A Polynomial-space Exact Algorithm for TSP in Degree-5 Graphs

Norhazwani Md Yunos Aleksandar Shurbevski Hiroshi Nagamochi

Department of Applied Mathematics and Physics, Kyoto University
{wanie, shurbevski, nag}@amp.i.kyoto-u.ac.jp

Abstract: The Traveling Salesman Problem (TSP) is one of the most well-known NP-hard optimization problems. Following a recent trend of research which focuses on developing algorithms for special types of TSP instances, namely graphs of limited degree, and thus alleviating a part of the time and space complexity, we present a polynomial-space branching algorithm for the TSP in graphs with degree at most 5, and show that it has a running time of $O^*(2.4531^n)$. To the best of our knowledge, this is the first exact algorithm specialized to graphs of such high degree. While the base of the exponent in the running time bound is greater than two, our algorithm uses space merely polynomial in an input instance size, and thus by far outperforms Gurevich and Shelah's $O^*(4^n n^{\log n})$ polynomial-space exact algorithm for the general TSP (Siam Journal of Computation, Vol. 16, No. 3, pp. 486-502, 1987). In the analysis of the running time, we use the measure-and-conquer method, and we develop a set of branching rules which foster the analysis of the running time.

Keywords: Traveling Salesman Problem, Exact Exponential Algorithm, Branch-and-reduce, Measure-and-conquer.

1 Introduction

The Traveling Salesman Problem (TSP) is one of the most extensively studied problems in optimization. It has been formulated as a mathematical problem in the 1930s. Many algorithmic methods have been investigated to beat the challenge of finding the fastest algorithm in terms of running time. On the other hand, it has proven even more challenging to devise fast algorithms that would use a manageable amount of computation space, bounded by a polynomial in an input instance's size. We will review previous algorithmic attempts, making a distinction between those which require space exponential in the size of a problem instance, and those requiring space merely polynomial in the input size. We use the O^* notation, which suppresses polynomial factors.

[1]Technical Report 2015-002, June 22, 2015.

The first non-trivial algorithm for the TSP in an n-vertex graph is the $O^*(2^n)$-time dynamic programming algorithm discovered independently by Bellman [1], and Held and Karp [9] in the early 1960s. This dynamic programming algorithm however, requires also an exponential amount of space. Ever since, this running time has only been improved for special types of graphs. Primarily, investigation efforts have been focused on graphs in which vertices have a limited degree. Henceforth, let degree-i graph stand for a graph in which vertices have maximum degree at most i. A recent improvement of the time bound to $O^*(1.2186^n)$ for degree-3 graphs has been presented by Bodlaender et al. [2]. They have used a general approach for speeding up straightforward dynamic programming algorithms. For TSP in degree-4 graphs, Gebauer [7] has shown a time bound of $O^*(1.733^n)$, by using a dynamic programming approach.

In the vein of polynomial space algorithms, Gurevich and Shelah [8] have shown that the TSP in a general n-vertex graph is solvable in time $O^*\left(4^n n^{\log n}\right)$. Eppstein [4] has started the exploration into polynomial space TSP algorithms specialized for graphs of bounded degree by designing an algorithm for degree-3 graphs that runs in $O^*(1.260^n)$-time. He introduced a branch-and-search method by considering a generalization of the TSP called the forced TSP. Iwama and Nakashima [10] have claimed an improvement of Eppstein's time bound to $O^*(1.251^n)$-time for TSP in degree-3 graphs. Later, Liskiewicz and Schuster [11] have uncovered some oversights made in Iwama and Nakashima's analysis, and proved that their algorithm actually runs in $O^*(1.257^n)$-time. Liskiewicz and Schuster then made some minor modifications of Eppstein's algorithm and showed that this modified algorithm runs in $O^*(1.2553^n)$-time, a slight improvement over Iwama and Nakashima's algorithm. Xiao and Nagamochi [14] have recently presented an $O^*(1.2312^n)$-time algorithm for TSP in degree-3 graphs, and this improved previous time bounds for polynomial-space algorithms. They used the basic steps of Eppstein's branch-and-search algorithm, and introduced a branching rule based on a cut-circuit structure. In the process of improving the time bound, they used simple analysis of measure and conquer, and effectively analyzed their algorithm by introducing an amortization scheme over the cut circuit structure, setting weights to both vertices and connected components of induced graphs.

For TSP in degree-4 graphs, Eppstein [4] designed an algorithm that runs in $O^*(1.890^n)$-time, based on a branch-and-search method. Later, Xiao and Nagamochi [13] showed an improved value for the upper bound of the running time and showed that their algorithm runs in $O^*(1.692^n)$-time. Currently, this is the fastest algorithm for TSP in degree-4 graphs. Basically, the idea behind their algorithm is to apply reduction rules until no further reduction is possible, and then branch on an edge by either including it to a solution or excluding it from a solution. This is similar to most previous branch-and-search algorithms for the TSP. To effectively analyze their algorithm, Xiao and Nagamochi used the measure and conquer method by setting a weight to each vertex in a graph. From each branching operation, they derived a branching vector using the assigned weight and evaluate how much weight can be decreased in each of the two instances obtained by branching on a selected edge e. In this way, they were able to analyze by how much the total weight would decrease in each branch. Moreover, they indicated that the measure will decrease more if we select a "good" edge to branch on, and gave a set of simple rules, based on a graph's topological properties, for choosing such an edge. However, the analysis of the running time itself is

not as straightforward [13].

To the best of our knowledge, there exist no reports in the literature of exact algorithms specialized to the TSP in degree-5 graphs. Therefore, this paper presents the first algorithm for the TSP in degree-5 graphs, and presents an upper bound on the running time of $O^*(2.4531^n)$. In this exploration, we use a deterministic branch-and-search algorithm for TSP in degree-5 graphs. Basically, our algorithm employs similar techniques to most previous branching algorithms for the TSP. When there are no vertices of degree 5 in an input graph, we call an existing algorithm for TSP in degree-4 graphs, and solve the remaining instance. In the analysis, we use the measure and conquer method as a tool to get an upper bound of the running time.

The remainder of this paper is organized as follows; Section 2 overviews the basic notation used in this paper and presents an introduction to the branching algorithm and measure and conquer method. Section 3 describes our polynomial-space branching algorithm. We state our main result in section 4, where we proceed with the analysis of the proposed algorithm. Finally, Section 5 concludes the paper.

2 Preliminaries

For a graph G, let $V(G)$ denote the set of vertices in G, and let $E(G)$ denote the set of edges in G. A pair of vertices v and u are called neighbors if v and u are adjacent by an edge uv. We denote the set of all neighbors of a vertex v by $N(v)$, and denote by $d(v)$ the cardinality $|N(v)|$ of $N(v)$, also called the *degree* of v. For a subset of vertices $W \subseteq V(G)$, let $N(v;W) = N(v) \cap W$. For a subset of edges $E' \subseteq E(G)$, let $N_{E'}(v) = N(v) \cap \{u \mid uv \in E'\}$, and let $d_{E'}(v) = |N_{E'}(v)|$. Analogously, let $N_{E'}(v;W) = N_{E'}(v) \cap W$, and $d_{E'}(v,W) = |N_{E'}(v,W)|$. Also, for a subset E' of $E(G)$, we denote by $G - E'$ the graph $(V, E \setminus E')$ obtained from G by removing the edges in E'.

We consider a generalization of the TSP, named the *forced* Traveling Salesman Problem. We define an instance $I = (G, F)$ that consists of a simple, edge weighted, undirected graph G, and a subset F of edges in G, called *forced*. A vertex is called *forced* if exactly one of its incident edges is forced. Similarly, it is called *unforced* if no forced edge is incident to it. A Hamiltonian cycle in G is called a *tour* if it passes through all the forced edges in F. Under these circumstances, the forced TSP requests to find a minimum cost tour of an instance (G, F).

In this paper, we assume that the maximum degree of a vertex in G is at most 5. We denote a forced (resp., unforced) vertex of degree i by fi (resp., ui). We are interested in six types of vertices in an instance of (G, F), namely, u5, f5, u4, f4, u3 and f3-vertices. As shall be seen in Subsection 3.1, forced and unforced vertices of degree 2 and 1 are treated as special cases. Let $V_{\mathrm{f}i}$ (resp., $V_{\mathrm{u}i}$), $i = 3, 4, 5$ denote the set of fi-vertices (resp., ui-vertices) in (G, F).

2.1 Essentials on Branching Algorithms

We here review how to derive an upper bound on the number of instances that can be generated from an initial instance by a branching algorithm.

We can represent the solution space in our branching algorithm as a search tree. This is a very useful way to illustrate the execution of the branching rules, and to aid the time analysis of the branching algorithm. The search tree is obtained by assigning the input instance of a problem as a root node, and recursively assigning a child to a node for each smaller instance obtained by applying the branching rules. For a single node of the search tree, the algorithm takes time polynomial in the size of the node instance, which in turn, is smaller than or equal to the original instance size. Thus, we can conclude that the running time of the branching algorithm is equal to the number of nodes of the search tree times a polynomial of the original input instance size.

Let I be a given instance with size μ, and let I' and I'' be instances obtained from I by a branching operation. We use $T(\mu)$ to denote the maximum number of nodes in the search tree of an input of size μ when we execute our branching algorithm. Let a and b be the amounts of decrease in size of instances I' and I'', respectively; these values directly determine the performance of the algorithm. Then, we call (a, b) the branching vector of the branching rules, and this implies the linear recurrence:

$$T(\mu) \leq T(\mu - a) + T(\mu - b). \tag{1}$$

To evaluate the performance of this branching vector, we can use any standard method for linear recurrence relations. In fact, it is known that $T(\mu)$ is of the form $O(\tau^{\mu})$, where τ is the unique positive real root of the function $f(x) = 1 - \left(x^{-a} + x^{-b}\right)$ [6]. The value τ is called the *branching factor* (of a given branching vector), and the running time of the algorithm decreases with the value of this branching factor.

2.2 The Measure-and-Conquer Method

To effectively analyze our search tree algorithm, we use the measure and conquer method. A complete description of this method is beyond the scope of this paper, and the interested reader might refer to the book of Fomin and Kratsch [6].

