An Exact Algorithm for TSP in Degree-3 Graphs via Circuit Procedure and Amortization on Connectivity Structure *

Mingyu Xiao[1] and Hiroshi Nagamochi[2]

[1] School of Computer Science and Engineering, University of Electronic Science and Technology of China, China, myxiao@gmail.com
[2] Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Japan, nag@amp.i.kyoto-u.ac.jp

Abstract. The paper presents an $O^*(1.2312^n)$-time and polynomial-space algorithm for the traveling salesman problem in an n-vertex graph with maximum degree 3. This improves the previous time bounds of $O^*(1.251^n)$ by Iwama and Nakashima and $O^*(1.260^n)$ by Eppstein. Our algorithm is a simple branch-and-search algorithm. The only branch rule is designed on a cut-circuit structure of a graph induced by unprocessed edges. To improve a time bound by a simple analysis on measure and conquer, we introduce an amortization scheme over the cut-circuit structure by defining the measure of an instance to be the sum of not only weights of vertices but also weights of connected components of the induced graph.

Key words. Traveling Salesman Problem, Exact Exponential Algorithm, Graph Algorithm, Connectivity, Measure and Conquer

1 Introduction

The traveling salesman problem (TSP) is one of the most famous and intensively studied problems in computational mathematics. Many algorithmic methods have been investigated to beat this challenge of finding the shortest route visiting each member of a collection of n locations and returning to the starting point. The first $O^*(2^n)$-time dynamic programming algorithm for TSP is back to early 1960s. However, in the last half of a century no one can break the barrier of 2 in the base of the running time. To make steps toward the long-standing and major open problem in exact exponential algorithms, TSP in special classes of graphs, especially degree bounded graphs, have also been intensively studied. Eppstein [5] showed that TSP in degree-3 graphs (a graph with maximum degree i is called a degree-i graph) can be solved in $O^*(1.260^n)$ time and polynomial space, and TSP in degree-4 graphs can be solved in $O^*(1.890^n)$

* Technical report 2012-009, December 4, 2012; Supported in part by Grant 60903007 of NSFC, China.

time and polynomial space. Iwama and Nakashima [9] refined Eppstein's algorithm for degree-3 graphs and improved the result to $O^*(1.251^n)$ by showing that the worst case in Eppstein's algorithm will not always happen. Gebauer [8] designed an $O^*(1.733^n)$-time exponential-space algorithm for TSP in degree-4 graphs, which is improved to $O^*(1.716^n)$ time and polynomial space by Xiao and Nagamochi [13]. Bjorklund *et al.* [2] also showed TSP in degree bounded graph can be solved in $O^*((2-\varepsilon)^n)$ time, where $\varepsilon > 0$ depends on the degree bound only. There is a Monte Carlo algorithm to decide a graph is Hamiltonian or not in $O^*(1.657^n)$ time [1]. For planar TSP and Euclidean TSP, there are sub-exponential algorithms based on small separators [3].

In this paper, we present an improved deterministic algorithm for TSP in degree-3 graphs, which runs in $O^*(2^{\frac{3}{10}n}) = O^*(1.2312^n)$ time and polynomial space. The algorithm is simple and contains only one branch rule that is designed on a cut-circuit structure of a graph induced by unprocessed edges. We will apply the measure and conquer method to analyze the running time. Note that our algorithm for TSP in degree-4 graphs in [13] is obtained by successfully applying the measure and conquer method to TSP for the first time. However, direct application of measure and conquer to TSP in degree-3 graphs may only lead to an $O^*(1.260^n)$-time algorithm. To effectively analyze our algorithm, we use an amortization scheme over the cut-circuit structures by setting weights to both vertices and connected components of the induced graph.

2 Preliminaries

In this paper, a graph means an undirected edge-weighted graph with maximum degree 3, which possibly has multiple edges, but no self-loops. Let $G = (V, E)$ be a graph with an edge weight. For a subset $V' \subseteq V$ of vertices and a subset $E' \subseteq E$ of edges, the subgraphs induced by V' and E' are denoted by $G[V']$ and $G[E']$ respectively. We also use $\text{cost}(E')$ to denote the total weight of edges in E'. For any graph G', the sets of vertices and edges in G' are denoted as $V(G')$ and $E(G')$ respectively. A graph consisting of a single vertex is called *trivial.* A *cycle* of length l (also denoted as l*-cycle*) is a graph with l vertices v_i and l edges v_iv_{i+1} ($i \in \{1, 2 \ldots, l\}$ and $v_{l+1} = v_1$). An edge v_iv_j ($|i-j| \geq 2$) between two vertices in the cycle but different from the l edges in it is called a *chord* of the cycle. Two vertices in a graph are k*-edge-connected* if there are k-edge-disjoint paths between them. A graph is k*-edge-connected* if every pair of vertices in it are k-edge-connected. We treat a trivial graph as a k-edge-connected graph for any $k \geq 1$. A Hamiltonian cycle is a cycle through every vertex. Given a graph with an edge weight, the *traveling salesman problem* (TSP) is to find a Hamiltonian cycle of minimum total weight in the edges.

