Vehicle Routing

Jacob Denson

August 23, 2017

The travelling salesman problem asks to find the shortest cycle in a connected graph passing through each vertex. A practical formulation of this problem is that a door to door deliveryman is attempting to deliver packages to each house in a neighbourhood, and wants to travel between houses as efficiently as possible. The *vehicle routing problem* adds an additional challenge to the scenario: the deliveryman can only carry a limited supply of goods at any once time, and must periodically return to a central depot to resupply. The deliveryman still wants to deliver goods as efficiently as possible, but must now simultaneously cluster potential customers together while optimizing routing within these clusters to ensure that time spent refueling is not wasted. As one of the most famous NP complete problems, most work on the travelling salesman problem is spent deriving heuristics which give approximation guarantees on solutions. In this work, we collect research on heuristics obtained for the vehicle routing problem.

In the vehicle routing problem, we take a complete graph G, with vertices $V \cup \{B\}$, where B is a distinguished *depot node*. We are also given a metric d between the vertices in the graph, as well as a capacity C. Each node on the graph, except for the depot node, has a fixed demand D_v. We shall let r_v denote $d(B, v)$, the *radius* from B to v. To begin with, we consider two variants of the vehicle routing problem. In the *split demand* problem, a feasible solution to this problem consists of a collection of cycles $T_1, \dots, T_n$, each cycle containing B, together with quantities D_v^i for each vertex v on the tour T_i, such that

- Each tour remains within the capacity of the delivery vehicle, so that $\sum_v D_v^i \leq C$ holds for each tour i.
- Each vertex is completely supplied with the capacity which it demands, so that $\sum_i D_v^i = D_v$.

In the *unsplit demand* version of the problem, we are unable to split demand satisfaction of a single node between different tours; each node must be completely supplied by a single vertex. In this case, we can assume a feasible solution is a collection of cycles $T_1, \dots, T_n$ forming a disjoint cover of the vertices V, intersecting only at B, such that

- Each tour remains within capacity, so that $\sum_{v \in T_i} D_v \leq C$ holds for each tour T_i. Of course, in this version of the problem we must therefore assume $D_v \leq C$ for each vertex v.

The goal of both the split and unsplit versions of vehicle routing is to find a feasible solution minimizing the combined total length of all cycles.

Part I
Basic Results

1 Lower Bounds to Vehicle Routing

In our introduction to vehicle routing, we emphasized how vehicle routing is an extension of the travelling salesman problem. The additional problem is to partition the graph into efficient, bounded demand clusters of nodes to travel through at once. Indeed, if the capacity of the salesman is greater than the sum of the demands of all vertices, than an optimal solution to the travelling salesman problem will give an optimal solution to the vehicle routing problem. Here we introduce strategies showing that feasible solutions to the travelling salesman problem can be adapted using certain refueling strategies to yield efficient approximations to vehicle routing. Given a vehicle routing instance G, define $T^*(G)$ to be the optimal length of a travelling salesman solution on the graph G, and let $V^*(G)$ denote the optimal length of a vehicle routing solution. The bounds below are established for the *split* version of the vehicle routing problem; since any solution to the unsplit problem is also a solution to the split version, the split optimal value is at least as optimal as the unsplit optimal value.

Proposition 1. $T^*(G) \leq V^*(G)$

Proof. The optimal solution of the vehicle routing is essentially just a feasible solution to the travelling salesman problem, because we can concatenate cycles to obtain a tour. □

Proposition 2. $\frac{2}{C}\sum_v D_v r_v \leq V^*$

Proof. We first note that for any vertex v on the tour T_i, the tour T_i consists of two paths from the depot to v, and hence the tour is longer than twice the distance from the depot to v. Thus $T^*(T_i) \geq 2\max_{v\in T_i} r_v$. What's more, the values D_v^i, once normalized by $\sum D_v^i$, constitute a probability distribution over the vertices in tour i, and employing the general fact that $\max_{v\in T_i} r_v \geq \mathbf{E}[r_X]$ for any random variable X distributed over the vertices in tour i, we conclude that

$$T^*(T_i) \geq 2\max_{v\in V_i} r_v \geq 2\frac{\sum_v D_v^i r_v}{\sum_v D_v^i} \geq 2\frac{\sum_v D_v^i r_v}{C}$$

where we have used the fact that $\sum_v D_v^i \leq C$. But now $V^*(G) = \sum T^*(T_i)$, and so we therefore conclude that

$$V^*(G) = \sum_i T^*(T_i) \geq \sum_i 2\frac{\sum_v D_v^i r_v}{C} = \frac{2}{C}\sum_v \left(\sum_i D_v^i\right) r_v = \frac{2}{C}\sum_v D_v r_v$$

This essentially says that the average radius of demands, weighted by the demands at each node with respect to the capacity, lower bounds the distance a routing problem must travel. □

This bound shows that, roughly, the expected radii of a random vertex in the graph, weighted by demand, lower bounds an optimum vehicle routing solution. The result implies that if we take a particular travelling salesman solution, and we are able to randomly refuel at a rate weighted by demand, then the expected resulting distance added by refueling along the cycle is bounded by the optimal vehicle routing solution. For brevity, we let $\text{LB} = \frac{2}{C}\sum_v D_v r_v$ stand for the left hand side of the inequality above. It is the canonical bound used to obtain approximations to the vehicle routing problem. However, one disadvantage with how this bound is often employed is that it isolates the *refill cost* of a particular solution to the vehicle routing problem, while discarding the *routing cost* of travelling between different nodes. This is why the bound works particularly well with vehicle routing approximations augmenting travelling salesman solutions, but is not sufficient to obtain better approximations in other scenarios.

