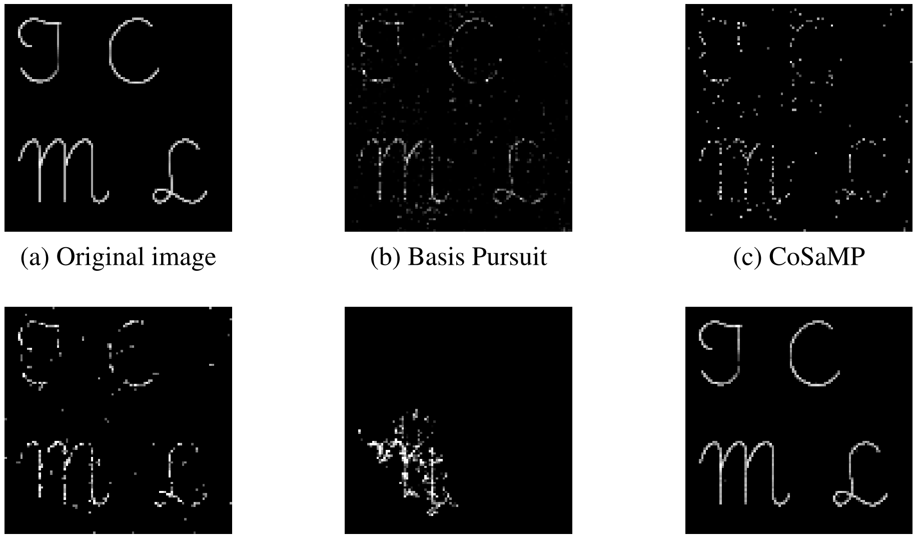


(a) Original image (b) Basis Pursuit (c) CoSaMP

(d) LaMP (e) StructOMP (f) Graph-CoSaMP

Figure 3. Recovery examples for the text image (see Figure 2) and $n = 3.3s$ noisy linear observations using different recovery algorithms. Only Graph-CoSaMP is able to recover the image correctly.

A. Further experimental results

We start with a more detailed description of our experimental setup. All three images used in Section 6 (Figure 2) are grayscale images of dimension 100×100 pixels with sparsity around 4% to 6%. The background-subtracted image was also used for the experimental evaluation in (Huang et al., 2011). The angiogram image is a slightly sparsified version of the image on the Wikipedia page about angiograms;[6] it shows cerebral blood vessels. The text image was created by us.

We used SPGL1[7] as implementation of Basis Pursuit. The implementation of CoSaMP was written by us, closely following (Needell & Tropp, 2009). Graph-CoSaMP and CoSaMP share the same code, only the projection methods differ (hard s-thresholding for CoSaMP and our model projections for Graph-CoSaMP). Empirically it is not necessary to "boost" the head-approximation algorithm as strongly as suggested by the analysis in (Hegde et al., 2014a), we use only a single approximate model projection in place of HEADAPPROX' (see Alg. 2). The timing experiments in Figure 2(d) were conducted on a Windows machine with a 2.30 GHz Intel Core i7 CPU, 8 MB of cache, and 32 GB of RAM.

Recovered images In order to illustrate the outcomes of unsuccessful recovery trials, we show examples in the regime where Graph-CoSaMP recovers correctly but the other algorithms fail. This is the most relevant regime because it demonstrates that Graph-CoSaMP accurately recovers the image while other methods still show significant errors. See Figure 3 for the corresponding results.

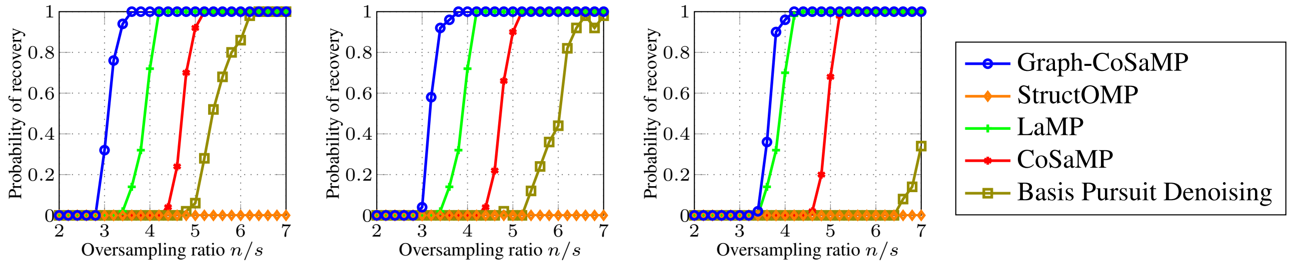
Noise tolerance We also investigate the performance of the recovery algorithms in the noisy setting (the error term e in (2)). For this, we add Gaussian noise at a measurement-SNR level of roughly 15dB. Since we cannot hope for exact recovery in the noisy setting, we consider different tolerance levels for declaring a trial as successful (the ratio $\|\beta - \widehat{\beta}\|^2 / \|\beta\|^2$). Figure 4 contains the phase transition plots for the text image from Figure 2(c). The results show that our algorithm also gives the best performance for noisy observations.

Graph Lasso Next, we compare our approach to the graph Lasso introduced in (Jacob et al., 2009). Since the implementation in the SPArse Modeling toolbox (SPAMS)[8] focuses on dense design matrices, we limit our experiments to a smaller image than those in Figure 2. In particular, we use a 30×30 pixel synthetic image similar to the experiment in Section 9.3

[6]http://commons.wikimedia.org/wiki/File:Cerebral_angiography,_arteria_vertebralis_sinister_injection.JPG

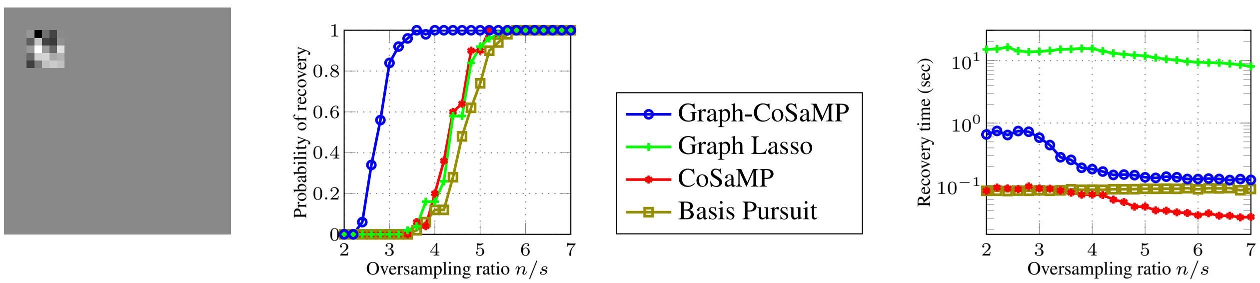
[7]https://www.math.ucdavis.edu/~mpf/spgl1/

[8]http://spams-devel.gforge.inria.fr/index.html



(a) Threshold 15%   (b) Threshold 10%   (c) Threshold 5%

Figure 4. Phase transitions for successful recovery under noisy observations. The three plots are for the same image (the text image from Fig. 2 (c)) but use different thresholds for declaring a trial as successful (the ratio $\|\beta - \widehat{\beta}\|^2 / \|\beta\|^2$). Our algorithm offers the best performance for all thresholds.



(a) Unknown β (b) Recovery performance   (c) Running time

Figure 5. Comparison of our algorithm Graph-CoSaMP with the graph Lasso. Subfigure (a) shows the synthetic test image (30×30 pixels). Graph-CoSaMP recovers the vector β from significantly fewer measurements than the other approaches (phase transition plot (b)). Moreover, Graph-CoSaMP is significantly faster than the variable replication implementation of the graph Lasso and essentially matches the performance of Basis Pursuit in the regime where both algorithms succeed ($n/s \geq 5$ in subfigure (c)).

of (Jacob et al., 2009). The nonzeros form a 5×5 square and hence correspond to a single component in the underlying grid graph. As suggested in (Jacob et al., 2009), we encode the graph structure by using all 4-cycles as groups and use the variable replication approach to implement the overlapping group penalty.

We record n observations $y = X\beta$ with an i.i.d. Gaussian design matrix and follow the experimental procedure outlined in Section 6 (recovery threshold 5%, 50 trials per data point). See Figure 5 for our results. While the graph Lasso improves over Basis Pursuit, our algorithm Graph-CoSaMP recovers the unknown vector β from significantly fewer observations. Moreover, our algorithm is significantly faster than this implementation of the graph Lasso via variable replication.[9] While there are faster algorithms for the overlapping group Lasso such as (Mosci et al., 2010), the recovery performance of the graph Lasso only matches Graph-CoSaMP for $n/s \geq 5$. In this regime, Graph-CoSaMP is already almost as fast as an efficient implementation of Basis Pursuit (SPGL1).

B. Sparse recovery with the WGM

We now give proofs for theorems in Section 5. First, we establish our general sample complexity bound.

Theorem 13. *Let $\mathbb{M}$ be the set of supports in the (G, s, g, B)-WGM. Then*

$$\log|\mathbb{M}| \;=\; O\left(s\left(\log\rho(G) + \log\frac{B}{s}\right) + g\log\frac{d}{g}\right).$$

Proof. Note that every support in the WGM corresponds to a g-forests, which contains exactly $s - g$ edges. We prove the theorem by counting the possible locations of g tree roots in the graph G, and then the local arrangements of the $s - g$

[9] As suggested by the documentation of the SPAMS toolbox, we ran this set of experiments under Linux. The corresponding machine has an Intel Core 2 Duo CPU with 2.93 GHz, 3 MB of cache, and 8 GB of RAM.

edges in the g trees.

Consider the following process:

1. Choose g root nodes out of the entire graph. There are $\binom{d}{g}$ possible choices.
2. Consider the $s-g$ edges as an ordered list and distribute the total weight budget B to the edges. There are $\binom{B+s-g-1}{s-g}$ possible allocations.
3. Assign a "target index" $t_e \in [\rho(G)]$ to each edge. There are $\rho(G)^{s-g}$ possible assignments. Note that the combination of edge weight and target index uniquely determines a neighbor of a fixed node v because there are at most $\rho(G)$ neighbors of v connected with edges of the same weight.
4. We now split the list of edges (together with their weight and target index assignments) into s sets. There are $\binom{2s-g}{s-1}$ possible partitions of the edge list.

We now have a list L consisting of s edge sets together with weight assignments and target indices. Moreover, we have a list of root nodes. We convert this structure to a g-forest (and hence a support in the WGM) according to the following rules, which essentially form a breadth-first search:

While there is a remaining root node, repeat the following:

1. Add the root node to a queue Q.
2. Initialize a new empty tree T_i.
3. While Q is non-empty, repeat the following
   (a) Let v be the first node in Q and remove v from Q.
   (b) Add v to T_i.
   (c) Let A be the first edge set in L and remove A from L.
   (d) For each pair of target index and weight in A, add the corresponding neighbor to Q.

Note that this process does not always succeed: for some weight allocations, there might be no neighbor connected by an edge with the corresponding weight. Nevertheless, it is easy to see that every possible support in the (G, s, g, B)-WGM can be constructed from at least one allocation via the process described above. Hence we have a surjection from the set of allocations to supports in the (G, s, g, B)-WGM $\mathbb{M}$, which gives the following bound:

$$|\mathbb{M}| \leq \binom{B+s-g-1}{s-g} \cdot \rho^{s-g} \cdot \binom{2s+g}{s-1} \cdot \binom{d}{g} .$$

Taking a logarithm on both sides and simplifying yields the bound in the theorem. □

The proof of the recovery result in Theorem 12 directly follows by combining the guarantees established for our tail- and head-approximation algortihms (Theorems 9 and 11) with the framework of (Hegde et al., 2014a).

