A 3-Approximation Algorithm for a Particular Case of the Hamiltonian $p$-Median Problem

Michel Wan Der Maas Soares
michewandermaas@gmail.com

Dilson Lucas Pereira
dilson.pereira@ufla.br
Departamento de Computação Aplicada
Universidade Federal de Lavras
2019

Abstract

Given a weighted graph $G$ with $n$ vertices and $m$ edges, and a positive integer $p$, the Hamiltonian $p$-median problem consists in finding $p$ cycles of minimum total weight such that each vertex of $G$ is in exactly one cycle. We introduce an $O(n^6)$ 3-approximation algorithm for the particular case in which $p \leq \lceil \frac{n-2}{3} \rceil$. An approximation ratio of 2 might be obtained depending on the number of components in the optimal 2-factor of $G$. We present computational experiments comparing the approximation algorithm to an exact algorithm from the literature. In practice much better ratios are obtained. For large values of $p$, the exact algorithm is outperformed by our approximation algorithm.

Keywords: Hamiltonian $p$-median problem, approximation algorithms, 2-factor, matching

1 Introduction

Given a graph $G = (V, E)$ with $n$ vertices and $m$ edges, a positive integer $p$, and a set of edge weights $\{c_e \in \mathbb{R}_+: e \in E\}$, the Hamiltonian $p$-median problem (HpMP) asks for $p$ cycles of minimum total weight, such that each vertex of $G$ is in exactly one cycle. A cycle is defined as a sequence of vertices $(v_1, v_2, \ldots, v_k, v_1)$ such that $k \geq 3$, $\{v_k, v_1\} \in E$, $\{v_i, v_{i+1}\} \in E$ for $1 \leq i < k$, and $v_i \neq v_j$, for $1 \leq i < j \leq k$. Note that the problem has a feasible solution only if $p \leq \lceil \frac{n}{3} \rceil$. In what follows, we assume that $E$ is complete and the edge weights satisfy the triangle inequality.

A particular case of the HpMP, when $p = 1$, is the traveling salesman problem (TSP), one of the best known NP-Hard combinatorial optimization problems. Consequently, the HpMP is also NP-Hard. In fact, the TSP can be reduced to the HpMP for any value of $p$.

The HpMP was introduced by Branco and Coelho. Formulations and exact approaches were studied in [2, 3, 5, 6, 7, 8]. A polyhedral study was conducted in [9]. An iterated local search heuristic was introduced in [10]. Not many heuristic approaches have been proposed for the problem.

Since polynomial time approximation algorithms exist for the TSP, a natural research question is whether such algorithms exist for the HpMP. In this paper, we introduce a 3-approximation algorithm that runs in $O(n^6)$ for the particular case of the HpMP in which $p \leq \lceil \frac{n-2}{3} \rceil$. To the best of our knowledge, this is the first approximation algorithm proposed for the problem.

The rest of the paper is organized as follows. The approximation algorithm is introduced in Section 2, along with complexity and approximation results. In Section 3 we present some experimental results, comparing the approximation to an exact algorithm. Concluding remarks are given in Section 4.
2 The Approximation Algorithm

Before presenting the algorithm, we introduce some notation. The notation $S \subseteq G$ is used to indicate that $S$ is a subgraph of $G$. Given $S \subseteq G$, we let $V(S)$ denote its set of vertices and $E(S)$ denote its set of edges. We define the weight of $S$ as $c(S) = \sum_{e \in E(S)} c_e$. We refer to cycles containing exactly $k$ vertices as $k$-cycles. The optimal HpMP solution is denoted $H^*$.

The approximation algorithm we propose is based on the concept of 2-factors. A 2-factor of a graph is a set of cycles such that each vertex is in exactly one cycle. A related concept is that of 2-matchings, a 2-factor having minimum total weight can be found in polynomial time, using matching techniques. One possible approach, presented in [2], is reproduced in Section 2.1. As any feasible HpMP solution is a 2-factor, the minimum weight 2-factor yields a lower bound on the optimal HpMP solution.

The approximation algorithm is presented in Algorithm 1.

Algorithm 1 HpMP approximation algorithm.

