Finding k Simple Shortest Paths and Cycles

Vijaya Ramachandran
University of Texas at Austin, USA

(Joint work with Udit Agarwal)

(http://arxiv.org/pdf/1512.02157v1.pdf)

k Simple Shortest Paths

Given: Directed graph $G = (V, E)$ with non-negative edge-weights, a pair of vertices $s, t \in V$, positive integer k; $|V| = n$, $|E| = m$.

- Find the k shortest paths from s to t.
  Easy: $O(m + n \log n + k)$ time [Eppstein'98]
- Find the k shortest paths with distinct path lengths from s to t.
  Hard: NP-hard even for $k = 2$ [Lalgudi-Papaefthymiou'97]
- Find the k simple shortest paths from s to t.
  $\tilde{O}(k \cdot mn)$ time algorithm [Yen'71]
  Even for $k = 2$, subcubic (for dense graphs) only if APSP has sub-cubic algorithm [Williams-Williams'10]

- For $k = 1$ all three problems are the same, and efficiently solvable using Dijkstra's algorithm.

This Talk: Finding k Simple Shortest Paths and Cycles

Prior work in this topic:

- k simple shortest paths from s to t (k-SiSP) [Yen'71, GL09, RZ12]]: $\tilde{O}(kmn)$ time.
- Enumeration of k simple cycles (in no particular order): $O(kmn)$ [Tarjan'73], improved to $O(km)$ in [Johnson'75].

We study the following natural variants:

- k simple shortest paths for <u>all pairs</u> (k-APSiSP).
- k simple shortest <u>cycles</u> through a given vertex (k-SiSC), or through each vertex in G (k-AVSiSC).
- Enumeration of k simple shortest cycles (k-All-SiSC) and
  k simple shortest paths (k-All-SiSP) in G.

Main Algorithmic Contributions

- New approach: Find simple shortest paths through path extensions:
  - Solves 2-APSiSP in $\tilde{O}(mn)$ time & 3-APSiSP in $\tilde{O}(mn^2)$ time. (Improves $\tilde{O}(n^3)$ for 2-APSiSP and $\tilde{O}(mn^3)$ for 3-APSiSP)
  - Solves k-All-SiSP in $O(m)$ time for the first path and $\tilde{O}(\min\{j, n\})$ for the j-th path. (uses different path extensions from the ones for k-APSiSP)
- Algorithms and reductions to obtain $\tilde{O}(mn)$ time algorithms for 2-AVSiSC and for k-SiSC, k-All-SiSC, for constant k.
- Also show that all of these problems as at least as hard as finding a minimum weight cycle (Min-Wt-Cyc) in a sparse graph, except k-All-SiSP (using $\leq_{(m,n)}$ reductions).

Reductions and Hardness Class

- The APSP hardness class contains a large collection of problems that are at least as hard as APSP for sub-cubic algorithms [WW'10].
- But this does not distinguish between dense and sparse graphs.
- We consider reductions that preserve sparsity, and the starting problem is Min-Wt-Cyc, which has an $\tilde{O}(mn)$ time algorithm.
- So, our hardness class is Sparse Min-Wt-Cyc hardness, and is with regard to sub-mn algorithms.

- $O(m^{3/2})$ is another (faster) sparse time bound that matches n^3 in the dense case, achieved by Min-Wt-Triangle [IR'78].
- But $O(mn)$ appears to be the most common time bound for sparse versions of problems equivalent to APSP under sub-cubic reductions.

Problem	Known Results	New Results
2-APSiSP	Upper Bound: $\tilde{O}(n^3)$ (using DSO) [BK]	**Upper Bound: $\tilde{O}(mn)$**
3-APSiSP	UB: $\tilde{O}(mn^3)$ [Yen]	**UB: $\tilde{O}(mn^2)$**
2-SiSP	LB: Min-Wt-$\Delta \leq$ 2-SiSP (for subcubic) [WW] UB: $\tilde{O}(mn)$ [Yen]	**LB: Min-Wt-Cyc $\leq_{(m,n)}$ 2-SiSP**
k-SiSP	LB: Same as 2-SiSP UB: $\tilde{O}(kmn)$ [Yen]	**LB: Same as 2-SiSP**
k-SiSC	—	**k-SiSP $\equiv_{(m,n)}$ k-SiSC**
k-AVSiSC	—	**LB: Min-Wt-Cyc $\leq_{(m,n)}$ 2-AVSiSC** **UB: $\tilde{O}(mn)$ for $(k = 2)$ and $\tilde{O}(kmn^2)$ for $(k > 2)$**
k-All-SiSC	—	**LB: Min-Wt-Cyc $\leq_{(m,n)}$ 2-All-SiSC** **UB: $\tilde{O}(mn)$ per cycle**
k-All-SiSP	—	**UB: amortized $\tilde{O}(k)$ if $k < n$ and $\tilde{O}(n)$ if $k \geq n$ per path after a startup cost of $O(m)$**

Table : Our Main Results. (DSO stands for Distance Sensitivity Oracles.)