2.1 Forced TSP

In some branch-and-search algorithms for TSP, we may branch on an edge in the graph by including it to the solution or excluding it from the solution. In this way, we need to maintain a set of edges that must be used in the solution. We introduce the *forced traveling salesman problem* as follows. An instance is a pair (G, F) of an edge-weighted undirected graph $G = (V, E)$ and a subset $F \subseteq E$ of edges, called *forced edges*. A Hamiltonian cycle of G is called a *tour* if it passes though all the forced edges in F. The objective of the problem is to compute a tour of minimum weight in the given instance (G, F). An instance is called *infeasible* if no tour exists. A vertex is called *forced* if there is a forced edge incident on it. For convenience, we say that the *sign* of an edge e is 1 if e is a forced edge and 0 if e is an unforced edge. We use $\mathrm{sign}(e)$ to denote the sign of e.

2.2 *U*-graphs and *U*-components

We consider an instance (G, F). Let $U = E(G) - F$ denote the set of unforced edges. A subgraph H of G is called a *U-graph* if H is a trivial graph or H is induced by a subset $U' \subseteq U$ of unforced edges (i.e., $H = G[U']$). A maximal connected U-graph is called a *U-component*. Note that each connected component in the graph $(V(G), U)$ is a U-component.

For a vertex subset X (or a subgraph X) of G, let $\mathrm{cut}(X)$ denote the set of edges in $E = F \cup U$ between X and $V(G) - X$, and denote $\mathrm{cut}_F(X) = \mathrm{cut}(X) \cap F$ and $\mathrm{cut}_U(X) = \mathrm{cut}(X) \cap U$. Edge set $\mathrm{cut}(X)$ is also called a *cut* of the graph. We say that an edge is *incident* on X if the edge is in $\mathrm{cut}(X)$. The *degree* $d(v)$ of a vertex v is defined to be $|\mathrm{cut}(\{v\})|$. We also denote $d_F(v) = |\mathrm{cut}_F(\{v\})|$ and $d_U(v) = |\mathrm{cut}_U(\{v\})|$. A U-graph H is *k-pendent* if $|\mathrm{cut}_U(H)| = k$. A U-graph H is called *even* (resp., *odd*) if $|\mathrm{cut}_F(H)|$ is even (resp., odd). A U-component is 0-pendent.

In this paper, we will always keep every U-component 2-edge-connected. For simplicity, we may regard a maximal path of forced edges between two vertices u and v as a single forced edge uv in an instance (G, F), since we can assume that $d_F(v) = 2$ always implies $d(v) = 2$ for any vertex v.

2.3 Circuits and blocks

We consider a nontrivial U-component H in an instance (G, F). A *circuit* $\mathcal{C}$ in H is a maximal sequence $e_1, e_2, \ldots, e_p$ of edges $e_i = u_i v_i \in E(H)$ $(1 \le i \le p)$ such that for each $e_i \in \mathcal{C}$ $(i \ne p)$, the next edge $e_{i+1} \in \mathcal{C}$ is given by a subgraph B_i of H such that $\mathrm{cut}_U(B_i) = \{e_i, e_{i+1}\}$. See Fig. 1 for an illustration. We say that each subgraph B_i is a *block* along $\mathcal{C}$ and vertices v_i and u_{i+1} are the *endpoints* of block B_i. By the maximality of $\mathcal{C}$, we know that any two vertices in each block B_i are 2-edge-connected in the induced subgraph $G[B_i]$. It is possible that a circuit in a 2-edge-connected graph H may contain only one edge $e = u_1 v_1$. For this case, vertices u_1 and v_1 are connected by three edge-disjoint paths in H and the circuit is called *trivial*, where the unique block is the U-component H. Each nontrivial circuit contains at least two blocks, each of which is a 2-pendent subgraph of H. In our algorithm, we will consider only nontrivial circuits $\mathcal{C}$. When H is 2-edge-connected, there are $p \ge 2$ different blocks along a nontrivial circuit $\mathcal{C}$, where u_1 and v_p are in the same block B_p and $\mathrm{cut}_U(B_p) = \{e_p, e_1\}$. A block B_i is called *trivial* if $|V(B_i)| = 1$ and $d_F(v) = 1$ for the only vertex v in it (v is of degree 3 in G). A block B_i is called *reducible* if $|V(B_i)| = 1$ and $d_F(v) = 0$ for the only vertex v in it (v is of degree 2 in G). A block B_i with $V(B_i) = \{v_i = u_{i+1}\}$ is either trivial or reducible in a 2-edge-connected graph.

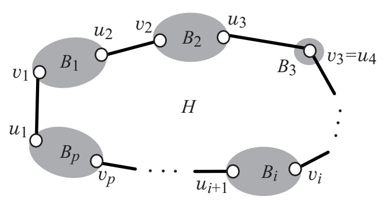

Fig. 1. A circuit in a 2-edge-connected graph H

We state more properties on circuits and blocks.

Lemma 1. *In a degree-3 graph, let H be a 2-edge-connected U-component and $\mathcal{C}$ be any circuit in it. For each block B_i of $\mathcal{C}$, B_i is not trivial or reducible if and only if the two endpoints v_i and u_{i+1} of it are two different vertices of degree 3 in H.*

Lemma 2. *Each edge in a 2-edge-connected U-component H of a degree-3 graph is contained in exactly one circuit. A partition of $E(H)$ into circuits can be obtained in linear time.*

Proof. It is known that the set of all minimum cuts (a set of k edges in a k-edge-connected multigraph is called a minimum cut if the graph becomes disconnected by removing the k edges) can be represented by a cactus structure (cf. [11]). In particular, when the size of a minimum cut is two, the cactus structure of minimum cuts can be obtained in linear time by contracting each 3-edge-connected component (a maximal set of vertices every two of which are 3-connected in the given graph) into a single vertex, and for each cycle C in the resulting graph, a pair of any two edges in C corresponds to a minimum cut in the original graph [10]. In a 2-edge-connected U-component H, (i) an edge $e \in E(H)$ forms a circuit $\mathcal{C}$ having only one block if and only if e is not in any minimum cut of H; and (ii) A circuit $\mathcal{C}$ with at least two edges in H corresponds to a cycle C in the cactus structure. Based on the cactus structure, we can obtain a partition of edge sets into circuits in linear time. ∎

