ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Semester Project

Approximation Algorithms for the Undirected Travelling Salesman Problem

Author:
Aleksandar Markovic*

Under the direction of:
Professor Mohammad Amin Shokrollahi
Responsible Assistant:
Dr. Ola Svensson

Fall 2011

*aleksandar.markovic@epfl.ch

Contents

Abstract

We present an overview of known approximation algorithms and linear programming relaxations of the Undirected Travelling Salesman Problem with the triangle inequality.

In particular, we review the classical algorithm by Christofides and explain how it upper bounds the integrality gap of the considered linear programming relaxation.

We also complement this by presenting better upper bounds for specific cases, such as subcubic graphs in the unweighed shortest path metric and two 5-cycles connected by five paths in a general metric.

1 Introduction

Imagine you are Santa Claus. You have to deliver gifts to the good children and coal to the others, and you have only one night to do it. Therefore you will have to find the shortest tour of the world which goes through every town exactly once, for you do not want to be seen too much. This problem is called the Travelling Salesman Problem (TSP for short) and is one of the most studied questions in combinatorial optimisation.

Let $G = (V, E)$ be a complete undirected graph and let

$$w : E \to \mathbb{R}_+$$

be the weight (or distance) function. We make the reasonable assumption that the triangle inequality is satisfied, i.e., for each three vertices v_1, v_2 and $v_3 \in V$, we have $w(v_1, v_2) + w(v_2, v_3) \geq w(v_1, v_3)$. We have now a more mathematical way to formulate the problem. Indeed, we want to find a tour $S \subset E$, which minimises:

$$\sum_{e \in S} w(e),$$

such that every vertex is visited exactly once by the tour S. We can first wonder which class of problems it belongs to.

The question $P = NP$ [15] is one of the most interesting open problems in computer science since it was introduced in 1971. We say that a problem is in class P if it is solvable in polynomial time and that it is in class NP if we can "verify" a solution to be feasible in polynomial time. We know P to be included in NP, but the other inclusion is unlikely, even if we cannot prove it. Then, we can define the class NP-complete to be the most "hard" problems of the class NP, i.e., a problem is NP-complete if any other problem in NP can be reduced to it in polynomial time. Thus, if we can find a polynomial time algorithm for any NP-complete problem, we have proven $P = NP$, but this will be very surprising since we think $P \neq NP$.

The TSP is known to be NP-complete [9]. Moreover, Sahni and Gonzales [7] prove that the TSP without the triangle inequality has no constant approximation guarantees, i.e., we can find a efficient algorithm which returns a solution of value at most r times the optimal solution with r a constant, if and only if $P = NP$. But as the problem is still interesting in practice, we are also interested in solutions not too far away from the optimal one. There are several methods to find such a solution: approximation algorithms, heuristics, etc. In this report, we will use approximation algorithms, which are algorithms running in polynomial time and returning a result not too far away from the optimal result. More formally, an r-approximation algorithm is an algorithm which returns a solution of value at most r times the optimal one.

We will present the best approximation algorithm known for the general case, found in 1976 by Christofides [2]. Then we will introduce an integer linear program and its relaxation, the Held-Karp relaxation [8, 16], and analyse how good it is. Since the relaxation is solvable in polynomial time, the better the relaxation is, the closer we are to a real tour; and this will be "measured" by the integrality gap. We will analyse this gap in order to bound it [1]. Then, we will consider some specific cases with the biggest known bound.

2 First approximation algorithms for the undirected TSP

As we stated in the introducation, the best known approximation algorithm has been found in the seventies by Christofides [2]. We will present this result and the naive one, both using a minimum spanning tree, which can be found in polynomial time with for instance Kruskal's Algorithm [10].

2.1 Preliminaries

First, we will present some basic definitions.

Definition 2.1. *Let $G = (V, E)$ be a connected undirected graph. A* Hamiltonian tour *is a closed path $a_0 - a_1 - ... - a_m = a_0$ such that for every vertex $v \in V$, it exists an $i \in \{0, 1, ..., m-1\}$ so that $v = a_i$ and $v \neq a_j$ for every $j \neq i$.*

Definition 2.2. *Let $G = (V, E)$ be a connected undirected graph. An* Eulerian tour *is a path $a_0 - a_1 - ... - a_m = a_0$ such that for every edge $e \in E$, it exists an $i \in \{0, 1, ..., m-1\}$ so that $e = (a_i, a_{i+1})$.*

Definition 2.3. *A graph $G = (V, E)$ is called an* Eulerian graph *if E contains an Eulerian tour.*

Theorem 2.4. *A graph $G = (V, E)$ is an Eulerian graph if and only if the degree of each vertex is even.*

Proof. First, we will prove that if the graph is Eulerian, the degree of each vertex is even. As the graph is Eulerian, we can find a tour which goes through every edge. Therefore, for every vertex, all edge "in" will correspond to an edge "out", which is the next edge in the Eulerian tour.

On the other hand, let us take a graph G with an even degree on each vertex. We will proceed on each connected part with more than one element. If there is no such part, there is no edges in the graph, so it is obviously an Eulerian graph. Then, for every vertex of this graph, there is an even number of edges. Let us do pairs of such edges, one in and one out. Then, the out one is obviously an in of the next vertex. As it is an in, there is an out linked to it. And so on, until we're back on the first vertex. Then, if we have not visited each edge, we take one of those edge and we do this again until there is no edges unvisited. Since the graph is connected, there is a way to collapse all those tours into a unique one as in Figure 1 in order to create an Eulerian tour. Thus, the graph is Eulerian.

□

Let us denote the optimal tour byOPT and the minimum spanning tree of the graph $G = (V, E)$ by MST, and let

$$val : \mathcal{P}(E) \to \mathbb{N}$$

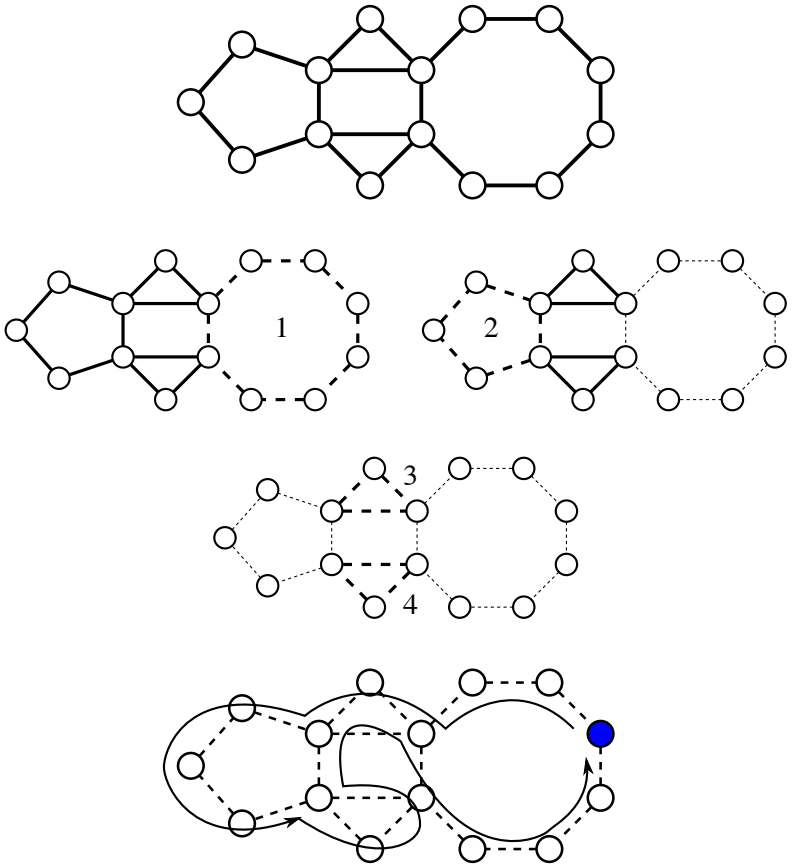

Figure 1*: Let us consider this graph. It is obviously Eulerian for all the degrees are even. Then we take any vertex and follow one path until we are back on the first vertex. We will have visited the tour 1, for instance. But in this case, all the edges have not been traversed yet. So we take another vertex and we highlight a tour until there is no more edges not traversed. Thus, we will have a certain number of tour, in this example, we will have four. Then, we take any vertex of the graph, for instance, the coloured one, and we follow a tour in which it appears until we reach a vertex which appears in a tour we have not begun to go through. We stop the first tour and begin with the second, until another such vertex appears or until we are back on the vertex where we have changed the tour. Finally when we have ended the last tour, we can go back on the others in the reversed order and thus complete the big tour.*

be a function defined by: $val(E') = \sum_{e \in E'} w(e)$, where $E' \subseteq E$. We start by showing that a MST is a lower bound of an optimal tour.

