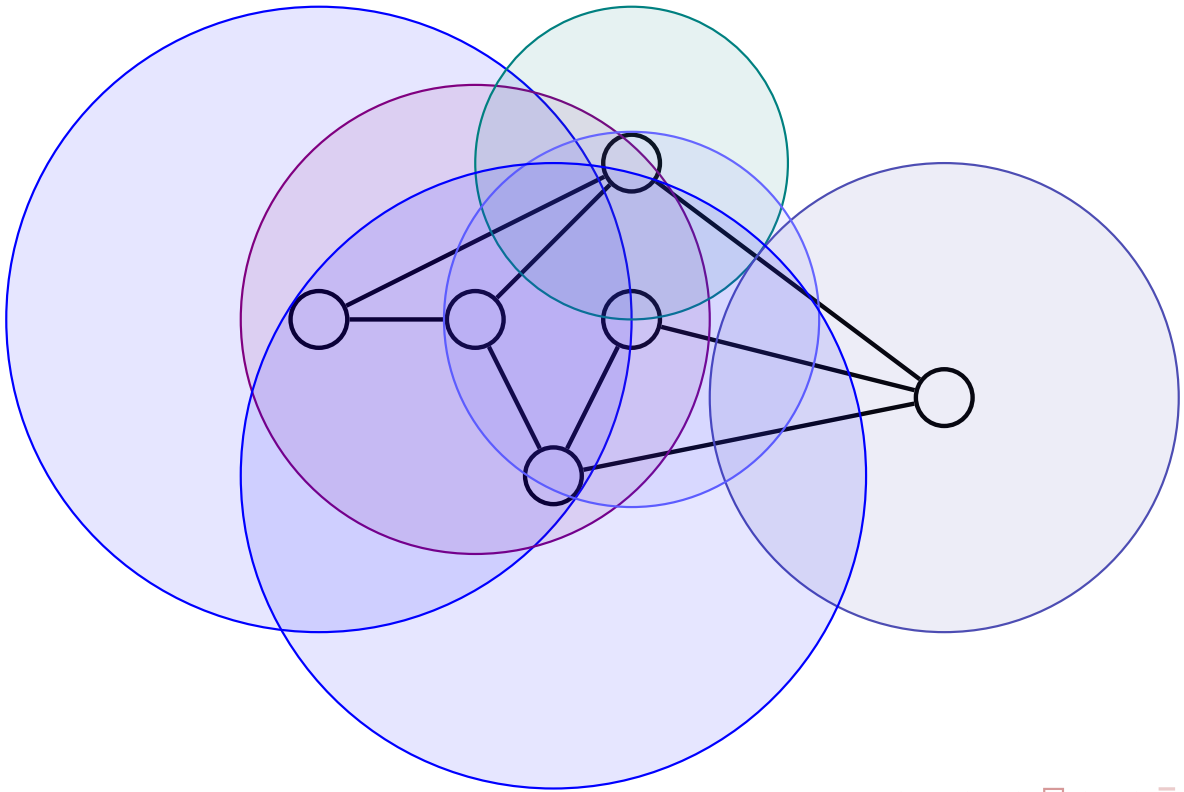


Red are a few deleted edges, black are leftover conns between blocks.

Low Diameter Decomposition (LDD)

