Prize-Collecting Walks and Branchings in Directed Graphs

Zac Friggstad

UNIVERSITY OF ALBERTA

Alberta-Montana Combinatorics & Algorithms Days at BIRS, 2022

Collaborators

Chaitanya Swamy - U. Waterloo (Faculty)

Sina Dezfuli - U. Alberta (M. Sc.)

Ian Post - U. Waterloo (PDF)

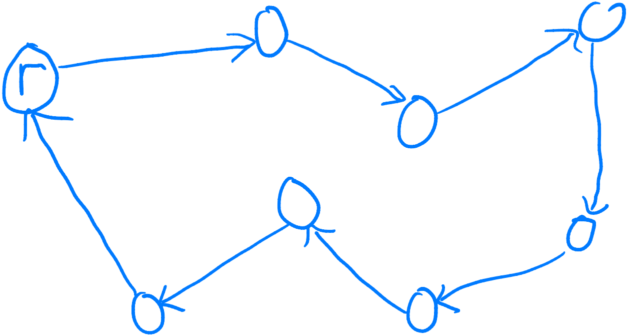
Part 1

A vehicle routing problem (i.e. motivation).

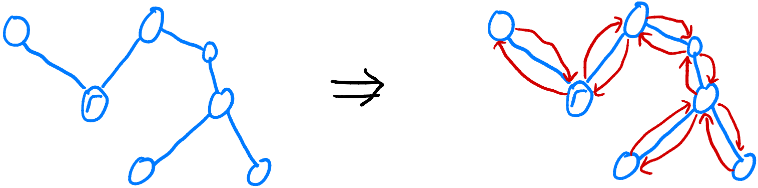
This talk is about finding “good” walks/trees in graphs with applications variants of the Traveling Salesperson Problem (TSP).

Classic TSP

Visit all locations and return home as cheaply as possible.



Very simple heuristic: find the cheapest connected subgraph T and do a depth-first traversal.



The solution has cost ≤ 2 times the optimum TSP tour cost.

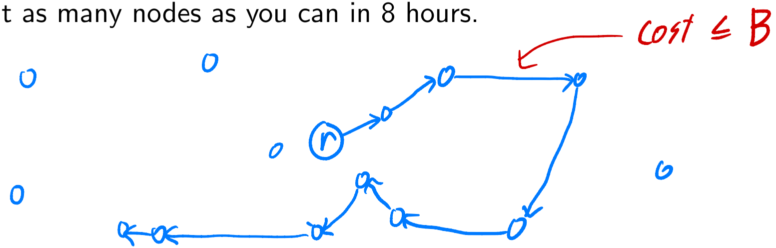
A very brief history:

- Christofides-Serdyukov (1976): a simple 1.5-approximation.
- Karlin, Klein, Oveis Gharan (2021): slightly better

All start with a low-cost connected subgraph and augment it as cheaply as possible to get a tour spanning all nodes.

Related Problem - Orienteering

Visit as many nodes as you can in 8 hours.



Precisely: Given a start node r (depot) and a budget B, find an r-walk with cost $\leq B$ visiting as many nodes as possible.

This Talk: Symmetric distances: $cost(u, v) = cost(v, u)$.

But good to think of edges as directed $(u, v) \neq (v, u)$.

Other variants are studied (eg. end where you start).

Brief history

- $O(1)$-approximations are possible. Best is by Chekuri, Korula, and Ene: $2 + \epsilon$ in $|V|^{O(1/\epsilon)}$ time ($n = \#nodes$). (2012)
- No approximation prior to our work would work in practice (way too slow). All fast heuristics that were proposed could behave terribly in some cases.