2.4 Branch-and-search algorithms

Our algorithm is a branch-and-search algorithm: we search the solution by iteratively branching on the current instance into several smaller instances until the current instance becomes trivial (or polynomially solvable). In this paradigm, we will get a search tree. In each leaf of the search tree, we can solve the problem directly. The size of the search tree is the exponential part of the running time of the search algorithm. Let μ be a measure of the instance (for graph problems, the measure can be the number of vertices or edges in the graph and so on). Let $C(\mu)$ denote the maximum number of leaves in the search tree generated by the algorithm for any instance with measure μ. We shall determine an upper bound on $C(\mu)$ by evaluating all the branches. When we branch on an instance (G, F) with k branches such that the i-th branch decreases the measure μ of (G, F) by at least a_i, we obtain the following recurrence

$$C(\mu) \le C(\mu - a_1) + C(\mu - a_2) + \cdots + C(\mu - a_k).$$

Solving this recurrence, we get $C(\mu) = [\alpha(a_1, a_2, \ldots, a_k)]^\mu$, where $\alpha(a_1, a_2, \ldots, a_k)$ is the largest root of the function $f(x) = 1 - \sum_{i=1}^{k} x^{-a_i}$. In this paper, we represent the above recurrence by a vector $(a_1; a_2; \cdots; a_k)$ of measure decreases, called a *branch vector* (cf. [7]). In particular, when $a_i = a_{i+1} = \cdots = a_j$ for some $i \le j$, it may be written as $(a_1; a_2; \cdots a_{i-1}; [a_i]_{j-i+1}; a_{j+1}; \cdots; a_k)$, and a vector $([a]_k)$ is simply written as $[a]_k$. When we compare two branch vectors $\mathbf{b} = (a_1; a_2)$ $(a_1 \le a_2)$ and $\mathbf{b}' = (a'_1; a'_2)$ such that "$a_i \le a'_i$ $(i = 1, 2)$" or "$a'_1 = a_1 + \varepsilon$ and $a'_2 = a_2 - \varepsilon$ for some $0 \le \varepsilon \le (a_2 - a_1)/2$," we only consider branch vector $\mathbf{b}$ in analysis, since a solution α from $\mathbf{b}$ is not smaller than that from $\mathbf{b}'$ (cf. [7]). We say that $\mathbf{b}$ *covers* $\mathbf{b}'$ in this case.

3 Reductions based on small cuts

For some special cases, we can reduce the instance directly without branching. Most of out reduction rules are based on the structures of small cuts in the graph. In fact, we will deal with cuts of size $1, 2, 3$ and 4.

3.1 Sufficient conditions for infeasibility

The *parity condition* on an instance is: (i) every U-component is even; and (ii) the number of odd blocks along every circuit is even.

Lemma 3. *An instance (G, F) is infeasible if G is not 2-edge-connected or it violates the parity condition.*

Proof. Since any tour is a 2-edge-connected spanning graph of G, it cannot exist when G is not 2-edge-connected. Since any tour is an Eulerian graph, it cannot exist in any instance with an odd U-component. For a circuit which has an odd number of odd blocks, we see that at least one odd block will be an odd U-component in any way of including/deleting edges in the circuit. ∎

3.2 Eliminable, reducible and parallel edges

The unique unforced edge incident on a 1-pendent U-graph is *eliminable*. From parity condition (i), we can decide whether each eliminable edge need to be included to F or deleted from the graph just by depending on the parity of $|\text{cut}_F(H)|$.

For any subgraph H of G with $|\text{cut}(H)| = 2$, we call the unforced edges in $\text{cut}(H)$ *reducible*. From the connectivity condition and parity condition (i), we see that all reducible edges need to be included to F. In particular, any edge uv incident to a vertex v with $d(v) = 2$ (or with a neighbor v' with multiple edges of $vv' \in F$ and $vv' \in U$) is reducible, since $uv \in \text{cut}(X)$ and $|\text{cut}(X)| = 2$ for $X = \{v\}$ (or $X = \{v, v'\}$).

If there are multiple edges with the same endpoints u and v, we can reduce the instance in the following way preserving the optimality: If the graph has only two vertices u and v, solve the problem directly; else if there are forced edges between u and v, the problem is infeasible; and otherwise remove all unforced edges between u and v except one with the smallest weight.