The basic idea behind the measure and conquer method is to assign a measure to an instance, as opposed to using simply its size when analyzing the branching vectors of the branching operations. A good choice for a measure might lead to a significantly improved analysis on the upper bound of the running time of a branching algorithm. For example, Fomin et al. [5] have presented simple polynomial-space algorithms for the Maximum Independent Set and the Minimum Dominating Set Problem, and obtained an impressive refinement of the time analysis by using the measure and conquer method. This shows that a good choice of measure is very important to the time bounds achievable.

For a given problem instance I of size μ, let $W(I)$ be the measure of I. When considering a branch and reduce algorithm for the concerned problem, intuitively we seek for a measure which satisfies the following properties

(i) $W(I) = 0$ if and only if I can be solved in polynomial time;

(ii) If I' is a sub-instance of I obtained through a reduction or a branching operation, then $W(I') \leq W(I)$.

We call a measure W satisfying conditions (i) and (ii) above a *proper measure*.

3 A Polynomial-Space Branching Algorithm

We assume that the maximum degree of a vertex in a given graph G is at most 5. Basically, our algorithm contains two major steps. In the first step, the algorithm applies reduction rules until no further reduction is possible. In the second step, the algorithm applies branching rules in a reduced instance to search for a solution. These two steps are repeated iteratively.

As a result of the reduction and branching operations, the degree of some vertices will decrease, while the degree of other vertices will remain unchanged. A forced edge will never disappear, neither by the reduction nor branching operations, but an unforced edge may be erased by either of the reduction or branching operation. Throughout the process of the reduction and branching operations, the measure of an instance will never increase.

Details about the reduction and branching procedures will be discussed in the following sub-sections.

3.1 Reduction Rules

Reduction is a process of transforming an instance to a smaller instance. It takes polynomial-time to obtain a solution of an original instance from a solution of a smaller instance that has been obtained by a reduction procedure from the original instance.

Not all forced TSP instances have a tour. If an instance has no tour, we called it *infeasible*. Lemma 1 gives two sufficient conditions for an instance to be infeasible.

Lemma 1 *If one of the following conditions holds, then the instance (G, F) is infeasible.*

(i) $d(v) \leq 1$ *for some vertex* $v \in V(G)$.

(ii) $d_F(v) \geq 3$ *for some vertex* $v \in V(G)$.

In this paper, there are two reduction rules applied in each of the branching operation. These reduction rules preserve the minimum cost tour of an instance, as stated in Lemma 2.

Lemma 2 *Each of the following reductions preserves the feasibility and a minimum cost tour of an instance (G, F).*

(i) *If $d(v) = 2$ for a vertex v, then add to F any unforced edge incident to vertex v; and*

(ii) *If $d(v) > 2$ and $d_F(v) = 2$ for a vertex v, then remove from G any unforced edge incident to vertex v.*

Proof. Statements (i) and (ii) immediately follow from the definition of tours. □

From Lemma 1 and Lemma 2, we form our reduction algorithm as described in Figure 1. An instance (G, F) which does not satisfy any of the conditions in Lemma 1 and Lemma 2 is called *reduced*.

Input: An instance (G, F) such that the maximum degree of G is at most 5.
Output: A message for the infeasibility of (G, F); or a reduced instance (G', F') of (G, F).

Initialize $(G', F') := (G, F)$;

while (G', F') is not a reduced instance **do**

If there is a vertex v in (G', F') such that $d(v) \leq 1$ or $d_{F'}(v) \geq 3$ **then**
Return message "Infeasible"

Elseif there is a vertex v in (G', F') such that $2 = d(v) > d_{F'}(v)$ **then**
Let $E^\dagger$ be the set of unforced edges incident to all such vertices;
Set $F' := F' \cup E^\dagger$

Elseif there is a vertex v in (G', F') such that $d(v) > d_{F'}(v) = 2$ **then**
Let $E^\dagger$ be the set of unforced edges incident to all such vertices;
Set $G' := G' - E^\dagger$

End while;

Return (G', F').

Figure 1: Algorithm Red(G, F)

3.2 Branching Rules

Our algorithm iteratively branches on an unforced edge e in a reduced instance $I = (G, F)$ by either including e into F, **force**(e), or excluding it from G, **delete**(e). By applying a branching operation, the algorithm generates two new instances, called branches, by adding an unforced edge to F, or by removing it from G.

To describe our branching algorithm, let (G, F) be a reduced instance such that the maximum degree of G is at most 5. In (G, F), an unforced edge $e = vt$ incident to a vertex v of degree 5 is called *optimal*, if it satisfies a condition (c-i) below with minimum index i, over all unforced edges vt in (G, F):

(c-1) $v \in V_{\mathrm{f}5}$ and $t \in N_U(v; V_{\mathrm{f}3})$ such that $N_U(v) \cap N_U(t) = \emptyset$;
(c-2) $v \in V_{\mathrm{f}5}$ and $t \in N_U(v; V_{\mathrm{f}3})$ such that $N_U(v) \cap N_U(t) \neq \emptyset$;
(c-3) $v \in V_{\mathrm{f}5}$ and $t \in N_U(v; V_{\mathrm{u}3})$;
(c-4) $v \in V_{\mathrm{f}5}$ and $t \in N_U(v; V_{\mathrm{f}4})$ such that $N_U(v) \cap N_U(t) = \emptyset$;
(c-5) $v \in V_{\mathrm{f}5}$ and $t \in N_U(v; V_{\mathrm{f}4})$ such that $N_U(v) \cap N_U(t) \neq \emptyset$;
  (I) $|N_U(v) \cap N_U(t)| = 1$; and
  (II) $|N_U(v) \cap N_U(t)| = 2$;
(c-6) $v \in V_{\mathrm{f}5}$ and $t \in N_U(v; V_{\mathrm{u}4})$;
(c-7) $v \in V_{\mathrm{f}5}$ and $t \in N_U(v; V_{\mathrm{f}5})$ such that $N_U(v) \cap N_U(t) = \emptyset$;
(c-8) $v \in V_{\mathrm{f}5}$ and $t \in N_U(v; V_{\mathrm{f}5})$ such that $N_U(v) \cap N_U(t) \neq \emptyset$;
  (I) $|N_U(v) \cap N_U(t)| = 1$;
  (II) $|N_U(v) \cap N_U(t)| = 2$; and

(III) $|N_U(v) \cap N_U(t)| = 3$;
(c-9) $v \in V_{\text{f5}}$ and $t \in N_U(v; V_{\text{u5}})$;
(c-10) $v \in V_{\text{u5}}$ and $t \in N_U(v; V_{\text{f3}})$;
(c-11) $v \in V_{\text{u5}}$ and $t \in N_U(v; V_{\text{f4}})$;
(c-12) $v \in V_{\text{u5}}$ and $t \in N_U(v; V_{\text{u3}})$;
(c-13) $v \in V_{\text{u5}}$ and $t \in N_U(v; V_{\text{u4}})$; and
(c-14) $v \in V_{\text{u5}}$ and $t \in N_U(v; V_{\text{u5}})$.

We refer to the above conditions for choosing an optimal edge to branch on, c-1 to c-14, as the *branching rules*. The collective set of branching rules are illustrated in Figure 2.

For convenience in the analysis of the algorithm, case (c-5) and case (c-8) have been subdivided into subcases according to the cardinality of the neighborhood intersection. Intersections of lower cardinality take precedence over higher ones.

Given a reduced instance $I = (G, F)$, our algorithm first checks whether there exists a vertex of degree 5, and if it does, chooses an optimal edge according to the branching rules. If there exists no optimal edge according to the branching rules, then the reduced instance has no more vertices of degree 5, and the maximum degree of the reduced instance at this point is at most 4. Then, we can call a polynomial space exact algorithm for the TSP that is specialized for degree-4 graphs, e.g., the algorithm specialized for degree-4 graphs by Xiao and Nagamochi [13]. Our branching algorithm is described in Figure 3.

4 Analysis

4.1 Main Result

In order to adopt the measure and conquer method in our algorithm, we need to set a measure W for a given an instance $I = (G, F)$ of the forced TSP. To this effect, we set a non-negative vertex weight function $\omega : V \to \mathbb{R}_+$ in the graph G, and we use the sum of weights of all vertices in the graph as the measure $W(I)$ of instance I. That is,

$$W(I) = \sum_{v \in V(G)} (\omega(v)) \,. \tag{2}$$

We bring to attention the fact that the number n of vertices in the graph G remains unmodified throughout the process of the reduction and branching operations. In addition to seeking a proper measure, we also require that the weight of each vertex is not greater than 1, and therefore, the measure $W(I)$ will not be greater than the number n of vertices in G. As a consequence, a running time bound as a function of the measure $W(I)$ implies the same running time bound as a function of n. The weight assigned to each vertex type plays an important role, since the value of the branching factor depends solely on these weights.