Lemma 2.5. *Let $K_n = (V, E)$ be the complete undirected graph of n vertices and let $E_e \subseteq E$ be an Eulerian tour in K_n. Let moreover w be a weight function on E which satisfies the triangle inequality. Then, we can find an Hamiltonian tour $E_h \subseteq E$ of G in a polynomial time such that $val(E_h) \leq val(E_e)$.*

Remark 2.6. *We are interested in Eulerian tours because they are more convenient to construct than the Hamiltonian ones. But without the triangle inequality, we cannot use shortcuts to transform them into Hamiltonian tours without increasing its cost.*

Proof. Let us take any vertex of the Eulerian tour and rename it a_0. Then for every a_i of this path, we call a_{i+1} the next vertex of the Eulerian tour that has not already been visited, i.e., if $a_{i+1} \neq a_j$, for all $j \leq i$. Else we continue on the Eulerian tour until we find such a vertex or have visited every vertex of the graph. We will now show that $a_o - a_1 - ... - a_n = a_0$ is an Hamiltonian tour E_h of cost at most the cost of the Eulerian tour. Indeed, for every pair (a_i, a_{i+1}), we have a corresponding edge because the graph is complete. Moreover, with the triangle inequality, this edge cannot cost more than the path between a_i and a_{i+1} in the Eulerian graph. Therefore, we have $val(E_h) \leq val(E_e)$. □

Definition 2.7. *Let $G = (V, E)$ be a complete undirected graph with a weight function w. A* spanning tree *in this graph is a subset $T \subseteq E$ such that (V, T) is connected and contains no cycle. Moreover, if the sum of all the weights of T is minimum, we call it a* Minimum Spanning Tree *(MST for short).*

Algorithm 1 describe how to find a MST of a graph.

Algorithm 1 Kruskal's MST algorithm

```
1: color the cheapest edge in red
2: while there is uncoloured edges do
3:    find the cheapest uncoloured edge
4:    if it creates a cycle in the red tree then
5:       color it in blue
6:    else
7:       color it in red
8:    end if
9: end while
10: return the red edges
```

This algorithm picks the cheapest edge (if there are several, it makes an arbitrary choice) in the graph and colors it red. Then it takes the cheapest

uncoloured edge, if it creates a cycle in the red tree, it colors it in blue and if not, in red. And so on, until there are no more uncoloured edges (we can also just ask we have enough red edges to make a connected tree, i.e., $|V| - 1$, for less coloring steps). So, the wanted MST is the set of the red edges returned.

The following lemma is easy to verify and well-known [15].

Lemma 2.8. *Kruskal's MST algorithm returns a minimum spanning tree in polynomial time.*

2.2 Christofides' approximation

We will present two different approaches, both using a MST. The first, more naive, is an approximation of at most twice the cost of an optimal tour, and the second, a little more elaborated, is the Christofides approximation [2]. The second one is the best approximation known for the general case of metric undirected TSP.

Lemma 2.9. $val(MST) \leq val(OPT)$.

Proof. Let us take the optimal tour and remove the heaviest edge. The result is a spanning tree T_1. Thus $val(MST) \leq val(T_1) \leq val(OPT)$. □

Lemma 2.10. (2-approximation) *We can find a tour $T_2 \subseteq E$ in polynomial time, such that $val(T_2) \leq 2val(OPT)$.*

Proof. Let us find a MST of our graph G. Then, we consider each edge of the MST twice to create an Eulerian graph, shown in Figure 2. The cost of this tour is at most twice the cost of OPT, by Lemma 2.9. Then, we apply Lemma 2.5 in order to obtain a Hamiltonian tour of no larger cost from the Eulerian tour. □

Lemma 2.11. (Christofides' 1.5-approximation [2]) *We can find a tour $T_{1.5} \subseteq E$ in polynomial time, such that $val(T_{1.5}) \leq 1.5val(OPT)$.*

Proof. Let us find a MST and consider all the vertices with an odd degree and color them in red. Since the number of such vertices is even, we can find a minimum perfect matching between the nodes in polynomial time [4]. Then, we can combine them both in order to have an Eulerian graph, since all the vertices will have an even degree. Finally, by Lemma 2.5, we can find in polynomial time a Hamiltonian tour.

In order to analyse the cost of the resulting tour, we will split the analysis in two steps: the MST and the MM (Minimum Matching). We already have proved in Lemma 2.9 that the value of the MST is at most the value of the optimal tour. In addition, we need to upper bound the cost of the MM in terms of the optimal tour cost.

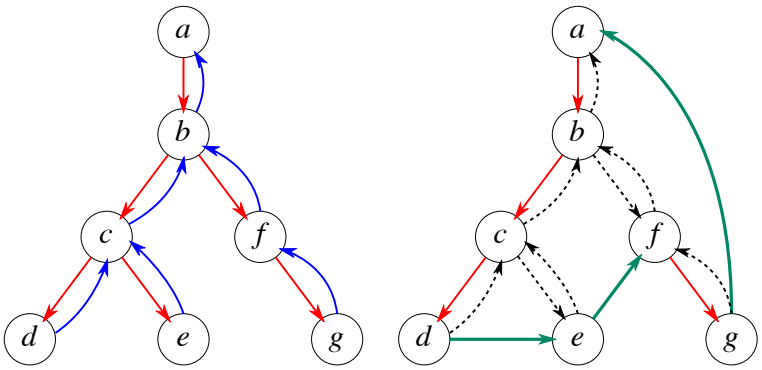

***Figure 2**: We first double each edge of the MST to obtain an Eulerian tour of cost at most twice $val(OPT)$. Here the tour will be a-b-c-d-c-e-c-b-f-g-f-b-a. Then, we consider all the vertices which appear more than once, and we avoid them by taking a shortcut. Thus, we obtain the tour a-b-c-d-e-f-g-a, our 2-approximation.*

Therefore, let us consider the red vertices in the optimal tour. We choose one of them and follow the tour until we find another red vertex. Using shortcuts, the weight of the edge between those two vertices is less or equal to the weight of the part of the tour we have traversed. Then we repeat this for all the following red vertices. We have thus a tour of the red vertices with cost at most $val(OPT)$. As the number of red vertices is even, we can find exactly two perfect matchings in the constructed tour between red vertices. Further, we chose the cheapest one to have a perfect matching of cost at most half the cost of OPT and hence the cost of a minimum perfect matching is at most half the cost of an optimal tour.

From the arguments above, we have a Hamiltonian tour of value at most $val(MST) + val(MM) \leq val(OPT) + 0.5val(OPT) = 1.5val(OPT)$. □

3 Linear Programming Relaxation

We will now introduce the Held-Karp relaxation [8] and analyse how good it is compared to the exact integral linear program formulation.

3.1 Integrality Gap

First of all, we need another definition, in order to formulate our linear program.