3.3 Reductions based on 3-cuts and 4-cuts

Lemma 4. *Let (G, F) be an instance where G is a graph with maximum degree 3. For any subgraph X with $|\text{cut}(X)| = 3$, we can replace X with a single vertex x and update the three edges incident on x preserving the optimality of the instance.*

Proof. Denote $\text{cut}(X)$ by $\{y_1x_1, y_2x_2, y_3x_3\}$ with $x_i \in V(X)$ and $y_i \in V - V(X)$. We will replace X and $\text{cut}(X)$ with a single vertex x and three new edges xy_1, xy_2 and xy_3. Let G' denote the new graph. We only need to decide the weights and signs of edges xy_1, xy_2 and xy_3 in G' to satisfy the lemma. Let I_i $(i = 1, 2, 3)$ denote the problem of finding a path P from x_{i_1} to x_{i_2} $(\{i, i_1, i_2\} = \{1, 2, 3\})$ of minimum total cost in X that passes through all vertices and forced edges in X. We say that I_i *infeasible* if it has no such path. We consider the three problems I_i $(i = 1, 2, 3)$. There are four possible cases. Case 1. None of the three problems is feasible: We can see that the original instance (G, F) is also infeasible. In G', we let $\text{sign}(xy_1) = \text{sign}(xy_2) = \text{sign}(xy_3) = 1$. Since the trivial U-component $\{x\}$ is odd, the new instance is infeasible by Lemma 3. Case 2. Only one of the three problems, say I_{j_1} $(\{j_1, j_2, j_3\} = \{1, 2, 3\})$, is feasible: Let S_{j_1} be an optimal solution to I_{j_1}. Then there is a solution S to (G, F) such that $S \cap E(X) = E(S_{j_1})$ (if (G, F) is feasible). Therefore, in G', we let $\text{sign}(xy_{j_2}) = \text{sign}(xy_{j_3}) = 1$, $\text{sign}(xy_{j_1}) = \text{sign}(x_{j_1}y_{j_1})$, $\text{cost}(xy_{j_2}) = \text{cost}(x_{j_2}y_{j_2}) + \text{cost}(S_{j_1})$, $\text{cost}(xy_{j_3}) = \text{cost}(x_{j_3}y_{j_3})$ and $\text{cost}(xy_{j_1}) = \text{cost}(x_{j_1}y_{j_1})$. Case 3. Exactly two of the three problems, say I_{j_1} and I_{j_2} $(\{j_1, j_2, j_3\} = \{1, 2, 3\})$, are feasible: Let S_{j_1} and S_{j_2} be an optimal solution to I_{j_1} and I_{j_2} respectively. Then there is a solution S to (G, F) such that either $S \cap E(X) = E(S_{j_1})$ or $S \cap E(X) = E(S_{j_2})$. Therefore, in G', we let $\text{sign}(xy_{j_3}) = 1$, $\text{sign}(xy_{j_1}) = \text{sign}(x_{j_1}y_{j_1})$, $\text{sign}(xy_{j_2}) = \text{sign}(x_{j_2}y_{j_2})$, $\text{cost}(xy_{j_3}) = \text{cost}(x_{j_3}y_{j_3})$, $\text{cost}(xy_{j_1}) = \text{cost}(x_{j_1}y_{j_1}) + \text{cost}(S_{j_2})$ and $\text{cost}(xy_{j_2}) = \text{cost}(x_{j_2}y_{j_2}) + \text{cost}(S_{j_1})$. Case 4. All of the three problems are feasible: Let S_1,

S_2 and S_3 be an optimal solution to I_1, I_2 and I_3 respectively. In G', we let $\mathrm{sign}(xy_i) = \mathrm{sign}(x_iy_i)$ and $\mathrm{cost}(xy_i) = \mathrm{cost}(x_iy_i) + \frac{1}{2}\sum_{j=1}^{3}\mathrm{cost}(S_j) - \mathrm{cost}(S_i)$ $(i = 1, 2, 3)$. Straightforward computation can verify that with these setting G' will preserve the optimality. ∎

Similar to Lemma 4, we can simplify a subgraph X with $|\mathrm{cut}(X)| = 4$. However, there are many cases needed to consider. In fact, in our algorithms, we only need to consider a special case.

We consider a subgraph X with $|\mathrm{cut}_F(X)| = 4$ and $|\mathrm{cut}_U(X)| = 0$. We want to reduce X. Denote $\mathrm{cut}(X)$ by $\{y_1x_1, y_2x_2, y_3x_3, y_4x_4\}$ with $x_i \in V(X)$ and $y_i \in V - V(X)$, where $x_i \neq x_j$ $(1 \le i < j \le 4)$. We define I_i $(i = 1, 2, 3)$ to be instances of the problem of finding two disjoint paths P and P' of minimum total cost in X such that all vertices and forced edges in X appear in exactly one of the two paths, and one of the two paths is from x_i to x_4 and the other one is from x_{j_1} to x_{j_2} $(\{j_1, j_2\} = \{1, 2, 3\} - \{i\})$. We say that I_i *infeasible* if it has no solution.

A subgraph X is 4-*cut reducible* if $|\mathrm{cut}_F(X)| = 4$, $|\mathrm{cut}_U(X)| = 0$, and at least one of the three problems I_1, I_2 and I_3 defined above is infeasible. We have the following lemma to reduce the 4-cut reducible subgraph.

Lemma 5. *Let (G, F) be an instance where G is a graph with maximum degree 3. A 4-cut reducible subgraph X can be replaced with one of the following subgraphs X' with four vertices and $|\mathrm{cut}_F(X')| = 4$ so that the optimality of the instance is preserved:*
(i) *four single vertices (i.e., there is no solution to this instance);*
(ii) *a pair of forced edges; and*
(iii) *a 4-cycle with four unforced edges.*