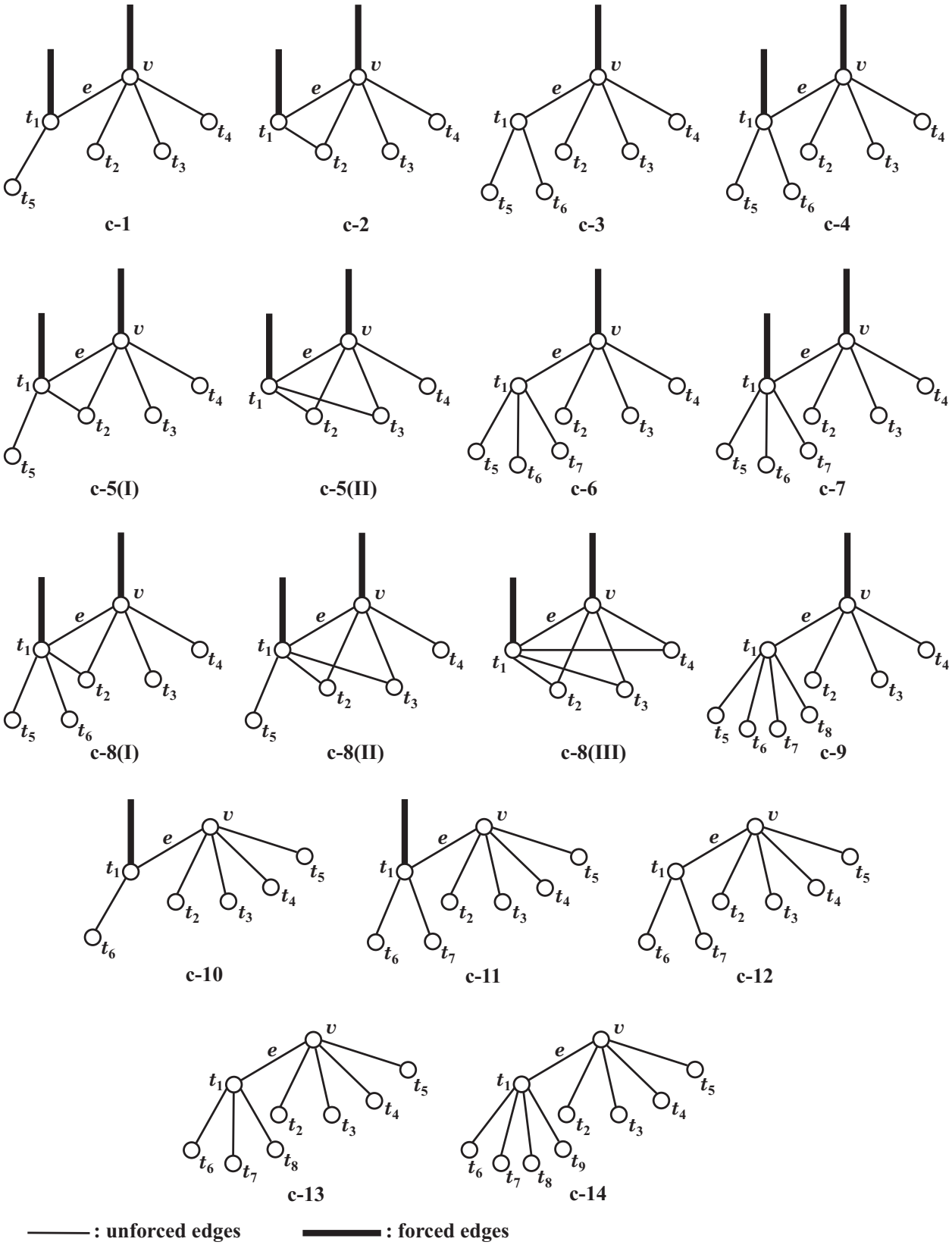


Figure 2: Illustration of the Branching Rules

Let the vertex weight function $\omega(v)$ be chosen as follows:

$$\omega(v) = \begin{cases} w_{3'} = 0.156687 & \text{for an f3-vertex } v \\ w_3 = 0.276915 & \text{for a u3-vertex } v \\ w_{4'} = 0.313373 & \text{for an f4-vertex } v \\ w_4 = 0.607542 & \text{for a u4-vertex } v \\ w_{5'} = 0.470059 & \text{for an f5-vertex } v \\ w_5 = 1 & \text{for a u5-vertex } v \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Lemma 3 *If the vertex weight function $\omega(v)$ is set as in Eq.* (3)*, then each branching operation in Figure 3 has a branching factor not greater than* 2.453051*.*

A proof of Lemma 3 will be derived analytically in the several subsections which follow. From the lemma, we get our main result:

Theorem 1 *The TSP in an n-vertex graph G with maximum degree 5 can be solved in $O^*(2.4531^n)$-time and polynomial-space.*

Input: An instance (G, F) such that the maximum degree of G is at most 5.
Output: A message for the infeasibility of (G, F); or the minimum cost of a tour of (G, F).

Run Red(G, F);

If Red(G', F') returns message "Infeasible" **then**

  Return message "Infeasible"

Else

  Let $(G', F') :=$ Red(G, F);

  If $V_{\text{u5}} \cup V_{\text{f5}} \neq \emptyset$ **then**
    Choose an optimal unforced edge e
    Return $\min\{\text{tsp5}(G', F' \cup \{e\}), \text{tsp5}(G' - \{e\},\ F')\}$
  Else /* there is no vertex of degree 5 in (G', F') */
    Return tsp4(G', F').

Note: The input and output of algorithm tsp4(G, F) are as follows
Input: An instance (G, F) such that the maximum degree of G is at most 4.
Output: A message for the infeasibility of (G, F); or the minimum cost of a tour of (G, F).

Figure 3: Algorithm tsp5(G, F)

4.2 Weight Constraints

We have stated that we confine the weights of individual vertices in an instance not to be greater than 1, in order to obtain a measure which will imply the same running time bound as a function of the size of a TSP instance. In what follows, we examine some necessary constraints which the vertex weights should satisfy in order to obtain a proper measure.

For $i = \{3, 4, 5\}$, we denote w_i to be the weight of a ui-vertex, and $w_{i'}$ to be the weight of an fi-vertex. The conditions for a proper measure require that the measure of an instance obtained through a branching or a reduction operation will not be greater than the measure of the original instance. Thus, vertex weights should satisfy the following relations;

$$w_5 \leq 1, \tag{4}$$

$$w_{5'} \leq w_5, \tag{5}$$

$$w_{4'} \leq w_4, \tag{6}$$

$$w_{3'} \leq w_3, \tag{7}$$

$$w_3 \leq w_4 \leq w_5, \text{ and} \tag{8}$$

$$w_{3'} \leq w_{4'} \leq w_{5'}. \tag{9}$$

The vertex weight for vertices of degree less than 3 is set to be 0.

We proceed to show in the algorithms given in Figures 1 and 3, setting vertex weights which satisfy the conditions of Eqs. (5) to (9) is sufficient to obtain a proper measure.

Lemma 4 *If the weights of vertices are chosen as in Eqs.* (5) *to* (9)*, then the measure* $W(I)$ *never increases as a result of the reduction or the branching operations of Figure 1 and Figure 3.*

Proof. Let $I = (G, F)$ be a given instance of the forced TSP. Due to our definition of the measure $W(I)$ of Eq.(2), it suffices to show that none of the individual vertex weights will increase as a result of a reduction or a branching operation in the algorithms of Figures 1 and 3.

The branching rules state that for an unforced edge e in $E(G) \setminus F$, two subinstances are generated by either setting $F := f \cap \{e\}$, termed as **force**(e), or by setting $G := G - \{e\}$, termed **delete**(e). In fact, we bring to attention that a reduction operation, if it does not return a message "Infeasible," is in fact a repeated application of the above two steps, **force**(e) or **delete**(e), for some unforced edge e, identified by the conditions in Lemma 2. Therefore, we proceed with analyzing the effects of applying the **force**(e) or the **delete**(e) operation.

Let $e = uv$ be an unforced edge to which one of the **force**(e) or **delete**(e) operations will be applied. Both u and v must have degree more than 2, otherwise by Lemma 1, the instance is infeasible. Without loss of generality, we observe the effect of the operation on the vertex weight $\omega(v)$.

In the case that operation **force**(e) is applied, the following cases may arise.

– If v is an unforced vertex, then v will become forced. By Eqs. (5) to (7), the weight $\omega(v)$ will not increase.

– If v is a forced vertex, then $\omega(v)$ will become 0.

– If $d_F(v) = 2$, then by Lemma 1 the instance will become infeasible.

On the other hand, if operation **delete**(e) is applied, we observe the following cases.

– If v is either forced or unforced, and $d(v) \geq 3$, then the degree of v will decrease by one, and by Eqs. (8) and (9) $\omega(v)$ will not increase.

– If v is either forced or unforced, and $d(v) \leq 2$, then by Lemma 1 the instance will become infeasible.

Following the above observations, we conclude that the complete measure $W(I)$ of a given instance $I = (G, F)$ of the forced TSP will not increase as a result of the reduction and branching operations of the algorithms in Figures 3 and 1. □

To simplify some arguments, we introduce the following notation;

$$\begin{aligned}
\Delta_3 &= w_3 - w_{3'}, \\
\Delta_4 &= w_4 - w_{4'}, \\
\Delta_5 &= w_5 - w_{5'}, \\
\Delta_{4-3} &= w_4 - w_3, \\
\Delta_{5-4} &= w_5 - w_4, \\
\Delta_{5-3} &= w_5 - w_3, \\
\Delta'_{4-3} &= w_{4'} - w_{3'}, \\
\Delta'_{5-4} &= w_{5'} - w_{4'}, \text{ and} \\
\Delta'_{5-3} &= w_{5'} - w_{3'}.
\end{aligned}$$

To simplify the list of our branching vectors, we use the following notation;

$$m_1 = \min\{w_{3'},\ w_3,\ \Delta'_{4-3},\ \Delta_{4-3},\ \Delta'_{5-4},\ \Delta_{5-4}\}, \tag{10}$$
$$m_2 = \min\{w_3,\ \Delta'_{4-3},\ \Delta_{4-3},\ \Delta'_{5-4},\ \Delta_{5-4}\}, \tag{11}$$
$$m_3 = \min\{w_{3'},\ \Delta_3,\ w_{4'},\ \Delta_4,\ w_{5'},\ \Delta_5\}, \tag{12}$$
$$m_4 = \min\{\Delta'_{4-3},\ \Delta_{4-3},\ \Delta'_{5-4},\ \Delta_{5-4}\}, \tag{13}$$
$$m_5 = \min\{w_{4'},\ w_4,\ \Delta'_{5-3},\ \Delta_{5-3}\}, \tag{14}$$
$$m_6 = \min\{\Delta_{4-3},\ \Delta'_{5-4},\ \Delta_{5-4}\}, \tag{15}$$
$$m_7 = \min\{\Delta'_{5-4},\ \Delta_{5-4}\}, \tag{16}$$
$$m_8 = \min\{\Delta'_{5-3},\ \Delta_{5-3}\}, \tag{17}$$
$$m_9 = \min\{w_{3'},\ w_3,\ \Delta'_{4-3},\ \Delta_{4-3},\ \Delta_{5-4}\}, \text{ and} \tag{18}$$
$$m_{10} = \min\{w_{3'},\ \Delta_3,\ w_{4'},\ \Delta_4,\ \Delta_5\}. \tag{19}$$

In the remainder of the analysis, for an optimal edge $e = vt_1$, we denote $N_U(v)$ by $\{t_1, t_2, \ldots, t_a\}$, $a = d_U(v)$, and $N_U(t_1) \setminus \{v\}$ by $\{t_{a+1}, t_{a+1}, \ldots, t_{a+b}\}$, $b = d_U(t_1) - 1$. We assume without loss of generality that $t_{1+i} = t_{a+i}$ for $i = 1, 2, \ldots, c$, where $c = |N_U(v) \cap N_U(t_1)|$, the number of good neighbors that v and t_1 have in common.