1: Find a minimum weight 2-factor $F$ of $G$. Note that $c(F) \leq c(H^*)$. Let $q$ be the number of cycles in $F$.
2: if $q = p$ then
3: $H \leftarrow F$.
4: else
5: if $q > p$ then
6: Find a spanning tree $T$ of $G$, remove its $p-1$ edges with largest weight. Note that after the removal $c(T) \leq c(H^*)$ and the number of components in $T$ is $p$, but some of its components might be singletons.
7: Since $T$ has $p$ components, its union with $F$ will create a graph with $p$ or less components. Add to $F$ edges of $T$ connecting different components of $F$ until it has $p$ components. Note that after this $c(F) \leq 2c(H^*)$.
8: Transform $F$ into a multigraph by duplicating the edges that were added from $T$, $c(F) \leq 3c(H^*)$.
9: else if $q < p$ then
10: while $F$ does not have $p$ components do
11: Select a component of $F$ with at least six vertices. For the special case of $p \leq \lceil \frac{n-2}{3}\rceil$ such a component always exists. This component is either a cycle $(v_1, v_2, \ldots, v_k, v_1)$ or a path $(v_1, v_2, \ldots, v_k)$. Irrespective of the case, replace it by two paths $(v_1, v_2, v_3)$ and $(v_4, \ldots, v_k)$. Note that this does not increase $c(F)$, i.e., $c(F) \leq c(H^*)$.
12: end while
13: Transform $F$ into a multigraph by duplicating all of its edges, $c(F) \leq 2c(H^*)$.
14: end if
15: Note that $F$ is a multigraph with $p$ connected components in which the degree of each vertex is even. The approximate solution $H$ is obtained by following an Eulerian tour for each component of $F$, skipping vertices already visited.
16: end if
17: Return $H$. If case $q = p$ was executed $H$ is an optimal solution. If case $q > p$ was executed, $H$ is a $3$-approximation. If case $q < p$ was executed, $H$ is a $2$-approximation.

For general values of $p$, it might happen that in step 11 the number of components in $F$ is less than $p$ and there are no components with six or more vertices. As an example, if $n = 10$, $p = 3$, and the optimal 2-factor has 2 connected components with 5 vertices each, the algorithm will fail. This type of problem is caused by components of $F$ whose number of vertices is not a multiple of 3. When splitting components in step 11, a component with $k$ vertices will have $k \mod 3$ vertices that cannot be left alone, as they are not enough to form a cycle. If we try to connect them to vertices coming from other components we might violate the approximation ratio guarantee. For the special case we consider, $p \leq \lceil \frac{n-2}{3}\rceil$, a component with at least six vertices will always exist in step 11. This is proven below, we will need the following result first:

Lemma 1. Given a 2-factor $F$ of $G$ composed of $q$ cycles $\{C_1, \ldots, C_q\}$ let $l(F) = \sum_{i=1}^{q}|V(C_i)| \mod 3$ be the number of vertices that cannot form cycles by themselves in step 11 of Algorithm 1. Then, the maximum possible value of $l(F)$ over all 2-factors of $G$ is not greater than $2 \lceil \frac{n}{3} \rceil$. 

2
Proof. We first show that the maximum of $l(F)$ is attained when $F$ is a 2-factor having the maximum possible number of 5-cycles. Let $F^*$ be a 2-factor attaining the maximum value of $l$. Let $r$ be the size of the largest cycle of $F^*$. If $F^*$ is not a 2-factor with the maximum possible number of 5-cycles, we can always construct another two factor from $F^*$ having one more 5-cycle than $F^*$ without decreasing $l(F^*)$. If $r \geq 8$, replace the corresponding cycle by two cycles, one containing 5 and the other containing the remaining vertices. If $r \leq 7$, there must be another cycle with number of vertices $r'$ such that $r' \leq 7$ and $r' \neq 5$, otherwise $F^*$ has the maximum possible number of 5-cycles. Now, it is a matter of examining the possibilities and showing that for each of them we can construct another 2-factor having more 5-cycles than $F^*$ yielding the same value (or a larger value) of $l(F^*)$. The possibilities are:

- $r = 7$ and $r' = 7$, we can generate cycles with 5, 5, and 4 vertices, respectively.
- $r = 7$ and $r' = 6$, we can generate cycles with 5, 5, and 3 vertices, respectively.
- $r = 7$ and $r' = 4$, we can generate cycles with 5 and 6 vertices, respectively.
- $r = 6$ and $r' = 6$, we can generate cycles with 5 and 5 vertices, respectively.
- $r = 6$ and $r' = 4$, we can generate cycles with 5 and 4 vertices, respectively.
- $r = 4$ and $r' = 4$, we can generate cycles with 5 and 3 vertices, respectively.
- $r = 4$ and $r' = 3$, $F^*$ already has the maximum number of 5-cycles.

