Partitioning Into Prescribed Number of Cycles and Mod k T-join With Slack

Jordan Barrett[1]

Department of Mathematics
McGill University
Montreal, Canada

Salomon Bendayan [2]

Department of Combinatorics & Optimization
University of Waterloo
Waterloo, Canada

Yanjia Li [3]

Department of Combinatorics & Optimization
University of Waterloo
Waterloo, Canada

Bruce Reed [4]

Department of Computer Science
McGill University
Montreal, Canada

Abstract

The input to a PPNC instance is integers n and p, and a non-negative real weighting of the edges of the clique K_n on the vertex set $\{1, ..., n\}$. We are asked to find a set of p disjoint cycles spanning $\{1, ..., n\}$ and subject to this such that the sum of the weights of the edges is minimized. We provide an efficient approximation algorithm for the metric version of this problem which has an approximation ratio of 4 if $p \le \frac{n}{5}$ and an approximation ratio of 51 for larger p.

For $p > \frac{n}{5}$, our algorithm uses a subroutine which approximately solves the Mod 3 T-join With Slack problem. The input to an instance of Mod k T-join with Slack consists of integers n and B, a non-negative weighting of the edges of the clique K_n, and a label $l(v)$ from $\{0, 1, ..., k-1\}$ on each vertex of K_n. We are asked to find the minimum weight spanning forest F from amongst those satisfying $\sum_{T \in F}((\sum_{v \in V(T)} l(v)) \mod k) \le B$. If $k = 2$ and $B = 0$ this is the well-studied T-join problem which can be solved exactly in polynomial time.

Keywords: Hamiltonian p-median problem, T-join, Approximation algorithm, Graph partition.

[1] Email: jordan.barrett@mail.mcgill.ca

[2] Email: s2bendayan@uwaterloo.ca

[3] Email: yanjia.li@uwaterloo.ca

[4] Email: breed@cs.mcgill.ca

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

1 Introduction

We focus on partitioning the vertices of a complete graph with weight on its edges into trees or cycles, so as to minimize the total weight of the edges of the partition elements, subject to restrictions on their size and number. In the decision version of such problems, we are given a value k, and asked if the minimum weight partition has weight at most k.

If our only restriction is that the partition has exactly one element, then we obtain a classical problem, in the case of both cycles and trees. In the case of trees this is the minimum weight spanning tree (MWST) problem. Polynomial time algorithms for this problem are presented in undergraduate textbooks such as [6]. For partitioning into cycles, the decision version is the well-known Travelling Salesman Problem (TSP), which is NP-complete [5].

If the only restriction is on the number p of components, then partitioning the graph into trees can be done using a simple modification to Kruskal's algorithm for MWST, which runs in $O(m \log n)$ time. We simply build a forest as in Kruskal's algorithm, stopping when it has p components instead of one.

Finding a partition into p cycles is not so easy and is precisely the problem which sparked our interest in the area and we now define it formally.

Partitioning into a Prescribed Number of Cycles(PPNC)

Input: Integers n and p, and a non-negative real weight function w on the edges of the clique K_n with vertex set $\{1, ..., n\}$.

Objective Find p cycles $\{C_1, \dots, C_p\} \subseteq K_n$ such that $\forall i \neq j$ we have $V(C_i) \cap V(C_j) = \emptyset$, and $\bigcup_{i=1}^{p} V(C_i) = \{1, ..., n\}$ minimizing

$$\sum_{i=1}^{p} \sum_{e \in E(C_i)} w(e)$$

Remark: The version of this problem where p is fixed and not a part of the input was introduced in [1] and is known as the Hamiltonian p-median problem (HpMP). Some authors incorrectly use this term for PPNC e.g. [14].

As previously mentioned, when $p = 1$, PPNC is the NP-complete TSP. When n is divisible by 3, and $p = \frac{n}{3}$, the problem is NP-complete because of a simple reduction from the partition into triangles problem [5]. For this reason we restrict our attention to the metric version of the problem, i.e we insist that for every three vertices, x, y, z, we have $w(xy) + w(yz) \geq w(xz)$. It is not difficult to show that this variant of the problem is NP-complete provided $p < \frac{n}{3} - \sqrt{n}$. Our focus thus becomes finding polynomial time algorithms with good approximation ratios. The case where $p = 1$, i.e. the metric TSP, has been profoundly studied and there is a slew of approximation algorithms available (see [15]). In fact, as is discussed more thoroughly below, we draw inspiration from classical algorithms for metric TSP and attempt to generalize the approach to arbitrary values of p.