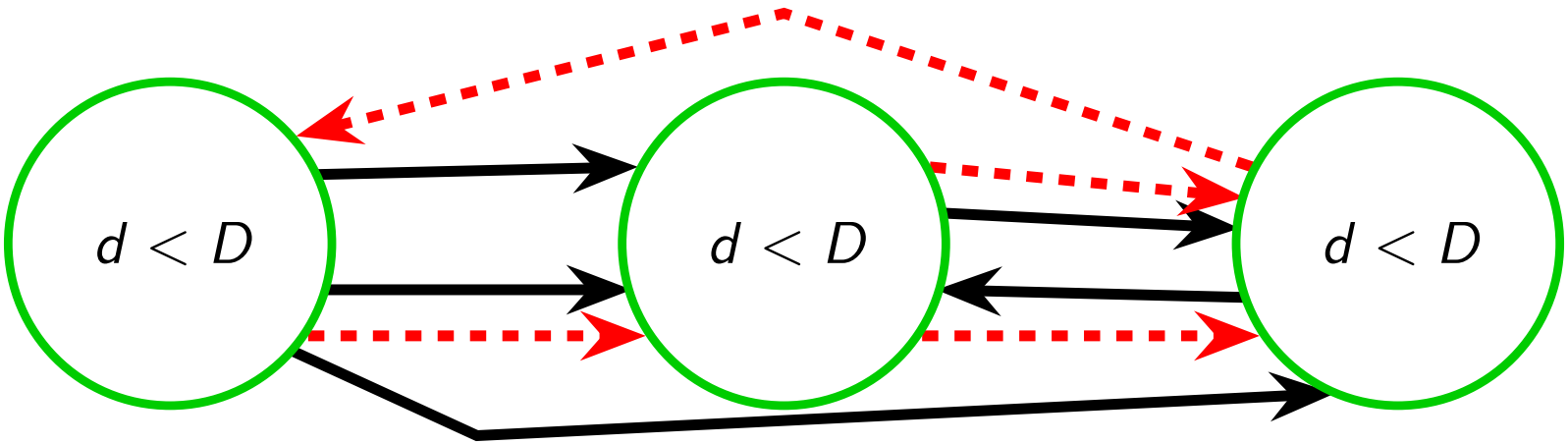
Algorithm: Probabilistic ball–carving

- creating random balls of radius $r \in [0, D)$ with geometric distribution
- cut border edges called LDD removed edges



Directed LDD

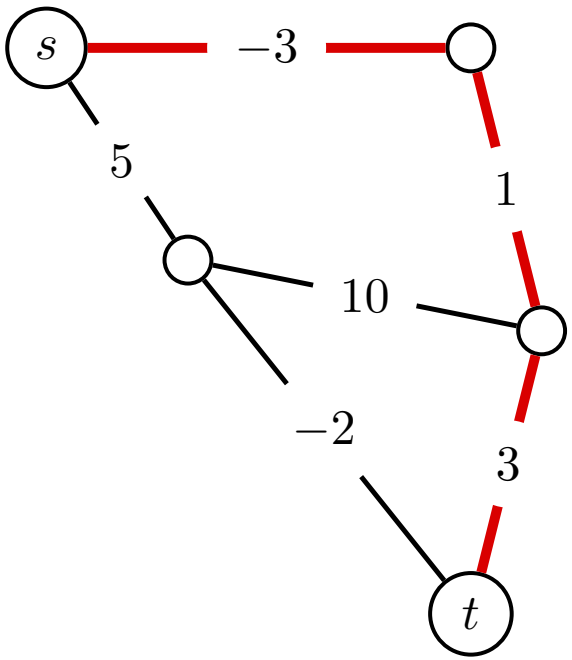
- First appearance: Near-Optimal Decremental SSSP ... (BGW 2020)
- Only uses: Decremental SSSP (2020), Negative-Weight SSSP (2022)

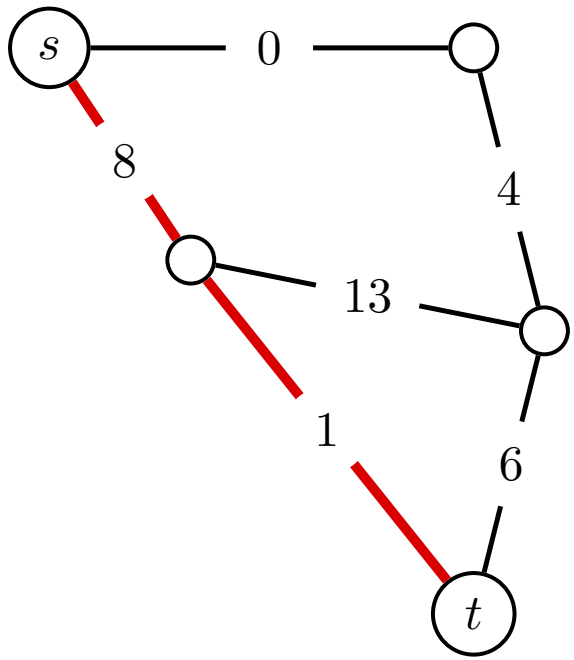


Problem! LDD works only on graphs with non-negative weights.

Price function Φ

Adding a constant to weights doesn't work for negative-weighted graphs.