2 Random Refill Approximations

As we mentioned, the bound above shows that an effective random refill policy results in a bounded approximation to the vehicle routing problem. We detail one such algorithm for the split vehicle routing problem below, assuming the existence of a polynomial time $1+\alpha$ approximation to the travelling salesman problem.

Algorithm 1 Augmenting a TSP to Obtain a Routing Approximation.

1: Compute an $1+\alpha$ optimal travelling salesman tour of V. Write this tour as a sequence $Q = \{v_1, \ldots, v_n\}$.
2: Uniformly randomly pick the starting supply $C' = C_0 \in [0, C]$.
3: Start the vehicle route at B.
4: **while** Q is non-empty **do**
5: Pick a new vertex v from the beginning of Q.
6: **if** $D_v \leq C'$ **then** supply v completely.
7: **else if** $D_v > C'$ **then**
8: Supply v with C' units.
9: Return to the depot and fill the vehicle up completely.
10: Return to v and completely supply v.
11: Return to B.

In the above algorithm, we say a vertex is a *breakpoint* if $D_v > C'$ when we visit the vertex, so we must return to the depot to refuel. The expected distance travelled by the algorithm above is then bounded by

$$(1+\alpha)T^*(G) + 2\sum_v \mathbf{P}(v \text{ is a breakpoint}) r_v$$

The event that the node v_i is a breakpoint is the same as the event that there is some integer n with $\sum_{j<i} D_{v_j} \leq C_0 + nC$, and $D_{v_i} + \sum_{j<i} d_{v_j} > C_0 + nC$, and we see that this occurs when $0 \leq C_0 < D_{v_i}$, so the probability that v_i is a breakpoint is D_{v_i}/C. This implies the expected distance of the route the algorithm returns is

$$(1+\alpha)T^*(G) + \frac{2}{C}\sum_v D_v r_v = (1+\alpha)T^*(G) + \text{LB} \leq (2+\alpha)V^*(G)$$

The current standard approximation algorithm for the travelling salesman problem is a $3/2$ approximation algorithm. Thus the random refill policy gives a $7/2$ approximation algorithm for the vehicle routing problem. If we work over graphs in Euclidean space, or more generally, we work over graphs with a bounded genus, then we have a polynomial time approximation scheme giving $1+\varepsilon$ approximation algorithms for each value of ε. This implies that there are $2+\varepsilon$ approximation algorithms for vehicle routing on graphs of bounded genus.

The random refill technique is certainly unnatural, especially when compared to just refilling to the maximum value each time. However, the random refill technique is useful for avoiding being 'stuck' in certain inefficient scenarios, and the technique is provably better: Refilling to maximum does result in an approximation algorithm with a constant approximation guarantee, but there are examples of graphs in which the full refuel strategy is unable to reach the $2+\alpha$ constant guaranteed by the random refill value (The constant value is left as an exercise in Gupta's *Approximation Algorithms for VRP with Stochastic Demands*, which I have yet to do).

3 Simulating Split Routing in Unsplit Scenarios

We can obtain an approximation algorithm for the unsplit version of vehicle routing by simulating the TSP approximation for the split version of the problem. We proceed as in the split case, until we end up with a vertex v with $D_v > C'$. In this case, we return to the depot, fill up our supply to D_v, return to v, supply v completely, then return back to the depot, fill up to $C-(D_v-C')$ capacity, and continue with the TSP cycle. The reason for this is that the algorithm will then follow the route provided by the split version of the algorithm, except that whenever the original algorithm goes back and forth from the depot, the split version of the algorithm goes back and forth twice. The expected length of the route will be

$$(1+\alpha)T^*(G) + (4/C)\sum_v d_v r_v = (1+\alpha)T^*(G) + 2\text{LB}$$

which is bounded by $3+\alpha$ times the length of the optimal solution to the *split* solution to the algorithm, and the split solution will always be shorter than the optimal solution to the unsplit algorithm[1].

[1]It might be a profitable question to ask when there is a measurable difference between the performance of split and unsplit optimal solutions on graphs, in which case we can bound the

4 NEW: Shortcutting in Unsplit Routing

The refill approximations to unsplit vehicle routing break down their approximations into two factors, the *refill cost*, which is obtained by returning to the vehicle depot, and the *TSP cost*. One idea to improving these approximation algorithms is to optimize the TSP route by skipping certain parts of the TSP journey when we travel to the depot to refuel. In the split case, we are unlikely to decrease the TSP cost without coming up with a completely new algorithm for VRP, because with random refills there is a zero percent chance of having to refuel from a vertex which is completely satisfied, so we must travel between the TSP tour completely to refuel all vertices. However, in the unsplit algorithm we are able to *shortcut* the parts of the TSP, because, when we travel back to the depot to refuel a second time, we have completely satisfied the current vertex, and so we have no need to return to it, so we may proceed directly to the next vertex, shortcutting the TSP path.