(m, n) Reductions

Definition. Given graph problems P and Q, an (m, n) reduction, $P \leq_{(m,n)} Q$, means that an input $G = (V, E)$ to P with $|V| = n$, $|E| = m$ can be transformed in $O(m + n)$ time to an input $G' = (V', E')$ to Q with $|V'| = O(n)$ and $|E'| = O(m)$ such that from a solution for Q on G' we can obtain a solution for P on G in $O(m + n)$ time.

- Our main reductions:

Min-Wt-Cycle $\leq_{(m,n)}$ 2-SiSP $\leq_{(m,n)}$ k-SiSP $\equiv_{(m,n)}$ k-SiSC

Trivially, APSP $\leq_{(m,n)}$ k-APSiSP, k-SiSC $\leq_{(m,n)}$ k-AVSiSC, Min-Wt-Cycle $\leq_{(m,n)}$ k-All-SiSC

- Prior related known results:

Min-Wt-Cycle $\leq_{(m,n)}$ APSP

2-SiSP $\leq_{(m,n)}$ APSP plus $O(n^2)$ processing [GL'09]

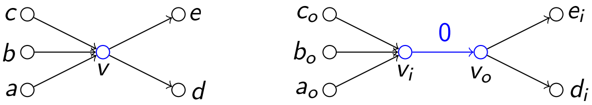
k-SiSP reduces to k calls to 2-SiSP [RZ'12]

Reductions: k-SiSP $\equiv_{(m,n)}$ k-SiSC

- k-SiSP $\leq_{(m,n)}$ k-SiSC:
  - Input is G, with source s and sink t.
  - Form G' by adding a new vertex u' and zero-weight edges (u', s), (t, u').
  - k-th simple s-t path in G is k-th simple cycle in G' though u'.
- k-SiSC $\leq_{(m,n)}$ k-SiSP:
- To compute k-SiSC through v in $G = (V, E)$:
  - Split v into v_i and v_o.
  - All incoming edges to v become incoming to v_i.
  - All outgoing edges from v become outgoing from v_o.
  - A simple cycle through vertex v in G is transformed into a simple path from v_o to v_i in G' with same weight.
  - So k-SiSC $\leq_{(m,n)}$ k-SiSP.

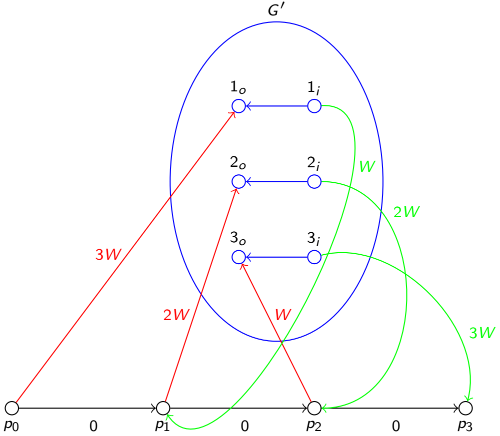
Min-Wt-Cycle $\leq_{(m,n)}$ 2-SiSP

- **Cycle to Path**: Basic transformation from G to G' converts each vertex v into v_i and v_o with zero-weight edge (v_i, v_o).
  - All incoming edges to v become incoming to v_i.
  - All outgoing edges from v become outgoing from v_o.



- A simple cycle through vertex v in G is transformed into a simple path from v_o to v_i in G' with same weight.

Min-Wt-Cycle $\leq_{(m,n)}$ 2-SiSP



- path $\langle p_0, \cdots, p_n \rangle$ with zero-weight edges.
- $W = n \cdot w$, where w is max edge-weight in G.
- edge of weight $(n - j + 1)W$ from p_{j-1} to j_o and an edge of weight jW from j_i to p_j.