Theorem 1.1 *For the metric version of PPNCP we have the following results:*

(i) *There is an 8/3-approximation algorithm with running time $O(n^2 \log n)$ for the restriction to instances with $p = \frac{n}{3}$,*

(ii) *there is a 4-approximation algorithm with running time $O(n^2 \log n)$ for the restriction to instances with $p \leq \frac{n}{5}$, and*

(iii) *there is a 51-approximation algorithm with running time $O(n^4)$ for the restriction to instances with $\frac{n}{3} > p > \frac{n}{5}$,*

Remark: The Euclidean HpMP is well studied [10,11,12,13], but no polynomial-time approximation algorithm for it is known.

In what follows we use $w(H) = \sum_{e \in E(H)} w(e)$ to denote the *weight* of H where H is a subgraph (usually a tree, cycle, forest, or set of cycles). We use OPT to denote the optimal solution of the PPNC instance under discussion.

If we focus on an approximation algorithm for the metric problem then there is a link between the problem of partitioning into cycles and partitioning into trees. As is covered in undergraduate courses, shortcutting a depth-first search traversal of a minimum weight spanning tree so it visits each vertex exactly once before returning to its starting point, yields a TSP tour whose weight is at most twice that of a minimum spanning tree. Since any TSP tour contains a spanning tree, this gives a polynomial 2-approximation algorithm for

[5] An instance of partition into triangles is a graph G. We want to know if its vertex set can be partitioned into triangles. The reduction involves assigning weight 1 to the edges of the graph and a larger weight to the other edges.

TSP [7]. We can improve this to a polynomial time $\frac{3}{2}$-approximation algorithm by finding a minimum weight matching M on the odd degree vertices of a minimum weight spanning tree T, and then shortcutting an Eulerian tour of $M \cup T$ [8].

It is natural to extend this approach to the more general problem of partitioning into p cycles for arbitrary p. We exploit the fact that there is an $O(n^2 \log n)$ time algorithm to find a minimum weight forest containing p trees. It is easy to see that the weight of this forest is a lower bound on OPT. If all the trees in the partition have at least three vertices then we obtain a solution to the PPNC instance which has weight at most 2OPT by shortcutting each tree.

Unfortunately, the optimal partition into trees may contain some of size one or two which cannot be turned into cycles. One way of dealing with this problem, which we use for all values of p except $\frac{n}{3}$, is to first construct a nearly minimum weight forest all of whose trees have size at least three, and then try and massage it, using a shortcutting approach, to obtain a set of p cycles whose weight is not that far from the weight of the forest returned.

It turns out that there is a 2-approximation algorithm due to Goemans and Williamson [4] for the problem of constructing a nearly minimum weight forest all of whose trees have size at least three which can be implemented in $O(n^2 \log n)$ time. We refer to this algorithm as GWALG. GWALG actually returns a solution whose weight is at most the weight of an optimal partition into cycles[6]. If GWALG returns a forest with exactly p trees then we shortcut a DFS traversal of each of them to obtain a solution to PPNC of weight at most 2OPT.

If GWALG returns a forest with more than p trees then we repeatedly add minimal weight edges joining trees until we have exactly p trees. The total weight of the edges added is at most OPT. Shortcutting a DFS traversal of each of the resultant trees yields a partition into cycles whose weight is at most 4OPT. If GWALG returns a forest with fewer than p trees, then we need to cut some large trees into more than one cycle. A tree T can be cut into at most $\lfloor \frac{|V(T)|}{3} \rfloor$ cycles. Furthermore, it turns out that a simple modification of the shortcutting approach discussed above allows us to cut any tree T into a cycles with total weight at most three times the weight of T for any a between 1 and $\lfloor \frac{|V(T)|}{3} \rfloor$.

Our concern then is whether or not the sum of $\lfloor \frac{|V(T)|}{3} \rfloor$ over all the trees in our forest is at least p. For example, if $p = \frac{n}{3}$ we need that the number of vertices in each tree is divisible by three. More generally, we need the number of vertices of each tree mod 3 over all trees in the forest to be at most $n - 3p$ i.e. $\sum_{T \in F}(|V(T)| \mod 3) \le n - 3p$. For $p \le \frac{n}{5}$ this will be true for any forest with fewer than p trees.