Example. *Consider a routing problem with uniform demands over vertices, equal to C/k for some integer k. If we follow a tour labelled with elements of $\mathbf{Z}_n$, then the probability that we skip an edge $(i\ i+1)$ is the probability that we begin a refill action at i or $i+1$. The event that we refill at i is disjoint from the event that we refill at $i+1$, because once we refill at i we will start with at least $1-1/k$ capacity at the next vertex. Thus we skip an edge $(i\ i+1)$ with probability $2/k$. The event that we refuel at i is $1/k$, and then we travel twice along the radius r_i, and once each along each radius r_{i-1} and r_{i+1}. Thus the expected distance travelled is*

$$(1-2/k)T^*(G) + \sum 1/k(r_{i-1} + 2r_i + r_{i+1}) = (1-2/k)T^*(G) + 2LB$$

Thus we obtain a $(1-2/k)(1+\alpha)+2 = (3-2/k)+(1-2/k)\alpha$ approximation algorithm. For $k=2$, we obtain a 2 approximation algorithm, and for $k=3$, we obtain a $2+1/3+\alpha/3$ approximation. However, these cases are solvable exactly in polynomial time (think of matching demand vertices), and therefore the approximations are just toy values. For $k=4$, we obtain a $2.5+\alpha/2$ approximation.

When the demands of vertices are high, the refuel cost of the algorithm should vastly outweigh the TSP cost of the approximation, and it is probable that shortcutting will yield similar approximation results. That is, we hope to obtain similar results assuming only that the demands of vertices are lower bounded by C/k. Then the event that we refill at two adjacent vertices i and $i+1$ no longer occurs with a non zero probability. If $D_i + D_{i+1} > C$, then the probability that we refill at both vertices is $(D_i+D_{i+1})/C-1$, so the probability

unsplit optimal algorithm performance even better, as well as understand what features of a graph limit the capabilities of the unsplit graph algorithm to perform well. I have yet to see an analysis of unsplit vehicle routing which does not just bound the output of an algorithm by the split demand cost.

that we skip the edge $i(i+1)$ is equal to

$$\frac{D_i + D_{i+1}}{C} - \max\left(0, \frac{D_i + D_{i+1}}{C} - 1\right) = \min\left(1, \frac{D_i + D_{i+1}}{C}\right)$$

This means that if $D_i + D_{i+1} > C$, then we skip an edge with probability one. Overall, we find the length of the route is

$$\left(1 - \sum \min\left(1, \frac{D_i + D_{i+1}}{C}\right)\right) T^*(G) + \frac{1}{C} \sum \left(D_{i-1} + 2D_i + D_{i+1}\right) r_i$$
$$\leq \left(1 - \sum \min\left(1, \frac{D_i + D_{i+1}}{C}\right)\right) T^*(G) + \mathrm{LB} + \frac{1}{C} \sum D_{i-1} r_i + \frac{1}{C} \sum D_{i+1} r_i$$

Unfortunately, we have yet to find a feasible way to bound the sums $\sum D_{i-1} r_i$ and $\sum D_{i+1} r_i$ in terms of the cost of the optimal VRP solution, because it is difficult to compare distances and weights when weighed against different vertices.

TODO: Describe method of directed arrows to strategize shortcutting the path, and how we haven't been able to succeed with this approach.

5 Reserving Refills in Small Demand Unsplit Vehicle Routing

TODO: Talk about reserve tank to reduce unsplit refill distances to two laps rather than 4.

Part II
Prize Collecting Vehicle Routing

In most combinatorial optimization problems, a feasible solution must accomplish the entire task thoroughly. In a feasible solution to the travelling salesman problem, we must visit *all* the vertices of the graph. But in a real life application of the travelling salesman problem, we might reject possible sales opportunities that are outlandishly far from our start point, unless these opportunities are significantly valuable. *Prize collection* versions of combinatorial optimization problems place values on subtasks of a particular problem, and ask to find an optimal solution, given that we need only solve a certain number of the subtasks of a problem. In a similar vein, *K-subtask* problems ask us to find the optimal solution to a task, given that we need only complete K subtasks. Certain techniques have been developed to reduce the K-subtask version of the travelling salesman problem to the prize collecting version. Good approximations for the prize collecting version of the vehicle routing problem have been developed, and one project we plan to work on this summer is to generalize the K-subtask prize

collecting reduction used in travelling salesman problem to yield a good approximation to the K-subtask vehicle routing problem. We begin by detailing the prize collecting travelling salesman problem reduction which is likely to yield good results in the vehicle routing arena.

6 K Minimal Spanning Tree

In the K minimal spanning tree problem, we are given a root node r, where we have some cost function d on the edges. The goal is to find a minimal edge set in the graph which connects the root to at least K other edges. We denote the optimum value of this problem by OPT_K. We begin by describing a basic linear program reduction. Let $X_e \in \{0,1\}$ be a variable denoting whether we choose an edge e, and $Z_v \in \{0,1\}$ describe whether v is connected to the root note. We can obtain an easy linear program relaxation by letting X_e denote choosing the edge e, and Z_v denote not connecting the vertex v to the root. The linear program is then to minimize $\sum d_e X_e$, such that for each vertex v, and for each subset S of edges containing v but not containing r, $\sum_{e \in \delta(S)} X_e + Z_v \geq 1$, and also $\sum Z_v \leq n - k$. Unfortunately, this approximation algorithm has a poor integrality gap.