Refinements Within $\tilde{O}(mn)$

TIME BOUND	PROBLEMS ACHIEVING THE TIME BOUND
$m \cdot n$	Min-Length-Cycle, Unweighted APSP (undirected and directed)
$m \cdot n \cdot \log \alpha(m, n)$	Undir Min-Wt-Cycle, Undir Wted APSP [PR'05]
$m \cdot n + n^2 \cdot \log\log n$	Min-Wt-Cyc, *k*-SiSP [Yen'71,GL'09], *k*-SiSC [here] (constant *k*), Directed APSP [Pettie'04]
$m \cdot n + n^2 \log n$	2-APSiSP, 2-AVSiSC, *k*-All-SiSC [all here] (constant *k*)
$(m \cdot n + n^2 \log n) \cdot \log n$	DSO [BK'09]
$n \cdot (m \cdot n + n^2 \log\log n)$	*k*-AVSiSC, $k \geq 3$ [here]
$n \cdot (m \cdot n + n^2 \log n)$	3-APSiSP [here]
$n^2(m \cdot n + n^2 \log n \log n)$	*k*-APSiSP, $k \geq 4$ [Yen'71, GL'09]

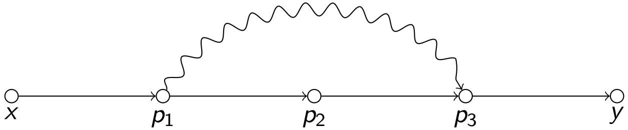
Path Extension Algorithms

The rest of the talk will cover:

- The 2-APSiSP algorithm.
  - Uses path extensions that may not be detours.
- 3-APSiSP, and k-APSiSP, $k \geq 3$.
  - Uses recursion, but inefficient for larger k.
- k-All-SiSP.
  - Uses a different type of path extension.

Background for SiSP

- k-SiSP. All known algorithms for k-SiSP (and 2-SiSP) from x to y compute detours around each edge in a shortest path, and then choose the shortest $x - y$ path generated by a detour.
- Replacement Paths. This computes, for each edge e on an x-y SP, a shortest path avoiding e. 2-SiSP from x to y can be computed as the minimum weight replacement path.



- **Lower Bound.** $O(m\sqrt{n})$ lower bound for both 2-SiSP and Replacement Paths in the path-weight comparison model, *assuming that the algorithm only examines these detours* [HSB'07].
- Our 2-APSiSP algorithm generates and examines paths that are not detours for any pair of vertices.

Replacement Paths

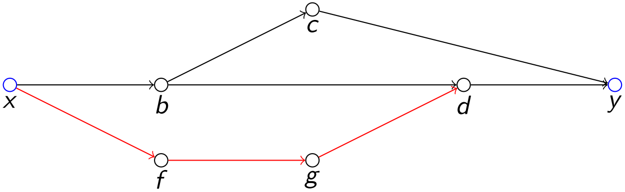
- **Replacement paths for a single pair x, y:** $O(mn + n^2 \log\log n)$ time [Yen'71, GL'09]
  - 2-SiSP is computed by the same algorithm plus $O(n)$ additional time to select a minimum weight replacement path.
- **Replacement paths for all pairs $x, y \in V$:**
  - The output can potentially have size $\Omega(n^3)$, simply for the weights of all replacement paths.
  - Instead use compact *distance sensitivity oracles* (DSO) [DTCR'08] of size $\tilde{O}(n^2)$.
  - Any specific replacement path can be found from DSO in constant time.
  - Current fastest algorithm for DSO runs in $O(mn \log n + n^2 \log^2 n)$ time [BK'09]
  - BUT: 2-APSiSP from DSO takes n^3 time.
    (to examine up to n^3 replacement paths)

Our Approach

- We first compute k *nearly SiSP* sets $Q_k(x, y)$ (to be defined).
- We then use an algorithm Compute-APSiSP (to be presented) that computes k-APSiSP from the $Q_k(x, y)$ sets in $O(kn^2 + n^2 \log n)$ time.

The Q_2 Sets and Distance Sensitivity Oracles

- **Definition.** The set $Q_2(x, y)$ of the two nearly shortest simple paths from x to y in G contains a shortest path π from x to y, and a shortest path from x to y in G that avoids the first edge on π (if such a path exists).



- Observation: Using DSO, we can compute the $Q_2(x, y)$ sets, in additional $O(n^2)$ time for all pairs.

  (We have another method – simpler than DSO – that computes $Q_2(x, y)$ sets directly in $O(mn + n^2 \log n)$.)