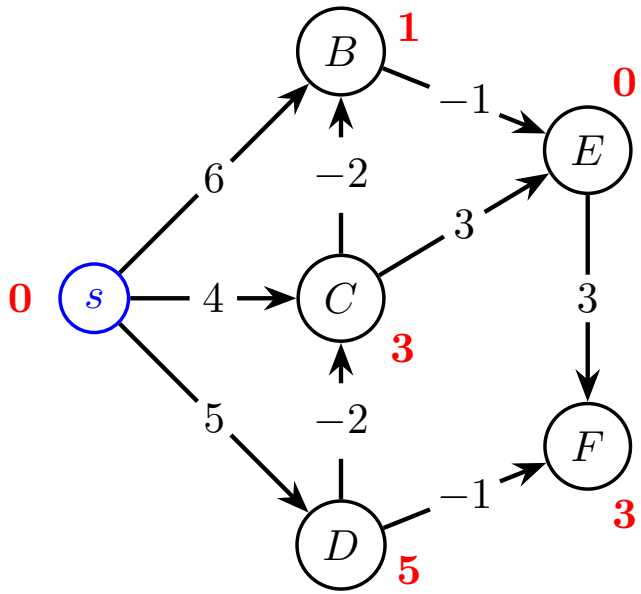
Shortest $s - t$ paths aren't the same!

Price function Φ

But we can assign values $\Phi(v)$ to vertices and redefine weights as

$$w_\Phi(u, v) = w(u, v) + \Phi(u) - \Phi(v).$$

Theorem: This transformation preserves the shortest paths.



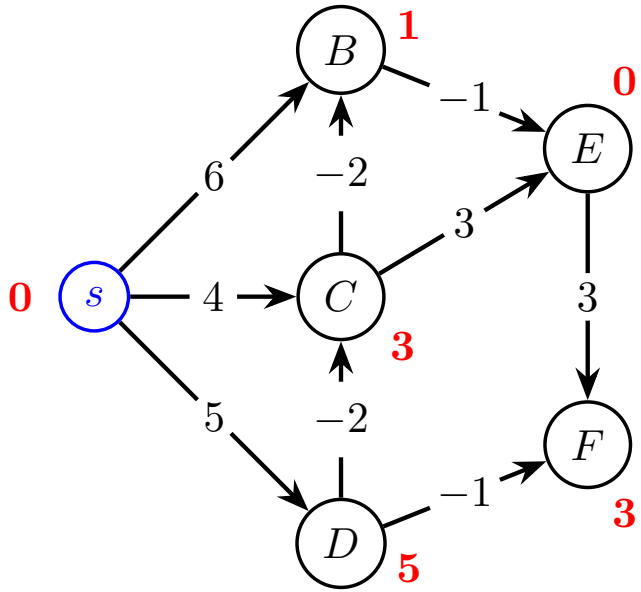
With $\Phi(v_j)$ just run Dijkstra as reduced weights $w_\Phi(e_i) \geq 0$ — we are done!

Price function Φ

Theorem: Distance from any node to all other is a good price function.

In example below we simply use Bellman–Ford distances from node s.

Check a few samples for $w_\Phi(u, v) = w(u, v) + \Phi(u) - \Phi(v) \geq 0$.