Now, if $F$ has the maximum number of 5-cycles, than it has at least $\left\lceil \frac{n}{5} \right\rceil - 1$ 5-cycles. The possible values for the remaining $n - \left\lceil \frac{n}{5} \right\lceil - 1$ vertices are 5, . . . , 9. If there are 5 remaining vertices, the maximum of $l(F)$ is attained by organizing them into a 5 cycle, in which case $l(F) = 2 \left\lceil \frac{n}{5} \right\rceil$. If there are 6 remaining vertices, the maximum of $l(F)$ is attained by organizing them into a 6-cycle, in which case $l(F) = 2 \left\lceil \frac{n}{5} \right\rceil - 1$. If there are 7, the maximum is attained by organizing them into a 7-cycle, $l(F) = 2 \left\lceil \frac{n}{5} \right\rceil + 1$. If there are 8, by organizing them into a 5-cycle and a 3-cycle, $l(F) = 2 \left\lceil \frac{n}{5} \right\rceil$. And if there are 9, by organizing them into a 5-cycle and a 4-cycle, $l(F) = 2 \left\lceil \frac{n}{5} \right\rceil + 1$. In all these cases, $l(F) \leq 2 \left\lceil \frac{n}{5} \right\rceil$.

Proposition 1. It is possible to split any 2-factor of a graph $G$ into at least $\left\lceil \frac{n-2\left\lceil \frac{n}{5} \right\rceil}{3} \right\rceil$ connected components containing at least 3 vertices each.

Proof. Given a 2-factor $F$ of $G$ composed of $q$ cycles $\{C_1, \ldots, C_q\}$, the maximum number of connected components with at least 3 vertices it can be split into is $\frac{n-2l(F)}{3} \geq \left\lceil \frac{n-2\left\lceil \frac{n}{5} \right\rceil}{3} \right\rceil$, since $l(F) \leq 2 \left\lceil \frac{n}{5} \right\rceil$ and the number of resulting components is integer.

Thus, for $p \leq \left\lceil \frac{n-2\left\lceil \frac{n}{5} \right\rceil}{3} \right\rceil$, Algorithm 1 is guaranteed to find a feasible solution.

Theorem 1. For the special case of the HpMP with $p \leq \left\lceil \frac{n-2\left\lceil \frac{n}{5} \right\rceil}{3} \right\rceil$, Algorithm 1 finds a solution $H$ with $c(H) \leq 3c(H^*)$ in $O(n^6)$ time.

Proof. The dominating term in the computational complexity of Algorithm 1 is given by step 1. The minimum weight 2-factor of a graph $G$ can be found by executing a minimum weight perfect matching (MWPM) algorithm on an auxiliary graph $G'$ with $O(n^2)$ vertices and $O(m)$ edges, obtained from $G$. The MWPM of a graph $G$ can be found in $O(n^2m)$ time, thus, finding the MWPM of $G'$ is $O(n^6)$ as we assume $E$ to be complete. The approximation factor was proven along the algorithm description.

For the sake of completeness, in the next section we present a scheme for finding a minimum weight 2-factor of $G$. 
2.1 Finding minimum weight 2-factors of $G$

Minimum weight 2-factors of $G$ can be found on a appropriately defined auxiliary graph, as described in [2]. First, define a graph $G' = \{V', E'\}$: Replace each edge $e = \{u, v\}$ of $E$ by two new vertices $a^u_e$ and $a^v_e$ and three new edges $\{u, a^u_e\}$, $\{a^u_e, a^v_e\}$, and $\{a^v_e, v\}$.

Now, define a graph $G'' = \{V'', E''\}$ from $G'$: For each original vertex $v \in V \cap V'$, replace it by two new vertices $b^u_v$ and $b^v_v$. Replace every edge $\{v, a^v_e\}$ by two new edges $\{b^v_v, a^v_e\}$ and $\{b^u_v, a^v_e\}$, both with weight $c''_{\{b^v_v, a^v_e\}} = c''_{\{b^u_v, a^v_e\}} = \frac{2}{p}$. Let $c''_{uv} = 0$ for any other edge in $E''$.

An edge $e = \{u, v\} \in E$ is in the minimum weight 2-factor of $G$ if and only if $\{b^u_v, a^v_e\}$ or $\{b^v_v, a^v_e\}$ is in the minimum weight perfect matching of $G''$, in which case either $\{b^v_v, a^v_e\}$ or $\{b^u_v, a^v_e\}$ will also be in the matching.

3 Computational Experiments

In this section, we present results from computational experiments that were conducted to evaluate the practical approximation ratio of the algorithm introduced here, as well as its performance in comparison to an exact algorithm.