Definition 3.1. *Let $G = (V, E)$ be a graph and let $S \subseteq V$ be a subset of the vertices of G. We denote $\delta(S) \subseteq E$ the subset of the edges that crosses the*

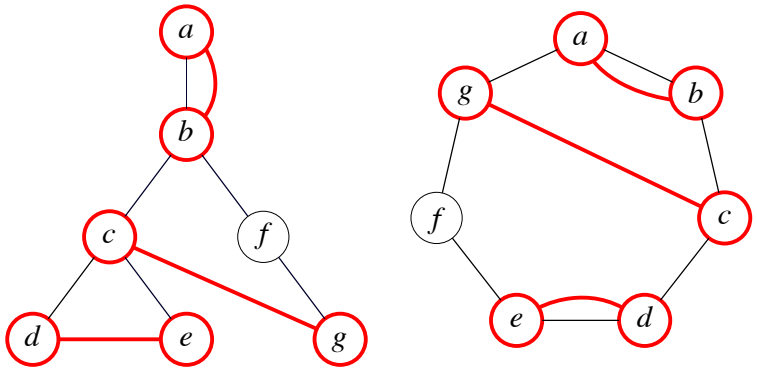

Figure 3: *We consider all the vertices of the MST with an odd degree and we find the minimum matching of those vertices. Then, we use shortcuts and the triangle inequality to find an approximation of the optimal tour. The cost of MST is at most the cost of OPT. Moreover, the cost of this matching is at most half the cost of OPT : we consider those vertices in an optimal tour a-b-c-d-e-f-g-a, for example. Then if g-a, b-c and e-d are more expensive than half the tour, we drop them and take a-b, c-d and e-g. Therefore, a minimum matching has value at most half the cost of an optimal tour.*

cut (S, S^c)*, i.e.,* $\delta(S) = \{e \in E \mid |e \cap S| = 1\}$ *We abbreviate* $\delta(\{v\})$ *by* $\delta(v)$*.*

We will now introduce the linear program (1) [8].

$$\begin{aligned} \min \sum_{e \in E} w(e) x_e, \quad & \text{subject to:} \\ \sum_{e \in \delta(v)} x_e = 2, \quad & \forall v \in V, \\ \sum_{e \in \delta(S)} x_e \geq 2, \quad & \forall S \subsetneq V,\ S \neq \emptyset, \\ x_e \in \{0,1\}. & \end{aligned} \tag{1}$$

If we find a solution vector x, we are done. But, we cannot hope to solve this Integral Linear Program (ILP) in polynomial time. So, we have to relax it, i.e., instead of taking $x_e \in \{0,1\}$, we take $x_e \in [0,1]$. Thus we have a Linear Program (LP) which is known to be solvable in polynomial time even with the exponentially many constraints, e.g., withe the ellipsoid method or by an equivalent more compact formulation. If we are lucky, all x_e are either 0 or 1, and we have the exact optimal tour, but in general, this is not the case. Obviously, the solution $\overline{x}$ of the LP has cost at most $val(OPT)$, since we have relaxed a condition. Thus, we can define the following.

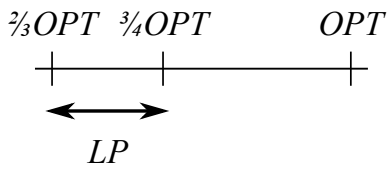

Figure 4: *The value of the LP takes place between* $\frac{2}{3}$ *and* $\frac{3}{4}$ *times the optimum value. The actual conjecture [12] claim the LP to be exactly* $\frac{3}{4}$.

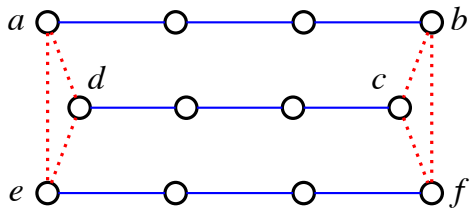

Figure 5: *We provide this graph with a simple weight function* w*, such that each drawn edge has a weight* 1*, and all undrawn edges have the same weight as the shortest path between the two vertices, e.g., the edge* (a, c) *has a weight* 4.

Definition 3.2. *For an instance* I *of the TSP, let* $\mathrm{ILP}(I)$ *be the value of the exact formulation, and let* $\mathrm{LP}(I)$ *be the value of the linear program relaxation. Then the* integrality gap *is the following number :*

$$IntGap = \sup_I \frac{\mathrm{ILP}(I)}{\mathrm{LP}(I)} \geq 1.$$

We will now investigate the strength of our LP, i.e., the closer the $IntGap$ is to 1, the better the relaxation is. We will prove the integrality gap to be between $\frac{2}{3}val(OPT)$ and $\frac{3}{4}val(OPT)$ as depicted in Figure 4.

3.2 Lower bound on the Integrality Gap

To find a lower bound, we just need to provide an instance of the problem in which this bound is reached.

Therefore, let us take the graph depicted in Figure 5, with the same weight function w. Such an instance is called graph-TSP or TSP on unweighted shortest path metric.

We want to calculate the integrality gap for this graph. If we let $\overline{x}$ be the solution of the LP, defined by

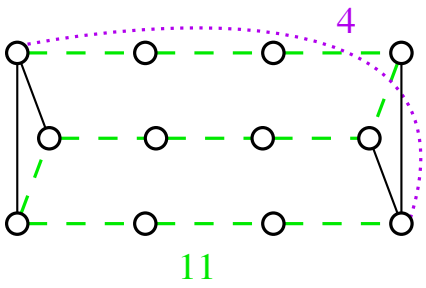

***Figure 6**: We link a and f with the obvious path (of cost 11) and then we come back to a with the edge (f, a), which has cost 4.*

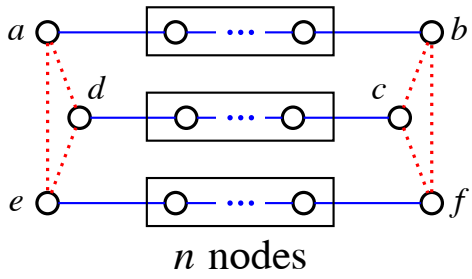

***Figure 7**: We provide this graph with the same weight function w as in Figure 5, where $n = 2$. Here, the edge (a, c) has a weight $n + 2$.*

$$\overline{x}_e = \begin{cases} 1 & \text{if } e \text{ is plain,} \\ \frac{1}{2} & \text{if } e \text{ is dotted,} \\ 0 & \text{otherwise.} \end{cases}$$

Obviously, $\overline{x}$ is a solution of the LP, and hence $\mathrm{LP}(I) \leq 6 \cdot \frac{1}{2} + 9 \cdot 1 = 12$. Then we consider the tour on Figure 6 which can be proved to be optimal. Thus, we have $IntGap \geq \frac{15}{12}$.

Let us generalise this result. Therefore, we consider the general case of the graph-TSP instance as shown in Figure 7. Then the same tour becomes a $4n + 7$-weighted path as on Figure 8, while $\mathrm{LP}(I) \leq 6 \cdot \frac{1}{2} + 3n \cdot 1 = 3n + 3$. Thus,

$$IntGap \geq \frac{4n+7}{3n+6} \underset{n \to +\infty}{\longrightarrow} \frac{4}{3}.$$

We have thus found a specific case to lower bound the integrality gap. In the next subsection, we will find an upper bound.

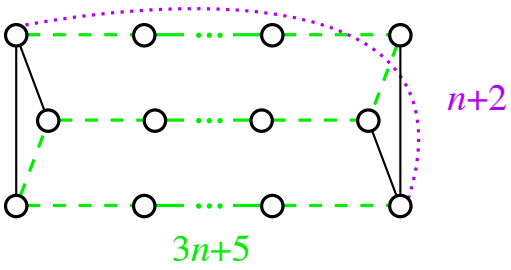

***Figure 8**: We use the same tour as in Figure 6. The path from the first to the last vertex has a length of $3n + 5$ and the last edge, $n + 2$.*

3.3 Upper bound on the Integrality Gap

Wolsey [17] in 1980 then Shmoys and Williamson [14] in 1990 proved this gap to be bounded by $\frac{3}{2}$. We will here present the proof given by Carr [1] in 2004, which is shorter and more elegant. We will use the fact that the MST and the T-join problems have integer polytopes. We will first remind the following definition.