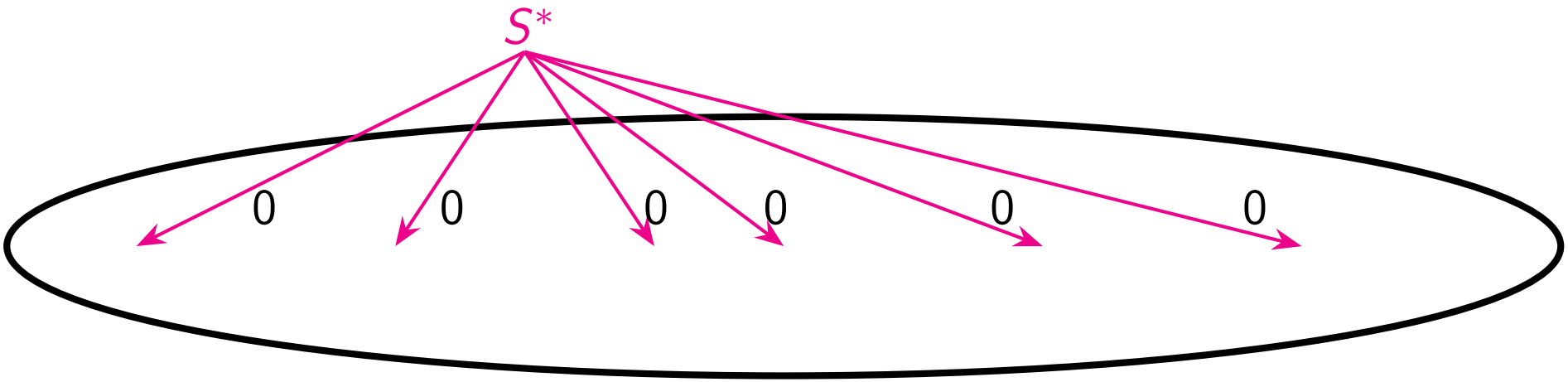


But this is our original problem — we don't know the distances!

Price function Φ

The trick — introduce a dummy source (super source) S^* with weights 0.

Special properties — connected to all nodes, $dist(S^*, v)$ either 0 or negative, connections into just one direction. . . a lot of tech in this step.



We have our Φ if we find a distance from S^* to all nodes.

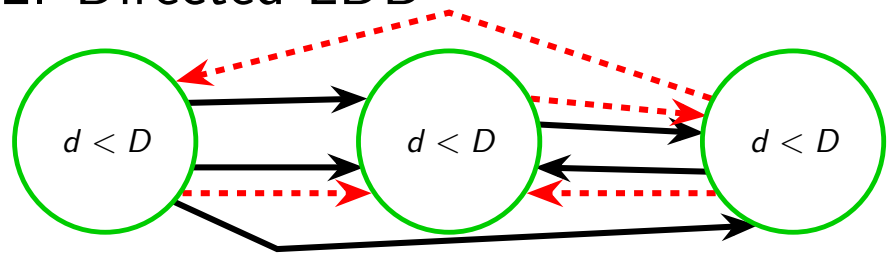
Original graph is just a sub-graph, now with weights fixed, i.e. $w_\Phi(e_i) \geq 0$.

