Approximation Algorithms for Prize-Collecting Steiner Tree

Sanjana Das

December 2024

§1 Introduction

In the Steiner tree problem, we're given a weighted graph and a subset of vertices designated as *terminals*; we want to find the minimum-weight tree that contains all terminals. This problem is NP-hard, so our goal is to get good approximation algorithms for it. There's a classic 2-approximation algorithm by [6]. Since then, the approximation factor has seen a number of improvements; most recently, in 2010 [3] obtained a $(\ln(4) + \varepsilon)$-approximation for arbitrary $\varepsilon > 0$, using a LP relaxation and randomized rounding.

The prize-collecting Steiner tree problem (PCST) is a generalization where instead of having some vertices we're required to include and others we don't care about, we have one special vertex (the *root*) which we're required to include, and every other vertex has a *penalty*; we can either include it or pay its penalty.

We can think of the ordinary Steiner tree problem as a special case of PCST where terminals have penalty ∞, and non-terminals have penalty 0. But the additional flexibility in PCST (compared to ordinary Steiner tree) can be useful for modelling real-life situations. For example, [5] studied PCST in relation to the real-life problem of access network design: we have a street map and various buildings, and we'd like to create a network — by laying cable across streets — that provides Internet to some (but not necessarily all) of the buildings. We can model this by PCST, where the penalty associated to each building corresponds to how profitable it would be to connect that building to the network.

In 1995, [4] found a 2-approximation algorithm for PCST. But unlike with the ordinary Steiner tree problem, for a long time this was the best we could do. Finally, [2] broke this barrier in 2011 by finding a 1.97-approximation; then in 2024, [1] substantially improved the approximation ratio to 1.79.

In this paper, we'll first explain the algorithm of [4]. We'll then explain the ideas of [2] that allow us to get some improvement over 2, and the ideas of [1] that allow us to get a more sizeable improvement.

To fix some notation, we'll use $G = (V, E)$ to denote the input graph, r to denote the root vertex, $w(e)$ to denote the weight of an edge e, and $\pi(v)$ to denote the penalty associated to a vertex v; all weights and penalties are nonnegative. For any algorithm A, we use $\mathcal{T}_{\mathsf{A}}$ to denote the tree it produces, $\mathsf{Incl}(\mathsf{A})$ to denote the set of *non-root* vertices that A includes in its tree, and $\mathsf{Excl}(\mathsf{A})$ to denote the set of vertices A excludes; so its total cost is

$$\operatorname{cost}(\mathsf{A}) = w(\mathcal{T}_{\mathsf{A}}) + \pi(\mathsf{Excl}(\mathsf{A}))$$

(where for a set of vertices $S \subseteq V$, we write $\pi(S)$ to mean $\sum_{v \in S} \pi(v)$).

There's also one definition that will be useful in the analyses: for a set $S \subseteq V$, we say S *cuts* an edge e if e has one endpoint in S and the other outside S; we define the *boundary* of S, denoted ∂S, as the set of edges S cuts. In other words, $\partial S = \{uv \in E \mid u \in S,\ v \notin S\}$.

§2 A 2-approximation for PCST

In this section, we'll describe the 2-approximation algorithm of [4].

§2.1 A starting point — the 2-approximation for Steiner tree

For motivation, we'll start by considering the classic 2-approximation for the ordinary Steiner tree problem. In this algorithm, we compute the shortest path between every pair of terminals, build a graph on just the terminals with these shortest path distances as edge weights, and take its minimum spanning tree.

It'll be useful to think of this algorithm as a dynamic process that looks at the original graph and in effect runs Kruskal's algorithm on the new graph: This process will gradually build up a forest F, which starts out empty and eventually becomes a tree. At any snapshot in time, we say a connected component of F is *active* if it contains a terminal, and *inactive* otherwise. Each active component S simultaneously 'paints' all the edges on its boundary at a constant rate (where we think of $w(e)$ as the length of the edge e — so if e is being painted by one component, then it'll take $w(e)$ time to be fully painted).

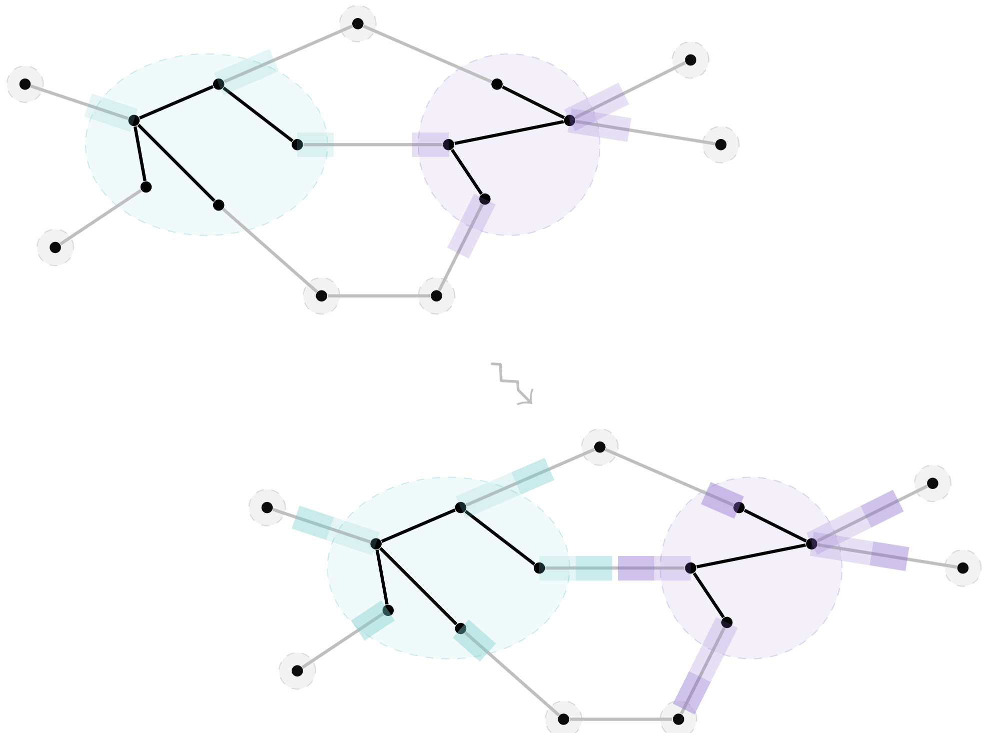

Figure 1: If we start with the situation on the top — where F consists of the black edges, the blue and purple sets represent active components, and the shading on the edges represents parts which have already been painted — then after a short time we'll get the situation on the bottom.

When an edge between two components (either active or inactive) becomes fully painted, we add it to F, merging these components. Note that F always remains a forest, since we only ever add edges between different components. We let this process run until F spans all terminals (meaning there's only one active component).

For the most part, this process directly implements Kruskal's minimum spanning tree algorithm — for any path between two terminals, the time it would take for the path to be fully painted is half its weight (the factor of half is because it's being painted from both ends), so we're adding the same paths between terminals that Kruskal's algorithm would. However, this process also might introduce a bunch of extraneous edges that aren't on paths between terminals. To fix this, we have a *pruning* step where we repeatedly remove all leaves which are non-terminals. After pruning, all the non-terminals left in the tree really are part of one of the shortest paths between terminals chosen by Kruskal's algorithm; so this accurately captures the classic 2-approximation algorithm.

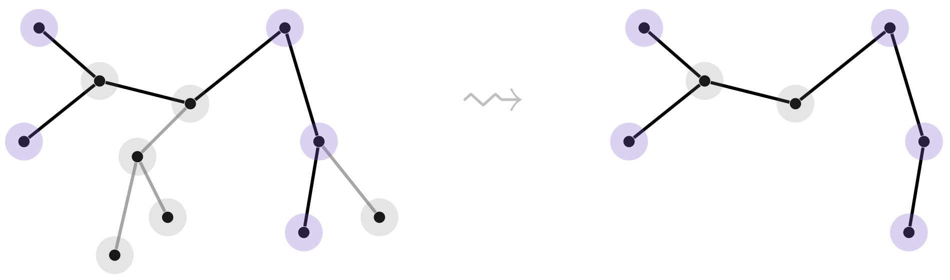

Figure 2: If we start with the tree on the left, with terminals highlighted in purple and non-terminals in gray, then pruning will result in the tree on the right.