Definition 3.3. *A* T-join *for a subset $T \subseteq V$ of a graph $G = (V, E)$ is a subgraph $H = (V, E')$ in which all vertices of odd degree are exactly those in T.*

We will now prove two lemmas in order to use them to prove Theorem 3.6.

Lemma 3.4. [1] *Given a solution x^* to the Held-Karp relaxation [8], $\left(1 - \frac{1}{n}\right) x^*$ is a feasible solution to minimum spanning tree polytope.*

Proof. Let $|V| = n$ and $E(S) \subseteq E$, for $S \subseteq V$, be the subset of edges e such that $|e \cap S| = 2$, i.e., the edges with both extremities in S. And, then, let us write the spanning tree polytope [11]:

1. $\sum_{e \in E(V)} x^*_e = n - 1$;
2. $\forall S \subsetneq V, S \neq \emptyset$, then $x^*(E(S)) \leq |S| - 1$;
3. $x^* \geq 0$.

We will now see if those conditions are satisfied. We first have

$$\begin{aligned}\sum_{e \in E(V)} x^*_e &= \left(\sum_{v \in V} \sum_{e \in \delta(v)} x^*_e\right) \frac{1}{2} \\ &= \left(\sum_{v \in V} 2\right) \frac{1}{2} = |V| = n\end{aligned}$$

and thus $\sum_{e\in E}\left(1-\frac{1}{n}\right)x^* = n-1$.

Then, in order to prove the second condition to be true, let us suppose that it exists a subset of vertices $S \subsetneq V$, such that $x^*(\mathbb{E}(S)) > |S|-1$. Therefore, we can write

$$\begin{aligned}|S|-1 &< \left(\sum_{v\in S}\left(\sum_{e\in\delta(v)} x_e^*\right) - x^*(\delta(S))\right)\frac{1}{2}\\ &= \left(\sum_{v\in S} 2 - x^*(\delta(S))\right)\frac{1}{2}\\ &= \Big(2|S| - x^*(\delta(S))\Big)\frac{1}{2}.\end{aligned}$$

Thus, $x^*(\delta(S)) < 2$ which contradicts that x^* is feasible to 1 and so, $\left(1-\frac{1}{n}\right)x^*$ is a feasible solution to spanning tree polytope.

□

Lemma 3.5. [1] *Given a solution x^* to the Held-Karp relaxation [8], $\frac{1}{2}x^*$ is a feasible solution to T-join polytope.*

Proof. Let us write the T-join polytope [6]:

1. $x(\delta(S)) \geq 1$ if $|S\cap T|$ is odd for $S \subseteq V$;
2. $x^* \geq 0$.

As $x^*(\delta(S)) \geq 2$ for all $S \subsetneq V$, if we divide the solution by 2, we have almost verified 1. But the case $S = V$ cannot happen, because $|S\cap T|$ must be odd, hence, all the conditions of a T-join polytope are verified for $\frac{1}{2}x^*$. □

Theorem 3.6. [1] *Given, a solution x^* to the Held-Karp relaxation [8], $\left(\frac{3}{2}-\frac{1}{n}\right)x^*$ can be expressed as a convex combination of Eulerian graphs.*

Proof. By Lemma 3.4, we know $\left(1-\frac{1}{n}\right)x^*$ to be a feasible solution for MST polytope :

$$\sum_{i\in I}\lambda_i T_i = \left(1-\frac{1}{n}\right)x^*,$$

where T_i is a spanning tree for all $i \in I$. Then, for all T_i, $\frac{1}{2}x^*$ is a feasible solution to T_i'-join polytope, where $T_i' \subseteq T_i$ is the subset of vertices of odd degree of T_i for all $i \in I$:

$$\sum_{i\in I, j\in J}\delta_{i,j}M_{i,j} = \frac{1}{2}x^*, \text{where } M_{i,j} \text{ is a } T_i\text{-join for } T_i'.$$

Hence, since $T_i \cup M_{i,j}$ is an Eulerian spanning graph for all $i \in I$ and $j \in J$, we have a convex combination of possible tours, using Lemma 2.5,

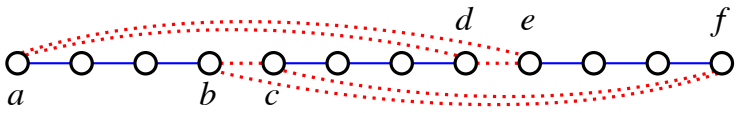

***Figure 9**: The same graph as in Figure 5, but the path from a to f is drawn as a straight line with the additional backedges, here doted.*

and thus $\left(\frac{3}{2} - \frac{1}{n}\right) x^*$ can be written as a convex combination of spanning trees T_i and T_i-join. □

Therefore, we can conclude with the following result.

Corollary 3.7. *The integrality gap of the TSP is at most $\frac{3}{2}$.*

Proof. As we have a convex combination of Eulerian graphs, and the expected cost is $\frac{3}{2} - \frac{1}{n}$ times the cost of the spanning tree, there exists at least one Eulerian graph of cost at most that. □

3.4 Special cases

Recently, Gharan, Saberi and Singh [13] presented an approximation algorithm for graph-TSP with a guarantee of $1.5 - \varepsilon$, for an ε of the order 10^{-12}. This major achievement was later improved by Mömke and Svensson [12]. We will present their approach. First, let us introduce some graph terminology.

Definition 3.8. *A* cubic graph *is a graph in which each vertex has degree 3.*

Definition 3.9. *A* subcubic graph *is a graph in which each vertex has degree at most 3.*

Here we want to find more instances where the integrality gap is $\frac{4}{3}$. Let us take the graph-TSP and draw it as a line with backedges, as depicted in Figure 9.

Then, let us drop two of the backedges and add and/or remove some vertices on the line, as on Figure 10.

There is three paths from b to c: the first one is the one going directly from b to c by the path, the second one use the backedge (b, d) and then the path from d to c and the last one goes back to a and then through the backedge (a, c). If we duplicate one of those three path, we would have an Eulerian graph. Thus, if we choose the path to duplicate at random with the same probability, the expectation cost of the Eulerian tour would be $\frac{4}{3}$ times the number of edges, since all edges have cost 1.

First, let us consider only cubic graphs.

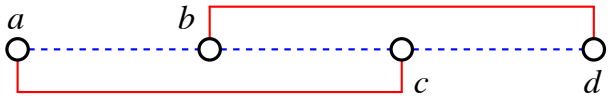

Figure 10: *We draw a path $a - ... - b - ... - c - ... - d$ with any number of vertices between those four. Then we add to this graph two shortcut edges (a, c) and (b, d).*

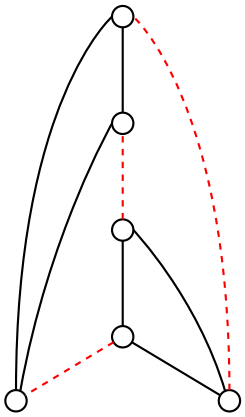

Figure 11: *We consider this graph with the perfect matching represented by the discontinued line. If we drop those three edges, we disconnect the graph.*

Theorem 3.10. [3] *In a cubic 2-vertex-connected graph $G = (V, E)$, we can pick a perfect matching at random so that*

$$\mathbb{P}r_M[e \in M] = \frac{1}{3}, \forall e \in E.$$

We now consider a cubic graph, e.g. Figure 11. We want to drop as many edges as possible without disconnecting the graph, so we can obtain a better tour and thus make a better bounding of the integrality gap.

Lemma 3.11. *Given a cubic 2-vertex-connected graph $G = (V, E)$, we can in polynomial time find a tour of value at most*

$$\frac{2}{3}(|E| + |V| - 1).$$

Proof. Let us apply Algorithm 2 to G and analyse the returned subset of edges.