Lemma 1.2 *Given a forest F containing t trees, each of size at least 3, if $p \le \frac{n}{5}$ then we can create p trees from F.*

Proof. Each tree can be split into at most $\lfloor \frac{|V(T)|}{3} \rfloor$ smaller trees of size at least three. Thus, since $t > p$ and $p \le \frac{n}{5}$:

$$\sum_{i \in [t]} \lfloor \frac{|V(T_i)|}{3} \rfloor \ge \sum_{i \in [t]} \frac{|V(T_i)| - 2}{3} = \frac{n - 2t}{3} \ge \frac{n - 2p}{3} \ge p$$

□

We then have that if $p \le \frac{n}{5}$, we can apply our modified shortcutting approach directly to the output of GWALG to obtain a solution to PPNC of weight at most 3OPT. For larger values of p it may not be possible to find enough cycles of size at least three.

For $\frac{n}{3} > p > \frac{n}{5}$, we see that OPT is at least the weight of the lowest weight forest F such that (a) F contains no trees with one or two vertices, and (b) $(\sum_{T \in F} |V(T)| \mod 3) \le n - 3p$. We will use as a subroutine an algorithm which, in $O(n^4)$ time, finds a forest satisfying (a) and (b) whose weight is at most 17OPT. Applying the modified shortcutting approach we obtain an efficient 51-approximation algorithm for instances of PPNC with $\frac{n}{3} > p > \frac{n}{5}$.

Our approximation algorithm for finding a forest F satisfying (a) and (b) first finds a forest F_1 satisfying (a) and a forest F_2 satisfying (b) separately. It then returns a spanning forest F of their union. Since the vertex set of every component in F is the union of some components of F_1 (resp. F_2), (a) (resp. (b)) remains true. We find F_1 satisfying (a) which has weight at most OPT, using GWALG. We find a forest F_2 satisfying (b) which has weight at most 16OPT by labelling each vertex with a 1 and applying an approximation algorithm for the following more general problem:

[6] We note that Couëtoux et al [3] obtained a $\frac{3}{2}$-approximation algorithm for the problem of finding a minimum weight partitioning into a set of trees all of size at least three. However, we prefer to use GWALG due to the fact that they solve for a partition into cycles, which allows us to compare it to a solution to PPNC.

Mod 3 T-join With Slack

Input: An integer B, a graph $G = (V, E)$, a non-negative weighting w of E, and a label $l(v) \in \{0, 1, 2\}$, for each v in V such that

$$\sum_{U \text{ a component of } G} (l(U) \mod 3) \leq B$$

Where $l(U) = \sum_{v \in U} l(v)$.

Objective: Find a minimum weight forest F such that

$$\sum_{T \in F} (l(V(T)) \mod 3) \leq B$$

We define Mod k T-join With Slack for any k analogously by replacing 3 with k. A Mod k T-join instance is an instance of Mod k T-join With Slack for which $B = 0$. Although Goemans and Williamson did not name Mod k T-join they did present a 2-approximation algorithm for it with running time $O(n^2 \log n)$ [4]. As we flesh out below, Mod 2 T-join is exactly the well known T-join problem which can be solved exactly in $O(n^3)$ time [2] by solving a related minimum weight matching problem. Furthermore, Mod 2 T-join with slack can also be solved exactly in $O(n^4)$ via matching techniques.

As is explained further in Section 3, a similar result holds for Mod 3 T-join With Slack. The minimum weight matching in an auxiliary hypergraph whose vertices have size two or three, points out a solution whose cost is at most twice that of the optimal solution. However, we do not know how to find this matching. Instead we solve Mod 3 T-join With Slack in $O(n^4)$ time using a greedy procedure. For $p = \frac{n}{3}$, we apply an algorithm which we call Exact-GWALG, which constructs a minimum weight partition into cycles of length exactly k. First presented in [4] for the exact cycle partition problem, the Exact-GWALG has an approximation ratio of $(4(1 - \frac{1}{k})(1 - \frac{1}{n}))$. We thus have for $k = 3$, a collection of p triangles, and a solution of cost at most $8/3$OPT.

This completes the informal description of our algorithm. In Section 2, we specify and give the details of our algorithm for partitioning a tree into cycles. In Section 4 we specify and give the details of our algorithm for solving Mod 3 T-Join With Slack. In Section 5, we formally specify our algorithm for solving PPNC.

2 Splitting a tree into cycles

In this section we discuss an algorithm with the following specifications:

Tree Splitter

Input Integers n and a with $a \in [1, \frac{n}{3}]$, a tree T such that $|V(T)| = n$ and a non-negative weighting w on $E(T)$.