Proof. We consider the three problems I_i $(i = 1, 2, 3)$. Since at least one of them is infeasible, there are three possible cases. Case 1. None of the three problems is feasible: We can see that the original instance (G, F) is also infeasible. Then we can replace X with a graph containing only four vertices $\{x_1, x_2, x_3, x_4\}$ and no edge. Now x_1 becomes a degree-1 vertex in the new graph and the new instance is infeasible. Case 2. Only one of the three problems, say I_{i_0} $(i_0 \in \{1, 2, 3\})$, is feasible: Let S_{i_0} be an optimal solution to I_{i_0}. Then there is a solution S to (G, F) such that $S \cap E(X) = E(S_{i_0})$. Therefore, we can replace X with a graph of four vertices $\{x_1, x_2, x_3, x_4\}$ and two edges x_ix_4 and $x_{j_1}x_{j_2}$ in G preserving the optimality of the instance, where the costs of $x_{i_0}x_4$ and $x_{j_1}x_{j_2}$ are the costs of the two paths in S_{i_0}. Note that in the new instance after the replacement, the four vertices $\{x_1, x_2, x_3, x_4\}$ become degree-2 vertices and the two new edges x_ix_4 and $x_{j_1}x_{j_2}$ should be included into F. Case 3. Two of the three problems, say I_{i_1} and I_{i_2} $(i_1, i_2 \in \{1, 2, 3\})$, are feasible: Let S_{i_1} and S_{i_2} be optimal solutions to I_{i_1} and I_{i_2} respectively. Then there is a solution S to (G, F) such that $S \cap E(X) = E(S_{i_1})$ or $S \cap E(X) = E(S_{i_2})$. Therefore, we can replace X with a 4-cycle $x_{i_1}x_4x_{i_2}x_j$ preserving the optimality of the instance, where $\{j\} = \{1, 2, 3\} - \{i_1, i_2\}$, the costs of $x_{i_1}x_4$ and $x_{i_2}x_j$ are the costs of the two paths in S_{i_1}, and the costs of $x_4x_{i_2}$ and $x_jx_{i_1}$ are the costs of the two paths in S_{i_2}. ∎

Lemma 6. *Let X be an induced subgraph of a degree-3 graph G such that X contains at most eight vertices of degree 3 in G. Then X is 4-cut reducible if $|\text{cut}_F(X)| = 4$, $|\text{cut}_U(X)| = 0$, and X contains at most two unforced vertices.*

Proof. We only need to show that at least one of the three problem instances I_1, I_2 and I_3 (defined before Lemma 5) is infeasible. Assume that no two edges in $\text{cut}_F(X)$ meet at a same vertex in X, since otherwise only one of I_1, I_2 and I_3 is feasible. Also assume that X has no multiple edges or induced triangles, since otherwise X can be reduced to a smaller graph preserving its optimality. Note that X contains an even number k of degree 3 vertices in G. Since $k = 4$ (i.e., $|V(X)| = 4$) implies the lemma, we consider the case of $k = 6, 8$. When $k = 6$, X is either a 6-cycle with a chord or a graph obtained from a 5-cycle with a chord by subdividing the chord with a new vertex. In any case, we see that one of I_1, I_2 and I_3 is infeasible. Let $k = 8$. In this case, there are four vertices u_i $(i = 1, 2, 3, 4)$ which are not incident to any of the four edges in $\text{cut}(B) = \text{cut}_F(B)$, and two of them, say u_1 and u_2 are joined by a forced edge $u_1u_2 \in F$ by the assumption on the number of unforced vertices in X. Then we see that there are three possible configurations for such an induced graph X with no induced triangles, and a straightforward inspection shows that none of them admits a set of three feasible instances I_1, I_2 and I_3. ∎

Lemma 4 and Lemma 5 imply a way of simplifying some local structures of an instance. However, it is not easy to find solutions to problems I_i in the above two lemmas. In our algorithm, we only do this replacement for X containing no more than 10 vertices and then the corresponding problems I_i can be solved in constant time by a brute force search.

We define the operation of 3/4-*cut reduction*: If there a subgraph X of G with $|V(X)| \leq 10$ such that $|\text{cut}(X)| = 3$ or X is 4-cut reducible, then we simplify the graph by replacing X with a graph according to Lemma 4 or Lemma 5. Note that a 3/4-cut can be found in polynomial time if it exists and then this reduction operation can be implemented in polynomial time.

3.4 A solvable case and reduced graphs

A 3/4-cut reduction reduce the subgraph X to a trivial graph except for the last case of Lemma 5 where X will become a 4-cycle. Eppstein has identified a polynomially solvable case of forced TSP [5], which can deal with U-components of 4-cycles.

Lemma 7. [5] *If every U-component is a component of a 4-cycle, then a minimum cost tour of the instance can be found in polynomial time.*

Based on this lemma, we do not need to deal with U-components of 4-cycles in our algorithms.

All above reduction rules can be applied in polynomial time. An instance (G, F) is called a *reduced* instance if G is 2-edge-connected, (G, F) satisfies the parity condition, and has none of reducible edges, eliminable edges and multiple edges, and the 3/4-cut reduction cannot be applied on it anymore. Note that a reduced instance has no triangle, otherwise 3-cut reduction would be applicable. An instance is called *2-edge-connected* if every U-component in it is 2-edge-connected. The initial instance $(G, F = \emptyset)$ is assumed to be 2-edge-connected, otherwise it is infeasible by Lemma 3. In our algorithm, we will guarantee that the input instance is always 2-edge-connected, and we branch on a reduced graph to search a solution.

4 The circuit procedure

The *circuit procedure* is one of the most important operations in our algorithm. The procedure will determinate each edge in a circuit to be included into F or to be deleted from the graph. It will be widely used as the only branching operation in our algorithm.