Algorithm 2 Cubic algorithm

1: Find a MST T of G.
2: Pick a perfect matching at random M, so that every edge has a probability of $\frac{1}{3}$ to appear.
3: **return** $(E \cup (M \cap T)) \setminus (M \setminus T)$.

First, the algorithm returns the set of all edges and it duplicates the edges of the matching that are in the minimum spanning tree. Then it removes the edges of E which appears in the matching but are not a part of the tree.

Then, as the MST is connected, we will always have a connected graph. Furthermore, the algorithm duplicates the edges of the matching in the tree and drop the others. Since it is a matching and all the vertices have degree 3, each vertex either drops or duplicates exactly one edge incident to it, i.e., each vertex has either degree two or four and the obtained graph is thus Eulerian by Theorem 2.4. Finally, let us calculate the expected number of edges of the given tour, using Theorem 3.10

$$\begin{aligned}\mathbb{E}[|(E \cup (M \cap T)|) \setminus (M \setminus T)] &= |E| + \mathbb{E}[|M \cap T|] - \mathbb{E}[|M \setminus T|] \\ &= |E| + \frac{1}{3}(|V| - 1) - \frac{1}{3}(|E| - (|V| - 1)) \\ &= \frac{2}{3}|E| + \frac{2}{3}(|V| - 1).\end{aligned}$$

□

3.5 Pairs

We now want to increase the number of removable edges, so that the expected cost of the returned edges will be even lower. For now, we have only allowed the edges not in the MST to be removed. But since the graph is two vertex connected, perhaps we can do better. Therefore, we will introduce some pairs in the graph, so that one is in the MST and the other is a backedge, i.e., an edge connecting a vertex to his ancestor, but not the direct one. We define them both to be removable, but they will not be removed at the same time, i.e., if we remove one, the other has to stay in the graph.

Therefore, we will first add some edges in R, the set of removable edges. Since all edges have the same cost, instead of using Kruskal's algorithm, we will simply use the Depth-First-Search (DFS) to find a minimum spanning tree, thus all the edges will either be in the MST or will be backedges. So, in this case, for every pair of vertices connected by a backedge, it exists a path going either always up or always down in the tree, i.e., if $(a, b) \in E$ is a backedge where a is the ancestor of b, it exists a path $a_0 = a - a_1 - ... - a_n = b$, such that a_{i+1} is a child of a_i for every $i = 0, ..., n - 1$. With this property, we can pair every backedge (a, b) with (a, a_1), as in Figure 12.

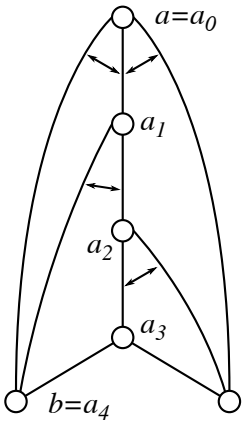

***Figure 12**: We pair each backedge with another one to increase the number of removable edges without disconnecting the graph. For instance, the backedge (a, b) is paired with (a, a_1). Here, we have made a pair more than we need.*

Let us consider Algorithm 3, and then we will analyse it.

Algorithm 3 Cubic algorithm with pairs

1: Obtain a MST T of G by Depth-First-Search.
2: Let $R = E \setminus T \cup P$, where P is the set of edges paired.
3: Pick a perfect matching at random M, so that every edges has a probability of $\frac{1}{3}$ to appear.
4: **return** $(E \cup (M \setminus R)) \setminus (M \cap R)$

The return of this algorithm is the set of edges E in which the edges of the matching M which are not removable, i.e., the MST without the paired edges, have been duplicates and and the edges of M which are in R, i.e., all the pairs, have been deleted.

Lemma 3.12. *Given a cubic graph $G = (V, E)$, Algorithm 3 returns a connected Eulerian graph with expected number of edges of $\frac{4}{3}n - \frac{2}{3}$, where $n = |V|$ is the number of vertices.*

Proof. First, we will prove that the graph stays connected by induction on the height of the tree. If the graph has a height of 0, i.e., it is a single vertex, then it is always connected. Now let us assume that the graph stays connected for every height strictly lower than h. Let us take a tree of height h. We have two cases: on the one hand, we have a full tree of height h and on the other, we only have a subtree of that height. If it is the whole tree, we will have two cases as shown in Figure 13. The last case with three subtrees is not 2-vertex connected.

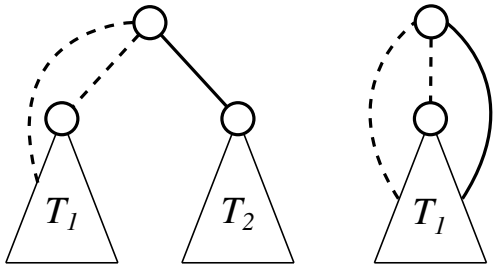

***Figure 13**: The two cases of height h with respectively two subtrees with one backedge and one subtree with two backedges. We first can notice that the left case is not 2-vertex-connected.*

If the root has two subtrees, then there is exactly one backedge to the root, without loss of generality, T_1. So it is paired to the edge from the root to T_1. Since both subtrees stay connected by the induction hypothesis, and as the only two other edges of R cannot be removed at the same time, this first graph stays connected. However, this case never happen because the tree is not 2-vertex connected. Then, if the root has one single subtree, the argument is the same, because the second backedge to the root can either be removed or not when the other is paired to the edge linking the root and T_1.

Otherwise, if it is just a subtree of a bigger tree there are also two cases as in Figure 14.

In the first case, we have no more removable edges, so there is nothing to be done. In the second case, as the two paired edges cannot be chosen at the same time and by induction hypothesis, the graph stays connected.

Furthermore, for every vertex, the algorithm either delete or duplicate exactly one edge incident to it, so the resulting degree of this vertex is either two or four, thus the graph is Eulerian.

Now we have proved that the graph stays connected and all the degrees are even, let us calculate the expected number of edges returned. So, using the fact that the probability of any edge to be picked by the matching is $\frac{1}{3}$, we have

$$\begin{aligned}\mathbb{E}[|(E \cup (M \setminus R)) \setminus (M \cap R)|] &= |E| + \mathbb{E}[|M \setminus R|] - \mathbb{E}[|M \cap R|] \\ &= |E| + \frac{1}{3}(|E| - |R|) - \frac{1}{3}|R| \\ &= \frac{4}{3}|E| - \frac{2}{3}|R|.\end{aligned} \tag{2}$$

Moreover, if we denote the number of backedges by b and as $|E| = \frac{3}{2}n$, we

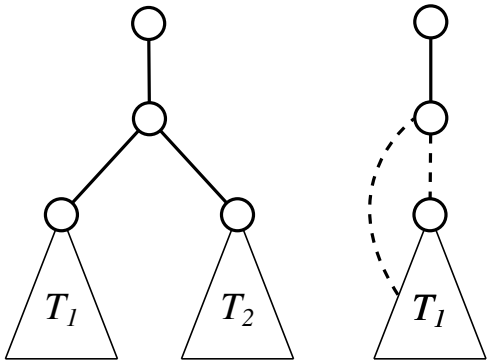

***Figure 14**: The two same cases of height $h+1$. In both, we remove a backedge, so that the graph stays cubic. In the first case, there are no more removable edges, and in the second one, we have still the same pair. We have not add the backedges to the new root because they no not have any purpose in what we consider.*

have

$$\begin{aligned} b &= |E| - (n-1) \\ &= \frac{2}{3}n - n + 1 \\ &= \frac{n}{2} + 1. \end{aligned}$$

Since all the removable edges are the pairs, for every backedge, we have exactly one other removable edge (exept for the root, because there is two backedges to it), that implies

$$|R| = 2b - 1 = n + 1.$$

Then, by replacing $|E|$ and $|R|$ in (2), we obtain

$$\begin{aligned} \mathbb{E}[|(E \cup (M \setminus R)) \setminus (M \cap R)|] &= \frac{4}{3}(\frac{3}{2}n) - \frac{2}{3}(n+1) \\ &= 2n - \frac{2}{3}n - \frac{2}{3} \\ &= \frac{4}{3}n - \frac{2}{3}. \end{aligned}$$

□

3.6 Subcubic graphs

All those results for cubic graphs can be extended to subcubic graphs using a trick. We will transform the subcubic graph in a cubic one without loosing

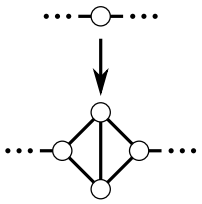

***Figure 15**: Diamond-element, which replace a two degree vertex into four three degree vertices.*

its structure as follows. Every vertex of degree two v will be replaced by a diamond structure consisting in a 4-cycle in which two opposite nodes will be incident to one edge incident to v, and the two other nodes of the 4-cycle will be linked by an edge, as depicted in Figure 15.