4.3 Branching on Edges Around f5-vertices (c-1 to c-9)

This section derives branching vectors for the branching operation on an optimal edge $e = vt_1$, incident to an f5-vertex v, distinguishing nine cases for conditions c-1 to c-9.

Case c-1: There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{f3}})$ such that $N_U(v) \cap N_U(t_1) = \emptyset$ (see Figure 4): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_5\}$.

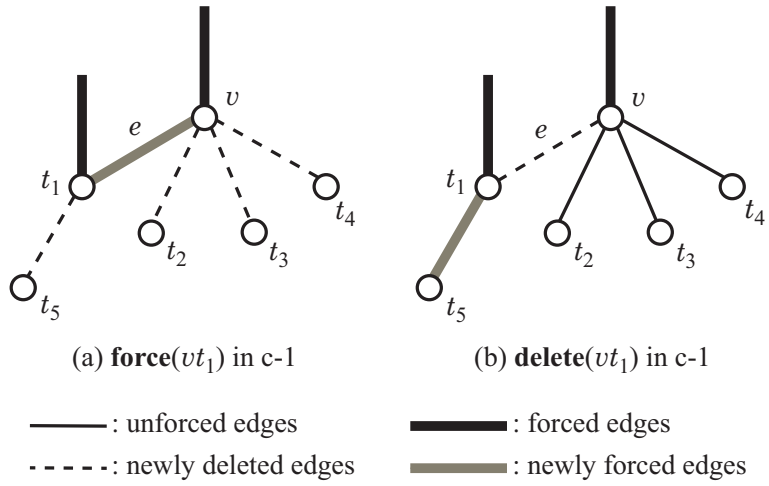


Figure 4: Illustration of branching rule c-1, where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f3}})$ such that $N_U(v) \cap N_U(t_1) = \emptyset$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3, vt_4 and t_1t_5 will be deleted from G' by the reduction rules. Both v and t_1 will become vertices of degree 2. From Eq. (3), the weight of vertices of degree 2 is 0. So, the weight of vertex v decreases by $w_{5'}$ and the weight of vertex t_1 decreases by $w_{3'}$. Each of the vertices t_2, t_3 and t_4 can be either a type f3, u3, f4, u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_1 = \min\{w_{3'}, w_3, \Delta'_{4-3}, \Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\}$. If vertex t_5 is an f3-vertex (resp., u3, f4, u4, f5, or a u5-vertex), then the weight decrease α of vertex t_5 would be $w_{3'}$ (resp., w_3, Δ'_{4-3}, Δ_{4-3}, Δ'_{5-4}, and Δ_{5-4}). Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_{5'} + w_{3'} + 3m_1 + \alpha)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation, and edge t_1t_5 will be added to F' by the reduction rules. The weight of vertex v decreases by Δ'_{5-4} and the weight of vertex t_1 decreases by $w_{3'}$. If vertex t_5 is an f3-vertex (resp., u3, f4, u4, f5, or a u5-vertex), then the weight decrease β of vertex t_5 would be $w_{3'}$ (resp., Δ_3, $w_{4'}$, Δ_4, $w_{5'}$, and Δ_5). Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_{5'} - w_{4'} + w_{3'} + \beta)$.

As a result, we get the following six branching vectors:

$$(w_{5'} + w_{3'} + 3m_1 + \alpha,\ w_{5'} - w_{4'} + w_{3'} + \beta) \tag{20}$$

for $(\alpha, \beta) \in \{(w_{3'}, w_{3'}), (w_3, \Delta_3), (\Delta'_{4-3}, w_{4'}), (\Delta_{4-3}, \Delta_4), (\Delta'_{5-4}, w_{5'}), (\Delta_{5-4}, \Delta_5)\}$.

c-2. There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{f3}})$ such that $N_U(v) \cap N_U(t_1) = \{t_2\}$ (see Figure 5): We branch on edge vt_1.

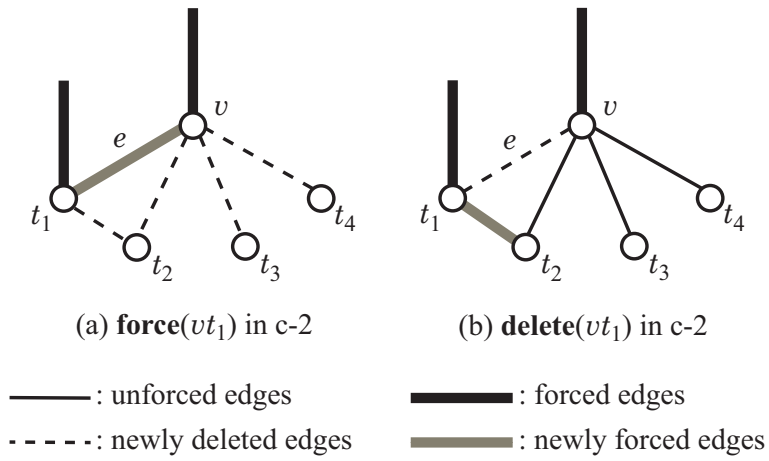


Figure 5: Illustration of branching rule c-2, where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f3}})$ such that $N_U(v) \cap N_U(t_1) = \{t_2\}$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3, vt_4 and t_1t_2 will be deleted from G' by the reduction rules. So, the weight of vertex v decreases by $w_{5'}$, and the weight of vertex t_1 decreases by $w_{3'}$. Each of the vertices t_3 and t_4 can be either a type f3, u3, f4, u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_1 = \min\{w_{3'}, w_3, \Delta'_{4-3}, \Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\}$.

There are two possible cases for the vertex type of vertex t_2. First, let t_2 be an f3 or u3-vertex. After performing the branching operation, t_2 would become a vertex of degree 1. By Lemma 1, case (i), this is infeasible, and the algorithm will return a message of infeasibility.

Second, let t_2 be an f4, u4, f5, or u5-vertex. If t_2 is an f4-vertex (resp., u4, f5, or a u5-vertex), then the weight decrease α of vertex t_2 would be $w_{4'}$ (resp., w_4, Δ'_{5-3}, and Δ_{5-3}). Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_{5'} + w_{3'} + 2m_1 + \alpha)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation, and edge t_1t_2 will be added to F' by the reduction rules. So, the weights of vertices v and t_1 decrease by Δ'_{5-4} and $w_{3'}$, respectively. If vertex t_2 is an f4-vertex (resp., u4, f5, or a u5-vertex), then the weight decrease β of vertex t_2 would be $w_{4'}$ (resp., Δ_4, $w_{5'}$, and Δ_5). Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_{5'} - w_{4'} + w_{3'} + \beta)$.

As a result, we get the following four branching vectors:

$$(w_{5'} + w_{3'} + 2m_1 + \alpha,\ w_{5'} - w_{4'} + w_{3'} + \beta) \tag{21}$$

for $(\alpha, \beta) \in \{(w_{4'}, w_{4'}), (w_4, \Delta_4), (\Delta'_{5-3}, w_{5'}), (\Delta_{5-3}, \Delta_5)\}$.

c-3. There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{u3}})$ (see Figure 6): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_5, t_6\}$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3 and vt_4 will be deleted from G' by the reduction rules. So, the weights of vertices v and t_1 decrease by $w_{5'}$ and Δ_3, respectively. Each of vertices t_2, t_3 and t_4 can be either a type u3, f4, u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_2 = \min\{w_3, \Delta'_{4-3}, \Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\}$. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_{5'} + w_3 - w_{3'} + 3m_2)$.

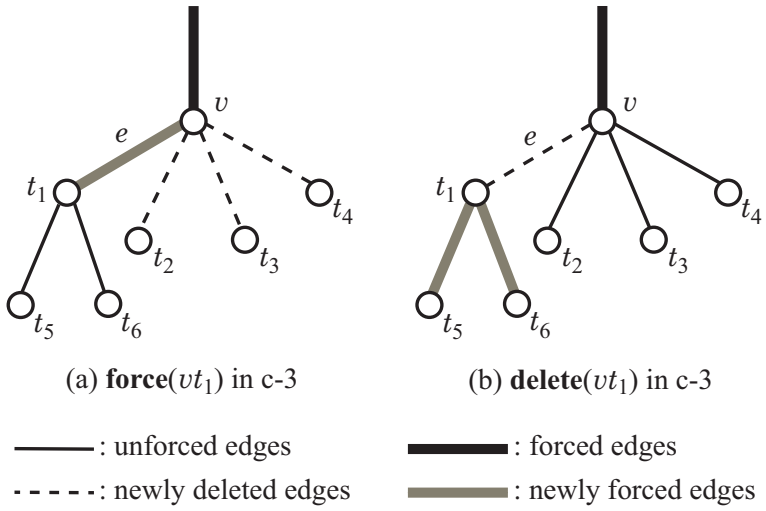


Figure 6: Illustration of branching rule c-3, where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{u3}})$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation and edges t_1t_5 and t_1t_6 will be added to F' by the reduction rules. So, the weight of vertex v decreases by Δ'_{5-4} and the weight of vertex t_1 decreases by w_3. Each of the vertices t_5 and t_6 can be either a type f3, u3, f4, u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_3 = \min\{w_{3'}, \Delta_3, w_{4'}, \Delta_4, w_{5'}, \Delta_5\}$. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_{5'} - w_{4'} + w_3 + 2m_3)$.