Example. *Consider a graph G consisting of vertices $\{r, v_1, \ldots, v_n\}$, with an edge (r, v_1) that costs n, and $n-1$ edges (v_1, v_i) each with cost 1. Then for $K = 2$, $OPT_K = n+1$, yet for fractional solutions we can set $Z = 1 - 2/n$ and $X = 2/n$, which is feasible, and has value $n(2/n) + (n-1)(2/n) \leq 4$.*

Nonetheless, we can still apply some tricks on the LP to obtain a constant factor approximation, by 'Lagrangifying' the cardinality constraint. Rather than forcing K vertices to be connected to the root, we will assign a penalty for each vertex not connected to the root. As we increase the penalty, the optimum will connect more and more vertices. If we can find the 'sweet spot' where the optimum to this problem has exactly K vertices, then we can actually substitute this answer in the original problem and obtain a good approximation.

For each λ, we consider the linear program K-MST(λ) which attempts to minimize $\sum d_e X_e + \lambda(\sum Z_v - (n-k))$, subject only to the subset constraint that $Z_v + \sum_{e \in \delta(S)} X_e \geq 1$ for any S with $r \notin S$, $v \in S$. We let the optimal value to this problem be $\text{OPT}_K(\lambda)$. It shall also be helpful to consider the problem PCST(λ), which has the same constraints but attempts to optimize $\sum d_e X_e + \lambda \sum Z_v$, with optimal value OPT_{ST}. The optimal solution for K-MST(λ) and PCST(λ) will always be the same, but the optimal solution values will be different, and as such they may have different integrality gaps. The reason for this name is that this problem is a linear relaxation of the *prize collecting Steiner tree problem*, which asks to find the minimal Steiner tree, subject to a uniform penalty λ for each terminal vertex we don't include in a solution.

Lemma 1. *For any $\lambda \geq 0$, $OPT_K(\lambda) \leq OPT_K$.*

Proof. The optimum solution to the K MST problem is certainly feasible to the linear program, and the cost of this solution is equal to

$$\sum d_e X_e + \lambda(\sum Z_v - (n-k)) = \text{OPT} + \lambda(\sum Z_v - (n-k)) \leq \text{OPT}_K$$

because we know $\sum Z_v \leq n - k$ in the optimal solution. In particular, this implies that the value of the optimal solution to K-MST(λ) is upper bounded by OPT_K. □

Lemma 2. *For any $\lambda \geq 0$, $OPT_K(\lambda) + \lambda(n - K) = OPT_{ST}(\lambda)$.*

Proof. Obvious. □

To analyze the K-MST problem, we will assume a result we will later prove, which is slightly stronger than saying that PCST(λ) linear programs have an integrality gap of 2.

Fact 1. *For any $\lambda \geq 0$, there is a polynomial time algorithm to find a set of edges E such that if V is the set of all the vertices not connected to the root by edges in E, then $d(E) + 2\lambda|V| \leq 2OPT_{ST}(\lambda)$.*

Such an algorithm is known as a *Lagrangian multiplier preserving approximation*. It is a stronger result that just the existence of a 2-approximation for the prize collecting Steiner tree, because we double the cost of missed vertices while still lower bounding the optimal result. This will be important to our approximation of K-MST.

Using the integrality gap, we present a $5 + \varepsilon$ algorithm with running time polynomial in the number of vertices in G and $\log(1/\varepsilon)$. As a preprocessing step, we guess the furthest node from the root spanned by the optimum solution, and discard all nodes further than this. Thus we may assume we know the radius of the optimum solution. In particular, we may assume that $d(r, v) \leq \text{OPT}$ for all vertices v. It is easy to check if there is a 0 cost solution to the problem, so we also assume $\text{OPT} > 0$. Let $\delta \leq \text{OPT}$ be the minimum non-zero distance $d(r, v)$ over all vertices $v \in V$.

If we invoke the algorithmic result with $\lambda = 0$, then we will obtain $E = \emptyset$. If $2\lambda > 2|\text{MST}(G)| + 2\lambda(n - k)$, i.e. if $\lambda \geq \text{OPT}_{MST}$, then the optimal solution to $\text{OPT}_{ST}(\lambda)$ is just a minimal spanning tree, because the penalty of missing a vertex is too much, and the algorithm will just return a minimal spanning tree. As we vary λ over $[0, \text{OPT}_{MST}]$, the number of vertices will vary between 0 and n. We would hope this varies continuously, so we can find a λ which misses exactly K vertices, but this is not always possible.

Example. *Consider the graph used to disprove the integrality gap for the basic linear relaxation of the K MST problem, now viewed as an instance of the prize collecting steiner tree problem. With some basic computation, we find that if if $\lambda \leq n$, an optimal solution will cover no vertices apart from the root, and for $\lambda \geq n$, the optimal solutions will be the minimal spanning tree covering all nodes. Thus we see a rapid transition in the optimal solutions to the graph,*

so we cannot expect the algorithm approximating this problem to have a smooth transition between vertices.

However, we can perform a binary search on the value of $\lambda \in [0, \mathrm{OPT}_{MST}]$ to find two values λ_1 and λ_2 which are incredibly close together, such that the algorithm for λ_1 returns a solution with slightly less than K vertices covered, and on the input λ_2 the algorithm returns a solution with slightly more than K vertices covered. More precisely, we binary search using λ_1 and λ_2 as lower and upper bounds until $\lambda_1 \leq \lambda_2 \leq \lambda_1 + \varepsilon\delta/4n$, such that if the solution on input λ_1 covers vertices V_1, and the solution for λ_2 is V_2, then $|V_1| \geq n - k \geq |V_2|$.