§2.2 Incorporating the penalties

To turn this into an algorithm for PCST, we need a way to incorporate the penalties. The intuition is that the amount of time a set spends painting roughly corresponds to how much it costs us to include it in the tree. So if a set spends a long time painting compared to the penalties of its vertices, then it's better to give up on trying to include it, and just pay these penalties instead.

To formalize this, instead of thinking of painting as being done by the component themselves, we'll think of it as being done by a vertex in that set — so at any time, for each active component S, there will be a representative vertex $v \in S$ painting all the edges in ∂S. We'll say that the component containing the root r always has representative r. For the other components, it doesn't matter who their representative is; we can choose arbitrarily (among the vertices which are 'allowed' to be painting — we'll elaborate on what this means shortly).

We also initialize each vertex $v \neq r$ with a *painting potential* equal to its penalty $\pi(v)$, and r with painting potential ∞. This represents the total amount of time it's allowed to spend painting — only vertices which still have leftover potential are allowed to be the representative of their set. If an active component S runs out of painting potential, meaning that all its vertices have completely used up their potential, then we *inactivate* it, and it stops painting its boundary.

With this modification, the growth phase of the process works in nearly the same way as before — when an edge becomes fully painted, we add it to F and merge its two components. (We say the new component is active, since at least one of the two original components must have been active.) We start with F being the empty forest, and run until it becomes a tree spanning all vertices.

To describe the pruning phase, we say a set $S \subseteq V$ is a *dead set* if there's any time during the process at which it's a component that gets inactivated. Intuitively, dead sets are the generalization of non-terminals from before — they're things that we've given up on including in our tree. So in the pruning phase, we simply prune out all the dead sets that we can — if there's any dead set such that deleting its vertices from our tree wouldn't disconnect the tree, then we do so (repeatedly, until there are no such dead sets left). (See Figure 3 for an illustration.)

This provides an algorithm for PCST, called the Goemans–Williamson algorithm (which we'll denote by GW). We'll now show that this is a 2-approximation, meaning that $\text{cost}(\mathsf{GW}) \leq 2 \cdot \text{cost}(\mathsf{OPT})$.

§2.3 The analysis

For the analysis, we'll need a bit more notation: for every vertex $v \in V$, we use $t(v)$ to denote the total time it spends painting. For a set of vertices $S \subseteq V$, we write $t(S) = \sum_{v \in S} t(v)$. We say v is *dead* if it is in any dead set, and *alive* otherwise. We must have $t(v) \leq \pi(v)$ for all v, and equality holds for all dead vertices (since dead vertices must have used up all their painting potential).

We'll first upper-bound the cost of GW; we'll start by considering its tree weight.

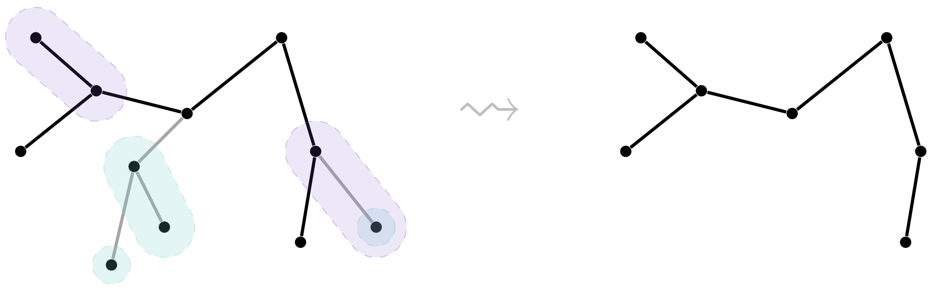

Figure 3: If we start with the tree on the left, with dead sets drawn in purple and blue, then we'll end up deleting the blue dead sets but not the purple ones (as shown on the right) — deleting the purple ones would disconnect the tree.

Lemma 2.1

We have $w(\mathcal{T}_{\mathsf{GW}}) \leq 2 \cdot t(\mathsf{Incl}(\mathsf{GW}))$ — in words, the weight of $\mathcal{T}_{\mathsf{GW}}$ is at most twice the total painting time of all non-root vertices that GW includes.

Proof. First, every edge $e \in \mathcal{T}_{\mathsf{GW}}$ receives $w(e)$ units of paint, since we only add edges to our forest F when they've been fully painted. This means the amount of paint placed on $\mathcal{T}_{\mathsf{GW}}$ is exactly $w(\mathcal{T}_{\mathsf{GW}})$; so in order to upper-bound $w(\mathcal{T}_{\mathsf{GW}})$, it suffices to upper-bound the total amount of paint that gets placed on $\mathcal{T}_{\mathsf{GW}}$. For this, we'll use the following claim.

Claim 2.2 — At every time during the process, the rate at which $\mathcal{T}_{\mathsf{GW}}$ is being painted is at most twice the number of vertices $v \in \mathsf{Incl}(\mathsf{GW})$ which are currently painting.

(When we say v is 'currently painting,' we mean it's currently the representative of its component. The 'rate' at which $\mathcal{T}_{\mathsf{GW}}$ is being painted is the number of vertex-edge pairs (v, e) with $e \in \mathcal{T}_{\mathsf{GW}}$ such that v is currently painting e.)

Proof. Let $\mathcal{A}$ and $\mathcal{I}$ be the set of active and inactive components which intersect $\mathcal{T}_{\mathsf{GW}}$, respectively (note that $\mathcal{T}_{\mathsf{GW}}$ is fixed — it's the final tree the process ends up with — but $\mathcal{A}$ and $\mathcal{I}$ depend on the moment in time we're considering). Then the edges of $\mathcal{T}_{\mathsf{GW}}$ form a tree structure on these components — this is because $\mathcal{T}_{\mathsf{GW}}$ is formed by adding more edges to our current forest, which links up its components in a tree structure, and then pruning, which doesn't affect this tree structure. (The pruning step might delete some of our current components entirely, but such components wouldn't be in $\mathcal{A}$ or $\mathcal{I}$.)

And for every active component S, the number of edges in $\mathcal{T}_{\mathsf{GW}}$ that it's painting is precisely its degree in this tree structure, which we denote by $\deg(S)$. This means the rate at which $\mathcal{T}_{\mathsf{GW}}$ is being painted is precisely $\sum_{S \in \mathcal{A}} \deg(S)$. (See Figure 4.)

Now we need to lower-bound the number of vertices $v \in \mathsf{Incl}(\mathsf{GW})$ which are currently painting. For this, we claim that for every component $S \in \mathcal{A}$ other than the one containing r, its representative is in $\mathsf{Incl}(\mathsf{GW})$. Assume this is not the case, i.e., for some $S \in \mathcal{A}$, its current representative v *doesn't* get included in $\mathcal{T}_{\mathsf{GW}}$. This means v must get pruned out during the pruning step, so at the end of the process, it must belong to a dead set S' that gets pruned out. But at the current moment in time, since v is currently painting, it hasn't died yet; and since components only expand as the process goes on, this means $S' \supseteq S$ (since S' is the set it's in when it dies, which occurs at some point after the current moment). So when S' gets pruned out of the tree at the end of the process, so does S. This is a contradiction, because we assumed that S intersects $\mathcal{T}_{\mathsf{GW}}$ (in the definition of $\mathcal{A}$).

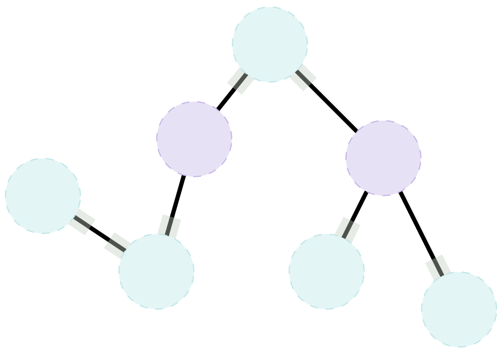

Figure 4: A possible snapshot in time, where the purple components form $\mathcal{I}$, the blue components form $\mathcal{A}$, and the edges of $\mathcal{T}_{\mathsf{GW}}$ between components are shown in black. Every blue component S is currently painting the black edges incident to it (as shown in green).