Our Work

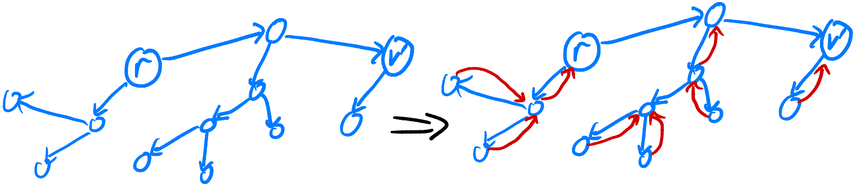
- A 3-approximation in time $\tilde{O}(|V|^4)$. Easy to implement. Trivial to parallelize to run in $\tilde{O}(|V|^3)$ time using $|V|$ processors.
- A fast, combinatorial algorithm that finds branchings (maybe-not-spanning trees) with low "cost" in directed graphs. Inspired by a particular directed graph decomposition by Bang-Jensen, Frank, and Jackson (1995).
- Numerical evaluation of our algorithm: performs much better than a 3-approximation in practice.

A Tree for Orienteering

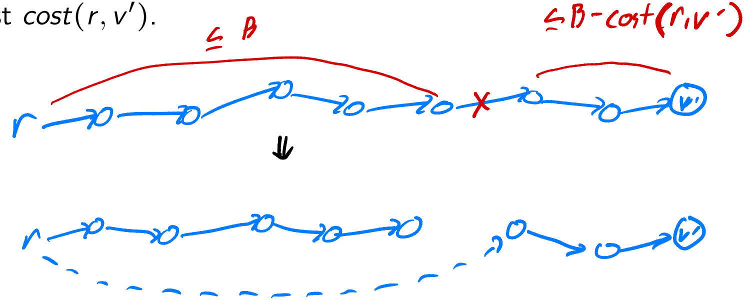
Throughout, let P^* be walk from r with length $\leq B$ visiting the maximum number (say OPT) of nodes.

This Talk: About finding a **tree/branching** T with $cost(T) \leq B$ that includes $\geq OPT$ nodes (not quite, but bear with me).

How does this help?
Let v' be the farthest (from r) node lying on P^*. Including the reverse of all edges not on the $r - v'$ walk yields a walk with cost $\leq D + (D - cost(r, v'))$.



Split the walk into two walks with costs $\leq D$ and $\leq D - cost(r, v')$. Turn the latter into a proper walk from r with additional cost at most $cost(r, v')$.



At least one of these two solutions (both having cost $\leq B$) will cover $\geq OPT/2$ nodes.

How to Get the Trees

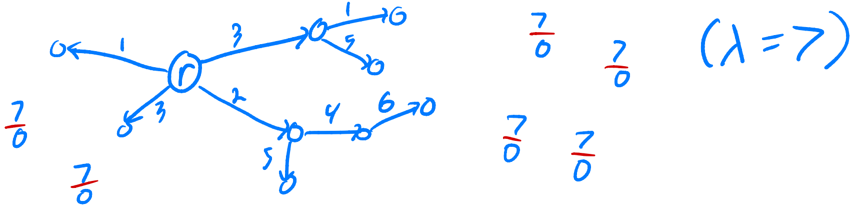
Instead of viewing the edge-cost as a hard constraint, do the following.

Lagrangian Relaxation
Let $\lambda \geq 0$ be some value.

Find a tree T with minimum **prize-collecting cost**:

$$cost(T) + \lambda \cdot (|V| - |V(T)|)$$

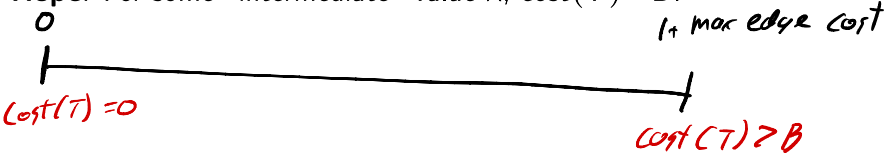
i.e. also pay λ for each node you do not include on T.



Observations

- $\lambda = 0$: $V(T) = \{r\}$. cost(T) = 0
- $\lambda \to \infty$: $V(T) = V$. cost(T) > B

Hope: For some "intermediate" value λ, $cost(T) = B$.



For such a tree T:

$$\begin{aligned} B + \lambda \cdot (|V| - |V(T)|) &\leq cost(P^*) + \lambda \cdot (|V| - OPT) \\ &\leq B + \lambda \cdot (|V| - OPT) \end{aligned}$$

Great! We can turn it into 2 feasible walks one of which visits $\geq OPT/2$ nodes.