Theorem 12. *Let $\beta \in \mathbb{R}^d$ be in the (G, s, g, B)-WGM $\mathcal{M}$ and let $X \in \mathbb{R}^{n\times d}$ be a matrix satisfying the model-RIP for a $(G, c_1 s, g, c_2 B)$-WGM and a fixed constant δ, where c_1 and c_2 are also fixed constants. Moreover, let $e \in \mathbb{R}^n$ be an arbitrary noise vector and let $y \in \mathbb{R}^n$ be defined as in* (2). *Then* GRAPH-COSAMP *returns a $\widehat{\beta}$ in the $(G, 5s, g, 5B)$-WGM such that $\left\|\beta - \widehat{\beta}\right\| \leq c_3\|e\|$, where c_3 is a fixed constant. Moreover,* GRAPH-COSAMP *runs in time*

$$O\left((T_X + |E| \log^3 d) \log \frac{\|\beta\|}{\|e\|}\right) ,$$

where T_X is the time complexity of a matrix-vector multiplication with X.

Proof. Note that both our head- and tail-approximation algorithms project into an output model with parameters bounded by constant multiples of s and B (we always maintain that the support corresponds to a g-forest), see Theorems 9 and 11. This allows us to use the CoSaMP version of Corollary 19 in (Hegde et al., 2014a) to establish the recovery result in our theorem. The claim about the running time follows from the near-linear running time of our model-projection algorithms and the running time analysis of CoSaMP in (Needell & Tropp, 2009). The $\log \frac{\|\beta\|}{\|e\|}$ term in the running time comes from the geometric convergence of Graph-CoSaMP. □

C. Approximate model-projection algorithms for the WGM

We now formally prove the head- and tail-approximation guarantees for our model-projection algorithms. We assume that we have access to an algorithm PCSF-GW for the PCSF problem with the approximation guarantee from Theorem 5, which we restate for completeness:

Theorem 5. *There is an algorithm for the PCSF problem that returns a g-forest F such that*

$$c(F) + 2\pi(\overline{F}) \;\leq \min_{F' \subseteq G,\, \gamma(F') \leq g} 2c(F') + 2\pi(\overline{F'}) \; . \tag{11}$$

We denote the running time of PCSF-GW with T_{PCSF}. See Section D for an algorithm that achieves guarantee (11) in nearly-linear time.

C.1. Tail-approximation

We first address the special case that there is a g-forest F^* with $c(F^*) \leq C$ and $\pi(\overline{F^*}) = 0$. In this case, we have to find a g-forest F with $\pi(\overline{F}) = 0$ in order to satisfy (12).

Lemma 14. *Let $\pi_{\min} = \min_{\pi(v)>0} \pi(v)$ and $\lambda_0 = \frac{\pi_{\min}}{2C}$. If there is a g-forest F^* with $c(F^*) \leq C$ and $\pi(\overline{F^*}) = 0$, then* PCSF-GW$(G, c_{\lambda_0}, \pi, g)$ *returns a g-forest F with $c(F) \leq 2C$ and $\pi(\overline{F}) = 0$.*

Proof. Applying the GW guarantee (11) gives

$$\begin{aligned} \lambda_0 \cdot c(F) + 2\pi(\overline{F}) \;&\leq\; 2\lambda_0 \cdot c_(F^*) + 2\pi(\overline{F^*}) \\ \pi(\overline{F}) \;&\leq\; \lambda_0 C \;=\; \frac{\pi_{\min}}{2} \; . \end{aligned}$$

Since $\pi_{\min} > 0$, we must have $\pi(\overline{F}) < \pi_{\min}$ and hence $\pi(\overline{F}) = 0$.

Applying (11) again then gives $c_{\lambda_0}(F) \leq 2c_{\lambda_0}(F^*)$, which shows that $c(F) \leq 2c(F^*) \leq 2C$ as desired. □

We can now proceed to prove an approximation guarantee for PCSF-TAIL.

Theorem 8. *Let $\nu > 2$ and $\delta > 0$. Then* PCSF-TAIL *returns a g-forest $F \subseteq G$ such that $c(F) \leq \nu \cdot C$ and*

$$\pi(\overline{F}) \;\leq\; \left(1 + \frac{2}{\nu - 2} + \delta\right) \min_{\gamma(F') = g, c(F') \leq C} \pi(\overline{F'}) \; . \tag{12}$$

Proof. We consider the three different cases in which PCSF-TAIL returns a forest. Note that the resulting forest is always the output of PCSF-GW with parameter g, so the resulting forest is always a g-forest. To simplify notation, in the following we use

$$OPT = \min_{\gamma(F') = g, c(F') \leq C} \pi(\overline{F'}) \; .$$

First, if PCSF-TAIL returns in Line 5, the forest F directly satisfies (12). Otherwise, there is no g forest F^* with $c(F^*) \leq C$ and $\pi(\overline{F^*}) = 0$ (contrapositive of Lemma 14). Hence in the following we can assume that $OPT \geq \pi_{\min}$.

If the algorithm returns in Line 10, we clearly have $c(F) \leq \nu \cdot C$. Moreover, the GW guarantee gives

$$\lambda_m \cdot c(F) + 2\pi(\overline{F}) \;\leq\; 2\lambda_m C + 2 \cdot OPT \; .$$

Since $c(F) \geq 2C$, we have $\pi(\overline{F}) \leq OPT$, satisfying (12).

Finally, consider the case that PCSF-TAIL returns in Line 13. Let F_l and F_r be the forests corresponding to λ_l and λ_r, respectively. We show that the final output F_l satisfies the desired approximation guarantee if $\lambda_r - \lambda_l$ is small. Note that during the binary search, we always maintain the invariant $c(F_l) \leq 2C$ and $c(F_r) \geq \nu \cdot C$.

Using the GW guarantee and $\pi(\overline{F_r}) \geq 0$ gives $\lambda_r c(F_r) \leq 2\lambda_r C + 2 \cdot OPT$. Therefore,

$$\lambda_r \;\leq\; \frac{2 \cdot OPT}{c(F_r) - 2C} \;\leq\; \frac{2 \cdot OPT}{C(\nu - 2)} \; . \tag{14}$$

At the end of the binary search, we have $\lambda_l \leq \lambda_r + \varepsilon$. Combining this with (14) above and the GW guarantee (11) gives

$$\pi(\overline{F}) \;\leq\; \lambda_l C + OPT \;\leq\; OPT + (\lambda_r + \varepsilon)C \;\leq\; OPT + \frac{2 \cdot OPT}{\nu - 2} + \varepsilon C \;\leq\; \left(1 + \frac{2}{\nu - 2} + \delta\right) OPT \; .$$

In the last inequality, we used $OPT \geq \pi_{\min}$ and $\varepsilon = \frac{\pi_{\min}\delta}{C}$. This concludes the proof. □

Finally, we consider the running time of PCSF-TAIL.

Theorem 15. PCSF-TAIL *runs in time* $O(T_{\text{PCSF}} \cdot \log \frac{C \cdot \pi(G)}{\delta \cdot \pi_{\min}})$.

Proof. The time complexity is dominated by the number of calls to PCSF-GW. Hence we bound the number of binary search iterations in order to establish the overall time complexity. Let $\lambda^{(0)}$ be the initial value of λ_l in PCSF-TAIL. Then the maximum number of iterations of the binary search is

$$\left\lceil \log \frac{\lambda_l^{(0)}}{\varepsilon} \right\rceil \;=\; \left\lceil \log \frac{3\pi(G) \cdot C}{\delta \cdot \pi_{\min}} \right\rceil \;=\; O\left(\log \frac{C \cdot \pi(G)}{\delta \cdot \pi_{\min}} \right) \; .$$

Since each iteration of the binary search takes $O(T_{\text{PCSF}})$ time, the time complexity stated in the theorem follows. □

If the node prizes π and edge costs c are polynomially bounded in $|V|$, the running time of PCSF-TAIL simplifies to $O(T_{\text{PCSF}} \cdot \log|V|)$ for constant δ.

We now have all results to complete our tail-approximation algorithm for the WGM.

Theorem 9. *Let $\mathcal{M}$ be a (G, s, g, B)-WGM, let $b \in \mathbb{R}^d$, and let $\nu > 2$. Then there is an algorithm that returns a support $S \subseteq [d]$ in the $(G,\, 2\nu \cdot s + g,\, g,\, 2\nu \cdot B)$-WGM satisfying* (8) *with $c_T = \sqrt{1 + 3/(\nu - 2)}$. Moreover, the algorithm runs in time $O(|E| \log^3 d)$.*

Proof. We run the algorithm PCSF-TAIL on the graph G with node prizes $\pi(i) = b_i^2$, edge costs $c(e) = w(e) + \frac{B}{s}$, a cost budget $C = 2B$, and the parameter $\delta = \min(\frac{1}{2}, \frac{1}{\nu})$. Let F be the resulting forest and S the corresponding support. The running time bound follows from combining Theorems 15 and 28.

First, we show that S is in the $(G,\, 2\nu \cdot s + g,\, g,\, 2\nu \cdot B)$-WGM. From Theorem 8 we know that F is a g-forest and that $c(F) \leq 2\nu \cdot B$. This directly implies that $w(F) \leq 2\nu \cdot B$. Moreover, the g-forest F has $|V_F| - g$ edges, all with cost at least $\frac{B}{s}$ because $w(e) \geq 0$ for all $e \in E$. Since $|V_F| = |S|$, this allows us to bound the sparsity of S as

$$(|S| - g) \cdot \frac{B}{s} \leq 2\nu \cdot B \; ,$$

which gives $|S| \leq 2s + g$ as desired.

Now, let S^* be an optimal support in the (G, s, g, B)-WGM $\mathbb{M}$ and let F^* be a corresponding g-forest, i.e.,

$$\pi(\overline{F^*}) \;=\; \|b - b_{S^*}\|^2 \;=\; \min_{S' \in \mathbb{M}} \|b - b_{S'}\|^2 \; .$$

Then we have

$$\pi(\overline{F^*}) \;\geq\; \min_{\gamma(F') = g,\, c(F') \leq 2B} \pi(\overline{F'})$$

because by construction, every support in $\mathbb{M}$ corresponds to a g-forest with cost at most $2B$. Since $\pi(\overline{F}) = \|b - b_S\|^2$, applying guarantee (12) gives

$$\|b - b_S\|^2 \;\leq\; \left(1 + \frac{2}{\nu - 2} + \delta\right) \min_{S' \in \mathbb{M}} \|b - b_{S'}\|^2 \,.$$

Simplifying this inequality with our choice of δ then completes the proof. $\square$

C.2. Head-approximation

We first state our head-approximation algorithm (see Alg. 3 and Alg. 4). In addition to a binary search over the Lagrangian parameter λ, the algorithm also uses the subroutines PRUNETREE and PRUNEFOREST in order to extract sub-forests with good "density" $\frac{\pi(F)}{c(F)}$.