For example, we can take the graph depicted in Figure 10. We replace vertices of degree two by diamond structures, then we can apply Algorithm 2, and finally go back on the original graph by dropping the edges of the diamond structures, as shown in Figure 16. We could also apply Algorithm 3 in order to have a better tour.

4 General metric

4.1 New condition

In Section 3, we considered those specific graphs with three restrictions:

1. 2-vertex connected;
2. subcubic;
3. and with the shortest path metric.

We will now consider any metric and replace this condition by the following one: we want the graph to be composed of some n_i-cycles linked by paths of 2-degree vertices. The relaxed linear program will have a value of $\frac{1}{2}$ on the edges in the cycles and 1 on all the others. In this case, we define the relaxed solution to be

$$\overline{x}_e = \begin{cases} 1 & \text{if } e \text{ is in a path linking two cycles,} \\ \frac{1}{2} & \text{if } e \text{ is in a cycle,} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

In theory, there is a conjecture which claims that the case where every n_i is odd is the most complicated one [5]. So let us consider the case with 3-cycles first. An example of such a graph is shown in Figure 17.

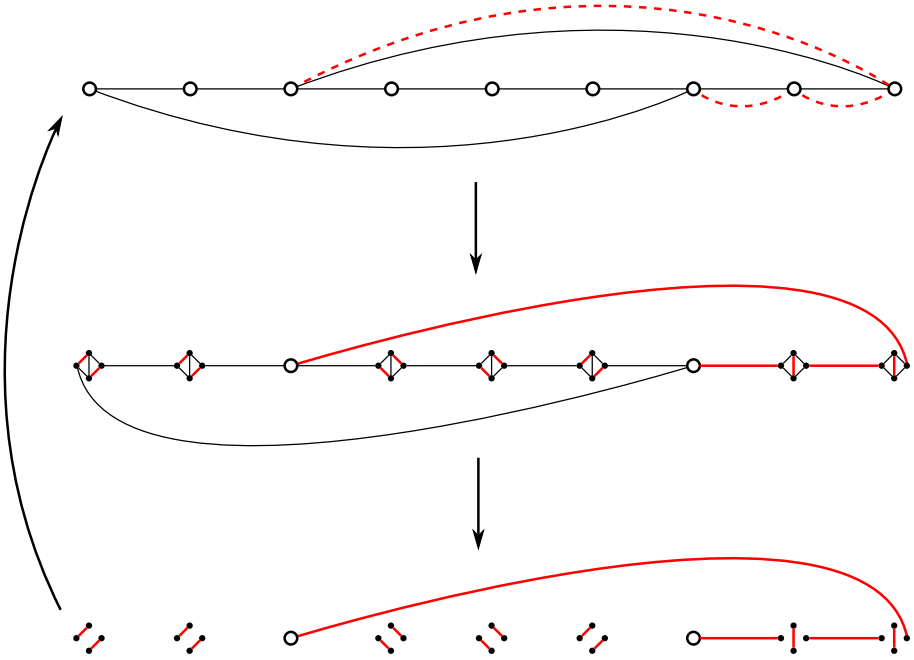

***Figure 16**: We replace every vertex of degree two by a four-vertex diamond such that every vertex has degree tree. Then, we can use Theorem 3.10 to pick up a matching. With this construction, we always pick exactly one of the tree paths between the "middle points" and no other edge of the original graph. Then we double the chosen path in the first graph. Since the matching has a probability of $\frac{1}{3}$ to chose any edge, this is also the case for the entire path, i.e., the expectancy of a random tour is $\frac{4}{3}|E|$.*

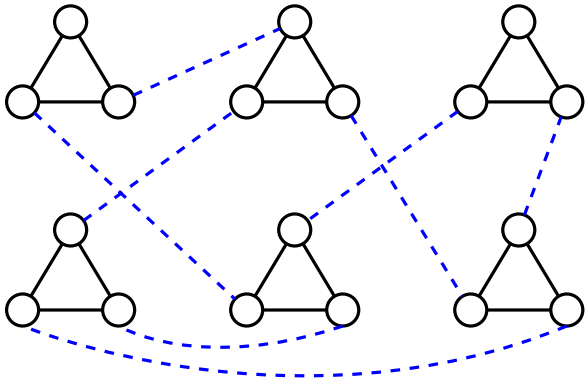

***Figure 17**: We have a 2-vertex-connected subcubic graph with a general metric in which there are some 3-cycles connected by paths of 2-degree vertices represented by doted lines.*

Lemma 4.1. *The integrality gap of this instance of the TSP is* $\frac{4}{3}$.

Proof. Let us consider Algorithm 4. This algorithm will duplicate some

Algorithm 4 3-cycles with general metric algorithm

1: Let R, the set of removable edges, be the set of all edges in the 3-cycles.
2: **if** There are vertices of degree two **then**
3: Obtain a cubic graph $G' = (V', E')$ by replacing each vertex of degree 2 in the paths by the diamond structure depicted in Figure 15
4: **end if**
5: Pick a perfect matching at random M', so that every edge has a probability of $\frac{1}{3}$ to appear
6: Obtain $M \subseteq E$ by restricting M' to E
7: **return** $(E \cup (M \setminus R)) \setminus (M \cap R)$

paths between cycles with a probability of $\frac{1}{3}$, so any edge on a path will appear with a probability of $\frac{4}{3}$. Moreover, in every cycle, the algorithm will choose exactly one edge and remove it. Moreover, as we only remove exactly one edge in every triangle, we cannot disconnect the graph. Thus, the probability that edges on cycles appear will be $\frac{2}{3}$. If we compare this with the relaxed solution $\overline{x}_e$ of (3), we will have an integrality gap of $\frac{4}{3}$. □

We now want to generalise this result for bigger cycles.

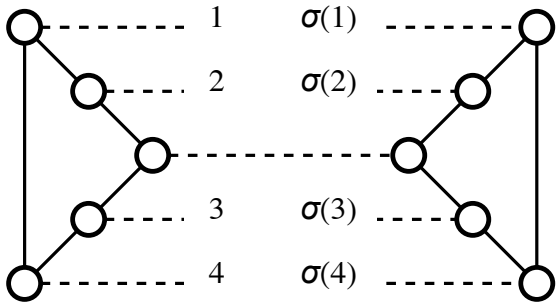

***Figure 18**: We consider two 5-cycles linked by five paths of 2-degree vertices. Without loss of generality, we can assume the node in the middle of the left cycle to be connected to the same node in the right one. The other nodes (from 1 to 4) are sent to them permutation by $\sigma \in \mathfrak{S}_4$ in the right side.*

4.2 5-cycles

Let us take two 5-cycles linked by five paths of any number of vertices (with any metric satisfying the triangle inequality), as in Figure 18. Without loss of generality, we can say that the middle vertex of the left cycle is linked to the middle vertex of the right cycle. Then we can enumerate from 1 to 4 the other vertices of the left cycle and link them to the right cycle with a permutation $\sigma \in \mathfrak{S}_4$. We define, furthermore, C_l, C_r and $P \subseteq E$ to be respectively the left cycle, the right cycle and the five paths.