Two Issues

1) We might not be able to get a λ such that $cost(T) = B$. Standard fix, not discussed here (but lose a bit: gets a feasible orienteering solution with $\geq OPT/3$ nodes).

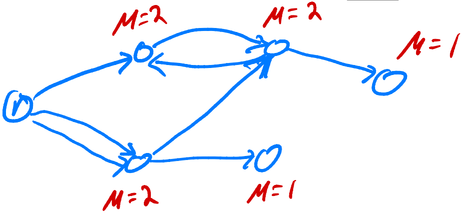
2) It is actually still hard to find the minimum prize-collecting cost tree T.

Our Real Result
An efficient combinatorial algorithm that finds a r-rooted tree T whose prize-collecting cost is at most the prize collecting cost of any r-walk.

But first, let's quickly see how this was done before our fast algorithm.

Decomposing Preflow Graphs

Preflow Graphs: Let $G = (V, E)$ be a directed graph such that $indegree(v) \geq outdegree(v)$ for all but one root node r.

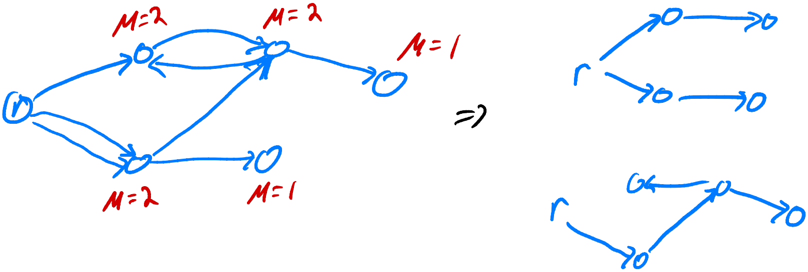


For a node $v \neq r$, let μ_v be the minimum number of edges we must delete to make v not reachable from r.

Bang-Jensen et al. Decompositions

Theorem (Bang-Jensen, Frank, Jackson, 1995)

*Can partition (a subset of) E into r-**branchings** so each $v \in V$ lies on $\geq \mu_v$ branchings.*



r-**Branching**: Has a unique path from r to every other node on the branching (directed tree), but maybe doesn't include all nodes.

Leads to a linear-programming based algorithm for the Orienteering problem.

Variables

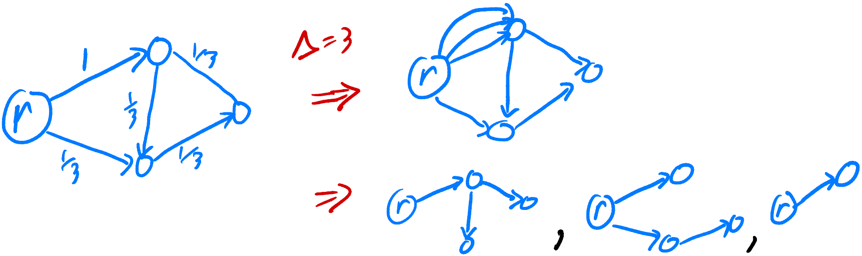
- $x_{u,v}$ for an arc (u,v) indicating we include (u,v) on the walk.
- z_v for a vertex v indicating v will be excluded from the walk.

minimize: $\sum_{(u,v)} cost(u,v) \cdot x_{u,v} + \sum_{v \neq r} \lambda \cdot z_v$
subject to:

$$\begin{array}{rcll}
x(\delta^{in}(v)) & \geq & x(\delta^{out}(v)) & \forall\ v \neq r & (\textbf{preflow}) \\
x(\delta^{in}(S)) & \geq & 1 - z_v & \forall\ v \neq r, \{v\} \subseteq S \subseteq V - \{r\} & (\textbf{connectivity} \\
x(\delta^{out}(r)) & = & 1 & & \\
z & \in & [0,1]^{V-\{r\}} & & \\
x & \geq & \mathbb{R}^{E}_{\geq 0} & &
\end{array}$$

Since the optimum Orienteering solution "is" a feasible solution, the optimum LP solution is at most $B + \lambda \cdot (|V| - OPT)$.