Algorithm 3 Head approximation for the WGM: main algorithm PCSF-HEAD

function PCSF-HEAD$(G, c, \pi, g, C, \delta)$
    We write $\pi_\lambda(i) = \lambda \cdot \pi(i)$.
    $\pi_{\min} \leftarrow \min_{\pi(i)>0} \pi(i)$
    $\lambda_r \leftarrow \frac{2C}{\pi_{\min}}$
    $F \leftarrow$ PCSF-GW$(G, c, \pi_{\lambda_r}, g)$
    if $c(F) \leq 2C$ **then** ▷ Ensure that we have the invariant $c(F_r) > 2C$ (see Theorem 17)
        return F
    end if
    $\varepsilon \leftarrow \frac{\delta \cdot C}{2\pi(G)}$
    $\lambda_l \leftarrow \frac{1}{4\pi(G)}$
    while $\lambda_r - \lambda_l > \varepsilon$ **do** ▷ Binary search over the Lagrange parameter λ
        $\lambda_m \leftarrow (\lambda_l + \lambda_r)/2$
        $F \leftarrow$ PCSF-GW$(G, c, \pi_{\lambda_m}, g)$
        if $c(F) > 2C$ **then**
            $\lambda_r \leftarrow \lambda_m$
        else
            $\lambda_l \leftarrow \lambda_m$
        end if
    end while
    $F_l \leftarrow$ PCSF-GW$(G, c, \pi_{\lambda_l}, g)$
    $F_r \leftarrow$ PCSF-GW$(G, c, \pi_{\lambda_r}, g)$
    $F'_r \leftarrow$ PRUNEFOREST(F, c, π, C) ▷ Prune the potentially large solution F_r (See Alg. 4)
    if $\pi(F_l) \geq \pi(F'_r)$ **then**
        return F_l
    else
        return F'_r
    end if
end function

We start our analysis by showing that PRUNETREE extracts sub-trees of good density $\frac{\pi(T')}{c(T')}$.

Lemma 10. *Let T be a tree and $C' \leq c(T)$. Then there is a subtree $T' \subseteq T$ such that $c(T') \leq C'$ and $\pi(T') \;\geq\; \frac{C'}{6} \cdot \frac{\pi(T)}{c(T)}$. Moreover, we can find such a subtree T' in linear time.*

Proof. We show that PRUNETREE satisfies the guarantees in the theorem. We use the definitions of L, π', c', and ϕ given in PRUNETREE (see Alg. 4). Moreover, let T' be the tree returned by PRUNETREE. First, note that PRUNETREE clearly runs in linear time by definition. Hence it remains to establish the approximation guarantee

$$\pi(T') \;\geq\; \frac{C'}{6} \cdot \frac{\pi(T)}{c(T)} \;=\; \frac{C' \cdot \phi}{6} \,.$$

Algorithm 4 Head approximation for the WGM: subroutine PRUNEFOREST

function PRUNEFOREST(F, c, π, C)
  Let $\{T_1, \ldots, T_{|F|}\}$ be the trees in F sorted by $\frac{\pi(T_i)}{c(T_i)}$ descendingly.
  $C_r \leftarrow C$
  for $i \leftarrow 1, \ldots, |F|$ **do**
    if $C_r \geq c(T_i)$ **then**
      $T'_i \leftarrow T_i$
      $C_r \leftarrow C_r - c(T_i)$ ▷ Cost budget $C^{(i)} = c(T_i)$
    else if $C_r > 0$ **then**
      $T'_i \leftarrow$ PRUNETREE(T_i, c, π, C_r)
      $C_r \leftarrow 0$ ▷ Cost budget $C^{(i)} = C_r$
    else
      $T'_i \leftarrow \{\arg\max_{j \in T_i} \pi(j)\}$ ▷ Cost budget $C^{(i)} = 0$
    end if
  end for
  return $\{T'_1, \ldots, T'_{|F|}\}$
end function

function PRUNETREE(T, c, π, C')
  Let $L = (v_1, \ldots, v_{2|V_T|-1})$ be a tour through the nodes of T. ▷ $T = (V_T, E_T)$
  Let $\pi'(j) = \begin{cases} \pi(v_j) & \text{if position } j \text{ is the first appearance of } v_j \text{ in } L \\ 0 & \text{otherwise} \end{cases}$
  Let $c'(P) = \sum_{i=1}^{|P|-1} c(P_i, P_{i+1})$
  Let $\phi = \frac{\pi(T)}{c(T)}$
  if there is a $v \in V_T$ with $\pi(v) \geq \frac{C' \cdot \phi}{6}$ **then** ▷ Check if there is a single good node (cost is automatically 0)
    return the tree $\{v\}$
  end if
  $l \leftarrow 1$
  $P^1 = ()$ ▷ Empty list
  for $i \leftarrow 1, \ldots, 2|V_T| - 1$ **do** ▷ Search for good sublists of L
    Append i to P^l
    if $c'(P^l) > C'$ **then** ▷ Start a new sublist if the cost reaches C
      $l \leftarrow l + 1$
      $P^l \leftarrow ()$
    else if $\pi'(P^l) \geq \frac{C' \cdot \phi}{6}$ **then** ▷ Return if we have found a good sublist
      return the subtree of T on the nodes in P^l
    end if
  end for
  Merge P^l and P^{l-1} ▷ The algorithm will never reach this point (see Lemma 10).
end function

Consider the first case in which PRUNETREE returns in line 23. Then T' is a tree consisting of a single node, so $c(T') = 0 \leq C'$. Moreover, we have $\pi(T') = \pi(v) \geq \frac{C' \cdot \phi}{6}$, which satisfies the guarantee in the theorem.

Next, we consider the case in which PRUNETREE returns in line 33. By definition of the algorithm, we have $c'(P^l) \leq C'$ and hence $c(T') \leq C'$ because the spanning tree T' of the nodes in P^l contains only edges that are also included at least once in $c'(P^l)$. Moreover, we have $\pi(T') \geq \pi'(P^l) \geq \frac{C' \cdot \phi}{6}$, so T' satisfies the guarantee in the theorem.

It remains to show that PRUNETREE always returns in one of the two cases above, i.e., never reaches line 36. We prove this statement by contradiction: assume that PRUNETREE reaches line 36. We first consider the partition of V_T induced by the lists P^i just before line 36. Note that there are no nodes $v \in V_T$ with $\pi(v) \geq \frac{C' \cdot \phi}{6}$ because otherwise PRUNETREE would have returned in line 23. Hence for every list P^i we have $\pi'(P^i) \leq \frac{C' \cdot \phi}{3}$ because the last element that was added to P^i can have increased $\pi'(P^i)$ by at most $\frac{C' \cdot \phi}{6}$, and we had $\pi'(P^i) < \frac{C' \cdot \phi}{6}$ before the last element was added to P^i because otherwise PRUNETREE would have returned in line 33. Moreover, every list P^i except P^l satisfies $c'(P^i) > C'$ by construction. Hence after merging the last two lists P^l and P^{l-1}, we have $c'(P^i) > C'$ for all P^i and also $\pi'(P^i) < \frac{C' \cdot \phi}{2}$.

We now derive the contradiction: note that all lists P^i have a low density $\frac{\pi'(P^i)}{c'(P^i)}$ but form a partition of the nodes in V_T. We can use this fact to show that the original tree had a density lower than $\frac{\pi(T)}{c(T)}$, which is a contradiction. More formally, we have

$$\pi(T) \; = \; \sum_{i=1}^{l-1} \pi'(P^i) \; < \; (l-1)\frac{C' \cdot \phi}{2}$$

and

$$2c(T) \; \geq \; \sum_{i=1}^{l-1} c'(P^i) \; > \; (l-1)C \; .$$

Combining these two inequalities gives

$$\frac{\phi}{2} \; = \; \frac{\pi(T)}{2c(T)} \; < \; \frac{(l-1)\frac{C'\phi}{2}}{(l-1)C'} \; = \; \frac{\phi}{2} \; ,$$

which is a contradiction. Hence PRUNETREE always returns in line 23 or 33 and satisfies the guarantee of the theorem. □

Extending the guarantee of PRUNETREE to forests is now straightforward: we can prune each tree in a forest F individually by assigning the correct cost budget to each tree. More formally, we get the following lemma.

Lemma 16. *Let F be a g-forest. Then* PRUNEFOREST(F, c, π, C) *returns a g-forest F' with $c(F') \leq C$ and*

$$\pi(F') \geq \frac{C}{6 \cdot c(F)} \pi(F) \; .$$

Proof. By construction, F' is a g-forest with $c(F) \leq C$. Let $C^{(i)}$ be the cost budget assigned to tree T'_i (see the comments in PRUNEFOREST). Using Lemma 10, we get

$$\pi(F') \; = \; \sum_{i=1}^{g} \pi(T'_i) \; \geq \; \sum_{i=1}^{g} \frac{C^{(i)}}{6} \frac{\pi(T_i)}{c(T_i)} \; .$$

Note that the $C^{(i)}$ are the optimal allocation of budgets to the ratios $\frac{\pi(T_i)}{c(T_i)}$ with $0 \leq C^{(i)} \leq c(T_i)$ and $\sum_{i=1}^{g} C^{(i)} = C$. In particular, we have

$$\sum_{i=1}^{g} \frac{C^{(i)}}{6} \frac{\pi(T_i)}{c(T_i)} \; \geq \; \sum_{i=1}^{g} \frac{C \cdot \frac{c(T_i)}{c(F)}}{6} \frac{\pi(T_i)}{c(T_i)} \; = \; \frac{C}{6 \cdot c(F)} \pi(F) \; ,$$

which completes the proof. □

We can now prove our main theorem about PCSF-HEAD.

Theorem 17. *Let* $0 < \delta < \frac{1}{13}$*. Then* PCSF-HEAD *returns a* g*-forest* F *such that* $c(F) \leq 2C$ *and*

$$\pi(F) \;\geq\; \left(1 - \frac{12}{13(1-\delta)}\right) \max_{\gamma(F')=g,\, c(F')\leq C} \pi(F') \;. \tag{15}$$

Proof. Let F^* be an optimal g-forest with $c(F) \leq C$ and $\pi(F^*) = OPT$, where

$$OPT = \max_{\gamma(F')=g,\, c(F')\leq C} \pi(F') \;.$$

In this proof, the following rearranged version of the GW guarantee 11 will be useful:

$$\begin{aligned} c(F) + 2(\pi(G) - \pi(F)) \;&\leq\; 2C + 2(\pi(G) - \pi(F^*)) \\ \pi(F) \;&\geq\; \pi(F^*) + \frac{c(F) - 2C}{2} \;. \end{aligned} \tag{16}$$

As in the definition of PCSF-HEAD, we write π_λ for the node prize function $\pi_\lambda(i) = \lambda \cdot \pi(i)$. Using such modified node prizes, (16) becomes

$$\pi(F) \;\geq\; OPT + \frac{c(F) - 2C}{2\lambda} \;. \tag{17}$$

We now analyze two cases: either PCSF-HEAD returns in line 7 or in one of the lines 24 and 26. Note that in all cases, the returned forest F is a g-forest because it is produced by PCSF-GW (and PRUNEFOREST maintains this property).

First, we consider the case that the algorithm returns in line 7. Then by definition we have $c(F) \leq 2C$. Moreover, the modified GW guarantee (17) gives

$$\pi(F) \;\geq\; OPT + \frac{c(F) - 2C}{2\lambda_r} \;\geq\; OPT - \frac{C}{\lambda_r} \;\geq\; OPT - \frac{\pi_{\min}}{2} \;\geq\; \frac{1}{2} OPT \;,$$

because clearly $OPT \geq \pi_{\min}$. Hence the guarantee in the theorem is satisfied.

Now, consider the case that the algorithm enters the binary search. Let F_l and F_r be the g-forests corresponding to λ_l and λ_r, respectively. During the binary search, we maintain the invariant that $c(F_l) \leq 2C$ and $c(F_r) > 2C$. Note that our initial choices for λ_l and λ_r satisfy this condition (provided the algorithm reaches the binary search).