This means the number of vertices $v \in \mathsf{Incl}(\mathsf{GW})$ which are currently painting is $|\mathcal{A}| - 1$ (since we have one such vertex for every component in $\mathcal{A}$ except the one containing r). This means our goal is to show that

$$\sum_{S \in \mathcal{A}} \deg(S) \leq 2(|\mathcal{A}| - 1).$$

For this, first note that since we've got a tree structure on $\mathcal{A} \cup \mathcal{I}$ (which has $|\mathcal{A}| + |\mathcal{I}| - 1$ edges), we have

$$\sum_{S \in \mathcal{A} \cup \mathcal{I}} \deg(S) = 2(|\mathcal{A}| + |\mathcal{I}| - 1). \tag{2.1}$$

Furthermore, no $S \in \mathcal{I}$ can be a leaf in the tree structure — this is because any such S is a dead set, and if S is a leaf in the tree structure then we can delete it without disconnecting $\mathcal{T}_{\mathsf{GW}}$, so we would have pruned it out in the pruning step (at the end of the process), contradicting the assumption that it intersects $\mathcal{T}_{\mathsf{GW}}$.

This means $\deg(S) \geq 2$ for every $S \in \mathcal{I}$, so the total contribution of such sets to the left-hand side of (2.1) is at least $2\,|\mathcal{I}|$. Removing this contribution, we get that $\sum_{S \in \mathcal{A}} \deg(S) \leq 2(|\mathcal{A}| - 1)$, as desired. □

So we've shown that at any time, the rate at which $\mathcal{T}_{\mathsf{GW}}$ is being painted is at most twice the number of vertices $v \in \mathsf{Incl}(\mathsf{GW})$ which are currently painting. Accumulating this over all moments in time, we get that the total amount of paint that ends up on $\mathcal{T}_{\mathsf{GW}}$ is at most twice the total painting time of all vertices $v \in \mathsf{Incl}(\mathsf{GW})$. And since this total amount of paint is $w(\mathcal{T}_{\mathsf{GW}})$, we get $w(\mathcal{T}_{\mathsf{GW}}) \leq 2 \cdot t(\mathsf{Incl}(\mathsf{GW}))$. □

Lemma 2.1 deals with the tree weight of GW; now we'll deal with its penalty.

Claim 2.3 — We have $\pi(\mathsf{Excl}(\mathsf{GW})) = t(\mathsf{Excl}(\mathsf{GW}))$ — in words, the total penalty GW pays, for all the vertices it excludes, is exactly the total painting time of those vertices.

Proof. Every vertex that GW excludes is dead (since we only prune out dead sets), which means $\pi(v) = t(v)$ for all such vertices. □

Putting these together, we get

$$\mathrm{cost}(\mathsf{GW}) \leq 2 \cdot t(\mathsf{Incl}(\mathsf{GW})) + 1 \cdot t(\mathsf{Excl}(\mathsf{GW})) \leq 2 \cdot t(V \setminus \{r\}) \tag{2.2}$$

(where $\mathsf{Incl}(\mathsf{GW})$ accounts for the tree weight and $\mathsf{Excl}(\mathsf{GW})$ accounts for the penalty, and these two sets partition $V \setminus \{r\}$).

Now we'll try to prove a corresponding *lower* bound on OPT. Again, we'll start with its tree weight.

Lemma 2.4

We have $w(\mathcal{T}_{\mathsf{OPT}}) \geq t(\mathsf{Incl}(\mathsf{OPT}))$ — in words, the weight of $\mathcal{T}_{\mathsf{OPT}}$ is at *least* the total painting time of all non-root vertices that it includes.

Proof. First, we claim that whenever a vertex $v \in \mathsf{Incl}(\mathsf{OPT})$ is painting, it's painting at least one edge of $\mathcal{T}_{\mathsf{OPT}}$. To see this, suppose that v is currently the representative of some component S. Then S contains v but not r (if S contained r, then r would be its representative instead). And since both v and r are in $\mathcal{T}_{\mathsf{OPT}}$, this means S must cut at least one edge of $\mathcal{T}_{\mathsf{OPT}}$; then v is painting that edge.

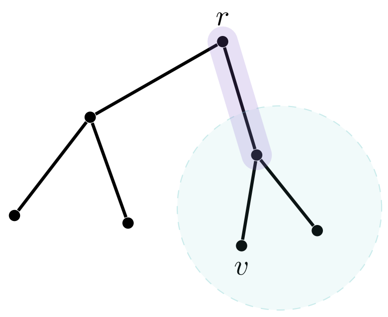

Figure 5: If $\mathcal{T}_{\mathsf{OPT}}$ is as shown in black and S is as shown in blue, then S cuts the edge of $\mathcal{T}_{\mathsf{OPT}}$ highlighted in purple, so v is painting it.

So the total amount of paint ending up on $\mathcal{T}_{\mathsf{OPT}}$ is at least the total amount of time vertices $v \in \mathsf{Incl}(\mathsf{OPT})$ spend painting, i.e., $t(\mathsf{Incl}(\mathsf{OPT}))$. But every edge is painted at most once, so this total amount of paint is at most $w(\mathcal{T}_{\mathsf{OPT}})$, giving the desired bound. $\square$

Next, we'll deal with OPT's penalty.

Claim 2.5 — We have $\pi(\mathsf{Excl}(\mathsf{OPT})) \geq t(\mathsf{Excl}(\mathsf{OPT}))$.

Proof. For *every* vertex v, we have $t(v) \leq \pi(v)$ by definition. $\square$

Putting these together, we get

$$\mathrm{cost}(\mathsf{OPT}) \geq t(\mathsf{Incl}(\mathsf{OPT})) + t(\mathsf{Excl}(\mathsf{OPT})) = t(V \setminus \{r\}) \tag{2.3}$$

(similarly to (2.2), vertices that get included account for the tree cost, and vertices that don't get included account for their own penalties). Combining this with (2.2), we get

$$\mathrm{cost}(\mathsf{GW}) \leq 2 \cdot t(V \setminus \{r\}) \leq 2 \cdot \mathrm{cost}(\mathsf{OPT}),$$

showing that GW is a 2-approximation.

§3 A first improvement

In this section, we'll explain the ideas of [2] used to break the barrier of 2. When describing the approach, we'll imagine that we're trying to end up with a $(2-\delta)$-approximation for *some* $\delta > 0$, and we'll compute what value of δ we end up with in the end.

§3.1 The high-penalty case

The first idea for how to improve over GW is that (2.2) is a bit lossy. There, we showed that $w(\mathcal{T}_{\mathsf{GW}}) \leq 2 \cdot t(\mathsf{Incl}(\mathsf{GW}))$ and $\pi(\mathsf{Excl}(\mathsf{GW})) = t(\mathsf{Excl}(\mathsf{GW}))$, and put these together to get $\operatorname{cost}(\mathsf{GW}) \leq 2 \cdot t(V \setminus \{r\})$. But we only need the factor of 2 in our bound on tree weights, not penalties; this means we can actually say

$$\operatorname{cost}(\mathsf{GW}) \leq 2 \cdot t(V \setminus \{r\}) - \pi(\mathsf{Excl}(\mathsf{GW})).$$

In particular, if $\pi(\mathsf{Excl}(\mathsf{GW}))$ is a substantial fraction of $\operatorname{cost}(\mathsf{GW})$ (in words, a substantial portion of our cost comes from penalties), then GW *already* gets an approximation ratio better than 2. So it suffices to figure out how to beat 2 in the case where only a tiny fraction of our cost comes from penalties.