If V_1 contains exactly $n - K$ vertices, then if E_1 are the respective edges returned by the algorithm, we find that

$$d(E_1)+2\lambda_1|V_1| = d(E_1)+2\lambda_1(n-K) \leq 2\mathrm{OPT}_{ST}(\lambda_1) = 2\mathrm{OPT}_K(\lambda_1)+2\lambda_1(n-K)$$

It follows that $d(E_1) \leq 2\mathrm{OPT}_K(\lambda_1) \leq 2\mathrm{OPT}$, so E_1 gives a 2-approximation solution. The same result holds for E_2 if V_2 contains exactly n_K vertices.

Given our construction of an approximation algorithm, we can now assume the strict inequality $|V_1| > n - k > |V_2|$. This means that the values

$$\alpha_1 = \frac{(n-K) - |V_2|}{|V_1| - |V_2|} \qquad \alpha_2 = \frac{|V_1| - (n-K)}{|V_1| - |V_2|}$$

which measure the relative closeness to an exact K MST, are well defined. Note that these values are both positive, and $\alpha_1 + \alpha_2 = 1$. Thus we may interpret these values as a probability distribution over the two solutions E_1 and E_2. Since $\alpha_1|V_1| + \alpha_2|V_2| = n - K$, 'on average' these solutions are exactly a K MST.

Lemma 3. $\alpha_1 d(E_1) + \alpha_2 d(E_2) \leq (2 + \varepsilon/2)\,OPT$.

Proof. Note that $\mathrm{OPT}_{ST}(\lambda_1) \leq \mathrm{OPT}_{ST}(\lambda_2)$, because the two problems have the exact same feasible solutions, except that the cost function of the second problem is always at least as large as the first. Next, note that

$$(\lambda_2 - \lambda_1)\alpha_2|V_2| \leq \frac{\varepsilon\delta}{4n}\alpha_2|V_2| \leq \frac{\varepsilon\mathrm{OPT}}{4n}\alpha_2|V_2| \leq (\varepsilon/4)\mathrm{OPT}$$

It follows that

$$\begin{aligned}
\alpha_1 d(E_1) + \alpha_2 d(E_2) &\leq 2[\alpha_1\mathrm{OPT}_{ST}(\lambda_1) - \lambda_1\alpha_1|V_1| + \alpha_2\mathrm{OPT}_{ST}(\lambda_2) - \lambda_2\alpha_2|V_2|] \\
&\leq 2[(\alpha_1 + \alpha_2)\mathrm{OPT}_{ST}(\lambda_2) - \lambda_1\alpha_1|V_1| - \lambda_2\alpha_2|V_2|] \\
&\leq 2[\mathrm{OPT}_{ST}(\lambda_2) - \lambda_2(\alpha_1|V_1| + \alpha_2|V_2|) + (\lambda_2 - \lambda_1)\alpha_1|V_1|] \\
&\leq 2[\mathrm{OPT}_{ST}(\lambda_2) - \lambda_2(n-K) + (\lambda_2 - \lambda_1)\alpha_1|V_1|] \\
&\leq 2\mathrm{OPT} + 2(\lambda_2 - \lambda_1)\alpha_1|V_1| \\
&\leq (2 + \varepsilon/2)\mathrm{OPT}
\end{aligned}$$

and this completes the proof. □

E_2 will always be a feasible solution to the K-MST problem. If $\alpha_2 \geq 1/2$, it is actually a good approximate solution, because the last lemma gives

$$d(E_2) \leq 2\alpha_2 d(E_2) \leq 2[(2+\varepsilon/2)\mathrm{OPT} - \alpha_1 d(E_1)] \leq (4+\varepsilon)\mathrm{OPT}$$

The only tricky case is where $\alpha_2 < 1/2$, in which case E_1 has a low cost, but is not a feasible solution to the problem. We shall find that we can *graft* part of the solution of E_2 onto E_1 to obtain a feasible solution, while only increasing the approximation ratio by OPT. This yields a $5+\varepsilon$ approximation for any input to the problem. First, notice that $|V_1 - V_2|$ contains at least $|V_1| - |V_2|$ nodes, and also $|V_2| \leq n - K$, so it makes sense to double the edges E_1, and to find the cheapest cycle C in E_1 which covers $|V_1| - (n-k)$ vertices in $V_1 - V_2$. We can view C as a cycle of these vertices in $V_1 - V_2$, by passing over vertices in V_2. By taking the cheapest subpath of length $|V_1| - (n-K)$ in this graph, we obtain a set of edges we can graft onto E_1, and after adding the edges to $E_1 \cup P$ forming the shortest path from r to P, so the graph is connected, we obtain a feasible solution. We know that the cost added by connecting r to P is bounded by OPT, because we have discarded all vertices of radius greater than OPT in a preprocessing step. Since C is a subset of the doubling of E_2, we obtain that $d(C) \leq d(E_2)$. What's more, since P is the cheapest subpath of the collapsing of C containing $|V_1| - (n-k)$ edges, and C contains $|V_1| - |V_2|$ edges, we find that

$$d(P) \leq \frac{|V_1| - (n-K)}{|V_1| - |V_2|} d(C) \leq 2\alpha_2 d(E_2)$$

and therefore

$$d(E_1) + 2\alpha_2 d(E_2) + \mathrm{OPT} \leq 2\alpha_1 d(E_1) + 2\alpha_2 d(E_2) + \mathrm{OPT} \leq (4+\varepsilon)\mathrm{OPT} + \mathrm{OPT}$$

this completes the analysis of the algorithm.