As a result, we get the following branching vector:

$$(w_{5'} + w_3 - w_{3'} + 3m_2,\ w_{5'} - w_{4'} + w_3 + 2m_3)\,. \tag{22}$$

c-4. There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{f4}})$ such that $N_U(v) \cap N_U(t_1) = \emptyset$ (see Figure 7): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_5, t_6\}$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3, vt_4, t_1t_5 and t_1t_6 will be deleted from G' by the reduction rules. So, the weight of vertex v decreases by $w_{5'}$ and the weight of vertex t_1 decreases by $w_{4'}$. Each of the vertices t_2, t_3 and t_4 can be either a type f4, u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_4 = \min\left\{\Delta'_{4-3}, \Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\right\}$. Each of vertices t_5 and t_6 can be either a type f3, u3, f4, u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_1 = \min\left\{w_{3'}, w_3, \Delta'_{4-3}, \Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\right\}$. Thus, the total decrease in the branch of **force**(vt_1) is at least $(w_{5'} + w_{4'} + 3m_4 + 2m_1)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, the weight of vertex v decreases by Δ'_{5-4} and the weight of vertex t_1 decreases by Δ'_{4-3}. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_{5'} - w_{3'})$. As a result, we get the following branching vector:

$$(w_{5'} + w_{4'} + 3m_4 + 2m_1,\ w_{5'} - w_{3'})\,. \tag{23}$$

c-5. There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{f4}})$ such that $N_U(v) \cap N_U(t_1) \neq \emptyset$. We

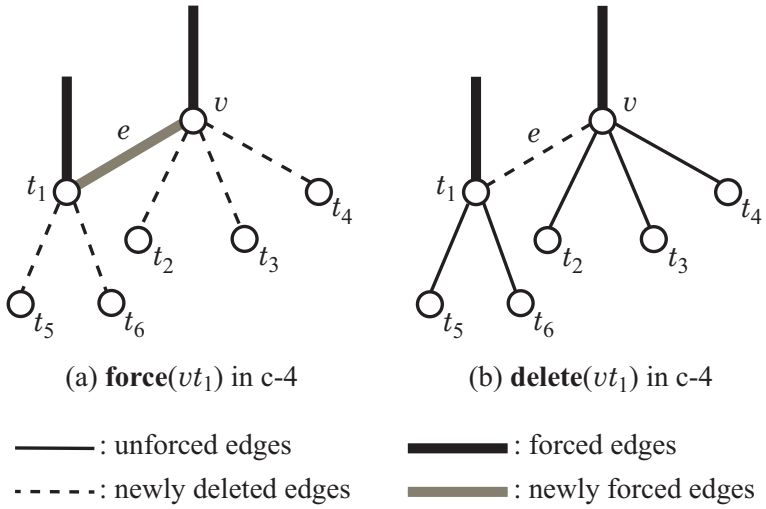


Figure 7: Illustration of branching rule c-4, where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f4}})$, such that $N_U(v) \cap N_U(t_1) = \emptyset$.

distinguish two sub cases, according to the cardinality of the intersection $N_U(v) \cap N_U(t_1)$, (c-5(I)), $|N_U(v) \cap N_U(t_1)| = 1$, and (c-5(II)), $|N_U(v) \cap N_U(t_1)| = 2$.

c-5(I). Without loss of generality, assume that $N_U(v) \cap N_U(t_1) = \{t_2\}$ (see Figure 8): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_5\}$.

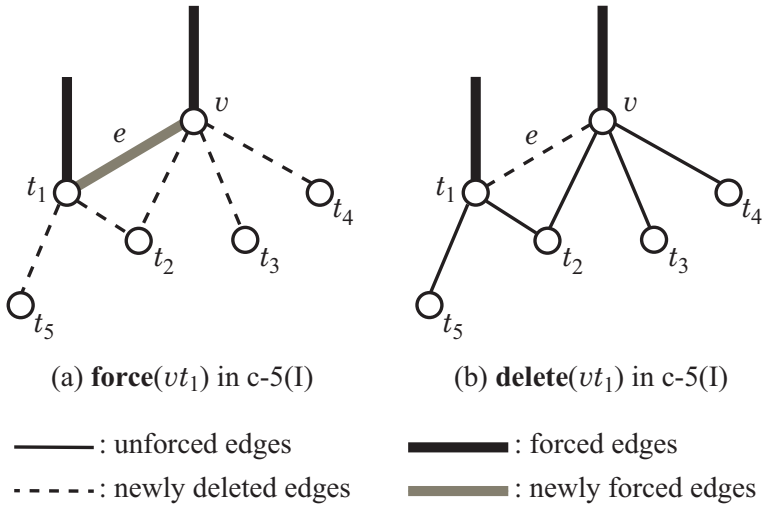


Figure 8: Illustration of branching rule c-5(I), where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f4}})$, such that $N_U(v) \cap N_U(t_1) = \{t_2\}$.

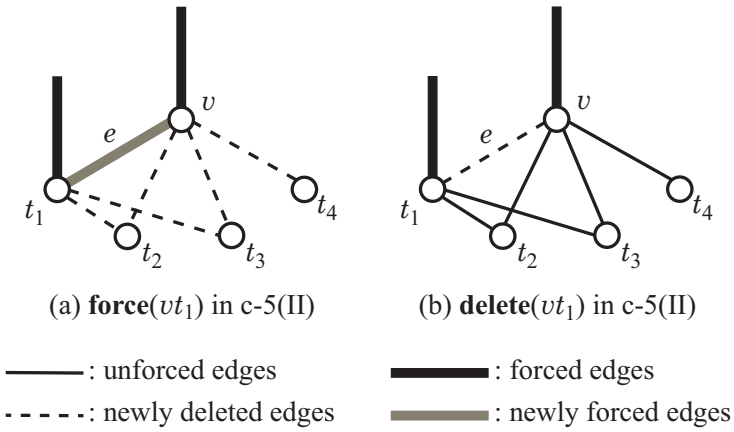
In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3, vt_4, t_1t_2, and t_1t_5 will be deleted from G' by the reduction rules. So, the weight of vertex v decreases by $w_{5'}$ and the weight of vertex t_1 decreases by $w_{4'}$. Vertex t_2 can be either a type f4, u4, f5, or u5-vertex, and its weight would decrease by at least $m_5 = \min\{w_{4'}, w_4, \Delta'_{5-3}, \Delta_{5-3}\}$. Each of the vertices t_3 and t_4 can be either a type f4, u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_4 =$

$\min\{\Delta'_{4-3}, \Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\}$. Vertex t_5 can be either a type f3, u3, f4, u4, f5, or u5-vertex, and its weight would decrease by at least $m_1 = \min\{w_{3'}, w_3, \Delta'_{4-3}, \Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\}$. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_{5'} + w_{4'} + m_5 + 2m_4 + m_1)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, the weight of vertex v decreases by Δ'_{5-4}, and the weight of vertex t_1 decreases by Δ'_{4-3}. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_{5'} - w_{3'})$. As a result, we get the following branching vector:

$$(w_{5'} + w_{4'} + m_5 + 2m_4 + m_1,\ w_{5'} - w_{3'})\,. \tag{24}$$

c-5(II). Without loss of generality, assume that $N_U(v) \cap N_U(t_1) = \{t_2, t_3\}$ (see Figure 9): We branch on edge vt_1.



(a) **force**(vt_1) in c-5(II) (b) **delete**(vt_1) in c-5(II)

Figure 9: Illustration of branching rule c-5(II), where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f4}})$ such that $N_U(v) \cap N_U(t_1) = \{t_2, t_3\}$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edge vt_2, vt_3, vt_4, t_1t_2 and t_1t_3 will be deleted from G' by the reduction rules. So, the weight of vertex v decreases by $w_{5'}$ and the weight of vertex t_1 decreases by $w_{4'}$. Each of vertices t_2 and t_3 can be either a type f4, u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_5 = \min\{w_{4'}, w_4, \Delta'_{5-3}, \Delta_{5-3}\}$. Vertex t_4 can be either a type f3, u3, f4, u4, f5, or u5-vertex, and its weight would decreases by at least $m_4 = \min\{\Delta'_{4-3}, \Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\}$. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_{5'} + w_{4'} + 2m_5 + m_4)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, the weight of vertex v decreases by Δ'_{5-4}, and the weight of vertex t_1 decreases by Δ'_{4-3}. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_{5'} - w_{3'})$. As a result, we get the following branching vector:

$$(w_{5'} + w_{4'} + 2m_5 + m_4,\ w_{5'} - w_{3'})\,. \tag{25}$$

c-6. There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{u4}})$ (see Figure 10): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_5, t_6, t_7\}$.

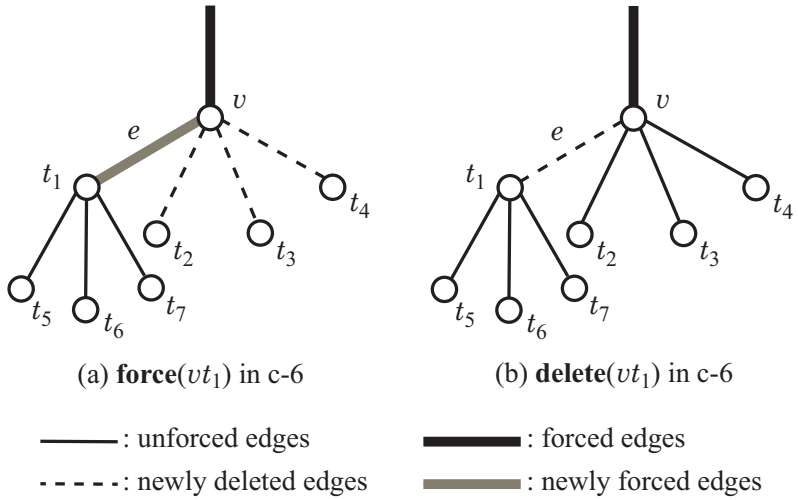


Figure 10: Illustration of branching rule c-6, where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{u4}})$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3 and vt_4 will be deleted from G' by the reduction rules. So, the weight of vertex v decreases by $w_{5'}$ and the weight of vertex t_1 decreases by Δ_4. Each of the vertices t_2, t_3 and t_4 can be either a type u4, f5, or u5-vertex, and each of their weights would decrease by at least $m_6 = \min\left\{\Delta_{4-3}, \Delta'_{5-4}, \Delta_{5-4}\right\}$. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_{5'} + w_4 - w_{4'} + 3m_6)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, the weight of vertex v decreases by Δ'_{5-4} and the weight of vertex t_1 decreases by Δ_{4-3}. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_{5'} - w_{4'} + w_4 - w_3)$. As a result, we get the following branching vector:

$$(w_{5'} + w_4 - w_{4'} + 3m_6,\ w_{5'} - w_{4'} + w_4 - w_3)\,. \tag{26}$$

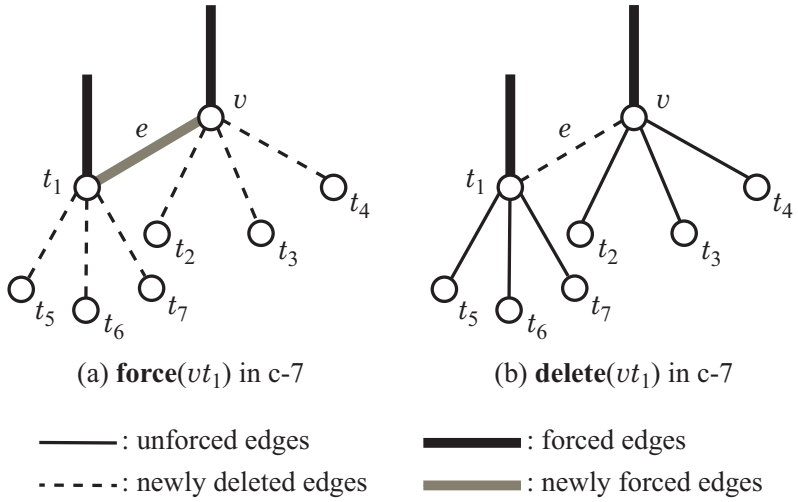
c-7. There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{f5}})$ such that $N_U(v) \cap N_U(t_1) = \emptyset$ (see Figure 11): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_5, t_6, t_7\}$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3, vt_4, t_1t_5, t_1t_6 and t_1t_7 will be deleted from G' by the reduction rules. So, both weights of vertex v and vertex t_1 decreases by $w_{5'}$, each. Each of vertices t_2, t_3, t_4, t_5, t_6 and t_7 can be either a type f5, or u5-vertex, and each of their weights would decrease by at least $m_7 = \min\left\{\Delta'_{5-4}, \Delta_{5-4}\right\}$. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(2w_{5'} + 6m_7)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, both weights of vertices v and t_1 decreases by Δ'_{5-4}, each. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(2w_{5'} - 2w_{4'})$. As a result, we get the following branching vector:

$$(2w_{5'} + 6m_7,\ 2w_{5'} - 2w_{4'})\,. \tag{27}$$

c-8. There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{f5}})$ such that $N_U(v) \cap N_U(t_1) \neq \emptyset$.



(a) **force**(vt_1) in c-7 (b) **delete**(vt_1) in c-7

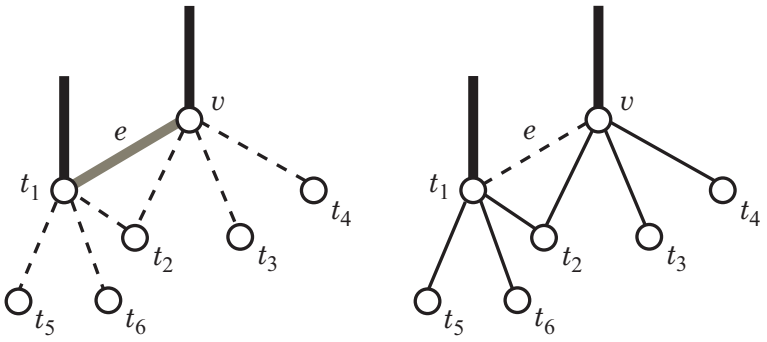
: unforced edges : forced edges

: newly deleted edges : newly forced edges

Figure 11: Illustration of branching rule c-7, where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f5}})$, such that $N_U(v) \cap N_U(t_1) = \emptyset$.

We distinguish three sub cases, according to the cardinality of the intersection $N_U(v) \cap N_U(t_1)$, (c-8(I)), $|N_U(v) \cap N_U(t_1)| = 1$, (c-8(II)), $|N_U(v) \cap N_U(t_1)| = 2$, and (c-8(III)), $|N_U(v) \cap N_U(t_1)| = 3$.

c-8(I). Without loss of generality, assume that $N_U(v) \cap N_U(t_1) = \{t_2\}$ (see Figure 12): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_5, t_6\}$.



(a) **force**(vt_1) in c-8(I) (b) **delete**(vt_1) in c-8(I)

: unforced edges : forced edges

: newly deleted edges : newly forced edges

Figure 12: Illustration of branching rule c-8(I), where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f5}})$, such that $N_U(v) \cap N_U(t_1) = \{t_2\}$.

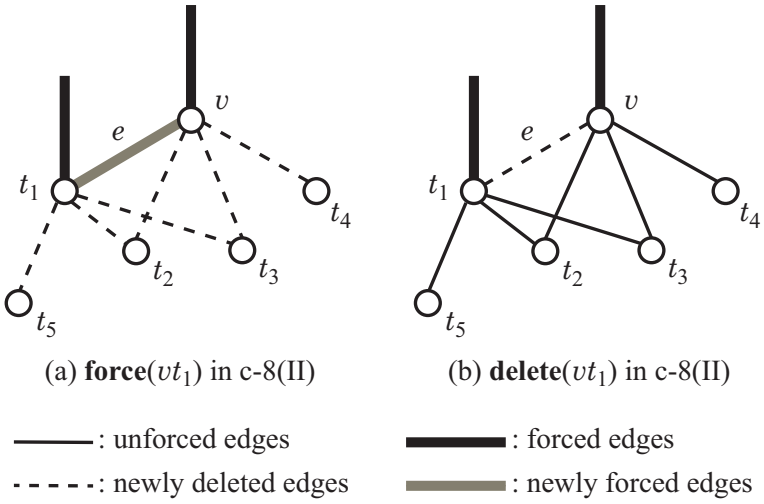
In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3, vt_4, t_1t_2, t_1t_5 and t_1t_6 will be deleted from G' by the reduction rules. Both weights of vertex v and vertex t_1 decreases by $w_{5'}$, each. Vertex t_2 can be either a type f5 or u5-vertex, and its weight would decreases by at least $m_8 = \min\left\{\Delta'_{5-3}, \Delta_{5-3}\right\}$. Each of vertices t_3, t_4, t_5 and t_6 can be either a type f5, or u5-vertex, and each of their weights would decrease by at least $m_7 = \min\left\{\Delta'_{5-4}, \Delta_{5-4}\right\}$. Thus, the total weight decrease in the

branch of **force**(vt_1) is at least $(2w_{5'} + 4m_7 + m_8)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. Both weights of vertices v and t_1 decreases by Δ'_{5-4}, each. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(2w_{5'} - 2w_{4'})$. As a result, we get the following branching vector:

$$(2w_{5'} + 4m_7 + m_8,\ 2w_{5'} - 2w_{4'})\,. \tag{28}$$

c-8(II). Without loss of generality, assume that $N_U(v) \cap N_U(t_1) = \{t_2, t_3\}$ (see Figure 13): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_5\}$.



(a) **force**(vt_1) in c-8(II) (b) **delete**(vt_1) in c-8(II)

Figure 13: Illustration of branching rule c-8(II), where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f5}})$, such that $N_U(v) \cap N_U(t_1) = \{t_2, t_3\}$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3, vt_4, t_1t_2, t_1t_3 and t_1t_5 will be deleted from G' by the reduction rules. So, both weights of vertex v and vertex t_1 decreases by $w_{5'}$, each. Each of vertices t_2 and t_3 can be either a type f5, or u5-vertex, and each of their weights would decrease by at least $m_8 = \min\{\Delta'_{5-3}, \Delta_{5-3}\}$. Each of vertices t_4 and t_5 can be either a type f5, or u5-vertex, and each of their weights would decrease by at least $m_7 = \min\{\Delta'_{5-4}, \Delta_{5-4}\}$. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(2w_{5'} + 2m_8 + 2m_7)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, both weights of vertex v and vertex t_1 decreases by Δ'_{5-4}, each. The total weight decrease in the branch of **delete**(vt_1) is at least $(2w_{5'} - 2w_{4'})$. As a result, we get the following branching vector:

$$(2w_{5'} + 2m_8 + 2m_7,\ 2w_{5'} - 2w_{4'})\,. \tag{29}$$

c-8(III). Without loss of generality, assume that $N_U(v) \cap N_U(t_1) = \{t_2, t_3, t_4\}$ (see Figure 14): We branch on edge vt_1.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3, vt_4, t_1t_2, t_1t_3 and t_1t_4 will be deleted from G' by the reduction rules.

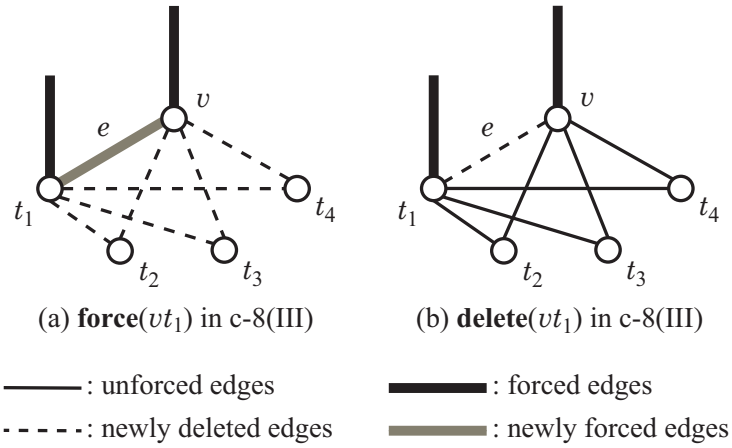


Figure 14: Illustration of branching rule c-8(III), where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{f5}})$, such that $N_U(v) \cap N_U(t_1) = \{t_2, t_3, t_4\}$.