Unfortunately, this assumption doesn't turn out to be particularly helpful. But a slight modification of it *does* — it turns out that it *is* helpful if we can make this assumption about OPT instead (i.e., assume that only a tiny fraction of OPT's cost comes from penalties), as we'll see in the following subsection.

In order to modify the above argument so that it lets us assume that OPT's penalty is small (rather than our own penalty), we'll define a version of GW with scaled potentials: for any parameter $\gamma > 0$, we define $\mathsf{GW}[\gamma]$ as the algorithm which works in the same way as GW, except that it initializes each vertex with painting potential $\gamma^{-1}\pi(v)$ rather than $\pi(v)$.

This scaling doesn't affect our analysis of tree weights — in particular, Lemmas 2.1 and 2.4 (regarding $w(\mathcal{T}_{\mathsf{GW}})$ and $w(\mathcal{T}_{\mathsf{OPT}})$, respectively) remain true as written. Meanwhile, when we're dealing with penalties, we now have $\pi(v) \geq \gamma \cdot t(v)$ for all vertices v, and equality holds for all dead vertices. This means Claim 2.3 now becomes

$$\pi(\mathsf{Excl}(\mathsf{GW}[2])) = \gamma \cdot t(\mathsf{Excl}(\mathsf{GW}[2])), \tag{3.1}$$

while Claim 2.5 becomes

$$\pi(\mathsf{Excl}(\mathsf{OPT})) \geq \gamma \cdot t(\mathsf{Excl}(\mathsf{OPT})). \tag{3.2}$$

Then it turns out that setting $\gamma = 2$ gives us a good approximation when OPT's penalty is a substantial fraction of its cost, as stated in the following lemma. (The point is essentially that setting $\gamma = 2$ fixes the loss in (2.2) that motivated this subsection, but in a more useful way than our first attempt.)

Lemma 3.1
If $\pi(\mathsf{Excl}(\mathsf{OPT})) \geq \delta \cdot \operatorname{cost}(\mathsf{OPT})$, then $\mathsf{GW}[2]$ achieves a $(2-\delta)$-approximation.

Proof. Combining Lemma 2.1 with (3.1) gives

$$\operatorname{cost}(\mathsf{GW}[2]) \leq 2 \cdot t(\mathsf{Incl}(\mathsf{GW}[2])) + 2 \cdot t(\mathsf{Excl}(\mathsf{GW}[2])) = 2t(V \setminus \{r\}).$$

We're roughly trying to compare this with $2 \cdot \operatorname{cost}(\mathsf{OPT})$. For this, we have

$$2 \cdot \operatorname{cost}(\mathsf{OPT}) = 2 \cdot w(\mathcal{T}_{\mathsf{OPT}}) + 2 \cdot \pi(\mathsf{OPT}).$$

If we apply Lemma 2.4 to replace $2 \cdot w(\mathcal{T}_{\mathsf{OPT}})$ with $2 \cdot t(\mathsf{Incl}(\mathsf{OPT}))$, and use (3.2) to replace one of the two $\pi(\mathsf{OPT})$ terms with $2 \cdot t(\mathsf{Excl}(\mathsf{OPT}))$ (but leave the second one as it is), we get

$$2 \cdot \operatorname{cost}(\mathsf{OPT}) \geq 2 \cdot t(V \setminus \{r\}) + \pi(\mathsf{Excl}(\mathsf{OPT})),$$

and combining this with our upper bound on $\operatorname{cost}(\mathsf{GW}[2])$ gives

$$\operatorname{cost}(\mathsf{GW}[2]) \leq 2 \cdot \operatorname{cost}(\mathsf{OPT}) - \pi(\mathsf{Excl}(\mathsf{OPT})) \leq (2-\delta) \cdot \operatorname{cost}(\mathsf{OPT}). \qquad \square$$

This means it suffices to figure out how to get a good approximatio in the case where OPT's penalty is at most a tiny fraction of its total cost. (When we're actually running the algorithm, we won't know whether we're in the high-penalty or low-penalty case, because we don't know OPT. But we don't need to — we'll simply run both $\mathsf{GW}[2]$ and the algorithm for the low-penalty case, and take whichever of the two outputs is better.)

§3.2 The low-penalty case

Now we suppose that OPT's penalty is a tiny fraction of its cost (specifically, $\pi(\mathsf{Excl}(\mathsf{OPT})) \leq \delta \cdot \operatorname{cost}(\mathsf{OPT})$ — we can assume δ is as small as we want, and this will dictate the improvement over 2 we end up with). Then $\mathsf{GW}[2]$ isn't guaranteed to get an approximation ratio substantially better than 2, so we need a different candidate solution.

The idea is to make use of the fact that we have better-than-2-approximations for the *ordinary* Steiner tree problem. We'll use such an approximation algorithm as a black box, and we'll let ρ denote its approximation ratio. This approach will work with any $\rho < 2$, with better values of ρ giving better bounds. The best ratio we currently have is essentially $\ln(4) \approx 1.39$, due to [3], so we'll set $\rho = 1.4$ in the final computations.

So we'd like to find a 'good' set of vertices to designate as terminals, give up on trying to include the non-terminals (and pay their penalties instead), and find a good Steiner tree on the terminals; we'll call this candidate solution ST.

For a set of terminals to be 'good,' we want it to satisfy two properties:

(1) The total penalty of all the non-terminals should be small.

(2) The weight of the optimal Steiner tree on the terminals isn't much more than $w(\mathcal{T}_{\mathsf{OPT}})$.

If we can attain these two goals, then we'll win. To see why, for concreteness, suppose that the total penalty of all non-terminals is at most $0.1 \cdot \operatorname{cost}(\mathsf{OPT})$, the weight of the optimal Steiner tree on the terminals is at most $1.1 \cdot w(\mathcal{T}_{\mathsf{OPT}})$, and we have a 1.4-approximation for ordinary Steiner tree. Then ST pays a penalty of at most $0.1 \cdot \operatorname{cost}(\mathsf{OPT})$. Meanwhile, its tree is at most a 1.4-factor off from the *optimal* Steiner tree on the terminals, which means

$$w(\mathcal{T}_{\mathsf{ST}}) \leq 1.4 \cdot 1.1 \cdot w(\mathcal{T}_{\mathsf{OPT}}) \leq 1.4 \cdot 1.1 \cdot \operatorname{cost}(\mathsf{OPT}).$$

Together, $\operatorname{cost}(\mathsf{ST}) \leq (1.4 \cdot 1.1 + 0.1) \cdot \operatorname{cost}(\mathsf{OPT})$, and we're happy because $1.4 \cdot 1.1 + 0.1 < 2$.

So it remains to figure out how to find a good set of terminals. For this, the idea is to run $\mathsf{GW}[\gamma]$ for a very *small* scaling parameter γ, and take its live and dead vertices as terminals and non-terminals. Why is this a reasonable thing to do? In $\mathsf{GW}[\gamma]$, we're giving each vertex painting potential $\gamma^{-1}\pi(v)$; if γ is small, then γ^{-1} is huge. And dead vertices run out of potential, so we'd expect their penalties $\pi(v)$ to be small; so this choice makes sense from the perspective of (1).

Now we'll show that this actually works. We'll use Live and Dead to denote the sets of live and dead vertices under $\mathsf{GW}[\gamma]$ (and $t(v)$ to denote the total painting time of v under $\mathsf{GW}[\gamma]$).

First we'll show that (1) holds — i.e., that the total penalty of dead vertices is small.

Lemma 3.2

We have $\pi(\mathsf{Dead}) \leq \gamma \cdot w(\mathcal{T}_{\mathsf{OPT}}) + \pi(\mathsf{Excl}(\mathsf{OPT}))$.

Note that we're working with the case where OPT's penalty is low (at most a δ-fraction of its cost), so the right-hand side is at most $(\gamma + \delta) \cdot \operatorname{cost}(\mathsf{OPT})$; so as long as γ and δ are both small, we do get (1).