There are two different ways to prove that the integrality gap is at most $\frac{4}{3}$. The first way will bound it by $\frac{6}{5}$, but we must look at each of the 24 possible permutations. The other way is more elegant but less precise, so we have, in this case, an upper bound of $\frac{4}{3}$.

Therefore, we will analyse Algorithm 5 in two different ways.

Lemma 4.2. *Given a subcubic 2-vertex-connected graph $G = (V, E)$, consisting of two 5-cycles C_l and C_r, linked by five paths, we can find an Eulerian tour of cost at most $\frac{4}{3}$ times the cost of the relaxation.*

Proof. As we have 5-cycles, the tour obtained will be connected. Then we have three possible cases. In the first one, we can obtain a connected graph with only $C_r(M)$, no matter which path we choose; in the second one, we have connected graph with $C_r \setminus C_r(M)$ for every paths. Those two cases are simple to analyse. In the first one, every edge of the paths is taken 6 times, while every edge of the cycles is taken 3 times.Since we have 5 possible choice for M, we have a bound of $\frac{5}{6}\overline{x}_e$. In the second case, each edge of the right

Algorithm 5 Two 5-cycles with general metric algorithm

Let P be the set of edges of the five paths, C_l be the set of edges on the left side and C_r, the set of edges on the right side.
Pick up one of the five paths with the same probability and put all those edges in M.
Let $C_l(M) \subset C_l$ and $C_r(M) \subset C_r$ be subsets with the three following edges: the two linked directly to M and the one which links the two vertices uncovered by the two first edges.
Add in M the edges of $C_l(M)$.
if $P \cup C_l(M) \cup C_r(M)$ is not connected, **then**
    add $C_r \setminus C_r(M)$ in M.
else
    if $P \cup C_l(M) \cup (C_r \setminus C_r(M))$ is not connected, **then**
        add $C_r(M)$ in M.
    else
        Add either $C_r(M)$ or $C_r \setminus C_r(M)$ in M with the same probability.
    end if
end if
return $P \cup M$

cycle is taken twice, so we can bound this by the same way. For the last case, we will take a specific permutation, but all the permutations of this case can be solved with the same argument. Let us consider the graph on Figure 19.

We now draw every possible tour, as in Figure 20. If we can choose to take either $C_r(M)$ or $C_r \setminus C_r(M)$, we can say that every of those edges is taken with a probability of $\frac{1}{2}$. Otherwise, we sum up every edge taken by the algorithm to obtain at most three times each edge on the right cycle. Then, as we have taken each edge on the left cycle three times too, and as we have taken each path six times, we could conclude the integrality gap to be at most $\frac{6}{5}$.

□

The alternative proof is as following.

Proof. This time, we just apply the algorithm twice, once as we already did, and once by replacing the left cycle by the right one and inversely. We can see, by case analysis, that every edge appears at most four times, but there is a more elegant way to see it. We suppose one edge appears five times. The only way for one edge to be chosen five times is shown in Figure 21. The two vertices of this edge will have to choose $C_r(M)$, as the opposite vertex and the two other will have to choose $C_r \setminus C_r(M)$. Then we count how many time each edge appear. Two edges appear just once and the two lasts appear three times. Thus, instead of adding five times the cost of the

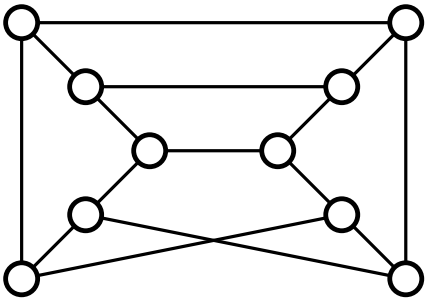

***Figure 19**: This permutation, i.e., $(34) \in \mathfrak{S}_4$, is one of the case where we cannot chose always the same subset of edges on the right side.*

considered edge, by the triangle inequality, we can replace once this cost by the sum of the other costs to have, in the final cost, four times three edges and twice two others, which is at most four times each edge. (Here, we do not change the matching by replacing one edge by the four others. We keep the matching but, by the triangle inequality, we replace one cost by the others.)

Now, we add to the two tours chosen with Algorithm 5, the two tours where we take exactly one cycle and we duplicate each edge of P. Then, we sum up the number of times each edge is taken. On the paths, we have taken 12 times each edge by Algorithm 5 and 4 times by this tour. So every edge on the paths is taken 16 times for 12 tours, that makes $\frac{4}{3}$. Then, we have taken each edge of the cycles 7 times by Algorithm 5, and once by this tour. So every of those edge is taken 8 times for 12 tours, that makes $\frac{2}{3}$. □

Unfortunately we have not been able to generalise this for bigger cycles. Indeed, we cannot expect the graph to be connected by one of those two tours. For instance, we can look at the graph in Figure 22. Therefore, we will certainly need some other tours which duplicate three, five or all of the seven paths.

Conjecture 4.3. *Given two 7-cycles linked by seven paths of 2-degree vertices, we can find a connected Eulerian graph by duplicating an odd number of paths and removing the right number of edges on the cycles, such that the cost of the Eulerian graph is at most $\frac{4}{3}$ times the cost of the original one.*

Remark 4.4. *The result is perhaps even better than this. As for 5-cycles, we have $\frac{6}{5}$, we can expect $\frac{7}{6}$ for 7-cycles, but with $\frac{4}{3}$, we already have the wanted result.*

And if this is true, maybe the following is as well.