2-APSiSP Algorithm

- The 2-APSiSP Algorithm:
  - Compute the first path in all Q_2 sets with an APSP computation.
  - Compute the second path in each Q_2 set in $O(1)$ time using distance oracles.
  - Compute 2-APSiSP from the Q_2 sets.
    (Need an algorithm for this – Compute-APSiSP)

The Q_k Sets

Assume that there are k simple paths from x to y, for all $x, y \in V$. Then,

- $P_k^*(x, y)$ is the set of k simple shortest paths from x to y in G.
- $Q_k(x, y)$ is the set of k nearly simple shortest paths from x to y, defined as follows:
  - if all paths in $P_{k-1}^*(x, y)$ share the same first edge (x, a), then $Q_k(x, y)$ contains all paths in $P_{k-1}^*(x, y)$, together with the shortest simple path from x to y that does not start with edge (x, a), if such a path exists.
  - Otherwise, $Q_k(x, y) = P_k^*(x, y)$.
- **Task for Algorithm** Compute-APSiSP.
  - If the $k - 1$ shortest paths in $Q_k(x, y)$ all start with the same edge (x, a) then we need to determine if the k-th simple shortest path from x to y also starts with edge (x, a).
  - Otherwise, $Q_k(x, y) = P_k^*(x, y)$.

Algorithm COMPUTE-APSISP

- Algorithm COMPUTE-APSISP computes k-APSiSP in $O(k \cdot n^2 + n^2 \log n)$ time, for any $k \geq 2$, given the $Q_k(x, y)$ sets.

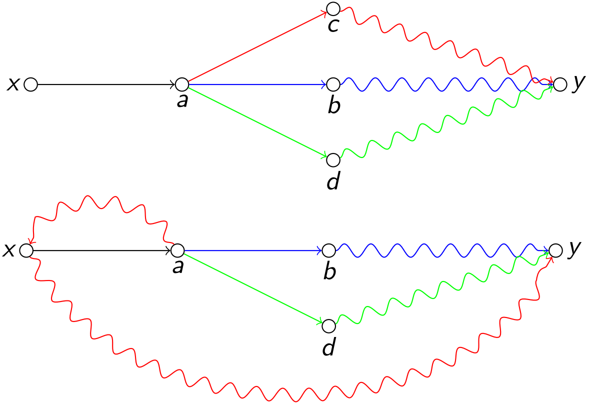
* **Recall:** Only if the $k-1$ shortest paths in $Q_k(x, y)$ all start with the same edge (x, a) then COMPUTE-APSISP needs to determine if the k-th simple shortest path from x to y also starts with edge (x, a). (Otherwise $Q_k(x, y) = P_k^*(x, y)$.)

- The pairs x, y for which * holds can be determined by scanning the Q_k sets, which are input to COMPUTE-APSISP.
- For these pairs, COMPUTE-APSISP uses the following Lemma 1 to find the k-th path.

Lemma 1. If all paths in $P_k^*(x, y)$ start with the same first edge (x, a) then $P_k^*(a, y)$ consists of the right subpaths of the paths in $P_k^*(x, y)$.

Lemma 1

Lemma 1. If all paths in $P_k^*(x, y)$ start with the same first edge (x, a) then the right subpath of the i-th simple shortest path from x to y has weight equal to the weight of the i-th simple shortest path from a to y, $1 \leq i \leq k$.



Algorithm Compute-APSiSP

- Algorithm COMPUTE-APSiSP maintains a set *Extensions*(a, y) for each pair of vertices a, y.
- *Extensions*(a, y) contains those edges (x, a), incoming to a, that are the first edge on the $k-1$ simple shortest paths from x to y.
  - So, if the $k-1$ shortest paths in $Q_k(x, y))$ all start with (x, a) then (x, a) is placed in *Extensions*(a, y)
- Lemma 1 shows that we may need to 'pre-extend to x,' the k-th simple shortest path from a to y in order to compute the k-th simple shortest path from x to y that uses (x, a) as the first edge.
  - COMPUTE-APSiSP performs these path extensions and may create paths that are not detours.