So, both weights of vertex v and vertex t_1 decreases by $w_{5'}$, each. Each of vertices t_2, t_3, and t_4 can be either a type f5, or u5-vertex, and each of their weights would decrease by at least $m_8 = \min\{\Delta'_{5-3}, \Delta_{5-3}\}$. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(2w_{5'} + 3m_8)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. Thus, both weights of vertex v and vertex t_1 decreases by Δ'_{5-4}, each. The total weight decrease in the branch of **delete**(vt_1) is at least $(2w_{5'} - 2w_{4'})$. As a result, we get the following branching vector:

$$(2w_{5'} + 3m_8,\ 2w_{5'} - 2w_{4'})\,. \tag{30}$$

c-9. There exist vertices $v \in V_{\text{f5}}$ and $t_1 \in N_U(v; V_{\text{u5}})$ (see Figure 15): We branch on edge vt_1.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges vt_2, vt_3 and vt_4 will be deleted from G' by the reduction rules. So, the weight of vertex v decreases by $w_{5'}$, and the weight of vertex t_1 decreases by Δ_5. Each of vertices t_2, t_3 and t_4 can only be a type u5-vertex, and each of their weights decrease by Δ_{5-4}. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(4w_5 - 3w_4)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. Thus, the weight of vertex v decreases by Δ'_{5-4}, and the weight of vertex t_1 decreases by Δ_{5-4}. The total weight decrease in the branch of **delete**(vt_1) is at least $(w_5 + w_{5'} - w_4 - w_{4'})$. Then, we get the following branching vector:

$$(4 - 3w_4,\ 1 + w_{5'} - w_4 - w_{4'})\,. \tag{31}$$

4.4 Branching on Edges Around u5-vertices (c-10 to c-14)

If none of the first nine conditions can be executed, this means that the graph has no f5-vertices. But this does not mean that the maximum degree of the graph has been reduced

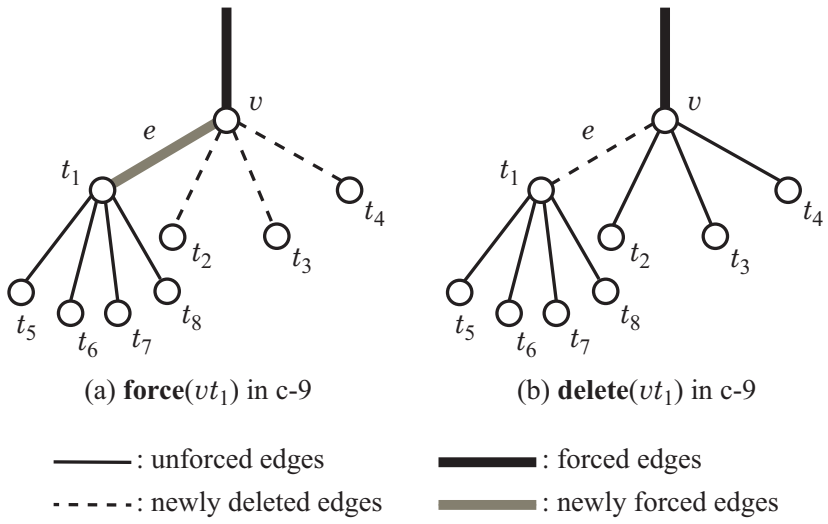


Figure 15: Illustration of branching rule c-9, where vertex $v \in V_{\text{f5}}$ and vertex $t_1 \in N_U(v; V_{\text{u5}})$.

to 4, since there might still be u5-vertices. This section derives branching vectors for branchings on an optimal edge $e = vt_1$ incident to a u5-vertex v, distinguishing the five cases for conditions c-10 to c-14.

c-10. There exist vertices $v \in V_{\text{u5}}$ and $t_1 \in N_U(v; V_{\text{f3}})$ (see Figure 16): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_6\}$.

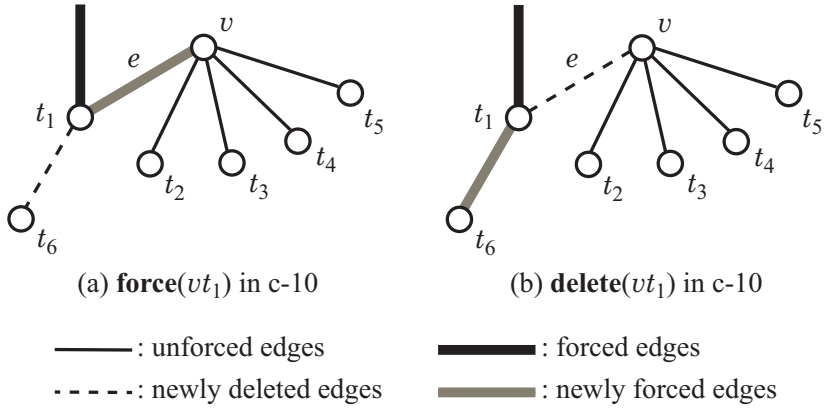


Figure 16: Illustration for c-10 where vertices $v \in V_{\text{u5}}$ and $t_1 \in N_U(v; V_{\text{f3}})$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edge t_1t_6 will be deleted from G' by the reduction rules. So, the weight of vertex v decreases by Δ_5, and the weight of vertex t_1 decreases by $w_{3'}$. If vertex t_6 is an f3-vertex (resp., u3, f4, u4, or a u5-vertex), then the weight decrease α of vertex t_6 would be $w_{3'}$ (resp., w_3, Δ'_{4-3}, Δ_{4-3}, and Δ_{5-4}). Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_5 - w_{5'} + w_{3'} + \alpha)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation, and edge t_1t_6 will be added to F' by the reduction rules. The weight of vertex v

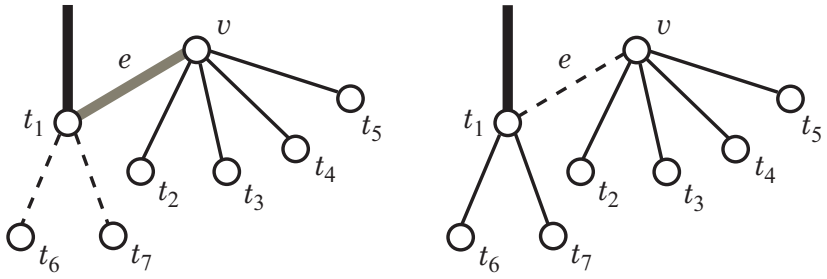
decreases by Δ_{5-4}, and the weight of vertex t_1 decreases by $w_{3'}$. If vertex t_6 is an f3-vertex (resp., u3, f4, u4, or a u5-vertex), then the weight decrease β of vertex t_6 would be $w_{3'}$ (resp., Δ_3, $w_{4'}$, Δ_4, and Δ_5). Thus, total weight decrease in the branch of **delete**(vt_1) is at least $(w_5 - w_4 + w_{3'} + \beta)$.

As a result, we get five branching vectors:

$$(1 - w_{5'} + w_{3'} + \alpha,\ 1 - w_4 + w_{3'} + \beta) \tag{32}$$

for $(\alpha, \beta) \in \{(w_{3'}, w_{3'}),\ (w_3, \Delta_3),\ (\Delta'_{4-3}, w_{4'}),\ (\Delta_{4-3}, \Delta_4),\ (\Delta_{5-4}, \Delta_5)\}$.

c-11. There exist vertices $v \in V_{\mathrm{u5}}$ and $t_1 \in N_U(v; V_{\mathrm{f4}})$ (see Figure 17): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_6, t_7\}$.



(a) **force**(vt_1) in c-11 (b) **delete**(vt_1) in c-11

: unforced edges : forced edges

: newly deleted edges : newly forced edges

Figure 17: Illustration for c-11 where vertices $v \in V_{\mathrm{u5}}$ and $t_1 \in N_U(v; V_{\mathrm{f4}})$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation, and edges t_1t_6 and t_1t_7 will be deleted from G' by the reduction rules. So, the weight of vertex v decreases by Δ_5, and the weight of vertex t_1 decreases by $w_{4'}$. Each of vertices t_6 and t_7 can be either a type f3, u3, f4, u4, or u5-vertex, and each of their weights would decrease by at least $m_9 = \min\{w_{3'}, w_3, \Delta'_{4-3}, \Delta_{4-3}, \Delta_{5-4}\}$. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_5 - w_{5'} + w_{4'} + 2m_9)$.

In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, the weight of vertex v decreases by Δ_{5-4}, and the weight of vertex t_1 decreases by Δ'_{4-3}. The total weight decrease in the branch of **delete**(vt_1) is at least $(w_5 - w_4 + w_{4'} - w_{3'})$. As a result, we get the following branching vector:

$$(1 - w_{5'} + w_{4'} + 2m_9,\ 1 - w_4 + w_{4'} - w_{3'})\,. \tag{33}$$

c-12. There exist vertices $v \in V_{\mathrm{u5}}$ and $t \in N_U(v; V_{\mathrm{u3}})$ (see Figure 18): We branch on edge vt_1. Note that $N_U(t_1) \setminus \{v\} = \{t_6, t_7\}$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation. So, the weight of vertex v decreases by Δ_5, and the weight of vertex t_1 decreases by Δ_3. The total weight decrease in the branch of **force**(vt_1) is at least $(w_5 - w_{5'} + w_3 - w_{3'})$. In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation, and edges t_1t_6 and t_1t_7 will be added to F' by the reduction rules. So, the weight of vertex v

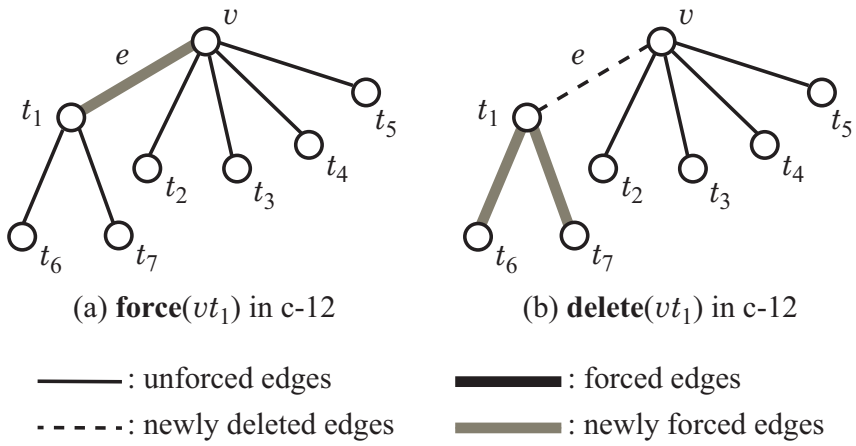


Figure 18: Illustration of branching rule c-12, where vertex $v \in V_{\text{u5}}$ and vertex $t \in N_U(v; V_{\text{u3}})$.

decreases by Δ_{5-4}, and the weight of vertex t_1 decreases by w_3. Each of vertices t_6 and t_7 can be either a type f3, u3, f4, u4, or u5-vertex, and each of their weights would decrease by at least $m_{10} = \min\{w_{3'}, \Delta_3, w_{4'}, \Delta_4, \Delta_5\}$. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_5 - w_4 + w_3 + 2m_{10})$. As a result, we get the following branching vector:

$$(1 - w_{5'} + w_3 - w_{3'},\ 1 - w_4 + w_3 + 2m_{10})\,. \tag{34}$$

c-13. There exist vertices $v \in V_{\text{u5}}$ and $t_1 \in N_U(v; V_{\text{u4}})$ (see Figure 19): We branch on edge vt_1.