Final Decomposition

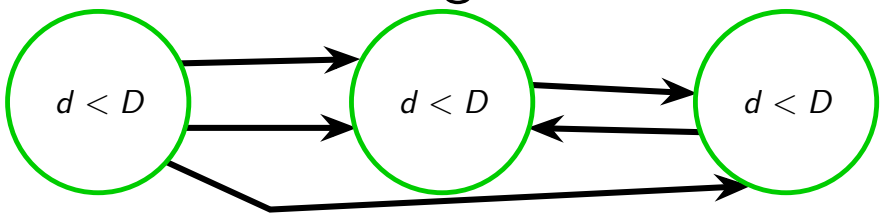
1. Add maximal $-W$ weight to edges

$+W$ *only for LDD*
$w(e) \geq 0$

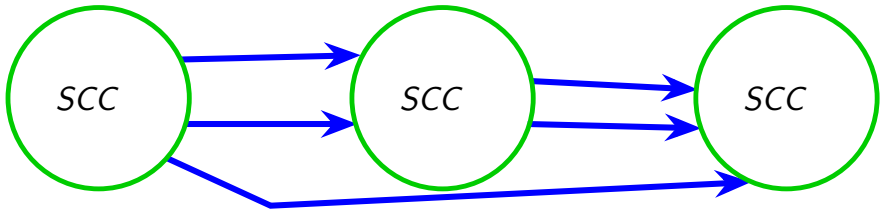
2. Directed LDD



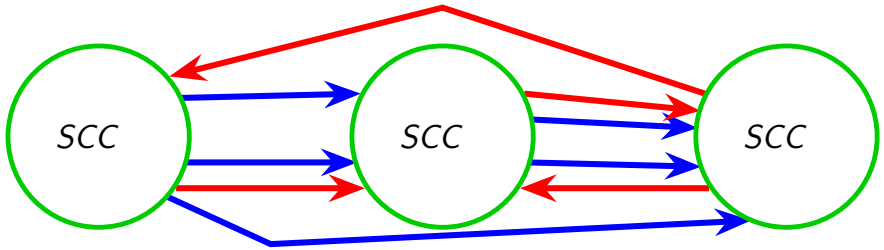
3. Remove cut edges



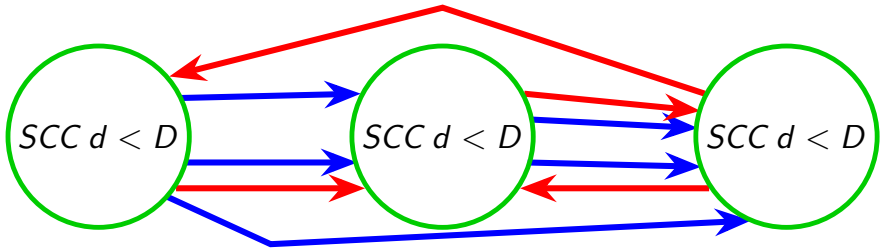
4. Find SCCs and topo-sort



5. Sorted SCCs with LDD edges



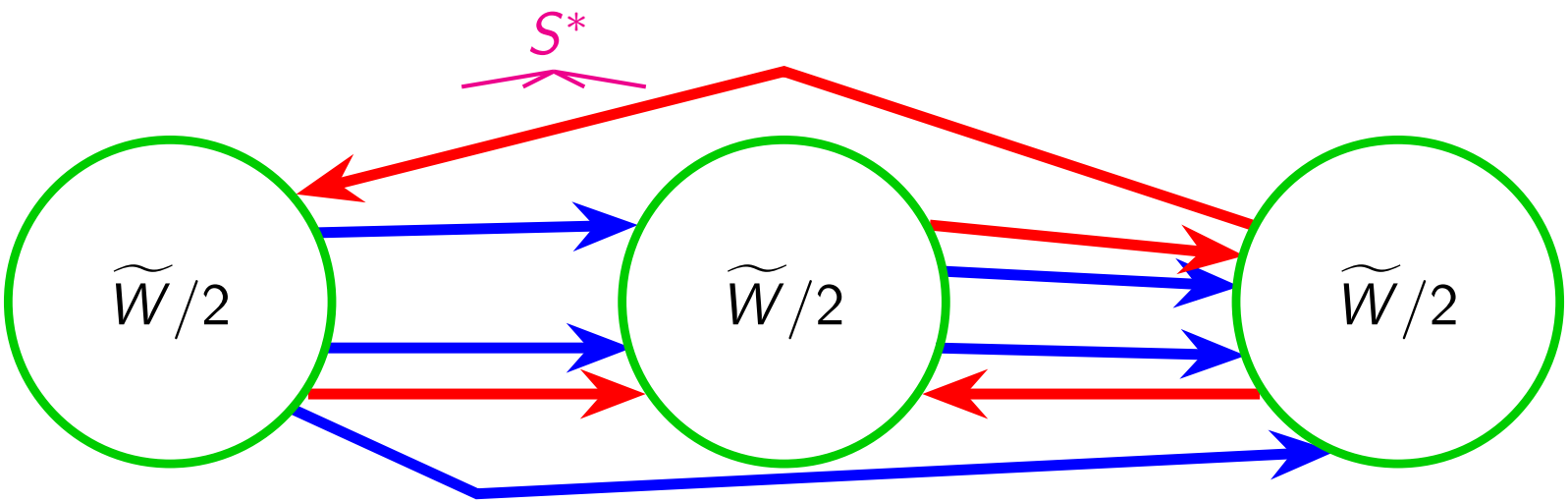
6. SCCs also have $d < D$



Phase 1 — Improve SCCs "negativity"

First we focus on finding distances from S^* to SCC nodes.

$$\text{for any SCC} \quad d < D \qquad d \geq -D \qquad \#(\text{negative distances to } S^*) < n/2$$