Algorithm $\textsc{Compute-APSiSP}(G = (V, E), wt, k, \{Q_k(x, y), \forall x, y\})$

Initialize:
$H \leftarrow \phi$ $\quad$ $\{H$ is a priority queue.$\}$
for all $x, y \in V, x \neq y$ **do**
    $P^*_k(x, y) \leftarrow Q_k(x, y)$
    if the $k - 1$ shortest paths in $P^*_k(x, y)$ have the same first edge, say (x, a) **then**
        Add (x, a) to the set $Extensions(a, y)$
        if $|Q_k(a, y)| = k$ **then**
            $\pi \leftarrow$ the path of largest weight in $Q_k(a, y)$
            $\pi' \leftarrow (x, a) \circ \pi$
            Add π' to H with weight $wt(x, a) + wt(\pi)$
Main Loop:
while $H \neq \phi$ **do**
    $\pi \leftarrow \textsc{Extract-min}(H)$
    Let $\pi = (xa, y)$ and let the path of largest weight in $P^*_k(x, y)$ be π'
    if $|P^*_k(x, y)| = k - 1$ **then** add π to $P^*_k(x, y)$ and set *update* flag
    else if $wt(\pi) < wt(\pi')$ **then** replace π' with π in $P^*_k(x, y)$ and set *update* flag
    if *update* flag is set **then**
        for all $(x', x) \in Extensions(x, y)$ **do**
            add $(x', x) \circ \pi$ to H with weight $wt(x', x) + wt(\pi)$

Analysis of Compute-APSiSP

- **Lemma 2.** Algorithm Compute-APSiSP correctly computes the sets $P_k^*(x, y)\ \forall x, y \in V$.
- **Lemma 3.** Algorithm Compute-APSiSP runs in $O(kn^2 + n^2 \log n)$ time.
  - **Corollary 1.** Using DSO, 2-APSiSP can be computed by an $O(mn \log n + n^2 \log^2 n)$ time randomized algorithm.
  - **Corollary 2.** 2-APSiSP can be computed in $O(mn + n^2 \log n)$ time.
    (This uses an algorithm that computes the Q_2 sets without using DSO.)

3-APSiSP and k-APSiSP

- 3-APSiSP:
  - Compute the Q_3 sets by recursively calling 2-APSiSP on G, with incoming edges to v removed, for each $v \in V$.
  - Call COMPUTE-APSISP with the Q_3 sets.
- Run-time is $O(mn^2 + n^3 \log n)$ (dominated by the recursive calls).
- Previous best method was to run the 3-SiSP algorithm $\Theta(n^2)$ times, which takes $O(mn^3 + n^4 \log\log n)$.
- k-APSiSP:

  The Q_k sets can be computed by the same recursive method, but the running time degrades with larger k.

Algorithm for k-All-SiSP

$\textsc{All-SiSP}(G = (V, E); wt)$

```
1: Initialization:
2: for all (x, y) ∈ E do
3:     Add (x, y) to priority queue H with wt(x, y) as key
4:     Add (x, y) to L(⟨y⟩) and R(⟨x⟩)
5: Main loop:
6: while H ≠ φ do
7:     π ← EXTRACT-MIN(H)
8:     Add π to the output sequence of simple paths
9:     Let π_xb = ℓ(π) and π_ay = r(π) ((x, a) and (b, y) are first and last edges on π)
10:    for all π_x'b ∈ L(π_xb) with x' ≠ y do
11:        Form π_x'y ← (x', x) ∘ π and add π_x'y to H with wt(π_x'y) as key
12:        Add π_x'y to L(π_xy) and to R(π_x'b)
13:    for all π_ay' ∈ R(π_ay) with y' ≠ x do perform steps complementary to Steps 11
   and 12
```

Lemma 4. Algorithm $\textsc{All-SiSP}$ computes the shortest path in $O(m)$ time and each succeeding simple shortest path in amortized $O(k + \log n)$ time if $k = O(n)$ and $O(n + \log k)$ time if $k = \Omega(n)$.

Summary

- Simple Shortest Paths and Cycles
  - New algorithm, using path extensions, for 2-APSiSP with the same time bound as 2-SiSP (to within a log factor), and for 3-APSiSP.
  - Reductions between sparse graphs for most versions of finding k simple shortest paths and cycles, showing hardness relative to Sparse Min-Wt-Cyc.
  - Very fast algorithm for k-All-SiSP, again with path extensions.
- Further Research
  - Can we compute the Q_k sets more efficiently?
  - Space usage is high in our all-pairs algorithms. Can we obtain more space-efficient algorithms?
  - Hardness relative to Sparse Min-Wt-Cycle.
    - Can we show equivalence to APSP in sparse graphs?
    - More generally, can we further extend the class of problems hard for 'sub-mn' computations?

- Udit Agarwal, Vijaya Ramachandran, "Finding k simple shortest paths and cycles," arXiv:1512.02157v1, 2015.