Output: A set S of a cycles partitioning $V(T)$ such that under any metric weighting of the clique on $V(T)$ extending w, the sum of the weights of the cycles of S is at most

$$3w(T) - \sum_{\substack{\{e \in E(T) | e \text{ is incident} \\ \text{to a leaf}\}}} w(e)$$

Running Time Bound: $O(n^2)$.

If $a = 1$, the algorithm simply returns a shortcutting of a DFS traversal of T. Otherwise the algorithm finds a triangle C and a tree T' which is the restriction of T to $V(T) - C$ such that the following property is satisfied

$$3w(T) - \sum_{\substack{\{e \in E(T) | e \text{ is incident} \\ \text{to a leaf}\}}} w(e) \geq w(C) + 3w(T') - \sum_{\substack{\{e \in E(T') | e \text{ is incident} \\ \text{to a leaf}\}}} w(e) \qquad (*)$$

It then recurses on $(n - 3, a - 1, T')$ and returns the union of C and the output of the recursive call. It is not difficult to see that each recursive call can be performed in linear time, which yields an $O(n^2)$ overall bound. We can improve this to $O(n)$ but this would not improve the running time of our algorithms.

To find C, the algorithm finds a leaf u of T with parent s which (i) is as far from the root (an arbitrarily chosen fixed node) as possible, and (ii) subject to this, maximizes the number of siblings it has. If u has a grandparent x subject to (i) and (ii) it maximizes $w(xs)$.

If u has at least two siblings, denoted v, w, then C is the triangle on u, v, w. If u has only one sibling v then C is the triangle on u, s, v. If u has no siblings and s has no siblings then C is a triangle on u, s, x. If u has no sibling and s has a sibling t, which is a leaf, then C is a triangle on u, s, t. In all these cases, C is a shortcutting of the tree formed by $E(T) - E(T')$ so it has weight at most $2(w(T) - w(T'))$ and $(*)$ holds.

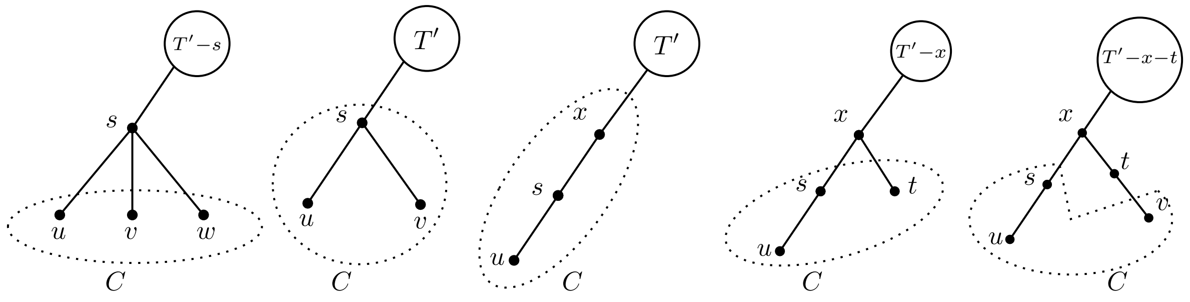

Fig. 1. The possible arrangements of the tree. The vertices making up C are circled.

If none of these cases occur then u has no siblings and there is a sibling t of s. By our choice of u, t has exactly one child v, and $w(xs) \geq w(xt)$. C is the triangle formed by u, s, v. The weight of C is at most $2(w(su) + w(xs) + w(xt) + w(tv)) \leq 2w(su) + 2w(tv) + 3w(xs) + w(xt)$. So, the condition $(*)$ still holds. The possible arrangements of the tree can be seen in Figure 1.

3 Solutions from Matchings in Auxiliary Hypergraphs: From T-Join to Mod 3 T-Join With Slack

The input to an instance of T-join is a graph $G = (V, E)$, a non-negative weighting of E and a set T of vertices of G. We want to find a spanning forest F in which the vertices of odd degree are precisely those in T. Since the sum of the degrees in each tree of F is even, clearly we can only do this if each such tree, and hence also every component of G, contains an even number of vertices of T.