7 Prize Collecting Steiner Tree

In order to obtain the integrality bound algorithm above, we have to look into the prize collecting Steiner tree problem in more detail. In general, the prize collecting Steiner tree problem assigns penalties π to each vertex not covered by the Steiner tree, in addition to the cost function c on edges. The problem then tries to minimize $c(E) + \pi(V)$, where E is a tree from a fixed root r, and V are the vertices not covered by E.

We will consider a nontrivial linear relaxation to obtain the required construction for the K-MST problem. We add the variables X_e for each edge, but also variables Z_S for each nonempty set S of vertices not containing r, and try to minimize $\sum c(e)X_e + \sum \pi(S)Z_S$, such that for each subset S of vertices, $\sum_{e \in \delta(S)} X_e + \sum_{S \subset R} Z_R \geq 1$, with $X, Z \geq 0$. This program has exponentially many constraints and variables, but we can still form a primal dual algorithm which can be solved in polynomial time; the dual asks to maximize $\sum y_S$, where y is a vector indexed by nonempty sets of vertices, such that $\sum_{e \in \delta(S)} y_S \leq c_e$

for each edge e, and $\sum_{R \subseteq S} y_R \leq \pi(S)$ for each nonempty set S. We denote this linear program by PCST, and the optimal value of this linear program by OPT_{PCST}.

Lemma 4. *If $\pi_v = \lambda \geq 0$ for each vertex v, then $OPT_{ST}(\lambda) = OPT_{PCST}$.*

Proof. Given an optimal solution (X^*, Z^*) to the PCST linear program we just defined, we construct an optimal solution to PCST(λ). We set $X_e = X^*_e$ and $Z_v = \sum_{v \in S} Z^*_S$. This gives a PCST($\lambda$) solution. On the other hand, given a PCST(λ) solution (X^*, Z^*), we can order the vertices $v_1, \ldots, v_n$ such that $Z^*_{v_1} \leq Z^*_{v_2} \leq \cdots \leq Z^*_{v_n}$, and we can then define $X_e = X^*_e$ and for $S_i = \{i, \ldots, n\}$, we can set $Z_{S_i} = Z^*_{v_i} - Z^*_{v_{i-1}}$, with $Z_{S_1} = Z^*_{v_1}$. It is clear that (X, Z) gives the same cost as (X^*, Z^*), and is a feasible solution. □

We now work on a polynomial time algorithm to find a feasible prize collecting Steiner tree instance (E, V) with $c(E) + 2\pi(V) \leq 2\mathrm{OPT}_{PCST}$. The idea is to grow 'moats' around the vertices in the graph until they connect to r. However, each moat has a certain growth limit, which if reached, allows us to discard the entire set of vertices in the moat from the tree. Once we have done this, we prune the solution, discarding vertices not connected to r and edges whose deletion produces a component which exceeded its growth limit in the course of the algorithm.

The general idea is to grow a forest while also constructing a dual solution y_S, modifying the terms y_T corresponding to the components of the forest. An *active component* will be a component whose dual constraint is slack. We begin by constructing a vector γ with $\gamma(v) = \pi_v$ for each vertex, and initializing the forest $F = \emptyset$, with components $\{v\}$. We set $y = 0$. While there are still active components in the graph, we take the active components $C_1, \ldots, C_m$, and increase y_{C_i} and decrease $\gamma(C_i)$ at the same rate simultaneously, until either

- $\gamma(C_i) = 0$ for some C_i, in which case C_i becomes *inactive*.
- The constraint corresponding to some edge e becomes tight. We then add e to the forest F, which identifies two components A and B. We set $\gamma(A \cup B) = \gamma(A) + \gamma(B)$, and then continue the algorithm.

Once there are no more active components, we begin the pruning process. We let C_r denote the component of the forest containing the root node, and let $F(C_r)$ denote its edges in the forest. For each edge $e \in F(C_r)$ such that the nonroot component C of $F - \{e\}$ was considered in the algorithm, and $\gamma(C) = 0$, we remove the edge e from the forest. Once this is done, we remove all the edges not connected to the root, and the remaining tree is our output.

Lemma 5. *For any two components C and C' considered in the algorithm, either C and C' are disjoint, C is a subset of C', or C' is a subset of C.*

Proof. From the construction of the components in the algorithm, we see that we successively form decompositions of the components, so the components considered are always decompositions of prior components. □

Now let F be the returned solution, where V is the set of vertices not connected to the root node. We set X to be the family of components C considered in the algorithm which aren't a subset of V, and we set Y to be the family of components C which are a subset of V.

Lemma 6. $\pi(V) = \sum_{C \in Y} y_C$.