When the algorithm terminates in line 24 or 26, we have $\lambda_r > \lambda_l > \lambda_r - \varepsilon$. Rearranging (17) gives

$$\begin{aligned} 2\lambda_r(\pi(F_r) - OPT) \;&\geq\; c(F_r) - 2C \\ \lambda_r \;&\geq\; \frac{c(F_r) - 2C}{2(\pi(F_r) - OPT)} \;. \end{aligned}$$

We now introduce a variable $\alpha \geq 2$ and distinguish two cases:

Case 1: Assume Assume $\frac{c(F_r)}{\pi(F_r)} > \alpha \frac{C}{OPT}$. Then Equation (17) gives

$$\begin{aligned} \lambda_r \;&\geq\; \frac{\frac{1}{2}\frac{c(F_r)}{OPT} - \frac{C}{OPT}}{\frac{\pi(F_r)}{OPT} - 1} \\ &=\; \frac{\frac{1}{2}\frac{\pi(F_r)}{OPT}\frac{c(F_r)}{\pi(F_r)} - \frac{C}{OPT}}{\frac{\pi(F_r)}{OPT} - 1} \\ &\geq\; \frac{\frac{1}{2}\frac{\pi(F_r)}{OPT}\alpha - 1}{\frac{\pi(F_r)}{OPT} - 1} \frac{C}{OPT} \\ &\geq\; \frac{\frac{\alpha}{2}\frac{\pi(F_r)}{OPT} - \frac{\alpha}{2}}{\frac{\pi(F_r)}{OPT} - 1} \frac{C}{OPT} \\ &=\; \frac{\alpha}{2}\frac{C}{OPT} \;. \end{aligned}$$

So we get $\lambda_l \;\geq\; \lambda_r - \varepsilon \;\geq\; \frac{\alpha}{2}\frac{C}{OPT} - \frac{\delta \cdot C}{2\pi(G)} \;\geq\; (1-\delta)\frac{\alpha C}{2OPT}$. We can now use this together with (17) to get:

$$\begin{aligned} \pi(F_l) \;&\geq\; OPT - \frac{C}{\lambda_l} \\ &\geq\; OPT - \frac{2OPT}{(1-\delta)\alpha} \\ &=\left(1 - \frac{2}{(1-\delta)\alpha}\right) OPT \; . \end{aligned} \tag{18}$$

Case 2: Assume $\frac{c(F_r)}{\pi(F_r)} \leq \alpha \frac{C}{OPT}$, which is equivalent to $\frac{\pi(F_r)}{c(F_r)} \geq \frac{1}{\alpha}\frac{OPT}{C}$. Since $c(F_r) > 2C$, we can invoke PRUNEFOREST on F_r with cost budget C. Let F'_r be the resulting g-forest. From Lemma 16 we have $c(F'_r) \leq C$. Moreover,

$$\pi(F'_r) \;\geq\; \frac{C}{6 \cdot c(F_r)} \pi(F_r) \;=\; \frac{C}{6}\frac{\pi(F_r)}{c(F_r)} \;\geq\; \frac{1}{6\alpha} OPT \; . \tag{19}$$

Either case 1 or case 2 must hold. Since HEADAPPROX chooses the better forest among F_l and F'_r, we can combine Equations (18) and (19) to get the following guarantee on the final result F:

$$\pi(F) \;\geq\; \min\left(1 - \frac{2}{(1-\delta)\alpha},\; \frac{1}{6\alpha}\right) OPT \; .$$

Choosing $\alpha = \frac{13}{6}$ to balance the two expressions (assuming δ is close to 0) then gives the approximation guarantee stated in the theorem. □

Next, we consider the running time of PCSF-HEAD.

Theorem 18. PCSF-HEAD *runs in time* $O\left(T_{\text{PCSF}} \cdot \log \frac{\pi(G)}{\delta \cdot \pi_{\min}}\right)$.

Proof. As in Theorem 15 it suffices to bound the number of iterations of the binary search. Let $\lambda_r^{(0)}$ be the initial value of λ_r in PCSF-HEAD. Then the maximum number of iterations is

$$\left\lceil \log \frac{\lambda_r^{(0)}}{\varepsilon} \right\rceil \;=\; \left\lceil \log \frac{4 \cdot C \cdot \pi(G)}{\delta \cdot C \cdot \pi_{\min}} \right\rceil \;=\; O\left(\frac{\pi(G)}{\delta \cdot \pi_{\min}} \right) .$$

□

As before, the running time simplifies to $O(T_{\text{PCSF}} \cdot \log |V|)$ for constant δ if the node prizes and edge costs are polynomially bounded in the size of the graph.

We can now conclude with our head-approximation algorithm for the WGM.

Theorem 11. *Let $\mathcal{M}$ be a (G, s, g, B)-WGM and let $b \in \mathbb{R}^d$. Then there is an algorithm that returns a support $S \subseteq [d]$ in the $(G, 2s+g, g, 2B)$-WGM satisfying* (9) *with $c_H = \sqrt{1/14}$. The algorithm runs in time $O(|E| \log^3 d)$.*

Proof. We embed the WGM into a PCSF instance similar to Theorem 9: we run PCSF-HEAD on the graph G with node prizes $\pi(i) = b_i^2$, edge costs $c(e) = w(e) + \frac{B}{s}$, a cost budget $C = 2B$, and the parameter $\delta = \frac{1}{169}$. Let F be the resulting forest and S be the corresponding support. The running time bound follows from combining Theorems 18 and 28.

From Theorem 17 we directly have that F is a g-forest with $w(F) \leq 2B$. Following a similar argument as in Theorem 9, we also get $|S| \leq 2s + g$. So S is in the $(G, 2s+g, g, 2B)$-WGM.

Now, let S^* be an optimal support in the (G, s, g, B)-WGM $\mathbb{M}$ and let F^* be a corresponding g-forest, i.e.,

$$\pi(F^*) \;=\; \|b_{S^*}\|^2 \;=\; \max_{S' \in \mathbb{M}} \|b_{S'}\|^2 \; .$$

By construction, every support in $\mathbb{M}$ corresponds to a g-forest with cost at most $2B$. Hence we have

$$\pi(F^*) \;\leq\; \max_{\gamma(F')=g, c(F')\leq 2B} \pi(F')$$

Since $\pi(F) = \|b_S\|^2$, applying Theorem 17 gives

$$\|b_S\|^2 \;\geq\; \left(1 - \frac{12}{13(1-\delta)}\right) \max_{S'\in\mathbb{M}} \|b_{S'}\|^2 \;.$$

Substituting $\delta = \frac{1}{169}$ completes the proof. $\square$

D. The prize-collecting Steiner forest problem (PCSF)

For completeness, we first review the relevant notation and the definition of the PCSF problem. Let $G = (V, E)$ be an undirected, weighted graph with edge costs $c : E \to \mathbb{R}_0^+$ and node prizes $\pi : V \to \mathbb{R}_0^+$. For a subset of edges $E' \subseteq E$, we write $c(E') = \sum_{e\in E'} c(e)$ and adopt the same convention for node subsets. Moreover, for a node subset $V' \subseteq V$, let $\overline{V'}$ be the complement $\overline{V'} = V \setminus V'$. We denote the number of connected components in the (sub-)graph F with $\gamma(F)$.

Definition 4 (The prize-collecting Steiner forest problem)**.** *Let $g \in \mathbb{N}$ be the target number of connected components. Then the goal of the prize-collecting Steiner forest (PCSF) problem is to find a subgraph $F = (V', E')$ with $\gamma(F) = g$ that minimizes $c(E') + \pi(\overline{V'})$.*

We divide our analysis in two parts: we first modify the Goemans-Williamson (GW) scheme to get an efficient algorithm with provable approximation guarantee for the PCSF problem (Subsection D.1). Then we show how to simulate the GW scheme in nearly-linear time (Subsection D.2).

D.1. The Goemans-Williamson (GW) scheme for PCSF

Before we introduce our variant of the GW scheme and prove the desired approximation guarantee, we introduce additional notation. For a set of nodes $U \subseteq V$ and a set of edges $D \subseteq E$, we write $\delta_D U$ to denote the set of edges contained in D with exactly one endpoint in U. If $D = E$, we write δU. The degree of a node v in an edge set D is $\deg_D(v) = |\delta_D\{v\}|$. We say that a (sub-)graph F is a g-forest if F is a forest with $\gamma(F) = g$.

At its core, the GW algorithm produces three results: a *laminar family* of clusters, a *dual value* for each cluster, and a forest connecting the nodes within each cluster.

Definition 19 (Laminar family)**.** *A family $\mathscr{L}$ of non-empty subsets of V is a* laminar *family if one of the following three cases holds for all $L_1, L_2 \in \mathscr{L}$: either $L_1 \cap L_2 = \{\}$, or $L_1 \subseteq L_2$, or $L_2 \subseteq L1$.*

Let U be a subset of V. Then we define the following two subsets of $\mathscr{L}$:

- $\mathscr{L}_{\downarrow U} := \{L \in \mathscr{L} \mid L \subseteq U\}$ *(going "down" in the laminar hierarchy).*
- $\mathscr{L}_{\uparrow U} := \{L \in \mathscr{L} \mid U \subseteq L\}$ *(going "up" in the laminar hierarchy).*

Let $\mathscr{L}^*$ be the family of maximal sets in $\mathscr{L}$, i.e., $L \in \mathscr{L}^*$ iff there is no $L' \in \mathscr{L}$ with $L \subsetneq L'$. If $\bigcup_{L\in\mathscr{L}} = V$, then $\mathscr{L}^*$ is a partition of V.

Let $e \in E$, then we write $\mathscr{L}(e) := \{L \in \mathscr{L} \mid e \in \delta L\}$ for the sub-family of sets that contain exactly one endpoint of e.

Definition 20 (Dual values)**.** *Let $\mathscr{L}$ be a laminar family. Then the dual values are a function $y : \mathscr{L} \to \mathbb{R}_0^+$ with the following two properties (as before, we write $y(\mathscr{L}') := \sum_{L\in\mathscr{L}'} y(L)$ for a sub-family $\mathscr{L}' \subseteq \mathscr{L}$).*

- $y(\mathscr{L}(e)) \leq c(e)$ *for each $e \in E$.*
- $y(\mathscr{L}_{\downarrow L}) \leq \pi(L)$ *for each $L \in \mathscr{L}$.*

We also define several properties of g-forests related to the new concepts introduced above.

Let $\mathscr{L}$ be a laminar family. We say a tree T is $\mathscr{L}$-connected iff for every $L \in \mathscr{L}$, the subgraph on $V_T \cap L$ is connected (we consider an empty graph to be connected). A g-forest F is $\mathscr{L}$-connected iff every $T \in F$ is $\mathscr{L}$-connected.

Let $L \in \mathscr{L}^*$ and let $L(F)$ be the trees in F with at least one node in L, i.e., $L(F) = \{T \in F \,|\, V_T \cap L \neq \{\}\}$. A g-tree F is $\mathscr{L}^*$-disjoint iff $|L(F)| \leq 1$ for every $L \in \mathscr{L}^*$.

Let $\mathscr{D}$ be a family of subsets of V. A tree T has a leaf component in $\mathscr{D}$ iff there is a $D \in \mathscr{D}$ with $|\delta_T D| = 1$. A g-forest F has a leaf component in $\mathscr{D}$ iff there is a tree $T \in F$ that has a leaf component in $\mathscr{D}$.

A tree T is contained in $\mathscr{D}$ iff there is a $D \in \mathscr{D}$ such that $V_T \subseteq D$. A g-forest F is contained in $\mathscr{D}$ iff there is a tree $T \in F$ that is contained in $\mathscr{D}$.