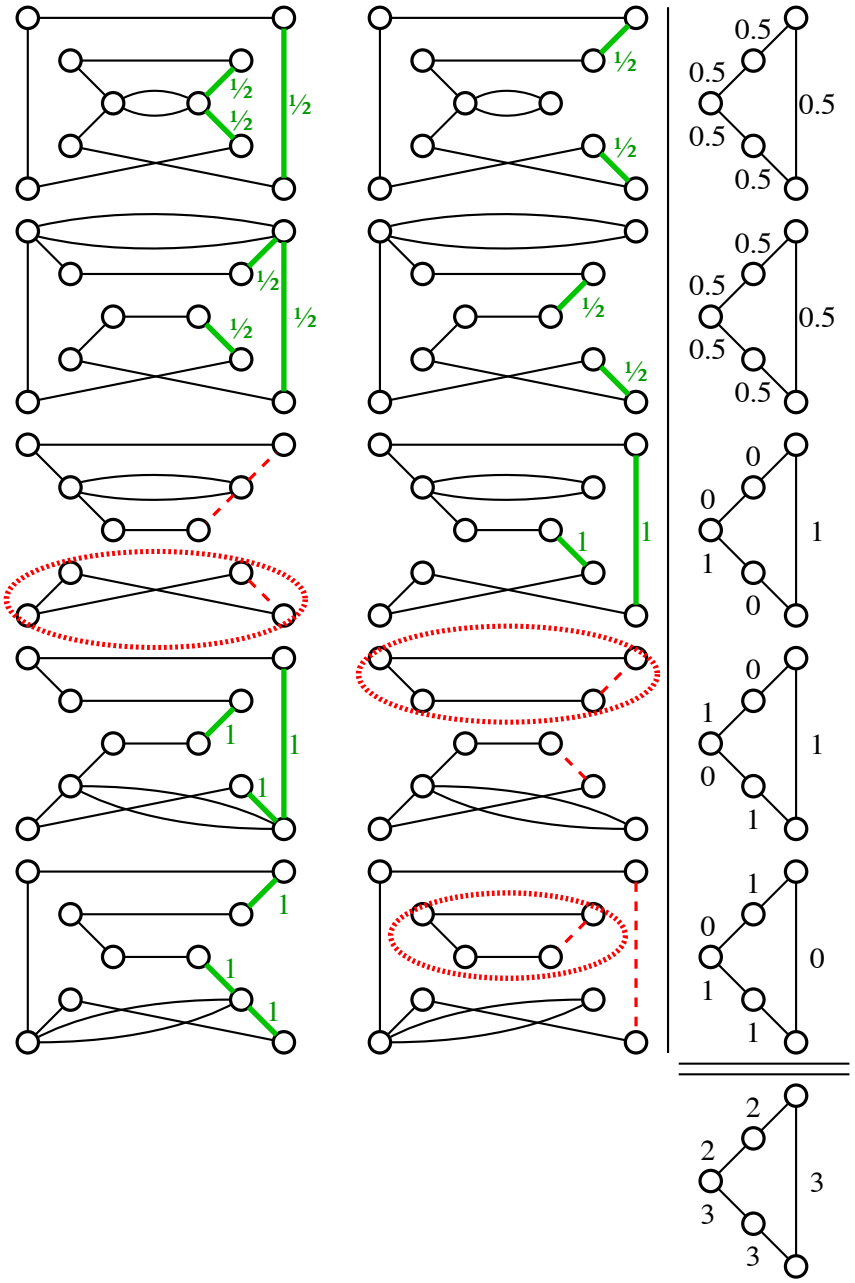

Figure 20: *All cases that could appear in Algorithm 5. If the tour disconnect the graph, we draw discontinued lines and we circle the disconnected part. Then, if the algorithm allows both possibilities, we consider every edge to be taken 0.5 times, else we count how many times each edge is taken. At the end of the day, we count how many times each edge appear for every path and we sum up everything to obtain at most three times each edge.*

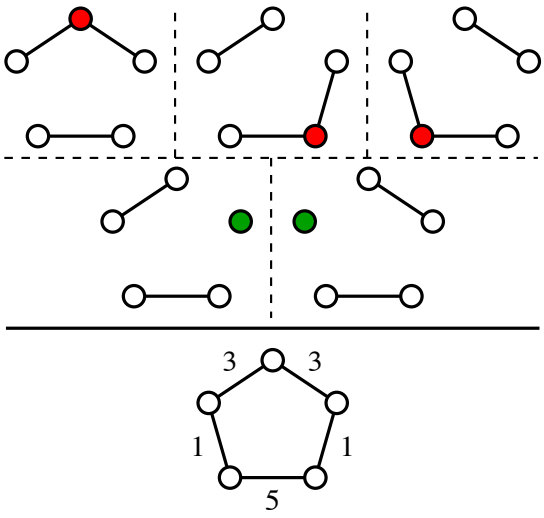

Figure 21*: We consider the case where one edge appears five times on the right cycle. The coloured vertex is the one where the duplicated path arrives. Then, in the three top cases, we have to pick $C_r(M)$, and in the three bottom cases, we have to pick $C_r \setminus C_r(M)$. Finally, we count how many times each edge appears, so we can replace once the cost of the five times chosen edge by the sum of the four others to have each cost added four times.*

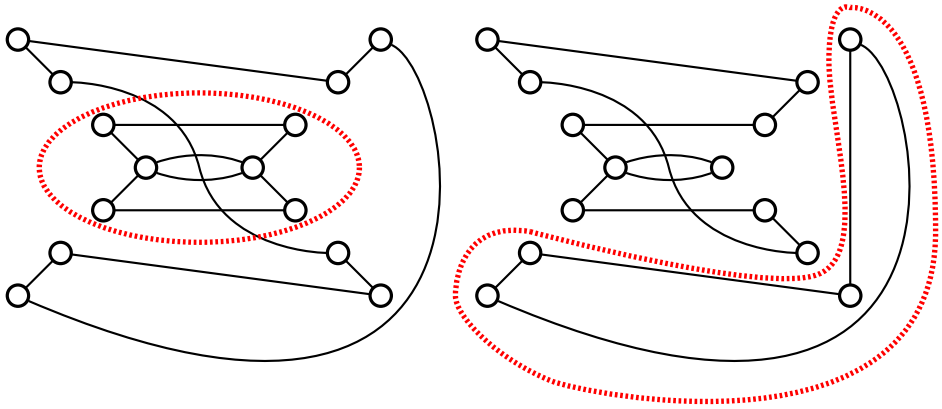

***Figure 22**: We have two 7-cycles, but both selections disconnect the graph.*

Conjecture 4.5. *Given two* $(2n+1)$*-cycles linked by seven paths of 2-degree vertices, where* $n \in \mathbb{N}$*, we can find a connected Eulerian graph by duplicating an odd number of paths and removing the right number of edges on the cycles, such that the cost of the Eulerian graph is at most* $\frac{4}{3}$ *times the cost of the original one.*

5 Conclusion

As we have seen, we can use a minimum spanning tree to find a 2-approximation and a 1.5-approximation [2], which is the best known result for the general case. This algorithm is very clear and elegant but in 40 years, we have not been able to improve it, even with something really sophisticated.

On the other hand, we have analysed the integrality gap of the relaxation in order to find a better approximation in another way. Even if all the specific cases we can think about do not exceed the $\frac{4}{3}$ bound, the upper bound is still $\frac{3}{2}$ for the general case.

Thus there are two ways of having a better understanding. We can think of a case where the gap is bigger than $\frac{4}{3}$, but this would be astonishing, since the consensus is that this bound is tight. Therefore we are trying to prove the integrality gap to be exactly $\frac{4}{3}$.

For some specific metric, we already have this result. We also proved it for some specific graphs: the 3-cycles connected to each other by paths to create a 2-vertex-connected graph and the two 5-cycles connected by five paths. If we could generalise either to more 5-cycles or for bigger cycles, it would be an interesting progress for understanding general metrics.

References

[1] Robert D. Carr and Goran Konjevod. Polyhedral combinatorics. In Harvey Greenberg, editor, *Tutorials on emerging methodologies and applications in Operations Research*, chapter 2, pages (2–1)–(2–48). Springer, 2004.

[2] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, CMU, 1976.

[3] Beth Spellman Munson Clyde L. Monma and William R. Pulleyblank. Minimum-weight two-connected spanning networks. *Mathematical Programming*, pages 153–171, 1990.

[4] Jack Edmonds. Maximum matching and a polyhedron with 0,1 vertices. *Journal of Research of the National Bureau of Standards*, 69:125–130, 1965.

[5] David Paul Williamson Frans Schalekamp and Anke van Zuylen. A proof of the boyd-carr conjecture. July 2011.

[6] A.M.H. Gerards. Matching. In *Network Models*, volume 7 of *Handbooks in Operations Research*, pages 503–615. Elsevier, 1995.

[7] Teofilo Gonzalez and Sartaj Sahni. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, July 1976.

[8] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[9] Richard M Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40(4):85–103, 1972.

[10] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, february 1956.

[11] Thomas L. Magnanti and Laurence A. Wolsey. Optimal trees. In *Network Models*, volume 7 of *Handbooks in Operations Research*, pages 135–224. Elsevier, 1995.

[12] Tobias Mömke and Ola Svensson. Approximating graphic tsp by matchings. *CoRR*, abs/1104.3090, 2011.

[13] Amin Saberi Shayan Oveis Gharan and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *FOCS*, pages 550–559, 2011.

[14] David B. Shmoys and David P. Williamson. Analyzing the held-karp tsp bound: a monotonicity property with application. *Information Processing Letters*, 35(6):281 – 285, 1990.

[15] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to Algorithms*. MIT Press, 1990 (first edition).

[16] David Paul Williamson. Analysis of the held-karp heuristic for the traveling salesman problem. Master's thesis, Massachusetts Institute of Technology, June 1990.

[17] Laurence A. Wolsey. Heuristic analysis, linear programming and branch and bound. In *Combinatorial Optimization II*, volume 13 of *Mathematical Programming Studies*, pages 121–134. Springer Berlin Heidelberg, 1980.