Processing circuits: Determining an unforced edge means either including it to F or deleting it from the graph. When an edge is determined, the other edges in the same circuit containing this edge can also be determined directly by reducing eliminable edges. We call the series of procedures applied to all edges in a circuit together as a *circuit procedure*. Thus, in the circuit procedure, after we start to process a circuit $\mathcal{C}$ either by including an edge $e_1 \in \mathcal{C}$ to F or by deleting e_1 from the graph, the next edge e_{i+1} of e_i becomes an eliminable edge and we continue to determine e_{i+1} either by deleting it from the graph if block B_i is odd and $e_i = u_i v_i$ is included to F (or B_i is even and e_i is deleted); or by including it to F otherwise. Circuit procedure is a fundamental operation to build up our proposed algorithm. Note that a circuit procedure determines only the edges in the circuit. During the procedure, some unforced edges outside the circuit may become reducible and so on, but we do not determine them in this execution.

Lemma 8. *Let H be a 2-edge-connected U-component in an instance (G, F) and $\mathcal{C}$ be a circuit in H. Let (G', F') be the resulting instance after applying circuit procedure on $\mathcal{C}$. Then*
(i) *each block B_i of $\mathcal{C}$ becomes a 2-edge-connected U-component in (G', F'); and*
(ii) *any other U-component H' than H in (G, F) remains unchanged in (G', F').*

Proof. Since H is 2-edge-connected, we know that each block B_i induces a 2-edge-connected subgraph from H according to the definition of circuits. Hence B_i will be a 2-edge-connected U-component in (G', F'). Then we get (i). Since H and H' are vertex-disjoint and only edges in H are determined, we see that (ii) holds. ∎

We call a circuit *reducible* if it contains at least one reducible edge. We can apply the circuit procedure on a reducible circuit directly starting by including a reducible edge to F. In our algorithm, we will deal with reducible edges by processing a reducible circuit. When the instance becomes a reduced instance, we may not be able to reduce the instance directly. Then we search the solution by "branching on a circuit." *Branching on a circuit* $\mathcal{C}$ *at edge* $e \in \mathcal{C}$ means branching on the current instance to generate two instances by applying the circuit procedure to $\mathcal{C}$ after including e to F and deleting e from the graph respectively. Branching on a circuit is the only branching operation used in our algorithm.

5 A simple algorithm based on circuit procedures

We first introduce a simple algorithm for forced TSP to show the effectiveness of the circuit procedures. Improved algorithms are given in the next sections.

The simple algorithm contains only two steps: First reduce the instance until it becomes a reduced one; and then select a U-component H that is neither trivial nor a 4-cycle and branch on a circuit $\mathcal{C}$ in H such that at least one block along $\mathcal{C}$ is trivial. Note that there is always a circuit having a trivial block as long as the forced edge set F is not empty.

Here we use a traditional method to analyze the simple algorithm. It is natural to consider how many edges can be added to F in each operation of the algorithm. The size of F will not decrease by applying the reduction rules. Let $r = n - |F|$ and $C(r)$ denote the maximum number of leaves in the search tree generated by the algorithm for any instance with measure r. We only need to consider the branching operation in the second step.

For convenience, we call a maximal sequence $P = \{e_1, e_2, \ldots, e_p\}$ of edges $e_i = u_i u_{i+1} \in E(H)$ $(1 \leq i \leq p-1)$ a *chain* if all vertices u_j $(j = 2, 3, \ldots, p-1)$ are forced vertices. In the definition of the chain, we allow $u_1 = u_p$. Observe that each chain is contained in the same circuit. Since the selected circuit $\mathcal{C}$ has some trivial block, we know that $\mathcal{C}$ contains at least one chain P of size ≥ 2. We distinguish two cases according to the size of P being even or odd.

Case 1. $|P|$ is even: If all the blocks of $\mathcal{C}$ are trivial, then the U-component H containing $\mathcal{C}$ is a cycle of even length. Since H cannot be a 4-cycle, the length of cycle H is at least 6 (see Fig. 2(a) for an illustration). In each branch, after processing the circuit $\mathcal{C}$, we can include at least 3 edges to F. This gives us branch vector

$$(3; 3). \tag{1}$$

Next, we assume that $\mathcal{C}$ has a nontrivial block. We look at the worst case where P is of size 2 and $\mathcal{C}$ has only one nontrivial block (see Fig. 2(b) for an illustration). Now $\mathcal{C} = P$. Let $\mathcal{C} = \{u_1 v_1, u_2 v_2\}$, where $v_1 = u_2$. We branch on the circuit at edge $u_1 v_1$. In the branch of deleting $u_1 v_1$, we include $u_2 v_2$ into F . Furthermore, we can include the remaining two edges incident on u_1 to F by simply applying reduction rules. In the other branch of including $u_1 v_1$ to F, we will delete $u_2 v_2$

and include the other two edges incident on v_2 to F. We still can get branch vector (1). Note that when $\mathcal{C}$ is not of the worst case, it is not hard to verify that we may include more edges to F and we can get a branch vector covered by (1). We omit the details here, since the detailed proof can also be derived from the analysis of the improved algorithm in the next sections.