D.1.1. ALGORITHM

Our algorithm is a modification of the unrooted GW PCST algorithm in (Johnson et al., 2000). In contrast to their unrooted prize-collecting Steiner tree algorithm, our algorithm stops the growth phase when exactly g active clusters are left. We use these active clusters as starting point in the pruning phase to identify a g-forest as the final result.

Since the focus of this section is the approximation guarantee rather than the time complexity, the pseudo code in Algorithm 5 is intentionally stated at a high level.

Algorithm 5 Prize-collecting Steiner forest

function PCSF-GW(G, c, π, g)
  $\mathscr{L} \leftarrow \{\{v\} \,|\, v \in V\}$ ▷ Laminar family of clusters.
  $y(C) \leftarrow 0$ for all $C \in \mathscr{L}$. ▷ Initial dual values.
  $V_F \leftarrow V, \quad E_F \leftarrow \{\}$ ▷ Initial forest.
  $\mathscr{D} \leftarrow \{\}$ ▷ Family of inactive clusters.
  $\mathscr{A} \leftarrow \mathscr{L}^* \setminus \mathscr{D}$ ▷ Family of active clusters.
  while $|\mathscr{A}| > g$ **do** ▷ Growth phase.
    $\varepsilon_d \leftarrow \min_{C \in \mathscr{A}} \pi(C) - y(\mathscr{L}_{\downarrow C})$ ▷ Next cluster deactivation time
    $\varepsilon_m \leftarrow \min_{\substack{e \in \delta C \\ C \in \mathscr{A}}} c(e) - y(\mathscr{L}(e))$ ▷ Next cluster merge time
    $\varepsilon \leftarrow \min(\varepsilon_d, \varepsilon_m)$
    for $C \in \mathscr{A}$ **do**
      $y(C) \leftarrow y(C) + \varepsilon$ ▷ Increase dual variables for active clusters.
    end for
    if $\varepsilon_c \leq \varepsilon_m$ **then** ▷ Cluster deactivation next.
      Let $C \in \mathscr{A}$ be such that $\pi(C) - y(\mathscr{L}_{\downarrow C}) = 0$.
      $\mathscr{D} \leftarrow \mathscr{D} \cup \{C\}$ ▷ Mark cluster as inactive.
    else ▷ Cluster merge next
      Let e be such that $c(e) - y(\mathscr{L}(e)) = 0$ and $e \in \delta C$ for some $C \in \mathscr{A}$.
      Let C_1 and C_2 be the endpoints of e in $\mathscr{L}^*$.
      $\mathscr{L} \leftarrow \mathscr{L} \cup \{C_1 \cup C_2\}$ ▷ Merge the two clusters.
      $y(C_1 \cup C_2) \leftarrow 0$
      $E_F \leftarrow E_F \cup \{e\}$ ▷ Add e to the forest.
    end if
    $\mathscr{A} \leftarrow \mathscr{L}^* \setminus \mathscr{D}$ ▷ Update active clusters.
  end while
  Restrict F to the g trees contained in $\mathscr{A}$. ▷ Discard trees spanning inactive clusters.
  while there is a $D \in \mathscr{D}$ such that $|\delta_F D| = 1$ **do** ▷ Pruning phase.
    $V_F \leftarrow V_F \setminus D$ ▷ Remove leaf component in $\mathscr{D}$.
    Remove all edges from E_F with at least one endpoint in D.
  end while
  return F
end function

D.1.2. Analysis

We now show that the forest returned by PCSF-GW has the desired properties: it is a g-forest and satisfies the guarantee in Equation (11). Our analysis follows the overall approach of (Feofiloff et al., 2010).

Lemma 21. *Let $H = (V_H, E_H)$ be a graph and let $A, B \subseteq V_H$ be a partition of V_H. Moreover, let $F = \{T_1, \ldots, T_g\}$ be a g-forest such that each T_i has no leaves in B and is not contained in B. Then*

$$\sum_{v \in A} \deg_F(v) + 2|A \setminus V_F| \;\leq\; 2|A| - 2g \;.$$

Proof. Since each T_i has no leaf in B and is not contained in B, every $v \in V_F \cap B$ satisfies $\deg_F(v) \geq 2$. Therefore,

$$\sum_{v \in V_F \cap B} \deg_F(v) \;\geq\; 2|V_F \cap B| \;.$$

Note that $\sum_{v \in V_F} \deg_F(v) = 2(|V_F| - g)$ because F divides V_F into g connected components. Hence

$$\begin{aligned}\sum_{v \in V_F \cap A} \deg_F(v) \;&=\; \sum_{v \in V_f} \deg_F(v) - \sum_{v \in V_F \cap B} \deg_F(v)\\ &\leq\; 2(|V_F| - g) - 2|V_F \cap B|\\ &=\; 2|V_F \cap A| - 2g \;.\end{aligned}$$

Moreover, $|A| = |A \cup V_F| + |A \setminus V_F|$. Combining this with the inequality above gives

$$\begin{aligned}\sum_{v \in V_F \cap A} \deg_F(v) + 2|A \setminus V_F| \;&\leq\; 2|V_F \cap A| - 2g + 2|A \setminus V_F|\\ &\leq\; 2|A| - 2g \;.\end{aligned}$$

Since $\sum_{v \in A} \deg_F(v) = \sum_{v \in V_F \cap A} \deg_F(v)$, the statement of the lemma follows. □

Lemma 22. *Let $\mathscr{L}$ be a laminar family, let $\mathscr{D} \subseteq \mathscr{L}$ be a sub-family, and let $\mathscr{A} = \mathscr{L}^* \setminus \mathscr{D}$. Let F be a g-forest which is $\mathscr{L}$-connected, $\mathscr{L}^*$-disjoint, has no leaf component in $\mathscr{D}$, and is not contained in $\mathscr{D}$. Then*

$$\sum_{C \in \mathscr{A}} |\delta_F C| + 2\left|\{C \in \mathscr{A} \,|\, C \in \mathscr{L}_{\downarrow \overline{F}}\}\right| \;\leq\; 2|\mathscr{A}| - 2g \;.$$

Proof. Contract each set $C \in \mathscr{L}^*$ into a single node, keeping only edges with endpoints in distinct sets in $\mathscr{L}^*$. Call the resulting graph H and let A and B be the sets of vertices corresponding to $\mathscr{A}$ and $\mathscr{L}^* \cap \mathscr{D}$, respectively.

Note that F is still a g-forest in H. Since F is $\mathscr{L}^*$-disjoint, no trees in T are connected by the contraction process. Moreover, no cycles are created because F is $\mathscr{L}$-connected. Let F' be the resulting g-forest in H. Since F has no leaf component in $\mathscr{D}$, F' has no leaves in B. Furthermore, no tree in F is contained in $\mathscr{D}$ and thus no tree in F' is contained in B. Therefore, F' satisfies the conditions of Lemma 21.

Since F is $\mathscr{L}$-connected, there is a bijection between edges in F' and edges in F with endpoints in distinct elements of $\mathscr{L}^*$. Thus we have

$$\sum_{C \in \mathscr{A}} |\delta_F C| \;=\; \sum_{v \in A} \deg_{F'}(v) \;.$$

Furthermore, the contraction process gives

$$\left|\{C \in \mathscr{A} \,|\, C \in \mathscr{L}_{\downarrow \overline{F}}\}\right| \;=\; |A \setminus V_{F'}|$$

and $|\mathscr{A}| = |A|$. Now the statement of the lemma follows directly from applying Lemma 21. □

Lemma 23. *At the beginning of every iteration of the growth phase in* PCSF-GW *(lines 7 to 24), the following invariant (I) holds:*

Let F be a g-forest which is $\mathscr{L}$-connected, $\mathscr{L}^*$-disjoint, has no leaf component in $\mathscr{D}$, and is not contained in $\mathscr{D}$. Moreover, let $A = \{v_1, \ldots, v_g\}$ be an arbitrary set of g nodes in G and let $\mathscr{B} = \bigcup_{v \in A} \mathscr{L}_{\uparrow\{v\}}$. Then

$$\sum_{e \in E_F} y(\mathscr{L}(e)) + 2y(\mathscr{L}_{\downarrow\overline{F}}) \;\leq\; y(\mathscr{L} \setminus \mathscr{B}) \;. \tag{20}$$

Proof. Clearly, (I) holds at the beginning of the first iteration because the dual values y are 0 for every element in $\mathscr{L}$. We now assume that (I) holds at the beginning of an arbitrary iteration and show that (I) then also holds at the beginning of the next iteration. By induction, this then completes the proof.

Let $\mathscr{L}'$, $\mathscr{D}'$, $\mathscr{A}'$, and y' be the values of $\mathscr{L}$, $\mathscr{D}$, $\mathscr{A}$, and y at the beginning of the iteration. We analyze two separate cases based on the current event in this iteration of the loop: either a cluster is deactivated (lines 15 to 16) or two clusters are merged (lines 18 to 22).

First, we consider the cluster deactivation case. Let F be a g-forest satisfying the conditions of invariant (I). Since $\mathscr{L}' = \mathscr{L}$ and $\mathscr{D}' \subseteq \mathscr{D}$, F is also $\mathscr{L}'$-connected, $\mathscr{L}'^*$-disjoint, has no leaf component in $\mathscr{D}'$, and is not contained in $\mathscr{D}'$. Hence we can invoke Equation (20):

$$\sum_{e \in E_F} y'(\mathscr{L}'(e)) + 2y'(\mathscr{L}'_{\downarrow\overline{F}}) \;\leq\; y'(\mathscr{L}' \setminus \mathscr{B}) \;. \tag{21}$$

Note that y and y' differ only on sets in $\mathscr{A}' = \mathscr{L}'^* \setminus \mathscr{D}'$. Therefore, we have the following three equations quantifying the differences between the three terms in Equations (20) and (21):

- $$\sum_{e \in E_F} y(\mathscr{L}'(e)) - \sum_{e \in E_F} y'(\mathscr{L}'(e)) \;=\; \sum_{e \in E_F} \sum_{C \in \mathscr{A}'} \varepsilon \cdot \mathbb{1}[e \in \delta_F C] \;=\; \varepsilon \sum_{C \in \mathscr{A}'} |\delta_F C| \tag{22}$$
- $$y(\mathscr{L}'_{\downarrow\overline{F}}) - y'(\mathscr{L}'_{\downarrow\overline{F}}) \;=\; \sum_{C \in \mathscr{A}'} \varepsilon \cdot \mathbb{1}[C \in \mathscr{L}'_{\downarrow\overline{F}}] \;=\; \varepsilon \Big|\{C \in \mathscr{A}' \,|\, C \in \mathscr{L}'_{\downarrow\overline{F}}\}\Big| \tag{23}$$
- $$y(\mathscr{L}' \setminus \mathscr{B}) - y'(\mathscr{L}' \setminus \mathscr{B}) \;=\; \sum_{C \in \mathscr{A}'} \varepsilon \cdot \mathbb{1}[C \notin \mathscr{B}] \;=\; \varepsilon|\mathscr{A}'| - \varepsilon|\mathscr{A}' \cap \mathscr{B}| \;\geq\; \varepsilon|\mathscr{A}'| - \varepsilon g \tag{24}$$

In the last inequality, we used the fact that $|A| = g$ and hence $\mathscr{B}$ can contain at most g maximal sets in the laminar family $\mathscr{L}'$. Combining the three equations above with Equation (21) and Lemma 22 then gives:

$$\sum_{e \in E_F} y(\mathscr{L}'(e)) + 2y(\mathscr{L}'_{\downarrow\overline{F}}) \;\leq\; y(\mathscr{L}' \setminus \mathscr{B}) \;. \tag{25}$$

Since $\mathscr{L}' = \mathscr{L}$, this is equivalent to Equation (20), completing the proof for this case.