Proof. We prove this as a loop algorithm of the construction, that for each component C of F (active or not), $\sum_{S \subset C} y_S + \gamma(C) = \pi(C)$. Initially this is obvious. Similarily, it holds when increasing and decreasing the y_{C_i} and $\gamma(C_i)$. Now we note that y_S is only nonzero if S was a component considered in the algorithm. In the last lemma, we showed this was a laminar family, which will become important when bridging components. If an edge e bridges two components A and B, then any $S \subset A \cup B$ with $y_S \neq 0$ must either be a subset of A or a subset of B. Thus

$$\sum_{S \subset A \cup B} y_S + \gamma(A \cup B) = \sum_{S \subset A} y_S + \sum_{S \subset B} y_S + \gamma(A) + \gamma(B) = \pi(A) + \pi(B) = \pi(A \cup B)$$

Thus the loop invariant holds throughout the algorithm. When our algorithm finishes, we have a sequence of inactive components $C_1, \ldots, C_m$, and we therefore know that $\sum_{S \subset C_i} y_S = \pi(C)$. But then the lemma is proved by summing over the components whose union is V. □

Lemma 7. $c(F) \leq 2 \sum_{S \in X} y_S$

Combining these two points, we find that

$$c(F) + 2\pi(V) \leq 2 \sum_{S \in X} y_S + 2 \sum_{C \in Y} y_S \leq 2 \sum y_S \leq 2\mathrm{OPT}_{PCST}$$

and this gives the inequality required in our analysis of K minimum spanning trees, while also giving us a 2 approximation for the prize collecting Steiner tree problem.

8 Prize Collecting Travelling Salesman

The analysis of the K travelling salesman and its relation to prize collecting travelling salesman is essentially the same as with the minimum spanning tree problem. The linear program for K travelling salesman is to minimize $\sum c_e X_e$ subject to the constraints that $\sum_{e \in \delta(S)} X_e + 2Z_v \geq 2$ for all sets S with $v \in S$, and $\sum_v Z_v \leq n - K$. The corresponding Lagragian problem is to minimize $\sum c_e X_e + \lambda(\sum Z_v - (n - K))$. We utilize the same notation as for MST, using OPT_K, $\mathrm{OPT}_K(\lambda)$.

9 Prize Collecting Vehicle Routing Problem

Consider a $1+\alpha$ approximation algorithm for K-MST. For instance, Arora's *A $2+\varepsilon$ approximation algorithm for the K-MST problem* introduces a polynomial time computable heuristic with $\alpha = 1+\varepsilon$. Given a particular instance of vehicle routing, define $d_{vw} = d(v,w) + \frac{d_v r_v + d_w r_w}{C} + d(v,w)$. This new metric now takes into account *both* the travelling salesman cost of an algorithm, and the refill cost of the algorithm. Indeed, a given travelling salesman tour using the metric gives rise to a vehicle routing solution whose expected cost using the random refill strategy. In fact, this metric should guarantee at least as good an approximation as any other vehicle routing strategy augmenting the travelling salesman solution, because all the other solutions are feasible in this solution! Thus we just need to explore which travelling salesman solutions are optimal with respect to this metric.

10 NEW: Zac Idea

Consider the following linear program for prize collecting vehicle routing with distances d and penalties π. We consider the original distance metric d, and consider a new *assymetric* metric $d'_{uv} = d_{uv} + 2d_v r_v/C$. Now double up edges in the original graph, so we have a complete digraph on the vertices, and consider minimizing $\sum d'_e X_e + \sum \pi_v Z_v$, such that $X(\delta^{\text{out}}(r)) \leq 1$ (SHOULD THIS BE GREATER THAN OR EQUAL TO ONE? IS IT EVEN NECESSARY TO INCLUDE IT?), $X(\delta^{\text{in}}(r)) = 0$, $X(\delta^{\text{in}}(v)) \geq X(\delta^{\text{out}}(v))$, for all $v \neq r$, and for any vertex v and subset S of vertices not containing r but containing v, $X(\delta^{\text{in}}(S)) + Z_v \geq 1$. The resulting output is a *preflow* X on the graph. Now we consider a result of Bang and Jensen.

Fact 2. *For any preflow X, for $v \neq r$ let λ_v be the minimum $r-v$ cut relative to X, and fix $K \in \mathbf{Q}$. Then we can obtain (SPANNING?) out-arborescenses $A_1, \ldots, A_n$ rooted at r, and positive weights $\lambda_1, \ldots, \lambda_n$ with*

$$\sum \lambda_i = K \qquad \sum_{e \in A_i} \lambda_i \leq X_e \qquad \sum_{v \in A_i} \lambda_i = \min(K, \lambda_v)$$

and these arborescences can be computed in time polynomial in $|V|$ and the bit complexity of K.

Part III
Metric Augmentation to Improve Vehicle Routing Approximations

11 A Lagrangian Preserving VRP Algorithm

Given a cost function c on the nodes of a vehicle routing problem, and taking two constants $\alpha, \beta > 0$ to be chosen later, define an assymetric cost function $\vec{c}(v,w) = c(v,w) + \alpha(2/C)D_w r_w$, giving us a bidirected metric structure on the graph which takes the refill cost of a TSP augmentation into consideration. In order to obtain a Lagrangian preserving approximation for vehicle routing, we design an LP with the aim of constructing prize collecting minimum spanning trees which can be turned into TSP tours, and then successively turned into prize collecting vehicle routing solution. We rely on a result of Bang-Jensen, which says that one can separate preflows on a graph into a random set of trees which have a high probability of containing vertices with a high $r - v$ flow, and whose probability of containing a particular edge is lower bounded by the flow itself.

Fact 3 (Bang-Jensen)**.** *Let x be a preflow on a directed graph. Then there exists a random r-rooted arboresence A such that $\mathbf{P}(e \in A) \leq x(e)$, and $\mathbf{P}(v \in A)$ is equal to the max $r - v$ flow with respect to x.*

The result implies that we can obtain spanning trees rooted at r by finding preflows with essentially the same required properties as required by the spanning tree. Since we intend to apply the random refill strategy to the tour obtained from doubling the spanning trees obtained from the preflows, we design an LP which discourages choosing vertices with a high refill cost.