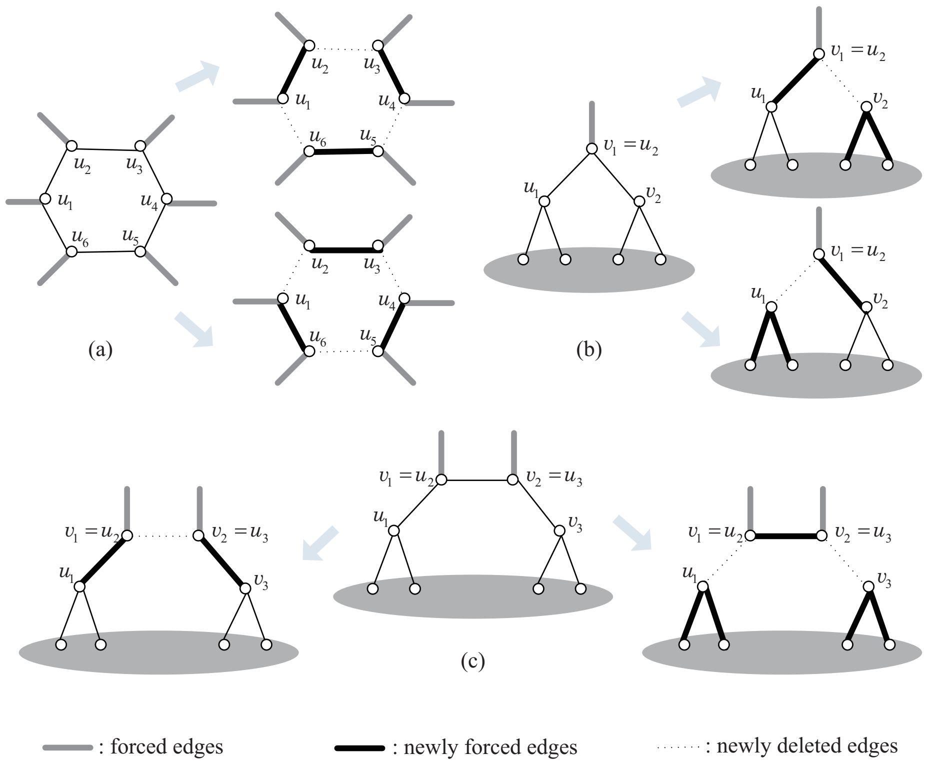

Fig. 2. Three bottleneck cases in branching on a ciruciut $\mathcal{C}$: (a) $\mathcal{C}$ is a 6-cycle; (b) $\mathcal{C}$ is a chain P of length 2; (c) $\mathcal{C}$ is a chain P of length 3.

Case 2. $|P|$ is odd: Now circuit $\mathcal{C}$ must contain at least one nontrivial block, otherwise the instance violates the parity condition. We also look at the worst case where P is of size 3 (since $|P| \geq 2$) and $\mathcal{C}$ has only one nontrivial block (see Fig. 2(c) for an illustration). Let $\mathcal{C} = \{u_1v_1, u_2v_2, u_3v_3\}$, where $v_1 = u_2$ and $v_2 = u_3$. We branch on the circuit at edge u_1v_1. In the branch of deleting u_1v_1, we will also delete u_3v_3 and include the following five edges to F: v_2u_2, the remaining two edges incident on u_1 and the remaining two edges incident on v_3 (note that since the graph is reduced, u_1 and v_3 are not adjacent and then the five edges are different to each other). In the other branch of including u_1v_1 to

F, we will delete u_2v_2 and include u_3v_3 to F. We can get branch vector

$$(5; 2). \tag{2}$$

When $\mathcal{C}$ is not of the worst case, we can reduce more edges and get (2) at least.

Since $C(r) = 1.260^r$ satisfies the two recurrences corresponding to (1) and (2), we know that the simple algorithm can solve the TSP problem in an n-vertex degree-3 graph in $O^*(1.260^n)$ time, which achieves the same running time bound of Eppstein's algorithm for TSP3 in [5].

6 The measure and conquer method

The measure and conquer method, first introduced by Fomin, Grandoni and Kratsch [6], is one of the most powerful tools to analyze exact algorithms. It can obtain improved running time bound for many branching-and-search algorithms without making any modification to the algorithms. Currently, many best exact algorithms for NP-hard problems are based on this method. In the measure and conquer method, we may set a weight of vertices in the instance and use the sum w of the total weight in the graph as the measure to evaluate the running time. In the algorithm, the measure w should satisfy the *measure condition*: (i) when $w \leq 0$ the instance can be solved in polynomial time; (ii) the measure w will never increase in each operation in the algorithm; and (iii) the measure will decrease in each of the subinstances generated by applying a branching rule. With these constraints, we may build recurrences for the branching operations. Next, we introduce a way of applying the measure and conquer method to the above simple algorithm.

The graph has three different *vertex-weight*. For each vertex v, we set its vertex-weight $w(v)$ to be

$$w(v) = \begin{cases} w_3 = 1 \text{ if } d_U(v) = 3 \\ \quad w_{3'} \quad \text{if } d_U(v) = 2 \text{ and } d_F(v) = 1 \\ \quad 0 \quad\;\; \text{otherwise.} \end{cases}$$

We will determine the best value of $w_{3'}$ such that the worst recurrence in our algorithm is best. Let $\Delta_3 = w_3 - w_{3'}$. For a subset of vertices (or a subgraph) X, we also use $w(X)$ to denote the total vertex-weight in X.

Now we analyze the simple algorithm presented in Section 5 by using this vertex-weight setting. Note that since we require that $w_3 = 1$, the total vertex weight w in the graph is not greater than the number n of vertices in the graph. We can get a running time bound related to n if we get a running time bound related to w. Here we only examine the three bottleneck cases in Fig. 2.