We show now that when $k = 2$, there is an optimal solutions to Mod 2 T-join which is an optimal solution to the T-join problem where T is the set of vertices of label 1. By the remarks of the above paragraph, any solution to the T-join problem is a solution to Mod 2 T-join, so the optimal T-join solution can't have lower weight than the optimal Mod 2 T-join solution. On the other hand, if F is an optimal solution to Mod 2 T-join with the fewest possible number of edges, then for every edge e of each tree T, for each component K of $T - e$, the labels on the edges of K sum to $1 \mod 2$. By applying this to the components of $T - x$ around a vertex x, it follows that x has odd degree in F precisely if it has label 1. Thus F is a solution to the T-join problem, and must solve it optimally.

The polynomial time algorithm to solve T-join involves reformulating it as a problem on an auxiliary graph. Any forest F can be partitioned into edge disjoint paths by repeatedly ripping out paths between a pair of distinct leaves. No vertex appears as an endpoint on two of these paths, and the vertices appearing as endpoints are precisely the vertices of odd degree. On the other hand, if G is the union of any set of edge disjoint paths in which each vertex is used as an endpoint at most once, then (i) the set of vertices of odd degree in G are precisely the set T of endpoints of these paths, and (ii) G is the union of a forest F and a set of edge-disjoint cycles. Note that the set of odd degree vertices in F is also T and that $w(F) \leq w(G)$. Thus, the weight of the optimal solution to a T-join instance is the minimum weight of a set of edge disjoint paths where each vertex of T appears exactly once as an endpoint in this set.

Now, if we take the symmetric difference of the edge sets of two paths with four distinct endpoints, then we still have the edge set of two paths with the same four endpoints, so we can drop the condition that the paths be disjoint here. Thus, we can solve T-join as follows[7]. We call a pair of vertices *relevant* if they lie in the same component of G and are both in T. We first compute, for every relevant pair (s, t) of vertices of G, the weight $w(st)$ of a shortest path from s to t. We form an auxiliary graph G' whose edges are the relevant pairs and where the weight on the edge st is $w(st)$. We then compute a minimum weight matching M with $V(M) = T$. Letting P_{st} be a shortest $s - t$ path, we can obtain our T-join solution by taking the subset of the union of $\{P_{st} \mid st \in M\}$ consisting of those edges appearing in an odd number of paths, and then repeatedly deleting cycles (of weight zero) until we obtain a forest.

It follows in exactly the same way, that the solution for an instance of Mod 2 T-join With Slack, consists of a minimum weight matching of size $\left\lceil \frac{|T|-B}{2} \right\rceil$ in G'.

Our approximation algorithm for Mod 3 T-join With Slack considers a minimum weight matching in a similar auxiliary hypergraph whose edges correspond to the family of pairs and triples of vertices of G. We

[7] Our description is more long-winded then it need be as we want to stress the connection to our algorithm for Mod 3 T-join With Slack.

call elements of this set *pots*[8]. The edges of our weighted auxiliary hypergraphs are the pots. The weight of a pot P, denoted $w(P)$, is the minimum weight of a tree containing the pot. Letting $D = l(V) - B$ we have:

Lemma 3.1 *A spanning forest for a subgraph J of G is a solution to the Mod 3 T-Join instance precisely if*

$$\sum_{\substack{U \text{ a component} \\ \text{of } J}} (l(U) - (l(U) \mod 3)) \geq D$$

Corollary 3.2 *For any set S of disjoint pots satisfying $\sum_{P \in S}(l(P) - (l(P) \mod 3)) \geq D$, any spanning forest for a subgraph J in which the vertices of each element of P lie in the same component is a feasible solution to the Mod 3 T-Join with slack instance. Thus, a spanning forest of a graph which is the union of a minimum weight tree containing P, for each P in S, is such a feasible solution of weight at most $\sum_{P \in S} w(P)$.*

We call a pot *relevant* if its elements lie in the same component of G and the labels of its elements form one of the multi-sets in $\{(1,2),(2,2),(1,1,1),(2,2,2)\}$. These sets of vertices are of interest to us since they are the minimal components such that label of the component is less than the sum of the individual labels of its vertices.

The key to our approximation algorithm for Mod 3 T-join With Slack is the following approximate analogue of the exact result which allows for an exact solution to Mod 2 T-join With Slack:

Lemma 3.3 *If F is an optimal solution to an instance of Mod 3 T-join With Slack then there is a set Z of disjoint relevant pots, each contained in a component of F such that (a) $\sum_{P \in Z}(l(P) - (l(P) \mod 3)) \geq D$, and (b) the sum of the weights of the pots in Z is at most twice the weight of F.*

Remark: By Corollary 3.2, any spanning forest of a graph obtained by (i) constructing for each pot P in Z, a tree of weight $w(P)$ containing the elements of P, and (ii) taking the union of these trees, is a solution to the Mod 3 T-Join With Slack instance with weight at most twice the optimum.