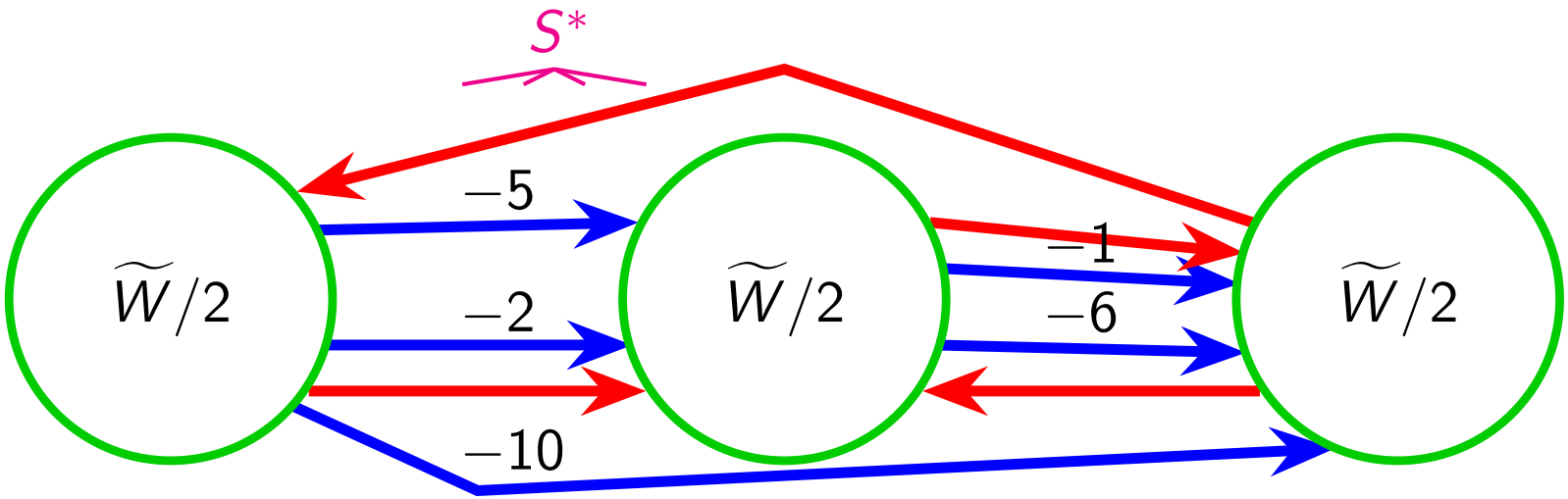
For proper parameters, SCCs "negativity measure" $\widetilde{W}$ reduced by 2.

We want $\widetilde{W}$ & $\#_{neg}$ to be very small — than we can apply Dijkstra + BF.

Phase 2 — Resolve DAG edges between SCCs

After (or assuming) we solved for edges inside SCCs,

Add same price to SCC nodes based on the most negative DAG weight.



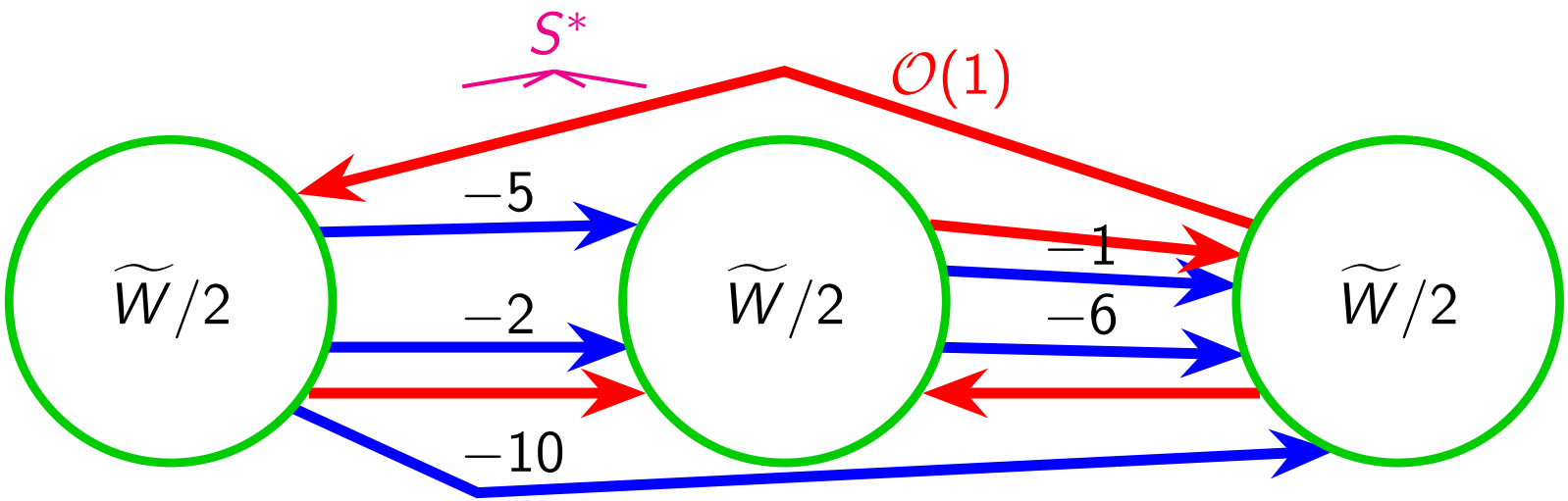
$\Phi_1 = -5 \qquad \Phi_2 = -5-10 = -15$

This price function ensures for all modified DAG edges to be positive.