Proof. Split the dead vertices into two sets based on whether OPT uses them or not — i.e., let

$$X = \mathsf{Dead} \cap \mathsf{Incl}(\mathsf{OPT}) \quad \text{and} \quad Y = \mathsf{Dead} \cap \mathsf{Excl}(\mathsf{OPT}),$$

so that $\pi(\mathsf{Dead}) = \pi(X) + \pi(Y)$.

For the latter, we have $\pi(Y) \leq \pi(\mathsf{Excl}(\mathsf{OPT}))$. For the former, we have $\pi(X) = \gamma \cdot t(X)$ (since all $v \in X$ are dead, so $\pi(v) = \gamma \cdot t(v)$). Meanwhile, Lemma 2.4 lower-bounds the weight of $\mathcal{T}_{\mathsf{OPT}}$ in terms of the painting times of the vertices OPT includes, giving

$$w(\mathcal{T}_{\mathsf{OPT}}) \geq t(\mathsf{Incl}(\mathsf{OPT})) \geq t(X).$$

Putting these together, we get $\pi(X) \leq \gamma \cdot w(\mathcal{T}_{\mathsf{OPT}})$, as desired. □

Next we'll show (2) — i.e., that the best Steiner tree on the live vertices isn't much worse than $\mathcal{T}_{\mathsf{OPT}}$.

Lemma 3.3

There is a tree $\mathcal{T}$ containing all vertices in Live such that

$$w(\mathcal{T}) \leq w(\mathcal{T}_{\mathsf{OPT}}) + 2\gamma^{-1} \cdot \pi(\mathsf{Excl}(\mathsf{OPT})).$$

Again, we're working with the case where OPT's penalty is low, so the second term is at most $2\gamma^{-1}\delta \cdot \operatorname{cost}(\mathsf{OPT})$, which we can make small by taking $\delta \ll \gamma$. So this does achieve (2).

Proof. We'll 'construct' $\mathcal{T}$ in the following way: imagine that we run $\mathsf{GW}[\gamma]$, except that instead of starting with the empty forest, we start with $\mathcal{T}_{\mathsf{OPT}}$.

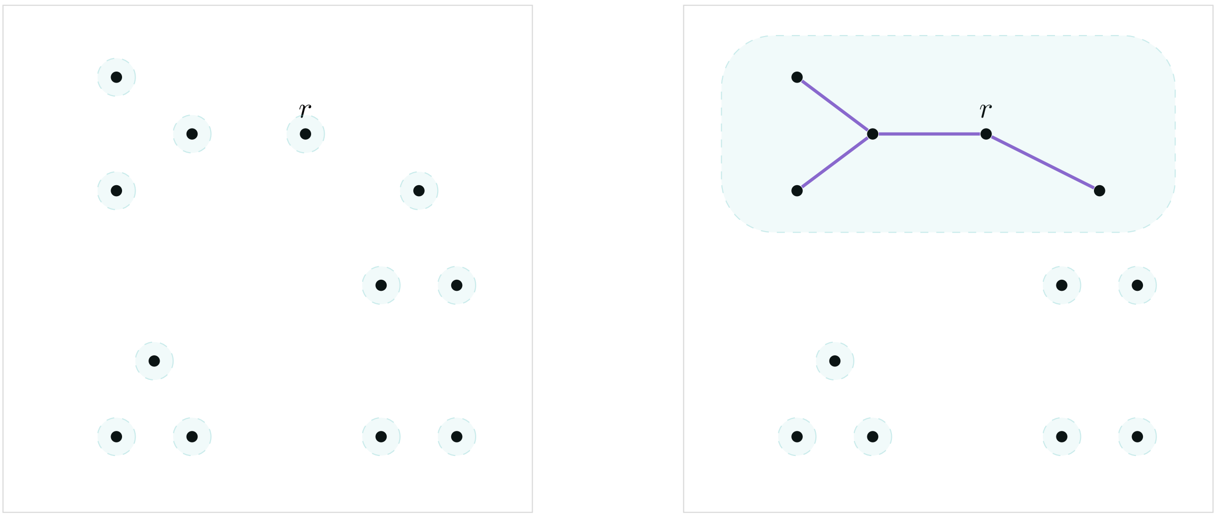

Figure 6: The original and modified processes at the beginning (on the left and right, respectively), with $\mathcal{T}_{\mathsf{OPT}}$ shown in purple.

More explicitly, at the beginning of the process, we have one component consisting of $\mathcal{T}_{\mathsf{OPT}}$ (this is the root's component), and every other vertex is in its own component. Other than this, the process works the same way as before — active components paint their boundaries, we add edges to our forest when they get fully painted and inactivate components when they run out of painting potential, and in the end we get a spanning tree, which we prune to obtain $\mathcal{T}$. We'll use $t'(v)$ to refer to the painting time of a vertex v in this modified process.

Then for every vertex v, its 'experience' in the modified process is the exact same as its experience in the original process until it comes into contact with the root's component; and this happens at least as early as

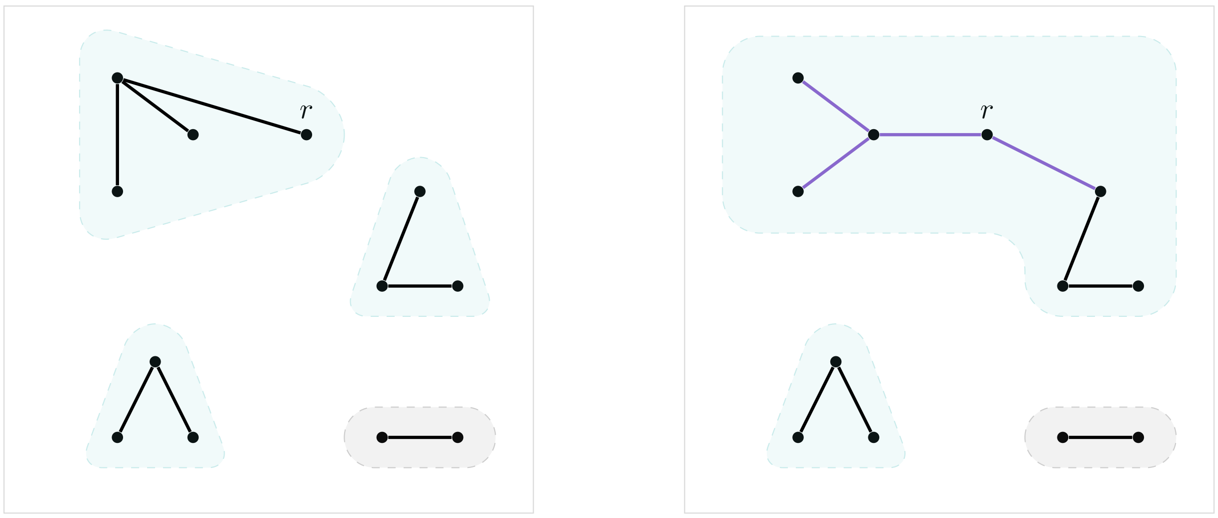

Figure 7: The original and modified processes after several steps, with $\mathcal{T}_{\mathsf{OPT}}$ in purple (on the right), the additional edges added to our forest in black, active components in blue, and inactive components in gray. All components on the right other than the one containing r are the same as their counterparts on the left.

it did in the original. (This is because the configuration at the beginning of the process is the same as the original except that we've enlarged the root's component, and this will remain true throughout the process.)

Also, once v comes into contact with the root's component, it stops painting (since r always represents its own component). In particular, if it hasn't died yet, then it never will. This means all live vertices for the original process are also live for the modified one. And since the final pruning step can only prune out vertices which are dead for the modified process, it won't prune out any vertices in Live (the vertices which were live for the original process); so $\mathcal{T}$ indeed contains all vertices in Live.