Now we consider the cluster merge case. As before, let F be a g-forest satisfying the conditions of invariant (I). Since $\mathscr{L} = \mathscr{L}' \cup \{C_1 \cup C_2\}$ and $\mathscr{D} = \mathscr{D}'$, F is also $\mathscr{L}'$-connected, $\mathscr{L}'^*$-disjoint, has no leaf component in $\mathscr{D}'$, and is not contained in $\mathscr{D}'$. Therefore, we can invoke Equation (20) again. Moreover, Equations (22), (23), and (24) also hold in this case. Combining these equations with (21) and Lemma 22 then again results in Equation (25). Furthermore, we have $y(C_1 \cup C_2) = 0$ and thus $y(\mathscr{L}(e)) = y(\mathscr{L}'(e)$, $y(\mathscr{L}_{\downarrow\overline{F}}) = y(\mathscr{L}'_{\downarrow\overline{F}})$, and $y(\mathscr{L} \setminus \mathscr{B}) = y(\mathscr{L}' \setminus \mathscr{B})$. Applying these equalities to Equation (25) completes the proof. □

The following lemma is essential for proving a lower bound on the value of the optimal solution.

Lemma 24. *Let $\mathscr{L}$ be a laminar family with dual values y. Let F be a g-forest and let $\mathscr{B} = \bigcup_{T \in F} \mathscr{L}_{\uparrow T}$. Then*

$$c(E_F) + \pi(\overline{V_F}) \;\geq\; y(\mathscr{L} \setminus \mathscr{B}) \;.$$

Proof. Let $\mathscr{M} = \{C \in \mathscr{L} \mid \delta_F C \neq \{\}\}$ and $\mathscr{N} = \mathscr{L}_{\downarrow \overline{F}}$. Then $\mathscr{L} = \mathscr{M} \cup \mathscr{N} \cup \mathscr{B}$.

Since the y are dual values, we have $c(e) \geq y(\mathscr{L}(e))$ for every $e \in E_F$. Therefore,

$$\begin{aligned} c(E_F) \;=\; \sum_{e \in E_F} c(e) \;\geq\; \sum_{e \in E_F} y(\mathscr{L}(e)) \;&=\; \sum_{e \in E_F} \sum_{C \in \mathscr{L}(e)} y(C) \\ &=\; \sum_{C \in \mathscr{L}} \sum_{e \in \delta_F C} y(C) \;\geq\; \sum_{C \in \mathscr{M}} y(C) \;=\; y(\mathscr{M}) \,. \end{aligned}$$

Moreover, we have $\pi(C) \geq y(\mathscr{L}_{\downarrow C})$ for every $C \in \mathscr{L}$. Thus,

$$\pi(\overline{V_F}) \;\geq\; \sum_{\substack{C \in \mathscr{L}^* \\ C \subseteq \overline{V_F}}} \pi(C) \;\geq\; \sum_{\substack{C \in \mathscr{L}^* \\ C \subseteq \overline{V_F}}} y(\mathscr{L}_{\downarrow C}) \;=\; y(\mathscr{L}_{\downarrow \overline{V_F}}) \;=\; y(\mathscr{N}) \,.$$

Finally, we get

$$c(E_F) + \pi(\overline{V_F}) \;\geq\; y(\mathscr{M}) + y(\mathscr{N}) \;\geq\; y(\mathscr{M} \cup \mathscr{N}) \;=\; y(\mathscr{L} \setminus (\mathscr{L} \setminus (\mathscr{M} \cup \mathscr{N}))) \;\geq\; y(\mathscr{L} \setminus \mathscr{B}) \,,$$

where we used $\mathscr{L} = \mathscr{M} \cup \mathscr{N} \cup \mathscr{B}$ in the final step. □

We can now prove the main theorem establishing an approximation guarantee for PCSF-GW, which also proves Theorem 5 from the main text of the paper.

Theorem 25. *Let F be the result of* PCSF-GW(G, c, π, g). *Then F is a g-forest and*

$$c(F) + 2\pi(\overline{F}) \;\leq\; 2c(F_{OPT}) + 2\pi(\overline{F_{OPT}}) \,,$$

where F_{OPT} is a g-forest minimizing $c(F_{OPT}) + \pi(\overline{F_{OPT}})$.

Proof. By construction in the growth phase of PCSF-GW (lines 7 to 24), F is a $\mathscr{L}$-connected forest at the end of the growth phase. Since at most one element is added to $\mathscr{D}$ in each iteration of the growth phase, we have $|\mathscr{A}| = g$ at the end of the growth phase. Hence restricting F to $\mathscr{A}$ in line 26 leads to F being a g-forest which is still $\mathscr{L}$-connected. Furthermore, F is $\mathscr{L}^*$-disjoint and no tree in F is contained in $\mathscr{D}$.

The pruning phase (lines 27 to 29) maintains that F is a g-forest, $\mathscr{L}$-connected, $\mathscr{L}^*$-disjoint, and not contained in $\mathscr{D}$. Moreover, the pruning phase removes all leaf components of F in $\mathscr{D}$. Hence at the end of the pruning phase, F satisfies the conditions of Lemma 23 ($\mathscr{L}$, $\mathscr{D}$, and $\mathscr{A}$ did not change in the pruning phase).

Now let $F_{OPT} = (T_1^{OPT}, \ldots, T_g^{OPT})$ be a g-forest minimizing $c(F_{OPT}) + \pi(\overline{F_{OPT}})$ and let $A = \{v_1, \ldots v_g\}$ with $v_i \in T_i^{OPT}$. Moreover, let $\mathscr{B}_1 = \bigcup_{v \in A} \mathscr{L}_{\uparrow \{v\}}$ as in Lemma 23. Invoking the Lemma then gives

$$\sum_{e \in E_F} y(\mathscr{L}(e)) + 2y(\mathscr{L}_{\downarrow \overline{F}}) \;\leq\; 2y(\mathscr{L} \setminus \mathscr{B}_1) \,. \tag{26}$$

Now, note that every $e \in E_F$ was added to F when we had $c(e) = y(\mathscr{L}(e))$. Hence

$$\sum_{e \in E_F} y(\mathscr{L}(e)) \;=\; \sum_{e \in E_F} c(e) \;=\; c(F) \,. \tag{27}$$

Moreover, $\overline{V_F}$ can be decomposed into elements in $\mathscr{D}$: after restricting F to $\mathscr{A} = \mathscr{L}^* \setminus \mathscr{D}$ in line 26 this clearly holds. During the pruning phase, all subtrees that are removed from trees in F are elements of $\mathscr{D}$. Therefore, there is a family of pairwise disjoint sets $\mathscr{Z} \subseteq \mathscr{D}$ such that $\bigcup_{C \in \mathscr{Z}} = \overline{V_F}$. Note that for every $C \in \mathscr{D}$ we have $\pi(C) = y(\mathscr{L}_{\downarrow C})$ because C was deactivated at some point in the growth phase. Therefore,

$$\pi(\overline{F}) \;=\; \sum_{C \in \mathscr{Z}} \pi(C) \;=\; \sum_{C \in \mathscr{Z}} y(\mathscr{L}_{\downarrow C}) \;\leq\; y(\mathscr{L}_{\downarrow \overline{F}}) \,. \tag{28}$$

Combining Equations (26), (27), and (28) then gives

$$c(F) + 2\pi(\overline{F}) \;\leq\; 2y(\mathscr{L} \setminus \mathscr{B}_1) \; . \tag{29}$$

We now relate this upper bound to the optimal solution F_{OPT}. Let $\mathscr{B}_2 = \bigcup_{T \in F_{OPT}} \mathscr{L}_{\uparrow T}$ as in Lemma 24. The y are valid dual values due to their construction in PCSF-GW. Thus Lemma 24 gives

$$y(\mathscr{L} \setminus \mathscr{B}_2) \;\leq\; c(F_{OPT}) + \pi(\overline{F_{OPT}}) \; . \tag{30}$$

Note that $\mathscr{B}_2 \subseteq \mathscr{B}_1$ and therefore $y(\mathscr{L} \setminus \mathscr{B}_1) \leq y(\mathscr{L} \setminus \mathscr{B}_2)$. The guarantee in the theorem now follows directly from Equations (29) and (30). □

D.2. A fast algorithm for Goemans-Williamson

We now introduce our fast variant of the GW scheme. To the best of our knowledge, our algorithm is the first practical implementation of a GW-like algorithm that runs in nearly linear time.

On a high level, our algorithm uses a more aggressive and *adaptive* dynamic edge splitting scheme than (Cole et al., 2001): our algorithm moves previously inserted splitting points in order to reach a tight edge constraint quicker than before. By analyzing the precision needed to represent merge and deactivation events in the GW algorithm, we prove that our algorithm runs in $O(\alpha \cdot |E| \log|V|)$ time, where α is the number of bits used to specify each value in the input. For constant bit precision α (as is often the case in practical applications) our algorithm hence has a running time of $O(|E| \log|V|)$. Furthermore, our algorithm achieves the approximation guarantee (11) exactly without the additional $\frac{2}{n^k}$ term present in the work of (Cole et al., 2001). From an empirical point of view, our more aggressive edge splitting scheme produces only very few additional edge pieces: we observed that the number of processed edge events is usually close to $2|E|$, the number of edge events initially created. We demonstrate this empirical benefit in our experiments (see Section D.3).

Since the pruning stage of the GW scheme can be implemented relatively easily in linear time (Johnson et al., 2000), we focus on the moat growing stage here. We also remark that there are algorithms for the PCST problem that achieve a nearly-linear time for *planar* graphs (Bateni et al., 2011; Eisenstat et al., 2012).

D.2.1. Algorithm

Similar to (Cole et al., 2001), our algorithm divides each edge $e = (u, v)$ into two *edge parts* e_u and e_v corresponding to the endpoints u and v. We say an edge part p is *active* if its endpoint is in an active cluster, otherwise the edge part p is *inactive*. The key advantage of this approach over considering entire edges is that *all active edge parts always grow at the same rate*. For each edge part p, we also maintain an *event value* $\mu(p)$. This event value is the total amount that the moats on edge part p are allowed to grow until the next event for this edge occurs. In order to ensure that the moats growing on the two corresponding edge parts e_u and e_v never overlap, we always set the event values so that $\mu(e_u) + \mu(e_v) = c(e)$. As for edges, we define the remaining slack of edge part e_u as $\mu(e_u) - \sum_{C \in \mathscr{C}} y_C$, where $\mathscr{C}$ is the set of clusters containing node u.

We say that an *edge event* occurs when an edge part has zero slack remaining. However, this does not necessarily mean that the corresponding edge constraint has become tight as the edge event might be "stale" since the other edge parts has become inactive and stopped growing since the last time the edge event was updated. Nevertheless, we will be able to show that the total number of edge events to be processed over the course of the algorithm is small. Note that we can find the next edge event by looking at the edge events with smallest remaining slack values in their clusters. This is an important property because it allows us to organize the edge parts in an efficient manner. In particular, we maintain a priority queue Q_C for each cluster C that contains the edge parts with endpoint in C, sorted by the time at which the next event on each edge part occurs. Furthermore, we arrange the cluster priority queues in an overall priority queue resulting in a "heap of heaps" data structure. This data structure allows us to quickly locate the next edge event and perform the necessary updates after cluster deactivation or merge events.