We consider the instances with 100 vertices among those proposed in Gollowitzer et al. [5], they comprise 5 complete Euclidean graphs whose vertices were randomly generated by sampling 100 points in the square $[0, 100]^2$. These instances are numbered 1-5. Besides these 5 instances, 5 more instances were generated following the same scheme. These instances are numbered 6-10. Since $n = 100$ in all instances, the special HpMP case for which our algorithm was devised consists of the instances with $p \leq 20$. Besides values of $p$ in this range we also considered larger values, in order to evaluate the algorithm’s behavior for more general values of $p$. The following values of $p$ were considered: $\{2, 10, 18, 26, 33\}$.

Gollowitzer et al. [6] propose several models from the HpMP. The exact algorithm we use in the experiments is the one based on “model 1”, the best performing algorithm in that reference. We had access to its source code and ported it to the version 12.63 of the IBM CPLEX solver.

The computational experiments were carried in an Intel XEON machine with 8GB of RAM and 8 cores running at 3.5GHz, running under Linux Ubuntu 14.04. Both the exact and the approximation algorithm were implemented in C++ and evaluated on this same computational environment. A time limit of 1 hour was imposed for the exact algorithm. The source code for the approximation algorithm can be found in [1].

The computational results are presented in Tables 4 and 5. Table 4 presents results for $p \in \{2, 10, 18\}$ while Table 5 presents results for $p \in \{26, 33\}$. Under the heading “Instance” we present instance information: the value of $p$ and a number identifying the instance. Under the heading “Exact Algorithm” we present results for the exact algorithm: the lower ($lb$) and upper bound ($ub$) upon the algorithm termination, and the total computational time in seconds ($t(s)$). Under the heading “Approximation Algorithm”, we present results for the approximation algorithm: the obtained upper bound ($ub$), the total computational time ($t(s)$), the approximation factor ($apx$) in relation to the lower bound provided the exact algorithm and the number of cycles ($q$) in the 2-factor computed in step 1 of Algorithm 1.

In terms of performance, the approximation algorithm took between 2.4 and 3.1 seconds to solve the instances, with no significant difference in time across the different values of $p$. The exact algorithm, on the other hand, was not able to find the optimal solution for large values of $p$ under the time limit of one hour, and for the smallest values the time taken varied significantly from instance to instance.

In terms of solution quality, the approximation algorithm achieved practical approximation ratios between 1.01 and 1.41. The exact algorithm was able to find the optimal solution for low values of $p$, and obtained relatively small optimality gaps for instances with mid-range values of $p$. However, under the time limit of one hour, for the $p$ value of 33, the approximation algorithm was able to find better solutions for all instances.

On average, the value of $q$ was 13.7, which explains the fact that a better practical approximation ratio was obtained for instances with $p = 10$ and $p = 18$. The quality of the approximation also seems to be better among the $p > q$ cases, in which the average gap was 1.10 as opposed to 1.16 for the other case.

These results highlight the efficiency of the approximation algorithm: small computational times and a good practical approximation factor, which in practice was much better than the theoretical factor of 3.
| Instance | Exact Algorithm | Approximation Algorithm |
|----------|-----------------|-------------------------|
| p | ID | lb | ub | t(s) | ub | t(s) | apx | q |
| 2 | 1 | 782.29 | 782.29 | 665.7 | 1018.72 | 2.57 | 1.3 | 16 |
|   | 2 | 781.52 | 781.52 | 38.7 | 920.02 | 2.99 | 1.17 | 11 |
|   | 3 | 732.24 | 732.24 | 132.95 | 850.01 | 2.49 | 1.16 | 9 |
|   | 4 | 789.62 | 789.62 | 559.4 | 873.28 | 2.87 | 1.1 | 9 |
|   | 5 | 748.19 | 748.19 | 134.2 | 962.85 | 2.62 | 1.28 | 16 |
|   | 6 | 761.12 | 761.12 | 73.12 | 815.12 | 2.63 | 1.07 | 7 |
|   | 7 | 754.79 | 754.79 | 76.56 | 922.95 | 2.57 | 1.22 | 14 |
|   | 8 | 752.55 | 783.93 | 3600 | 1067.03 | 2.62 | 1.41 | 22 |
|   | 9 | 778.12 | 778.12 | 235.31 | 1059.02 | 2.61 | 1.36 | 19 |
|   | 10 | 787.75 | 787.75 | 19.31 | 965.22 | 2.45 | 1.22 | 14 |
| avg. | 766.81 | 769.95 | 553.52 | 945.42 | 2.64 | 1.22 | 13.7 |