Phase 3 — Resolve LDD removed edges

Theorem: There are only $\mathcal{O}(1)$ *negative* LDD removed edges!

Ball–carving usually finishes with a heavy positive edge.

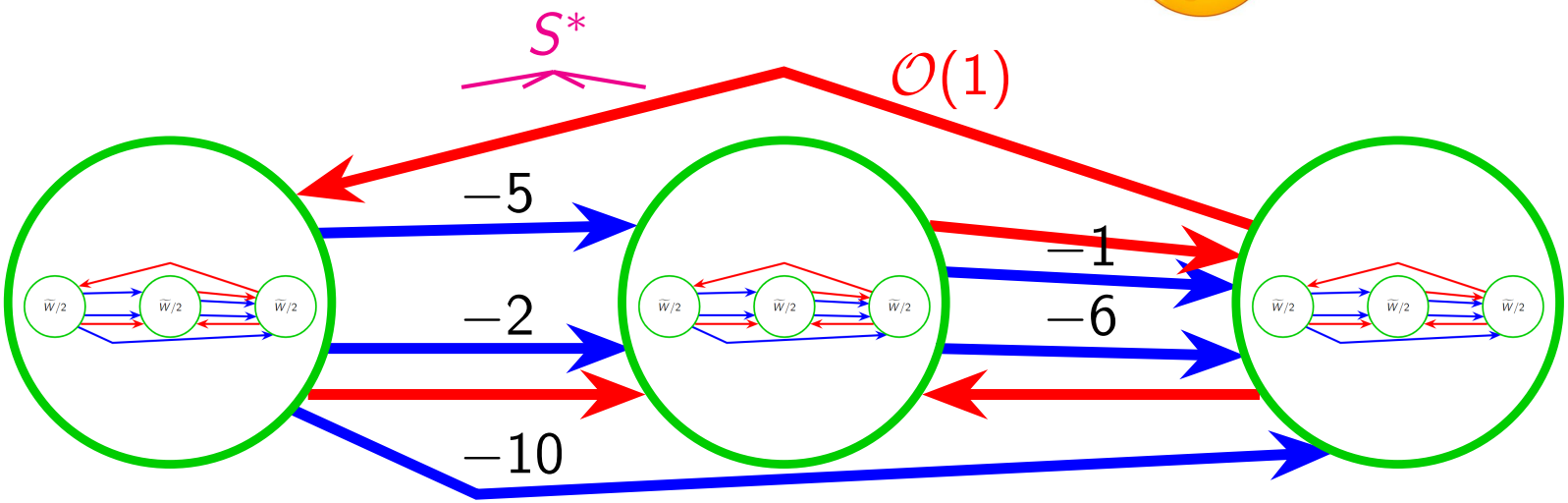


$\Phi_1 = -5$ $\Phi_2 = -5-10 = -15$

They can be resolved by $\mathcal{O}(1)$ Bellman–Ford steps.

The Recursion

And now do all that recursively within SCCs.



$$\Phi_1 = -5 \qquad \Phi_2 = -5-10 = -15$$

Negativity drops exponentially as $\widetilde{W}/2^k$ in k steps.

Recurse until $W/2^k < 1/n^2$ — path length induced error below integer resolution — reason for integer condition and $\mathcal{O}(\log W)$ time complexity.