Can compute an optimal solution (x^*, z^*) with rational entries.
Let Δ be such that $\Delta \cdot (x^*, z^*)$ is an integer vector.



Consider the preflow multigraph having $\Delta \cdot x^*_{u,v}$ copies of (u, v).
Do the decomposition: get Δ edge-disjoint branchings such that each v lies on $\Delta \cdot (1 - z_v)$ of them.

Keep the branching with minimum prize-collecting cost.

Phew, that's quite a bit of work to get a single tree.

Involves solving a large linear program ($O(n^2)$ variables, many constraints). Very impractical.

In what follows, we discuss an approach that does not use linear programming. It can't be applied to all problems that use the Bang-Jensen et al. decomposition, but it can for Orienteering and a few other vehicle routing problems.

Part 2

A combinatorial algorithm to find such an r-branching.

Can be seen as a generalization of Edmonds' minimum-cost arborescence algorithm.

Restating the Problem

Let $G = (V, E)$ be a directed graph with a root node r, edge costs $cost(u, v)$, vertex penalties $\lambda(v)$.

Want to find a r-branching T minimizing:

$$cost(T) + \sum_{v \notin V(T)} \lambda(v).$$

That's hard to do, but for vehicle-routing applications it suffices to find such a branching whose prize-collecting cost is at most that of any walk P.

Adjusts Costs/Penalties

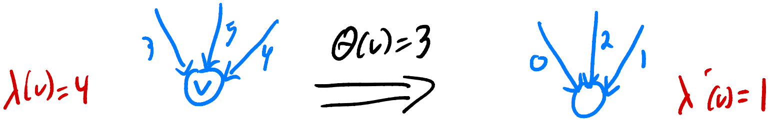
Suppose we subtracted $\theta(v)$ from all edges (u, v) and also from $\lambda(v)$ for each $v \in V - \{r\}$.

Let $\Theta = \sum_{v \in V - \{r\}} \theta(v)$ and $cost', \lambda'$ denote the new costs/penalties.

Lemma

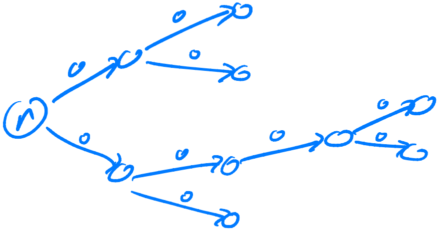
For any r-walk P,

$$cost'(P) + \sum_{v \notin V(P)} \lambda'(v) + \Theta \leq cost(P) + \sum_{v \notin V(P)} \lambda(v).$$



Super-Easy Case

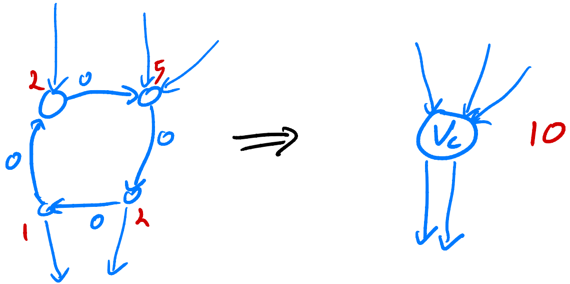
In the modified graph, if r can reach every node using only 0-cost edges, then output any single r-branching T spanning V using these edges.



The original cost of these edges is exactly Θ and the previous lemma shows $\Theta \leq cost(P) + \sum_{v \notin V(P)} \lambda(v)$.

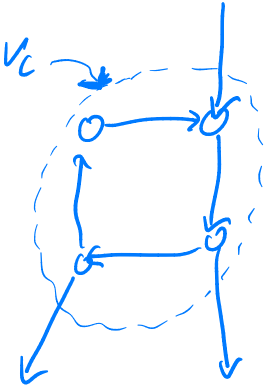
0-Cost Cycles

If there is a cycle C of 0-cost edges in the modified graph, contract them to a single vertex v_C with penalty $\sum_{v \in C} \lambda'(v)$.