To bound $w(\mathcal{T})$, applying Lemma 2.1 (which upper-bounds the weight of the tree produced by GW, or more generally $\mathsf{GW}[\gamma]$) to the *modified* process gives that

$$w(\mathcal{T}) \leq w(\mathcal{T}_{\mathsf{OPT}}) + 2 \cdot t'(V \setminus \{r\}).$$

(The term of $w(\mathcal{T}_{\mathsf{OPT}})$ is because we're starting with $\mathcal{T}_{\mathsf{OPT}}$; the term of $2 \cdot t'(V \setminus \{r\})$ comes from bounding the weight of the additional edges that get added, in the same manner as Lemma 2.1. Lemma 2.1 actually lets us refine $V \setminus \{r\}$ to only the non-root vertices that end up in $\mathcal{T}$, but we don't need this refinement.)

Finally, we have $t'(v) \leq t(v)$ for all v (since v has the same experience until it comes into contact with the root's component, at which point it stops painting); and $t'(v) = 0$ for all $v \in \mathsf{Incl}(\mathsf{OPT})$ (since these vertices are in the root's component to begin with). Plugging this into $t'(V \setminus \{r\})$ in the above bound gives

$$w(\mathcal{T}) \leq w(\mathcal{T}_{\mathsf{OPT}}) + 2 \cdot t(\mathsf{Excl}(\mathsf{OPT})).$$

And since $t(v) \leq \gamma^{-1} \cdot \pi(v)$ for all v, we have $t(\mathsf{Excl}(\mathsf{OPT})) \leq \gamma^{-1} \cdot \pi(\mathsf{Excl}(\mathsf{OPT}))$, completing the proof. $\square$

So we've shown that setting Live and Dead as terminals and non-terminals satisfies the two properties (1) and (2) that we wanted — Dead has low penalties, and the best Steiner tree on Live isn't too much worse than $\mathcal{T}_{\mathsf{OPT}}$. This means for an appropriate choice of parameters, ST really is a better-than-2-approximation.

§3.3 Calculating the approximation ratio

To finish, we'll give a loose estimate on what value of δ we end up with (i.e., how much we can beat 2 by). First, the bound on penalties we get from Lemma 3.2 is

$$\pi(\mathsf{Excl}(\mathsf{ST})) \leq (\gamma + \delta) \cdot \mathrm{cost}(\mathsf{OPT}).$$

Meanwhile, the bound on tree weight we get from Lemma 3.3 is

$$w(\mathcal{T}_{\mathsf{ST}}) \le \rho(1 + 2\gamma^{-1}\delta) \cdot \mathrm{cost}(\mathsf{OPT})$$

(since the optimal Steiner tree on Live has weight at most $(1 + 2\gamma^{-1}\delta) \cdot \mathrm{cost}(\mathsf{OPT})$, and our approximation algorithm for Steiner tree is at most a factor of ρ off). Putting these together, we get a bound of

$$\mathrm{cost}(\mathsf{ST}) \le (\rho(1 + 2\gamma^{-1}\delta) + \gamma + \delta) \cdot \mathrm{cost}(\mathsf{OPT}),$$

so we successfully get a $(2 - \delta)$-approximation as long as

$$\rho(1 + 2\gamma^{-1}\delta) + \gamma + \delta \le 2 - \delta.$$

For this to hold, we want γ and $2\gamma^{-1}\delta$ to both be small. With $\rho = 1.4$, taking $\gamma = 0.1$ and $\delta = 0.01$ works; so this gives a 1.99-approximation. By doing computations more carefully, [2] shows this actually gives a 1.97-approximation. But the point is that we've broken the barrier of 2, but by a tiny amount.

§4 A more sizeable improvement

In this section, we'll explain the ideas of [1] that get a more substantial improvement over 2. As in the previous section, we'll imagine that we're trying to obtain a $(2 - \delta)$-approximation for some δ, and only compute δ in the end; but this time, we'll hope δ is less tiny than before. (They obtain a ratio of 1.79; we'll present a slightly modified version of their algorithm where the computations are more intuitive, which ends up with 1.84.)

§4.1 Motivation and overview

We're still going to use the high-level framework of [2] where we split into a high-penalty case (where OPT's cost is at least a δ-fraction of its penalty, in which case GW[2] is a good approximation) and a low-penalty case (where OPT's cost is at most a δ-fraction of its penalty), but this time, we want our low-penalty case to be able to handle less tiny values of δ.

For motivation, let's think about why the low-penalty case of [2] forced δ to be tiny. There, we got our solution by running GW[γ] to partition the vertices into Live and Dead, giving up on trying to include the vertices in Dead (instead paying their penalties), and finding a good Steiner tree on Live. We showed that:

(1) The penalty $\pi(\mathsf{Dead})$ is small, so paying it doesn't hurt us too much. This came from Lemma 3.2, and required γ to be small.

(2) The *optimal* Steiner tree on Live isn't much worse than $\mathcal{T}_{\mathsf{OPT}}$, so the one that our algorithm found isn't much worse than a ρ-factor off $\mathcal{T}_{\mathsf{OPT}}$. This came from Lemma 3.3, and required $\gamma^{-1}\delta$ to be small.

Combining these gave that our solution isn't much worse than a ρ-factor off OPT; in particular, it beats 2 fairly substantially. But the bottleneck was that for this to work, we need both γ and $\gamma^{-1}\delta$ to be *moderately* small, and that forces δ to be *really* small.

The idea of [1] is to give up on (1) — we'll use the same approach to get a candidate solution ST, but instead of taking γ to be small, we'll take $\gamma = 1$. In other words, we'll run GW with no scaling, let Live and Dead be the sets of live and dead vertices it produces, and obtain a candidate solution ST by running a ρ-approximation algorithm for ordinary Steiner tree on Live.

Then making (2) work — i.e., making sure the optimal Steiner tree on Live isn't much worse than $\mathcal{T}_{\mathsf{OPT}}$, so $w(\mathcal{T}_{\mathsf{ST}})$ isn't much worse than a ρ-factor off — only requires δ to be moderately small, which is great. And it's still true that ST is a good solution when $\pi(\mathsf{Dead})$ is small (we can no longer say that (1) is always true, but when it is true, ST is a good solution for the same reason as before). But we need new candidate solutions to handle the case where $\pi(\mathsf{Dead})$ is substantial.

The main idea of [1] is to imagine we pay the penalties of all the dead vertices, and consider the *reduced problem* where we set these penalties to 0. We run our algorithm recursively on this reduced problem, and this gives us a third candidate solution IT.

This might seem counterintuitive — if $\pi(\mathsf{Dead})$ is large, why would it be a good idea to pay it? But the main insight is that we might hope that moving from the original to the reduced problem substantially improves the value of the *optimal* solution. Before we get to why we might expect this, let's see why this would make us happy. Suppose we can guarantee a decrease of Δ, so the new optimum is at most $\text{cost}(\mathsf{OPT}) - \Delta$. When we're trying to prove that our algorithm is a $(2-\delta)$-approximation, we can inductively assume that it's a $(2-\delta)$-approximation on the reduced problem; this means it'll return a solution whose cost in the reduced problem is at most $(2-\delta)(\text{cost}(\mathsf{OPT}) - \Delta)$, and therefore whose cost in the *actual* problem (where we pay the penalties of the dead vertices) is at most

$$(2-\delta)\,\text{cost}(\mathsf{OPT}) - (2-\delta)\Delta + \pi(\mathsf{Dead}).$$

So if we can guarantee a decrease of $\Delta = \frac{1}{2-\delta} \cdot \pi(\mathsf{Dead})$, then IT will really be a $(2-\delta)$-approximation. For some intuition, the maximum decrease we could possibly hope for is $\pi(\mathsf{Dead})$ (we're decreasing the total penalties by $\pi(\mathsf{Dead})$, so we can't possibly decrease the optimum value by more than that); and this argument shows that as long as we can guarantee a decrease somewhat close to this, we'll win.

Now, why might we expect that when $\pi(\mathsf{Dead})$ is large, moving to the reduced problem substantially improves the optimum? The main idea is that $\pi(\mathsf{Dead}) = t(\mathsf{Dead})$ (since every dead vertex v runs out of potential, meaning $\pi(v) = t(v)$), so if $\pi(\mathsf{Dead})$ is large, then dead vertices spend a long time painting; and intuitively, this means they 'account' for a large portion of the weight of $\mathcal{T}_{\mathsf{OPT}}$. (We'll formalize what this means in the analysis.) And when we move to the reduced problem, now that we've set their penalties to 0, we can try to improve OPT by pruning out dead vertices; and we might hope this substantially decreases its tree weight.