Consider the LP for prize collecting unsplit vehicle routing[2] which minimizes $\sum \vec{d}(e)x_e + \beta \sum \pi_v z_v$, where $x \geq 0$ is a preflow on the graph, that is, a function on edges satisfying $x(\delta^{\text{in}})(v) \geq x(\delta^{\text{out}}(v))$ for all non-root vertices v. We also add the constraints that $x(\delta^{\text{in}}(S)) + z_v \geq 1$ and $x(\delta^{\text{out}}(r)) = 1$. The variable $z \geq 0$ is designed to designate the vertices we ignore in our prize collecting solution. Let (x^*, z^*) denote the optimal solution. Now for any subset $W \subset V$, the optimal VRP tour $T^*(W)$ satisfying all vertices in W is a feasible solution to this LP, and since the VRP tour has no need to return to any non-root vertex more than once, since we must completely the vertex in one go in the unsplit problem, we conclude that

$$\begin{aligned} c(x^*) + \alpha\frac{2}{C}\sum_{vw} x^*_{vw} D_w r_w + \beta\pi(v^*) &\leq c(T^*(W)) + \alpha \text{LB}(T^*(W)) + \beta\pi(W^c) \\ &\leq (1+\alpha)c(T^*(W)) + \beta\pi(W^c) \end{aligned}$$

[2]Note that this algorithm takes advantage of the unsplit vehicle routing optimum rather than the split vehicle routing optimum

where we have used the lower bound inequality $\mathrm{LB}(T^*(W)) \leq c(T^*(W))$ common to most vehicle routing lower analyses. Taking the inequality over all vertex sets W and optimal VRP solutions $T^*(W)$,

$$c(x^*) + \alpha \frac{2}{C} \sum_{vw} x^*_{vw} D_w r_w + \beta \pi(v^*) \leq \min_W c(T^*(W)) + \alpha \mathrm{LB}(T^*(W)) + \beta \pi(W^c)$$

Using the result of Bang-Jensen, we now find a random arboresence A. This arboresence has the following useful properties:

- Directly taking the promises of Bang-Jensen gives $\mathbf{E}(c(A)) \leq c(x^*)$.
- The collection of constraints $x(\delta^{\mathrm{in}}(S)) + z_v \geq 1$, for a fixed v, and letting S range over all subsets of V containing v, can be interpreted as saying that the max $r - v$ flow with respect to a feasible solution to the LP is lower bounded by $1 - z_v$. Thus $\mathbf{P}(v \in A) \geq 1 - z_v$.
- Applying a union bound gives $\mathbf{P}(v \in A) \leq \sum_{v \in e} \mathbf{P}(e \in A) \leq \sum_{v \in e} x^*(e)$.

Using the doubling and shortcutting technique, we can turn the random arboresence into a random tour covering the same vertices, using each edge at most twice. Furthermore, we can perform the random refill strategy on the tour to obtain a random vehicle routing solution, which, with respect to the prize collecting VRP problem, has expected cost upper bounded by

$$2c(A) + \frac{2}{C} \sum_v \mathbf{P}(v \in A) D_v r_v + \pi(T^c) \leq 2c(x^*) + \frac{2}{C} \sum_{vw} x^*_{wv} D_v r_v + \pi(z^*)$$

Choosing the constants α and β carefully, we can turn this into a 3 Lagrangian preserving approximation for prize collecting TSP. In particular, setting $\alpha = 1/2$ and $\beta = 3/2$, and doubling the initial inequality with the optimal VRP tours considered just after forming the LP, we obtain

$$2c(x^*) + \frac{2}{C} \sum x^*_{vw} D_w r_w + 3\pi(v^*) \leq 3 \min_W c(T^*(W)) + \pi(W^c)$$

and so to conclude, if we denote the random VRP solution by T, then we find that

$$\mathbf{E}[c(T) + 3\pi(T^c)] \leq 3 \min_W c(T^*(W)) + \pi(W^c) \leq 3\mathrm{OPT}$$

we will use this to form an approximation algorithm to the K VRP problem.

12 A K-VRP Approximation

To summarize, in terms of uniform Lagrangian penalties λ in place of the variable π, we can find a VRP solution (T, W) with $c(T) + 3\pi(W^c) \leq 3\mathrm{OPT}_{\mathrm{PCVRP}}(\lambda)$. As with the 'standard' technique of forming K VRPs, we find $\lambda_1 < \lambda_2$ infinitisimally close with corresponding tours T_1, T_2, with $|T_1| \leq K \leq |T_2|$. If

$|T_1| = K$ or $|T_2| = K$, we obtain a 3 approximation to VRP. Otherwise, we form the constants α_1 and α_2 like in the K MST algorithm. As with the K MST algorithm, $\alpha_1, \alpha_2 \geq 0$, and $\alpha_1 + \alpha_2 = 1$, $\alpha_1|T_1| + \alpha_2|T_2| = K$, and $\alpha_1 c(T_1) + \alpha_2 c(T_2) \leq (3+\varepsilon)\mathrm{OPT}$, where ε depends in some way on how close we choose the values λ_1 and λ_2. If $\alpha_2 \geq 1/t$, we can simply choose T_2 as our output as a $t(3+\varepsilon)$ approximation to the optimal solution. To obtain an alternative solution if α_2 is small, we have to find a grafting strategy to place part of T_2 onto T_1.