We prove Lemma 3.3 using a similar, but more complicated, procedure than that presented in the last section. We note that we do not require a metric assumption on the edge weighting of G as the weight of a pot is determined by the weight of a minimum weight tree containing its elements so there is no need to shortcut the tree containing the vertices of C. We omit the details of the proof.

Example 3.4 Consider now the graph G made up of 4 vertices lying on a line, labelled $1, 1, 2, 2$ from left to right. Consecutive vertices are at a distance 1 apart. The weight of the edge joining any pair of vertices is given by the Euclidean distance between them. Then, subject to the constraint $B = 0$, the optimal solution will include all edges of the path.

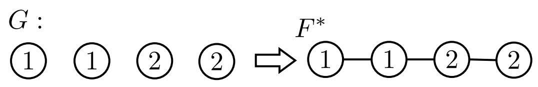

Fig. 2. Example 3.4 with no optimal pot matching solution

As the example above demonstrates, for an instance of Mod 3 T-join With Slack, there may be no matching of relevant pots whose weight is equal to the weight of the optimal solution.

4 Our Algorithm for Mod 3 T-Join With Slack

We now present an algorithm with the following specifications:

***Mod* 3 *T-Join With Slack Solver* Input:** A graph $G = (V, E)$ with a non-negative weighting of E and a labelling of vertices $l(v) \in \{0, 1, 2\}$, a non-negative integer B.

Output: A forest F satisfying $\sum_{T \in F}(l(T) \mod 3) \leq B$ with weight at most 16 times the weight of the minimum weight such forest.

Running time: $O(n^4)$.

While we exploit the fact that there is a matching in our auxiliary hypergraph whose weight is at most twice the weight of an optimum solution, we cannot actually find this matching. Instead, we apply a greedy approach similar to that used in Kruskal's algorithm, although it is implemented back to front rather than front to back. The algorithm proceeds as follows.

[8] The term comes from the first letter of each word : **p**airs **o**r **t**riples.

We begin by creating the auxiliary hypergraph H. Letting $d(x,y)$ be the length of a lightest path from x to y in the weighted graph, $w(xy)$ is simply $d(x,y)$ and $T_{x,y}$ is a lightest $x-y$ path. For $P=\{x,y,z\}$, $w(P)$ is the minimum of $d(u,x)+d(u,y)+d(u,z)$ over all u in V. For T_P we choose the union of shortest paths from u to each of x,z and z for a u which achieves this minimum. Computing all the lightest paths can be done in $O(|V|^3)$ time [9], thus finding the $w(P)$ and T_P can be done in $O(|V|^4)$ time.

Our greedy iterative procedure maintains a set Z of pots and a sub-hypergraph H_Z whose edges are the elements of Z. Initially Z is the entire hypergraph. The recursive algorithms orders the relevant pots by their weights in non-increasing order and examines them accordingly. We define a function to calculate the amount "saved" by a collection of pots:

$$f(Z)=\sum_{\substack{U \text{ a component}\\ \text{of } H_Z}} (l(U)-(l(U) \mod 3))$$

When considering a pot P, the algorithm computes $f(Z-P)$ and determines whether $f(Z-P)\geq D$. If so, P is deleted from Z, otherwise, P is kept and we move onto the next pot. When the algorithm finishes, it first constructs the graph J formed by the union of $\{T_P \mid P\in Z\}$. It then constructs and returns a spanning tree F of J.

Claim 4.1 *The output of our algorithm F is a solution of cost at most $16OPT$.*

To prove our claim, we consider a matching M of relevant pots which points out a solution with cost at most 2OPT. We enumerate M as $P_1,\dots,P_m$ where P_i is the $(m+1-i)^{th}$ pot of M considered by the algorithm. We let w_i be the weight of P_i and note that $w(M)=mw_1+\sum_{j=2}^{m}(m+1-j)(w_j-w_{j-1})$. We note that we will discard all the pots of weight greater than w_m. So, letting a_j be the number of pots of Z of weight exceeding w_{j-1}, the cost of our solution is at most $w_1|Z|+\sum_{j=2}^{m}a_j(w_j-w_{j-1})$. To complete the proof of our claim it remains to show that $|Z|\leq 8m$ and for every j, we have $a_j\leq 8(m+1-j)$. We note that $m\geq\lceil\frac{D}{6}\rceil$, and for every j, letting

$$D_j=\sum_{t=j}^{m}(l(P_t)-(l(P_t) \mod 3))$$

we have that $(m+1-j)\geq\lceil\frac{D_j}{6}\rceil$.