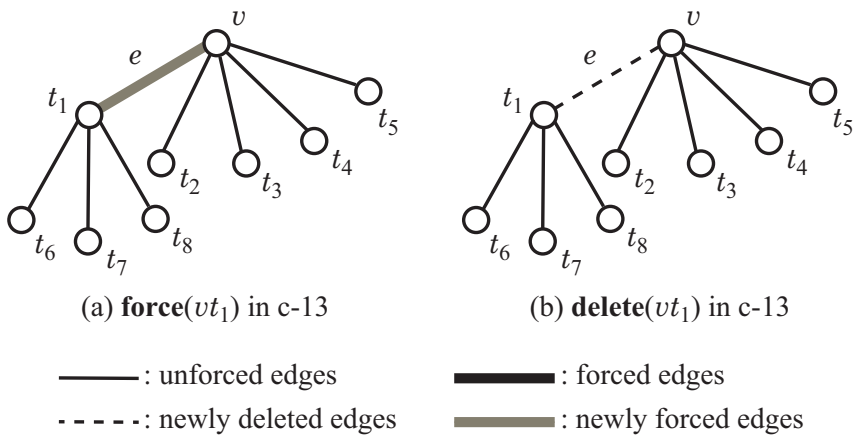


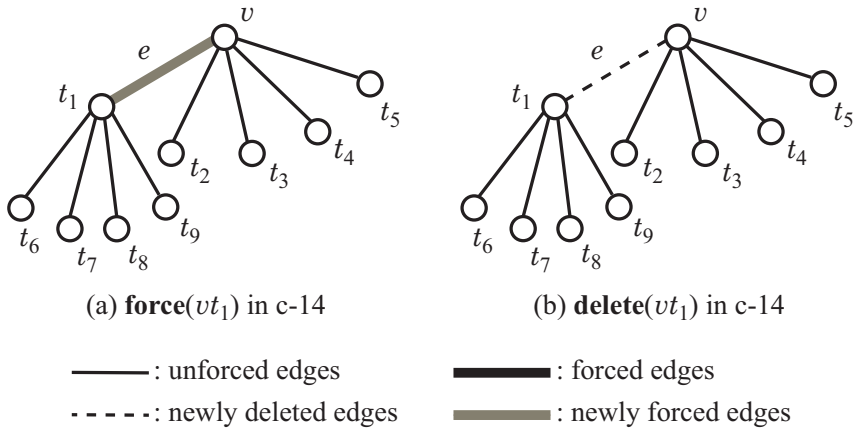
Figure 19: Illustration of branching rule c-13, where vertex $v \in V_{\text{u5}}$ and vertex $t_1 \in N_U(v; V_{\text{u4}})$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation. So, the weight of vertex v decreases by Δ_5, and the weight of vertex t_1 decreases by Δ_4. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(w_5 - w_{5'} + w_4 - w_{4'})$. In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, the weight of vertex v decreases by Δ_{5-4}, and the weight of vertex t_1 decreases by Δ_{4-3}.

Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(w_5 - w_3)$. Then, we get the following branching vector:

$$(1 - w_{5'} + w_4 - w_{4'},\ 1 - w_3)\,. \tag{35}$$

c-14. There exist vertices $v \in V_{\mathrm{u}5}$ and $t_1 \in N_U(v; V_{\mathrm{u}5})$ (see Figure 20): We branch on edge vt_1.



(a) **force**(vt_1) in c-14 (b) **delete**(vt_1) in c-14

Figure 20: Illustration of branching rule c-14, where vertex $v \in V_{\mathrm{u}5}$ and vertex $t_1 \in N_U(v; V_{\mathrm{u}5})$.

In the branch of **force**(vt_1), edge vt_1 will be added to F' by the branching operation. So, both weights of vertex v and vertex t_1 decreases by Δ_5, each. Thus, the total weight decrease in the branch of **force**(vt_1) is at least $(2w_5 - 2w_{5'})$. In the branch of **delete**(vt_1), edge vt_1 will be deleted from G' by the branching operation. So, both weights of vertex v and vertex t_1 decreases by Δ_{5-4}, each. Thus, the total weight decrease in the branch of **delete**(vt_1) is at least $(2w_5 - 2w_4)$. Then, we get the following branching vector:

$$(2 - 2w_{5'},\ 2 - 2w_4)\,. \tag{36}$$

4.5 Switching to TSP4

If none of these 14 cases can be executed, this means that the graph has no more degree-5 vertices. In that case, we can switch and use a fast algorithm for TSP in degree-4 graphs (tsp4(G, F)) to solve the remaining instances. Xiao and Nagamochi [15, Lemma 3] have shown how to leverage results obtained by a measure-and-conquer analysis, and that an algorithm can be used as a sub-procedure, given that we know the respective weight setting mechanism. To get a combination of total running time bound of these two algorithms, we can use the maximum branching factor for TSP in degree-4 graphs algorithm and a measure μ is calculated based on the maximum ratio of vertex weights for TSP in degree-4 graphs and TSP in degree-5 graphs [12].

Here we use the $O^*(1.69193^n)$-time algorithm by Xiao and Nagamochi [13], where the weights of vertices in degree-4 graphs are set as follows; $w_{3'} = 0.21968$, $w_3 = 0.45540$, $w_{4'} = 0.59804$, and $w_4 = 1$. For this step, the running time bound is

$$T(\mu) \leq O\left(1.69193^{\max\left\{\frac{0.21968}{w_{3'}}, \frac{0.45540}{w_3}, \frac{0.59804}{w_{4'}}, \frac{1}{w_4}\right\}}\right). \tag{37}$$

4.6 Overall Analysis

The branching factor of each of the branching vectors from (20) to (37) does not exceed 2.453051. The tight constraints in the quasiconvex program are in conditions c-4, c-5(I), c-5(II), c-10, c-12 and c-13. This completes a proof of Theorem 1.

5 Conclusion

In this paper, we have presented an exact algorithm for TSP in degree-5 graphs. Our algorithm is a simple branching algorithm, following the branch-and-reduce paradigm, and it operates in space which is polynomial of the size of an input instance. To the best of our knowledge, this is the first polynomial space exact algorithm developed specifically for graphs of maximum degree at most 5, and extends previous algorithms for degree 3 [11, 14], and degree-4 graphs [13].

We used the measure and conquer method for the analysis of the running time of the proposed algorithm, and have obtained an upper bound of $O^*(2.4531^n)$, where n is the number of vertices in a given instance. This result compares favorably with the polynomial-space TSP algorithm for general graphs by Gurevich and Shelah [8], which runs in $O^*(4^n n^{\log n})$-time.

It remains an open question whether this time bound can be further improved by a modified analysis technique, or by a careful re-examination of the branching rules. Indeed, it would be most interesting to obtain a polynomial-space algorithm with a running time of $O^*(2^n)$ or less, or simply show that this cannot be achieved.

References

[1] Bellman, R. : Combinatorial Processes and Dynamic Programming. In: Proceeding of the 10th Symposium in Applied Mathematics, Amer. Math. Soc., Providence, RI, 1960.

[2] Bodlaender, H. L., Cygan, M., Kratsch, S. and Nederlof, J. : Solving Weighted and Counting Variants of Connectivity Problems Parameterized by Treewidth Deterministically in Single Exponential Time. In: CoRR abs/1211.1505, 2012.

[3] Eppstein, D.: Quasiconvex Analysis of Multivariate Recurrence Equations for Backtracking Algorithms. In: ACM Transactions on Algorithms, Vol. 2, No. 4, pp. 492-509, 2006.

[4] Eppstein, D. : The Traveling Salesman Problem for Cubic Graphs. In: Journal of Graph Algorithms and Application, Vol. 11, No. 1, pp. 61-81, 2007.

[5] Fomin, F. V., Grandoni, F. and Kratsch, D. : A Measure and Conquer Approach for the Analysis of Exact Algorithms. In: J. ACM, Vol. 56, No. 5, Article 25, 2009.

[6] Fomin, F. V. and Kratsch, D. : Exact Exponential Algorithms. In: Berlin Heidelberg: Springer, 2010.

[7] Gebauer, H. : Finding and Enumerating Hamilton Cycles in 4-regular Graphs. In: Theoretical Computer Science, Vol. 412, No. 35, pp. 4579-4591, 2011.

[8] Gurevich, Y. and Shelah, S. : Expected Computation Time for Hamiltonian Path Problem. In: Siam Journal of Computation, Vol. 16, No. 3, pp. 486-502, 1987.

[9] Held, M. and Karp, R. M. : A Dynamic Programming Approach to Sequencing Problems. In: Journal of the Society for Industrial and Applied Mathematics, Vol. 10, No. 1, pp. 196-210, 1962.

[10] Iwama, K. and Nakashima, T.: An Improved Exact Algorithm for Cubic Graph TSP. In: COCOON, LNCS 4598, pp. 108-117, 2007.

[11] Liskiewicz, M. and Schuster, M. R.: A New Upper Bound for the Traveling Salesman Problem in Cubic Graphs. In: CoRR abs/1207.4694v2, 2012.

[12] Xiao, M. and Nagamochi, H.: Further Improvement on Maximum Independent Set in Degree-4 Graphs. In: COCOA 2011, LNCS 6831, pp. 163-178, 2011.

[13] Xiao, M. and Nagamochi, H.: An Improved Exact Algorithm for TSP in Graphs of Maximum Degree-4. In: Theory Comput Syst, DOI 10.1007/s00224-015-9612-x, 2015.

[14] Xiao, M. and Nagamochi, H.: An Exact Algorithm for TSP in Degree-3 Graphs via Circuit Procedure and Amortization on Connectivity Structure. In: TAMC 2013, LNCS 7876, pp. 96-107, 2013.

[15] Xiao, M. and Nagamochi, H.: Exact Algorithms for Maximum Independent Set. In: ISAAC 2013. LNCS 8283, pp. 328-338, 2013.