Complexity

LDD	$\mathcal{O}(m \log^{\alpha} n)$
Misc. steps	$\mathcal{O}(\log^{\beta} n)$
Recursion	$\mathcal{O}(\log W)$
TOTAL	$\mathcal{O}(m \log^{8} n \log W)$

Summary

Three Pillars:

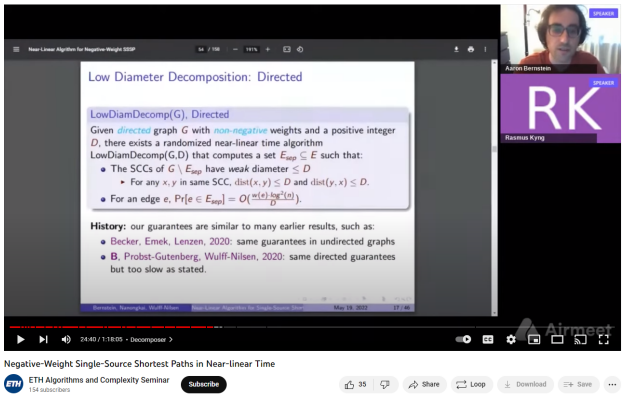
1 2 3

Dummy Source Price Function + $\widetilde{W}/2$ effect + The Recursion

Resources

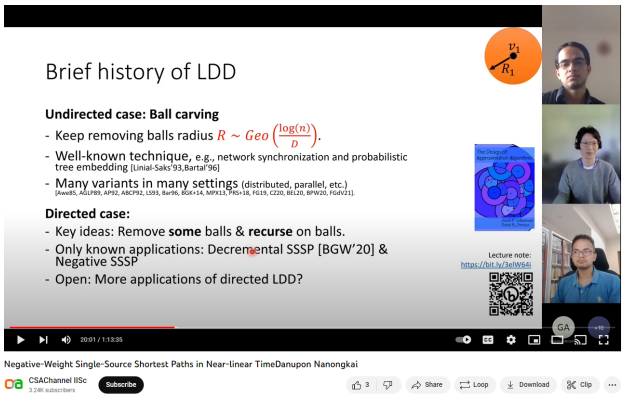
Aaron Bernstein on YouTube

`https://www.youtube.com/watch?v=Bpw3yqWT_d0`



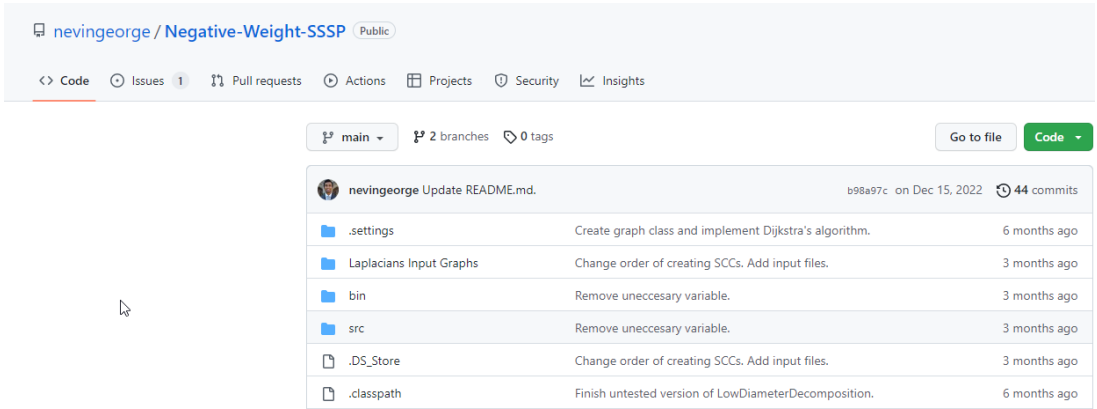
Danupon Nanongkai on YouTube

`https://www.youtube.com/watch?v=awvBpvlbG1M`



George Nevin's implementation on GitHub

`https://github.com/nevingeorge/Negative-Weight-SSSP`



README.md

Negative-Weight Single-Source Shortest Paths

Code implements the near-linear time algorithm of Aaron Bernstein, Danupon Nanongkai, Christian Wulff-Nilsen that computes the shortest paths from a single source vertex to all other vertices in a graph with integral (potentially negative) edge weights. Link to paper: https://doi.org/10.48550/arXiv.2203.03456

To run the code, enter the directory containing the executable negativeWeightSSSP.jar, and run the command "java -jar negativeWeightSSSP.jar" in a Terminal or Command Prompt window.

Input file format

The first 10 lines of the input file are parameters for the algorithm.

cca. 2000 lines of Java

Thank You So Much!

Questions, comments, reflections...