To bound the size of Z, we consider the graph H' which has vertex set $V(G)$ and in which two vertices are adjacent if they lie in a pot of Z together. We let F' be a spanning forest of H'. We observe that for every pot of Z there must be an edge of F' contained in P that does not appear in any other pot. As otherwise we would have discarded P. Thus letting c be the number of components of F', $|Z|\leq|V(F')|-c$. We observe further that since every non-singleton component of F' contains a relevant pot, F' contains at most $\frac{D}{3}$ non-singleton components. Moreover, since for any pot P, $f(Z-P)\geq f(Z)-6$, we must have $f(Z)\leq D+5$.

It follows that if d is the number of non-singleton components of F' then the sum of the labels of the vertices in these components is at most $D+5+2d\leq\frac{4D+15}{3}+d$ Hence the number of vertices in the non-singleton components is also at most $\frac{4D+15}{3}+d$ and hence $|Z|\leq\frac{4D+15}{3}$. This yields that $|Z|\leq 8m+5$.

Case 1: There is no pot P of Z for which there is only one edge of F' contained in no pot of $Z-P$.

This means $|Z|\leq\frac{|V(F')|-c}{2}$, which means $|Z|\leq|V(F')|-c-|Z|$. Replacing the inequality $|Z|\leq|V(F')|-c$ by $|Z|\leq|V(F')|-c-|Z|$ in our earlier derivation, we get $|Z|\leq\frac{4D+15}{3}-|Z|$. So we can assume that $|Z|$ is at most 4. We are done since if m is 0, so is $|Z|$.

Case 2: There is a pot P of Z for which there is only one edge of F' that is not contained in any pot of $Z-P$.

This implies that H_{Z-P} is obtained from H_Z by splitting some component into two, so $f(Z-P)\geq f(Z)-3$ so $f(Z)\leq D+2$ and hence we can replace $D+5$ by $D+2$ in our derivation above to obtain $|Z|\leq\frac{4d+6}{3}$.

Now if U is a component of F' for which the labels on its vertices sum to 2 mod 3 then for any edge e of U, our choice of Z ensures that there is no pot P' for which e is the only edge of F' contained in no pot $Z-P'$. So for each such component our bound on $|Z|$ is decreased by $\lceil(|U|-1)/2\rceil$. This means that there is at most one such component which has at most 3 vertices and we have obtained that $|Z|\leq\frac{4d+3}{3}$. This implies that there is no vertex of label 2 in F', as otherwise the sum of the labels on the vertices of F' actually exceeds the number of vertices of F' by 1, and putting this into the derivation, we are done. But this implies that the sum of the labels on U is 2 which is impossible as it contains the vertices of a pot. It follows that no such U exists.

Hence we can replace the occurrence of $2d$ in the derivation by d thereby improving our bound on $|Z|$ by d. This means that F' has a unique component or we are done. Furthermore, we have improved our bound to

$|Z| \leq \frac{4d+3}{3}$. So, the sum of the weights of the vertices in F' is 1, or we can replace $2d$ by 0 instead of d and we are done. Moreover, we can assume there is no vertex with label 2, or we again improve our bound by 1 and are done. So for an edge e incident to a leaf of F', $F' - e$ has two components. One is a singleton which has label 1, and another for which the sum of the labels on the vertices is 0. So, by our choice of Z, there is no pot P' for which there is no edge of $F' - e$ not contained in any of $Z - P'$. This improves our bound on $|Z|$ by 1, and we are done. In the same way, we can show that $a_j \leq 8(m+1-j)$ for every j and the claim is proved.

5 The Specification of our Algorithm to Solve PPNC

Having described its subroutines, we can now present the details of our algorithm to solve PPNC. Its output and running time depend on p and are given by Theorem 1.1.