However, this isn't exactly true — it's possible that $\mathcal{T}_{\mathsf{OPT}}$ has lots of dead vertices which account for lots of its weight, but they're embedded somewhere in the center of the tree where they can't be pruned out. But it turns out that in this case, we can actually improve the bound in Lemma 2.4, our lower bound on $w(\mathcal{T}_{\mathsf{OPT}})$. (Roughly, the way we obtained this lower bound originally was by saying that for every vertex v that ends up in $\mathcal{T}_{\mathsf{OPT}}$, whenever v is painting, it's painting at least one edge of $\mathcal{T}_{\mathsf{OPT}}$; this bad case will correspond to vertices that are painting at least *two* edges of $\mathcal{T}_{\mathsf{OPT}}$, which will improve the bound.) Using this improvement, we can conclude that GW itself is a good approximation (i.e., its ratio is substantially better than 2). So we win in this case as well.

§4.2 The analysis

To summarize the previous subsection, our final algorithm works as follows:

- Run $\mathsf{GW}[2]$; this gives one candidate solution, which we denote by $\mathsf{GW}[2]$.
- Run GW; this gives a second candidate solution, which we denote by GW, as well as a partition of the vertices into two sets Live and Dead.
- Run a ρ-approximation algorithm for ordinary Steiner tree on the live vertices; this gives a third candidate solution ST.
- Consider the reduced problem where we modify all the penalties of dead vertices to 0, and run our algorithm recursively on this reduced problem; this gives a fourth candidate solution IT. (If all dead vertices already had penalty 0, then we skip this step.)

Our goal is to show that in all cases, one of these four candidates is a $(2-\delta)$-approximation (for some hopefully not too small δ that we'll compute in the end).

First, we've seen that $\mathsf{GW}[2]$ is good when OPT's penalty is large, and that ST is good when OPT's penalty is small and $\pi(\mathsf{Dead})$ (the penalty of the dead vertices) is also small. So it suffices to consider the case where $\pi(\mathsf{Dead})$ is large; and in this case, we'll show that either GW or IT is good.

For the analysis, we partition the dead vertices X and Y into two sets, depending on whether OPT includes them or not — i.e., we define

$$X = \mathsf{Dead} \cap \mathsf{Incl}(\mathsf{OPT}) \quad \text{and} \quad Y = \mathsf{Dead} \cap \mathsf{Excl}(\mathsf{OPT}).$$

We say a set $S \subseteq V$ is a *single-cut set* if S cuts $\mathcal{T}_{\mathsf{OPT}}$ in exactly one edge, and a *multi-cut set* if S cuts $\mathcal{T}_{\mathsf{OPT}}$ in at least two edges.

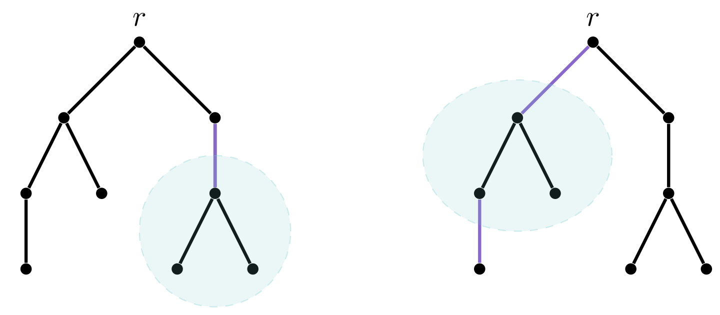

Figure 8: A single-cut set on the left, and a multi-cut set on the right (where only the edges and vertices of $\mathcal{T}_{\mathsf{OPT}}$ are shown, and the cut edges are in purple).

We'll split $t(X)$ (the total time all vertices $v \in X$ spend painting under GW) into two parts: we define $t_1(X)$ as the total time vertices $v \in X$ spend painting as the representative of a single-cut set, and $t_2(X)$ as the total time they spend painting as the representative of a multi-cut set. Note that for any $v \in \mathsf{Incl}(\mathsf{OPT})$, whenever v is painting as the representative of some set S, this set has to cut $\mathcal{T}_{\mathsf{OPT}}$ in at least one edge (we used this in the proof of Lemma 2.4; it's true because S contains v but can't contain r, and both are in $\mathcal{T}_{\mathsf{OPT}}$), meaning it's either a single-cut or multi-cut set; so $t(X) = t_1(X) + t_2(X)$.

We're trying to handle the case where $t(\mathsf{Dead}) = t_1(X) + t_2(X) + t(Y)$ is large. We'll show that when $t_2(X)$ is large, GW is good (this corresponds to the last case discussed in the previous subsection), while when $t_1(X) + t(Y)$ is large, IT is good (this corresponds to the second-last case discussed there).

First we'll consider the case where $t_2(X)$ is large. The main idea is that we can improve the lower bound on $w(\mathcal{T}_{\mathsf{OPT}})$ from Lemma 2.4 (by adding an extra term of $t_2(X)$ to it).

Claim 4.1 — We have $w(\mathcal{T}_{\mathsf{OPT}}) \geq t(\mathsf{Incl}(\mathsf{OPT})) + t_2(X)$.

Proof. As mentioned above, whenever any vertex $v \in \mathsf{Incl}(\mathsf{OPT})$ is painting, it's painting at least one edge in $\mathcal{T}_{\mathsf{OPT}}$; and $t_2(X)$ captures the time when vertices are painting at least *two* edges in $\mathcal{T}_{\mathsf{OPT}}$. So the total amount of paint that gets placed on $\mathcal{T}_{\mathsf{OPT}}$ is at least $t(\mathsf{Incl}(\mathsf{OPT})) + t_2(X)$. Meanwhile, this total amount of paint is at most $w(\mathcal{T}_{\mathsf{OPT}})$, because each edge gets painted at most once. □

The original bound of Lemma 2.4 implied that GW was a 2-approximation; using this improved bound, we can show that it does substantially better when $t_2(X)$ is large.

Claim 4.2 — If $t_2(X) \geq \frac{1}{2}\delta \cdot \mathrm{cost}(\mathsf{OPT})$, then GW is a $(2-\delta)$-approximation.

Proof. In our original analysis of GW, we saw that $\mathrm{cost}(\mathsf{GW}) \le 2 \cdot t(V \setminus \{r\})$. Meanwhile, we have

$$t(\mathsf{Incl}(\mathsf{OPT})) \le w(\mathcal{T}_{\mathsf{OPT}}) - t_2(X) \quad \text{and} \quad t(\mathsf{Excl}(\mathsf{OPT})) \le \pi(\mathsf{Excl}(\mathsf{OPT}))$$

(the first statement is Claim 4.1, and the second is the fact that $t(v) \le \pi(v)$ for all v), and adding these together gives $t(V \setminus \{r\}) \le \mathrm{cost}(\mathsf{OPT}) - t_2(X)$. So

$$\mathrm{cost}(\mathsf{GW}) \le 2 \cdot \mathrm{cost}(\mathsf{OPT}) - 2 \cdot t_2(X) \le (2 - \delta) \cdot \mathrm{cost}(\mathsf{OPT}). \qquad \square$$

So we've handled the case where $t_2(X)$ is large; all that's left is the case where $t_1(X) + t(Y)$ is large and $t_2(X)$ is small. In this case, we want to show that IT — the solution obtained by paying the penalties of all the dead vertices, setting their penalties to 0, and running our algorithm recursively on the resulting problem (which we call the *reduced problem*). And the key insight is that in this case, the optimum value substantially drops when we move from the original to the reduced problem.

Lemma 4.3

The optimum value for the reduced problem is at most $\mathrm{cost}(\mathsf{OPT}) - t_1(X) - t(Y)$.