Table 1: Results for instances with n = 100 and p ∈ {2, 10, 18}.

| Instance | Exact Algorithm | Approximation Algorithm |
|----------|-----------------|-------------------------|
| p | ID | lb | ub | t(s) | ub | t(s) | apx | q |
| 18 | 1 | 754.15 | 754.15 | 11.04 | 779.62 | 2.55 | 1.03 | 16 |
|   | 2 | 776.21 | 782.54 | 3600 | 843.03 | 2.98 | 1.08 | 11 |
|   | 3 | 720.43 | 735.43 | 3600 | 764.55 | 2.64 | 1.06 | 9 |
|   | 4 | 769.5 | 770.23 | 3600 | 883.84 | 2.69 | 1.14 | 9 |
|   | 5 | 723.34 | 723.34 | 12.19 | 773.38 | 2.56 | 1.06 | 16 |
|   | 6 | 750.09 | 750.09 | 32.62 | 871.85 | 2.63 | 1.16 | 7 |
|   | 7 | 734.91 | 734.91 | 18.61 | 817.68 | 2.51 | 1.14 | 14 |
|   | 8 | 709.13 | 709.13 | 1.63 | 740.2 | 2.53 | 1.04 | 22 |
|   | 9 | 744.04 | 744.04 | 1.83 | 754.58 | 2.6 | 1.01 | 19 |
|   | 10 | 772.45 | 772.72 | 3600 | 820.37 | 2.45 | 1.06 | 14 |
| avg. | 745.99 | 745.99 | 20.23 | 808.18 | 2.60 | 1.07 | 13.7 |

Table 2: Results for instances with n = 100 and p ∈ {26, 33}.
4 Conclusion

In this paper, we introduced a 3-approximation algorithm for the special $p \leq \left\lceil \frac{n - 2\lfloor \frac{n}{3} \rfloor}{3} \right\rceil$ case of the H_pMP. The first H_pMP approximation algorithm in the literature to the best of our knowledge.

We conducted computation experiments to assess the practical approximation ratio of the proposed algorithm, comparing it to an exact algorithm from the literature. The practical approximation ratio was 1.1, on average, with computational times orders of magnitude smaller than the exact algorithm. In fact, the approximation algorithm proves to be specially helpful for higher values of $p$, for which the exact algorithm has a hard time finding the optimal solution.

As future research, it might be interesting to look for new approximation algorithms, with better approximation ratios and time complexities, and capable of handling general values of $p$. Another possible line of research is the combination of the approximation algorithm with local search procedures.

References

[1] I. M. Branco and J. D. Coelho. The hamiltonian p-median problem. *European Journal of Operational Research*, 47(1):86 – 95, 1990.

[2] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley, 1997.

[3] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[4] Jack Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, pages 449–467, 1965.

[5] G. Erdoğan, G. Laporte, and A. M. R. Chía. Exact and heuristic algorithms for the hamiltonian p-median problem. *European Journal of Operational Research*, 253(2):280 – 289, 2016.

[6] H. Glaab and A. Pott. The Hamiltonian p-median problem. *The Electronic Journal of Combinatorics*, 7(R42):2, 2000.

[7] S. Gollowitzer, L. Gouveia, G. Laporte, D. L. Pereira, and A. Wojciechowski. A comparison of several models for the hamiltonian p-median problem. *Networks*, 63(4):350–363, 2014.

[8] Stefan Gollowitzer, Dilson Lucas Pereira, and Adam Wojciechowski. New models for and numerical tests of the hamiltonian p-median problem. In *Volume 6701 of Lecture Notes in Computer Science*, 2011. Proceedings of the International Network Optimization Conference (INOC) 2011.

[9] L. Hupp and F. Liers. A polyhedral study of the hamiltonian p-median problem. *Electronic Notes in Discrete Mathematics*, 41:213 – 220, 2013.

[10] Michel Wan Der Maas. Implementation of the approximation algorithm. URL https://github.com/michelwandermaas/p_hamiltonian_heuristic.

[11] A. M. Marzouk, E. Moreno-Centeno, and H. Üster. A branch-and-price algorithm for solving the hamiltonian p-median problem. *Informs Journal on Computing*, 28(4), 2016.

[12] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001.

[13] M. Zohrehbandian. A new formulation of the Hamiltonian p-median problem. *Applied Mathematical Sciences*, 1(8):355–361, 2007.