In addition to the edge events, we also maintain a priority queue of *cluster events*. This priority queue contains each active cluster with the time at which the corresponding cluster constraint becomes tight. Using these definitions, we can now state the high-level structure of our algorithm in pseudo code (see Algorithm 6) and then describe the two subroutines MergeClusters and GenerateNewEdgeEvents in more detail.

Algorithm 6 Fast variant of the GW algorithm for PCSF.

function PCSF-FAST(G, c, π, g)
    INITPCST(G, c, π)
    $t \leftarrow 0$ ▷ *Current time*
    $g' \leftarrow |V|$ ▷ *Number of active clusters*
    while $g' > g$ **do**
        ▷ *Returns event time and corresponding edge part*
        $(t_e, p_u) \leftarrow$ GETNEXTEDGEEVENT$()$
        ▷ *Returns event time and corresponding cluster*
        $(t_c, C) \leftarrow$ GETNEXTCLUSTEREVENT$()$
        if $t_e < t_c$ **then**
            $t \leftarrow t_e$
            REMOVENEXTEDGEEVENT$()$
            $p_v \leftarrow$ GETOTHEREDGEPART(p_u)
            ▷ GETSUMONEDGEPART *returns the current moat sum on the edge part*
            ▷ p_u *and the maximal cluster containing* u
            $(s, C_u) \leftarrow$ GETSUMONEDGEPART(p_u)
            $(s', C_v) \leftarrow$ GETSUMONEDGEPART(p_v)
            $r \leftarrow$ GETEDGECOST$(p_u) - s - s'$ ▷ *Remaining amount on the edge*
            if $C_u = C_v$ **then** ▷ *The two endpoints are already in the same cluster*
                continue ▷ *Skip to beginning of while-loop*
            end if
            if $r = 0$ **then**
                MERGECLUSTERS(C_u, C_v)
            else
                GENERATENEWEDGEEVENTS(p_u, p_v)
            end if
        else
            $t \leftarrow t_c$
            REMOVENEXTCLUSTEREVENT$()$
            DEACTIVATECLUSTER(C)
            $g' \leftarrow g' - 1$
        end if
    end while
    PRUNING$()$
end function

MERGECLUSTERS(C_u, C_v) : As a first step, we mark C_u and C_v as inactive and remove them from the priority queue keeping track of cluster deactivation events. Furthermore, we remove the priority queues Q_{C_u} and Q_{C_v} from the heap of heaps for edge events. Before we merge the heaps of C_u and C_v, we have to ensure that both heaps contain edge events on the "global" time frame. If C_u (or C_v) is inactive since time t' when the merge occurs, the edge event times in Q_{C_u} will have become "stale" because the moat on edge parts incident to C_u did not grow since t'. In order to correct for this offset and bring the keys in Q_{C_u} back to the global time frame, we first increase all keys in Q_{C_u} by $t - t'$. Then, we merge Q_{C_u} and Q_{C_v}, which results in the heap for the new merged cluster. Finally, we insert the new heap into the heap of heaps and add a new entry to the cluster deactivation heap.

GENERATENEWEDGEEVENTS(p_u, p_v) : This function is invoked when an edge event occurs, but the corresponding edge constraint is not yet tight. Since the edge part p_u has no slack left, this means that there is slack remaining on p_v. Let $\mathscr{C}_u$ and $\mathscr{C}_v$ be the set of clusters containing u and v, respectively. Then $r = c(e) - \sum_{C \in \mathscr{C}_u \cup \mathscr{C}_v} y_C$ is the length of the part of edge e not covered by moats yet. We distinguish two cases:

1. The cluster containing the endpoint v is active.
   Since both endpoints are active, we expect both edge parts to grow at the same rate until they meet and the edge constraint becomes tight. Therefore, we set the new event values to $\mu(p_u) = \sum_{C \in \mathscr{C}_u} + \frac{r}{2}$ and $\mu(p_v) = \sum_{C \in \mathscr{C}_v} + \frac{r}{2}$. Note that this maintains the invariant $\mu(p_u) + \mu(p_v) = c(e)$. Using the new event values for p_u and p_v, we update the priority queues Q_{C_u} and Q_{C_v} accordingly and then also update the heap of heaps.

2. The cluster containing the endpoint v is inactive.
   In this case, we assume that v stays inactive until the moat growing on edge part p_u makes the edge constraint for e tight. Hence, we set the new event values to $\mu(p_u) = \sum_{C \in \mathscr{C}_u} + r$ and $\mu(p_v) = \sum_{C \in \mathscr{C}_v}$. As in the previous case, this maintains the invariant $\mu(p_u) + \mu(p_v) = c(e)$ and we update the relevant heaps accordingly. It is worth noting our setting of $\mu(p_v)$ reduces the slack for p_v to zero. This ensures that as soon as the cluster C_v becomes active again, the edge event for p_v will be processed next.

Crucially, in GENERATENEWEDGEEVENTS, we set the new event values for p_u and p_v so that the next edge event on e would merge the clusters C_u and C_v, *assuming both clusters maintain their current activity status*. If one of the two clusters changes its activity status, this will not hold:

1. If both clusters were active and cluster C_u has become inactive since then, the next event on edge e will be part p_v reaching the common midpoint. However, due to the deactivation of C_u, the edge part p_u will not have reached the common midpoint yet.

2. If C_v was inactive and becomes active before the edge event for p_u occurs, the edge event for p_v will also immediately occur after the activation for C_v. At this time, the moat on p_u has not reached the new, size-0 moat of C_v, and thus the edge constraint is not tight.

However, in the next section we show that if all input values are specified with d bits of precision then at most $O(d)$ edge events can occur per edge. Moreover, even in the general case our experiments in Section 6 show that the pathological cases described above occur very rarely in practice. In most instances, only two edge events are processed per edge on average.

D.2.2. ANALYSIS

We now study the theoretical properties of our algorithm PCSF-FAST. Note that by construction, the result of our algorithm exactly matches the output of PCSF-GW and hence also satisfies guarantee (11).

First, we establish the following structural result for the growth stage of the GW algorithm (the "exact" algorithm PCSF-GW, not yet PCSF-FAST). Informally, we show that a single additional bit of precision suffices to exactly represent all important events in the moat growth process. The following result is equivalent to Theorem 6.

Theorem 26. *Let all node prizes $\pi(v)$ and edge costs $c(e)$ be even integers. Then all cluster merge and deactivation events occur at integer times.*

Proof. We prove the theorem by induction over the cluster merge and deactivation events occuring in the GW scheme, sorted by the time at which the events happen. We will show that the updates caused by every event maintain the following invariant:

Induction hypothesis Based on the current state of the algorithm, let t_e be the time at which the edge constraint for edge e becomes tight and t_C be the time at which the cluster constraint for cluster C becomes tight. Then t_e and t_C are integers. Moreover, if the merge event at t_e is a merge event between an active cluster and an inactive cluster C, then $t_e - t_{\text{inactive}(C)}$ is even, where $t_{\text{inactive}(C)}$ is the time at which cluster C became inactive.

Clearly, the induction hypothesis holds at the beginning of the algorithm: all edge costs are even, so $t_e = \frac{c(e)}{2}$ is an integer. Since the node prizes are integers, so are the t_C. The assumption on merge events with inactive clusters trivially holds because there are no inactive clusters at the beginning of the algorithm. Next, we perform the induction step by a case analysis over the possible events:

- **Active-active**: a merge event between two active clusters. Since this event modifies no edge events, we only have to consider the new deactivation event for the new cluster C. By the induction hypothesis, all events so far have occured at integer times, so all moats have integer size. Since the sum of prizes in C is also an integer, the new cluster constraint becomes tight at an integer time.
- **Active-inactive**: a merge event between an active cluster and an inactive cluster. Let e be the current edge, t_e be the current time, and C be the inactive cluster. The deactivation time for the new cluster is the same as that of the current active cluster, so it is also integer. Since every edge e' incident to C now has a new growing moat, we have to consider the change in the event time for e'. We denote the previous event time of e' with $t'_{e'}$. We distinguish two cases:
  - If the other endpoint of e' is in an active cluster, the part of e' remaining has size $t'_{e'} - t_e$ and e' becomes tight at time $t_e + \frac{t'_{e'} - t_e}{2}$ because e' has two growing moats. We have
  $$t'_{e'} - t_e \;=\; \big(t'_{e'} - t_{\text{inactive}(C)}\big) - \big(t_e - t_{\text{inactive}(C)}\big)\;.$$
  Note that both terms on the right hand side are even by the induction hypothesis, and therefore their difference is also even. Hence the new event time for edge e' is an integer.
  - If the other endpoint of e' is an inactive cluster, say C', we have to show that $t_{e'} - t_{\text{inactive}(C')}$ is even, where $t_{e'}$ is the new edge event time for e'. We consider whether C or C' became inactive last:
    * C became inactive last: from the time at which C became inactive we know that $t'_{e'} - t_{\text{inactive}(C')}$ is even. Moreover, we have that $t_{e'} = t'_{e'} + (t_e - t_{\text{inactive}(C)})$. Since $t_e - t_{\text{inactive}(C)}$ is even by the induction hypothesis, so is $t_{e'} - t_{\text{inactive}(C')}$.
    * C' became inactive last: from the time at which C' became inactive we know that $t'_{e'} - t_{\text{inactive}(C)}$ is even. The time of the new edge event can be written as $t_{e'} = t_e + t'_{e'} - t_{\text{inactive}(C')}$ (an integer by the induction hypothesis), which is equivalent to $t_{e'} - t'_{e'} = t_e - t_{\text{inactive}(C')}$. We now use this equality in the second line of the following derivation:
    $$\begin{aligned} t_{e'} - t_{\text{inactive}(C')} \;&=\; t_{e'} - t'_{e'} + t'_{e'} - t_e + t_e - t_{\text{inactive}(C')} \\ &=\; 2(t_{e'} - t'_{e'}) + t'_{e'} - t_e \\ &=\; 2(t_{e'} - t'_{e'}) + \big(t'_{e'} - t_{\text{inactive}(C)}\big) - \big(t_e - t_{\text{inactive}(C)}\big)\;. \end{aligned}$$
    Since $t_e - t_{\text{inactive}(C)}$ is even by the induction hypothesis, all three terms on the right hand side are even.
- **Cluster deactivation**: Clearly, a deactivation of cluster C leads to no changes in other cluster deactivation times. Moreover, edges incident to C and another inactive cluster will never become tight based on the current state of the algorithm. The only quantities remaining are the edge event times for edges e with another cluster endpoint that is active. Note that up to time t_C, the edge e had two growing moats and t_e was an integer. Therefore, the part of e remaining has length $2(t_e - t_C)$, which is an even integer. The new value of t_e is $t_C + 2(t_e - t_C)$, and since $t_{\text{inactive}(C)} = t_C$ the induction hypothesis is restored.

Since the induction hypothesis is maintained throughout the algorithm and implies the statement of the theorem, the proof is complete. $\square$

We now use this result to show that the number of edge part events occuring in PCSF-FAST is small.

Corollary 27. *Let all node prizes $\pi(v)$ and edge costs $c(e)$ be specified with α bits of precision. Then the number of edge part events processed in* PCSF-FAST *is bounded by $O(\alpha \cdot |E|)$.*

Proof. We look at each edge e individually. For every edge part event A on e that does not merge two clusters, the following holds: either A reduces the remaining slack of e by at least a factor of two or the event directly preceeding A reduced the remaining slack on e by at least a factor of two. In the second case, we charge A to the predecessor event of A.