We saw in the previous subsection that if we could get our drop to be at least a $\frac{1}{2-\delta}$-fraction of $\pi(\mathsf{Dead})$, we'd be happy (i.e., we could guarantee IT is a $(2 - \delta)$-approximation). When $t_1(X) + t(Y)$ is large and $t_2(X)$ is small, then this bound accomplishes that — our drop is $t_1(X) + t(Y)$, while

$$\pi(\mathsf{Dead}) = t_1(X) + t_2(X) + t(Y).$$

Proof. First, the penalty of OPT itself drops by $t(Y)$ when we move from the original to the reduced problem, simply because OPT was originally paying penalties for all the vertices in Y, and we've now zeroed out those penalties. Now imagine that we take OPT, and obtain a new tree $\mathcal{T}'_{\mathsf{OPT}}$ by repeatedly pruning out dead vertices — we delete any dead vertices at leaves of $\mathcal{T}_{\mathsf{OPT}}$ repeatedly, until there are none left.

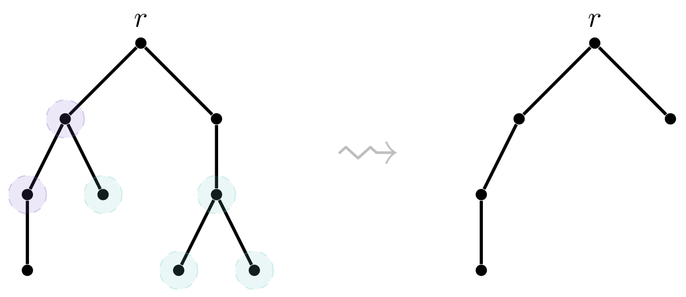

Figure 9: If we start with $\mathcal{T}_{\mathsf{OPT}}$ as shown on the left (with dead vertices circled), we'll end up pruning out the blue dead vertices but not the purple ones (pruning the purple ones would disconnect the tree), giving the tree on the right.

Our goal is to show that the weight of the edges that get pruned out is at least $t_1(X)$; we'll do this by showing the total amount of paint on those edges is at least $t_1(X)$.

Claim 4.4 — At any moment, if a vertex $v \in X$ is painting as a representative of a single-cut set S, then the edge of $\mathcal{T}_{\mathsf{OPT}}$ that it's painting gets pruned out.

Proof. First, since v is in X, it eventually dies during the process. But since it's currently painting, it hasn't died yet. So this means it dies at some point in the future. Let S' be the component it belongs to when it dies; since components can only grow with time, S' must contain S.

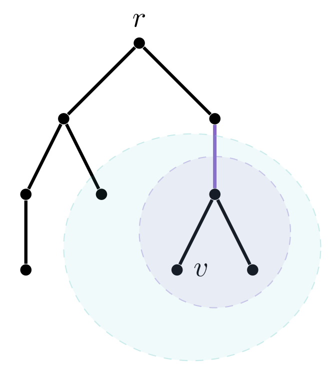

Figure 10: Here S (the single-cut set v is currently painting for) is shown in purple, and S' (the set v is in when it dies) is shown in blue; we have $S' \supseteq S$. The edge of $\mathcal{T}_{\mathsf{OPT}}$ that v is painting is drawn in purple.

But when S' dies, all the vertices inside it die (unless they were already dead); and since $S' \supseteq S$, this means all the vertices in S end up dead.

And since S is a single-cut set, it can be pruned out of $\mathcal{T}_{\mathsf{OPT}}$ without disconnecting $\mathcal{T}_{\mathsf{OPT}}$; so we will do so when pruning $\mathcal{T}_{\mathsf{OPT}}$, which will delete the one edge of $\mathcal{T}_{\mathsf{OPT}}$ that v is painting. □

And since $t_1(X)$ is the total amount of painting time of the form this claim describes, we get that the total amount of paint placed on pruned edges is at least $t_1(X)$; and therefore pruning removes at least $t_1(X)$ weight from $\mathcal{T}_{\mathsf{OPT}}$. □

So this completes the (qualitative) analysis — in all cases, we've shown that one of our four candidate solutions is good.

§4.3 Calculating the approximation ratio

Finally, we'll estimate the value of δ we end up with from this approach.

First, we've seen that $\mathsf{GW}[2]$ handles the case where $\pi(\mathsf{Excl}(\mathsf{OPT})) \geq \delta \cdot \mathrm{cost}(\mathsf{OPT})$, so from now on we can assume not.

Next, we've seen that GW handles the case where $t_2(X) \geq \frac{1}{2}\delta \cdot \mathrm{cost}(\mathsf{OPT})$ (by Claim 4.2). Meanwhile, IT handles the case where moving to the reduced problem gets a drop of at least $\frac{1}{2-\delta} \cdot \pi(\mathsf{Dead})$, and by Lemma 4.3 this occurs when

$$\frac{t_1(X) + t(Y)}{\pi(\mathsf{Dead})} = \frac{\pi(\mathsf{Dead}) - t_2(X)}{\pi(\mathsf{Dead})} \geq \frac{1}{2-\delta}.$$

To avoid computations, we'll assume $\delta \leq \frac{1}{4}$ and replace $\frac{1}{2-\delta}$ with $\frac{4}{7}$; then this says IT is good as long as $\pi(\mathsf{Dead}) \geq \frac{7}{3}t_2(X)$. Combining these two cases, either GW or IT is good as long as

$$\pi(\mathsf{Dead}) \geq \frac{7}{6}\delta \cdot \mathrm{cost}(\mathsf{OPT}),$$

so from now on we can assume not.

Finally, we're left with the case where both OPT's penalty and the penalty of the dead vertices are small, and in this case we want to show that ST is good. Lemma 3.3, together with the fact that ST is a ρ-approximation of the optimal Steiner tree, bounds the tree weight of ST by

$$w(\mathcal{T}_{\mathsf{ST}}) \leq \rho(w(\mathcal{T}_{\mathsf{OPT}}) + 2 \cdot \pi(\mathsf{Excl}(\mathsf{OPT}))) \leq \rho(1+\delta) \cdot \mathrm{cost}(\mathsf{OPT})$$

(in the last equality, we're adding the bounds $w(\mathcal{T}_{\mathsf{OPT}}) + \pi(\mathsf{Excl}(\mathsf{OPT})) = \mathrm{cost}(\mathsf{OPT})$ and $\pi(\mathsf{Excl}(\mathsf{OPT})) \leq \delta \cdot \mathrm{cost}(\mathsf{OPT})$). Meanwhile, the total penalty of ST is at most $\pi(\mathsf{Dead}) \leq \frac{7}{6}\delta \cdot \mathrm{cost}(\mathsf{OPT})$, so in total

$$\mathrm{cost}(\mathsf{ST}) \leq \left(\rho(1+\delta) + \frac{7}{6}\delta\right) \cdot \mathrm{cost}(\mathsf{OPT}).$$

For ST to be a $(2-\delta)$-approximation, we want to have $\rho(1+\delta) + \frac{7}{6}\delta \leq 2 - \delta$. Plugging in $\rho = 1.4$ (it's our approximation factor for ordinary Steiner tree) and solving for δ gives $\delta \approx 0.16$. This is much more substantial than our value of $\delta = 0.01$ from the previous algorithm, so we're happy.

References

[1] Ali Ahmadi, Iman Gholami, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Mohammad Mahdavi. Prize-collecting Steiner tree: a 1.79 approximation. In *Proceedings of the 56th annual ACM symposium on theory of computing*, STOC 2024, pages 1641–1652, 2024.

[2] Aaron Archer, Mohammadhossein Bateni, MohammadTaghi Hajiaghayi, and Howard Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM Journal on Computing*, 40(2):309–332, 2011.

[3] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved LP-based approximation for Steiner tree. In *Proceedings of the 42nd ACM symposium on theory of computing*, STOC 2010, pages 583–592, 2010.

[4] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[5] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting Steiner tree problem: theory and practice. In *Proceedings of the eleventh annual ACM-SIAM symposium on discrete algorithms*, SODA 2000, pages 760–769, 2000.

[6] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.