Algorithm 1 PPNCSolver($G = (V, E), n, p$)

```
1: Let $C = \emptyset$ be our result set, $F = (V, \emptyset)$ be a forest.
2: if $p = n/3$ then
3:     Apply Exact-GWALG, add the output cycles to $C$.
4: else
5:     Let $k = 3$. Apply GWALG to $G$, get a forest $F_1$. Let $t_1$ be the number of trees in $F_1$.
6:     if $t_1 < p$ then
7:         Calculate $L(F_1) := \sum_{T \in F_1, T:tree}((\sum_{v \in V(T)} l(v)) \mod 3)$
8:         if $L(F_1) \leq n - 3p$ then
9:             Set $F = F_1$.
10:        else
11:            Call Mod3$T$-JoinWithSlackSolver$(G, n-3p)$. Let the returned forest be $F_2$
12:            Set $F = F_1 \cup F_2$. Let $t_1$ now denote the number of components of $F$
13:        Let $t = p - t_1$.
14:        for $1 \leq i \leq t_1$ do
15:            Take any tree $T_i \in F$, delete it from $F$. Let $a = \min\{\lfloor \frac{|V(T)|}{3} \rfloor, t+1\}$.
16:            Call TreeSplitter$(n, a, T)$. Add returned cycles to $C$.
17:            $t \leftarrow t - a + 1$
18:    else
19:        Construct an array $K$ indexed by $V$.
20:        Initialize $K[v]$ to be the index of the component containing $v$.
21:        Order the edges by weight in non-decreasing order.
22:        while $t_1 > p$ do
23:            if  for the next edge $e = (u, v)$ on the list $K[u] \neq K[v]$ then
24:                Add $e$ to $F_1$. Set $t_1 \leftarrow t_1 - 1$. Update $K$.
25:        Set $F = F_1$
26:        for every tree $T_i$ in $F$ do
27:            Shortcut $T_i$ to get a cycle $C_i$. Add $C_i$ to $C$.
28: return  $C$
```

References

[1] Branco, I. and J. Coelho, *The Hamiltonian p-Median Problem*, European Journal of Operational Research. **47** (1990), 85-95

[2] Cook, William J., William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver, "Combinatorial Optimization," John Wiley & Sons, Inc. (1998).

[3] Couëtoux, Basile, James M. Davis, and David P. Williamson, *A 3/2-approximation algorithm for some minimum-cost graph problems*, Math. Program. **150** (2015), 19-34.

[4] Goemans, Michel X. and David P. Williamson, *A General Approximation Technique for Constrained Forest Problems*, SIAM J. Computing **24** (1995), 296-317.

[5] Sahni, Sartaj and Teofilo Gonzalez, *P-Complete Approximation Problems*, J. ACM **23** (1976), 555-565.

[6] Kleinberg, J. and E. Tardos, "Algorithm Design," Pearson Education (2006).

[7] Rosenkrantz, Daniel J., Richard E. Stearns, and Philip M. Lewis, II, *An analysis of several heuristics for the traveling salesman problem*, SIAM Journal on Computing **6** (1977).

[8] Christofides, Nicos, “Worst-case analysis of a new heuristic for the travelling salesman problem,” Carnegie-Mellon University, Pittsburgh Pa. Management Sciences Research Group, (1976).

[9] Floyd, Robert W., *Algorithm 97: Shortest Path*, Commun. ACM **5** (1962), 345-350.

[10] Marzouk, Ahmed M. and Erick Moreno-Centeno, and Halit Üster, *A Branch-and-Price Algorithm for Solving the Hamiltonian p-Median Problem*, INFORMS Journal on Computing **28** (2016), 674-686.

[11] Gouveia, Luís, Tolga Bektaş, and Daniel Santos, *Revisiting the Hamiltonian p-median problem: A new formulation on directed graphs and a branch-and-cut algorithm*, European Journal of Operational Research **276** (2019), 40-64.

[12] Erdoğan, Güneş, Gilbert Laporte, and Antonio M. Rodríguez Chía, *Exact and heuristic algorithms for the Hamiltonian p-median problem*, European Journal of Operational Research **253** (2016), 280-289.

[13] Glaab, Holger, and Alexander Pott, *The Hamiltonian p-Median Problem*, Electr. J. Comb. **7** (2000).

[14] Gollowitzer, Stefan and Gouveia, Luis and Laporte, Gilbert and Pereira, Dilson Lucas and Wojciechowski, Adam, *A comparison of several models for the hamiltonian p-median problem*, Networks **63** (2014), 350-363.

[15] Beresneva, E.N.. and Sergey Avdoshin, *The Metric Travelling Salesman Problem: The Experiment on Pareto-optimal Algorithms.* Proceedings of the Institute for System Programming of the RAS. **29** (2017), 123-138.