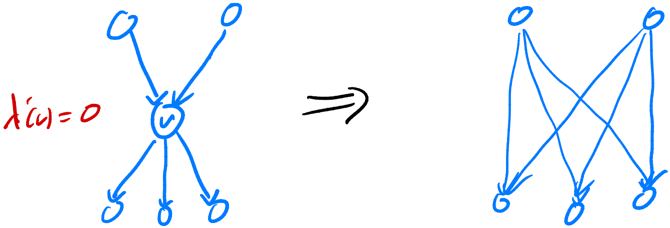
Any walk P in G naturally maps to a walk in this new graph with no worse prize-collecting cost.

Conversely, when we eventuallyt an r-branching T in this contracted graph we can turn it into an r-branching in G with no greater prize-collecting cost. If $v_C \notin V(T)$, do nothing. Otherwise, expand it as follows:



Final Case: A Node Dies

If $\lambda'(v) = 0$ for some $v \in V - \{r\}$, we "bypass it" and remove it.



Any walk P naturally maps to a walk in this new graph with no greater prize-collecting cost.

Final Case: A Node Dies

After finding r-branching T, do the following. For every **bypassing edge** (u, w) used on T, remove it and replace with $(u, v), (v, w)$.

Summary

Given $(G, cost, \lambda, r)$, compute $\theta(v)$ and modified costs/penalties $cost', \lambda'$.

- If r can reach every $v \in V$ using 0-cost edges, pick any r-branching (eg. a search tree).
- Else, if there is a cycle C of 0-cost edges then contract it, recursively find an r-branching, and expand v_C as described if it lies on T.
- Else, pick any $\lambda'(v) = 0$ node, bypass it, recursively find an r-branching T, and adjust any **bypassing edge** as described.

In any case, we get an r-branching in G.

Summary

A careful inspection shows this runs in cubic time (in $|V|$):

- At most $|V|$ "reductions" (cycle contractions or node deletions).
- Each runs in $O(|E|)$ time, note $|E| < |V|^2$ since we ensure the graph is simple.

Theorem (Dezfuli, F., Post, Swamy, 2022)

There is an $O(|V|^3)$ time algorithm that finds an r-branching T such that

$$cost(T) + \sum_{v \notin V(T)} \lambda(v) \leq cost(P) + \sum_{v \notin V(P)}$$

for any r-walk P.

Notes

The bottleneck in the running time bypassing a dead node: even if $|E| = O(|V|)$ (like a road network), it could be that $|E'| = \Omega(|V|^2)$ after a single bypassing step.

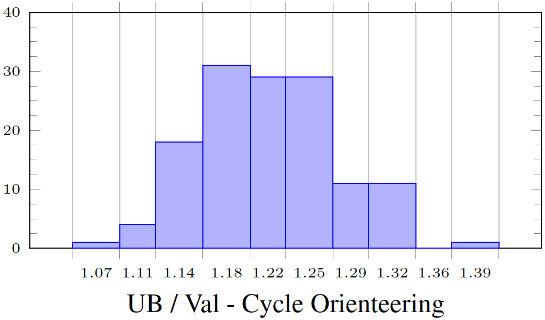
Open Problem: Do the bypassing implicitly, manage necessary information about bypassing/restoring with dynamic trees (eg. link/cut trees) rather than generating a bunch of new edges.

Standard techniques can handle other parts. The hope would be to reduce the running time to $O(|E| \log |E|)$.

Open Problem: If you look at the reason we lost a factor of 3 instead of 2 in the "Lagrangification step", it seems unsatisfactory. Better analysis? Better approach?

Notes

Numerical evaluation. Using TSPLIB datasets with $B = OPT_{TSP}/2$ (as in previous work).



i.e. often visits at least $0.8 \cdot OPT$ nodes.

Works in a few minutes on instances with ~ 200 nodes (recall the final running time of Orienteering is $\tilde{O}(n^4)$: a linear-factor is lost in the Lagrangification part).