So after $O(\alpha)$ edge parts events on e, the remaining slack on e is at most $\frac{c(e)}{2^\alpha}$. Theorem 26 implies that the minimum time between two cluster merge or deactivation events is $\frac{c(e)}{2^{\alpha+1}}$. So after a constant number of additional edge part events on e, the edge constraint of e must be the next constraint to become tight, which is the last edge part event on e to be processed. Therefore, the total number of edge part events on e is $O(\alpha)$. $\square$

We now show that all subroutines in PCSF-FAST can be implemented in $O(\log|V|)$ amortized time, which leads to our final bound on the running time.

Theorem 28. *Let all node prizes $\pi(v)$ and edge costs $c(e)$ be specified with α bits of precision. Then* PCSF-FAST *runs in $O(\alpha \cdot |E| \log|V|)$ time.*

Proof. The requirements for the priority queue maintaining edge parts events are the standard operations of a mergeable heap data structure, combined with an operation that adds a constant offset to all elements in a heap in $O(\log|V|)$ amortized time. We can build such a data structure by augmenting a pairing heap with an offset value at each node. Due to space constraints, we omit the details of this construction here. For the outer heap in the heap of heaps and the priority queue containing cluster deactivation events, a standard binomial heap suffices.

We represent the laminar family of clusters in a tree structure: each cluster C is a node, the child nodes are the two clusters that were merged to form C, and the parent is the cluster C was merged into. The initial clusters, i.e., the individual nodes, form the leaves of the tree. By also storing the moat values at each node, the GETSUMONEDGEPART operation for edge part p_u can be implemented by traversing the tree from leaf u to the root of its subtree. However, the depth of this tree can be up to $\Omega(|V|)$. In order to speed up the data structure, we use path compression in essentially the same way as standard union-find data structures. The resulting amortized running time for GETSUMONEDGEPART and merging clusters then becomes $O(\log|V|)$ via a standard analysis of union-find data structures (with path compression only).

This shows that all subroutines in PCSF-FAST (Algorithm 6) can be implemented to run in $O(\log|V|)$ amortized time. Since there are at most $O(\alpha|E|)$ events to be processed in total, the overall running time bound of $O(\alpha \cdot |E| \log|V|)$ follows. $\square$

D.3. Experimental results

We also investigate the performance of our algorithm PCSF-FAST outside sparse recovery. As test data, we use the public instances of the DIMACS challenge on Steiner tree problems[10]. We record both the total running times and the number of edge events processed by our algorithm. All experiments were conducted on a laptop computer from 2010 (Intel Core i7 with 2.66 GHz, 4 MB of cache, and 8 GB of RAM). All reported running times are averaged over 11 trials after removing the slowest run. Since the GW scheme has a provable approximation guarantee, we focus on the running time results here.

Running times Figure 6 shows the running times of our algorithm on the public DIMACS instances for the unrooted prize-collecting Steiner tree problem (PCSPG). For a single instance, the maximum running time of our algorithm is roughly 1.3 seconds and most instances are solved significantly faster. The scatter plots also demonstrates the nearly-linear scaling of our running time with respect to the input size.

Effectiveness of our edge splitting heuristic As pointed out in our running time analysis, the number of edge part events determines the overall running time of our algorithm. For input values specified with α bits of precision, our analysis shows that the algorithm encounters at most $O(\alpha)$ events per edge. In order to get a better understanding of our

[10] `http://dimacs11.cs.princeton.edu/`

empirical performance, we now look at the number of edge part events encountered by our algorithm (see Figure 7). The scatter plots show that the average number of events per edge is less than 3 for all instances. These results demonstrate the effectiveness of our more adaptive edge splitting heuristics. Moreover, the number of edge events encountered explains the small running times on the large i640 instances in Figure 6.

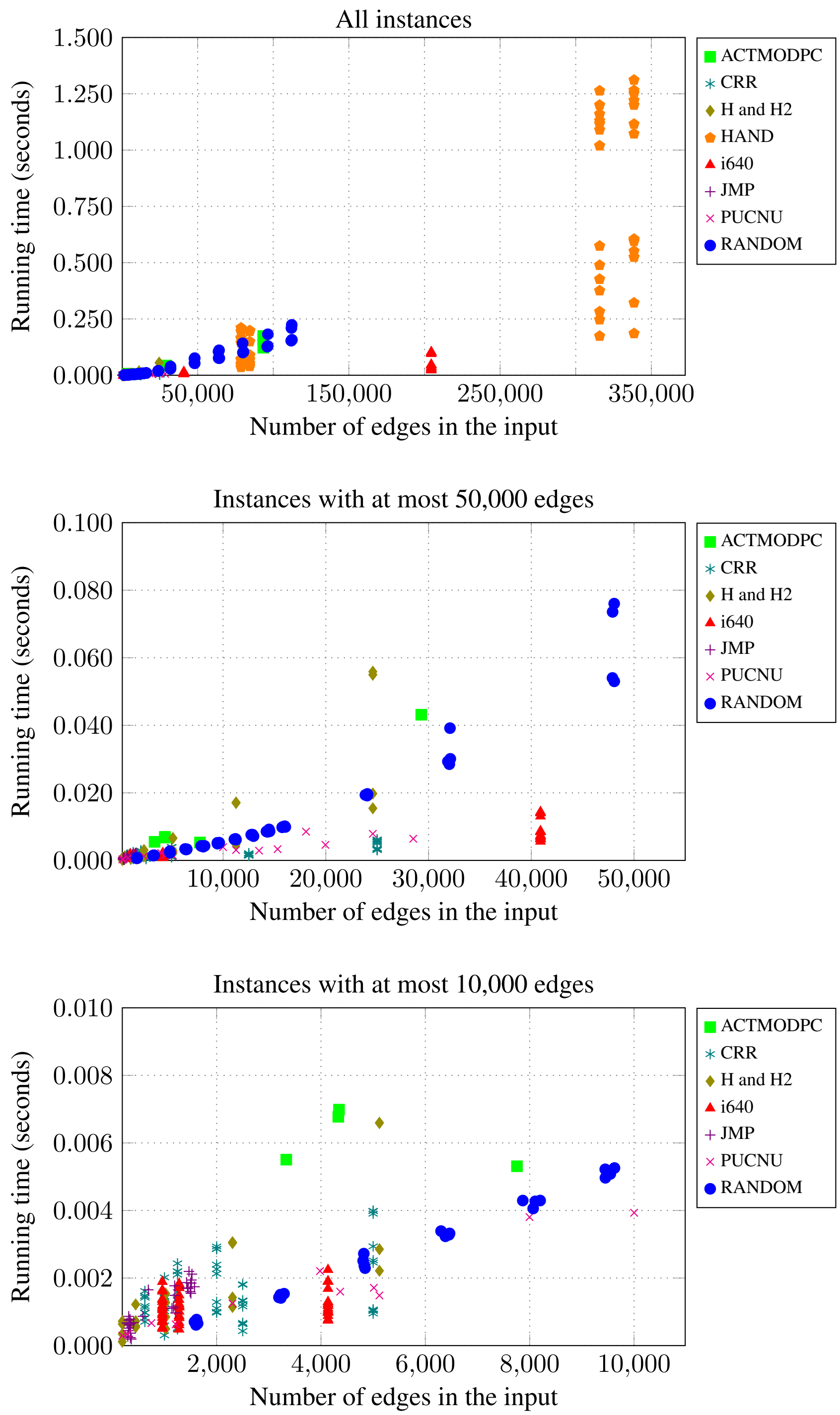


Figure 6. Running times for the PCSPG instances of the DIMACS challenge. Each color corresponds to one test case group. Our algorithm runs for at most 1.3s on any instance and clearly shows nearly-linear scaling with the input size.

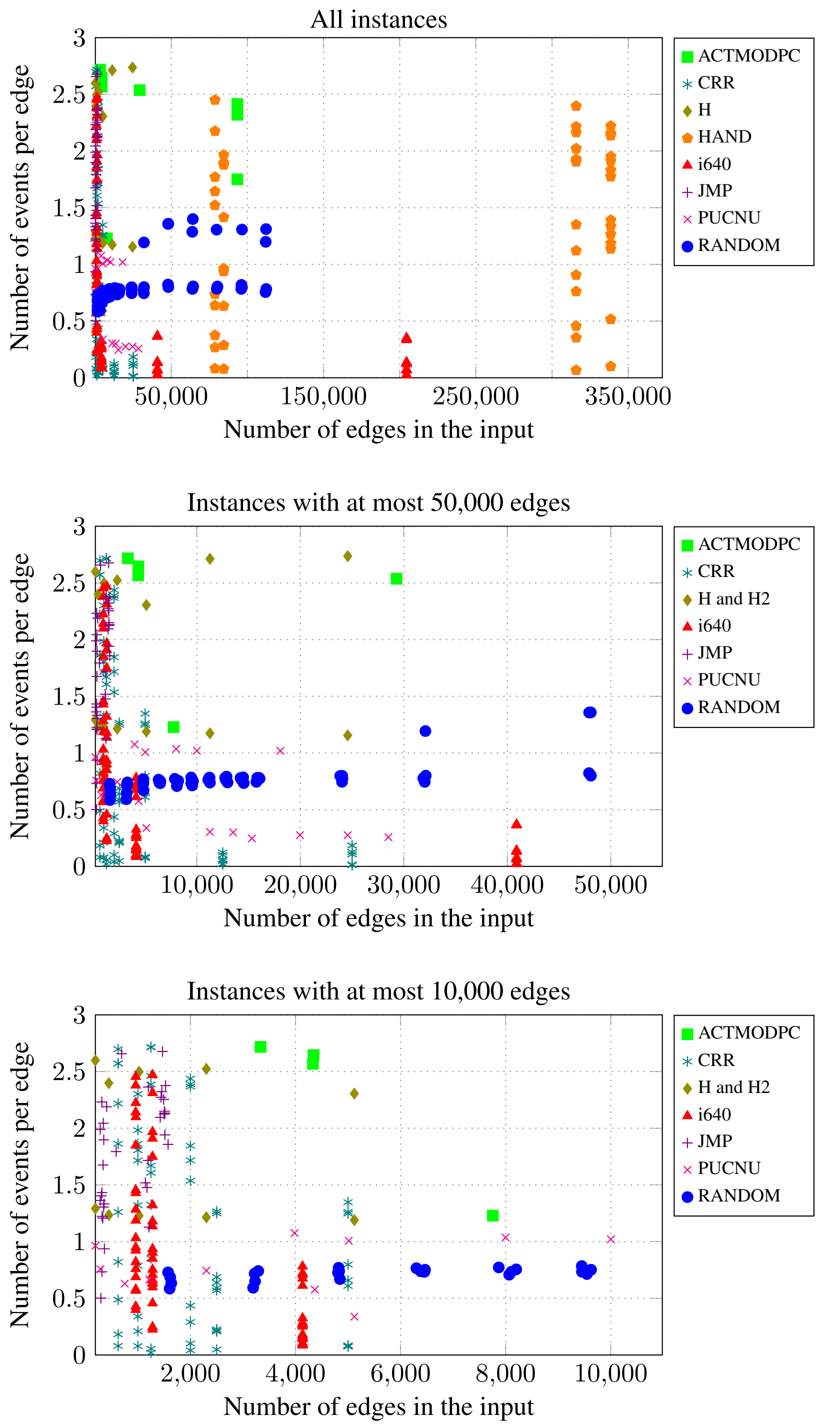


Figure 7. Average number of edge events processed per edge for the PCSPG instances of the DIMACS challenge. Each color corresponds to one test case group. The results demonstrate the effectiveness of our edge splitting approach and show that the average number of edge events is less than 3 for every instance.