When we branch on a circuit of 6-cycle in Fig. 2(a), in each branch, all the six forced vertices will be reduced and then we can reduce w by $6w_{3'}$. We get the following branching vector:

$$[6w_{3'}]_2. \tag{3}$$

When we branch on a circuit of chain of length 2 in Fig. 2(b), in the branch where u_1v_1 is included to F, u_2v_2 is deleted, and v_2v_2' and v_2v_2'' are also included to F by reduction rules, where v_2' and v_2'' are the two neighbors of v_2 other than u_2. Then we can reduce w by $w_{3'}$ from v_1, Δ_3 from u_1, w_3 from v_2, and $2\delta_1$ from v_2' and v_2'', where $\delta_1 \geq \min\{\Delta_3, w_{3'}\}$. The second branch can be analyzed in a similar manner. We get

$$[w_{3'} + \Delta_3 + w_3 + 2\delta_1]_2 = [2 + 2\delta_1]_2. \tag{4}$$

When we branch on a circuit of chain of length 3 in Fig. 2(c), in the branch of including u_1v_1 to F, we reduce w by $2\Delta_3$ from u_1 and v_3 and $2w_{3'}$ from v_1 and v_2. In the other branch, we can reduce w by $2w_3$ from u_1 and v_3, $2w_{3'}$ from v_1 and v_2, and $4\delta_1$ from $\{u_1', u_1'', v_3', v_3''\}$, where u_1' and u_1'' are the two neighbors of u_1 other than v_1, and v_3' and v_3'' are the two neighbors of v_3 other than u_3. We get

$$(2; 2w_3 + 2w_{3'} + 4\delta_1). \tag{5}$$

We can verify that under that above three constraints, the best value of $w_{3'}$ is $\frac{1}{2}w_3 = \frac{1}{2}$. With this setting, we can see that (3) and (4) become (1), and (5) becomes (2). This also tells us that the measure and conquer method cannot directly derive a better running time bound of the simple algorithm. In the next section, we present a new technique and show an improvement by combining the new technique with the traditional measure and conquer method.

7 Amortization on connectivity structures

To improve the time bound by the above simple analysis, we need to use more structural properties of the graph. Note that for the bottleneck case of Fig. 2(a), the above algorithm reduces all the vertices in this U-component. It is impossible to improve by reducing more vertices (or edges) and so on. But we also reduce the number of U-components by 1. This observation gives us an idea of an amortization scheme over the cut-circuit structure by setting a weight on each U-component in the graph.

In this method, each vertex in the graph receives a nonnegative weight as shown in Section 6. We also set a weight (which is possibly negative, but bounded from by a constant $c \geq 0$) to each U-component. Let μ be the sum of all vertex weight and U-component weight. We will use μ to measure the size of the search tree generated by our algorithm. The measure μ will also satisfy the measure condition. Initially there is only one U-component and $\mu < n + c$ holds. If we get a running time bound related to μ for our algorithm, then we get a running time bound related to n.

A simple idea is to set the same weight to each nontrivial U-component. It is possible to improve the previous best result by using this simple idea. However, to get further improvement, in this paper, we set several different component-weights. Our purpose is to distinguish some "bad" U-components, which will be characterized as "critical" U-components.

An *extension* of a 6-cycle is obtained from a 6-cycle $v_1v_2v_3v_4v_5v_6$ and a 2-clique ab by joining them with two independent edges av_i and bv_j $(i \neq j)$. An extension of a 6-cycle always has exactly eight vertices. A chord of an extension of a 6-cycle is an edge between two vertices in it but different from the eight edges $v_1v_2, v_2v_3, v_3v_4, v_4v_5, v_5v_6, v_6v_1, av_i$, bv_j and ab.

A subgraph H of a U-component in an instance (G, F) is *k-pendent critical*, if it is a 6-cycle or an extension of a 6-cycle with $|\mathrm{cut}_U(H)| = k$ and $|\mathrm{cut}_F(H)| = 6 - k$ (i.e., H has no chord of unforced/forced edge). A 0-pendent critical U-component is also simply called a *critical graph* or *critical U-component*. Fig. 3 illustrates two examples of critical graphs of extensions of a 6-cycle. Branching on a critical U-component may lead to a bottleneck recurrence in our algorithm. So we set a different component-weight to this kind of components to get improvement.

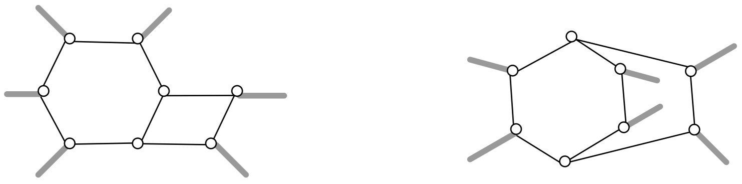

Fig. 3. Extensions of a 6-cycle

For each U-component H, we set its component-weight $c(H)$ to be

$$c(H) = \begin{cases} 0 & \text{if } H \text{ is trivial} \\ -4w_{3'} & \text{if } H \text{ is a 4-cycle} \\ \gamma & \text{if } H \text{ is a critical } U\text{-component} \\ \delta & \text{otherwise,} \end{cases}$$

where we set $c(H) = -4w_{3'}$ so that $c(H) + w(H) = 0$ holds for every 4-cycle U-component H.

We also require that the vertex-weight and component-weight satisfy the following requirements

$$2\Delta_3 \geq \gamma \geq \delta \geq \Delta_3 \geq \frac{1}{2}w_3, \; w_{3'} \geq \frac{1}{5}w_3 \; \text{ and } \; \gamma - \delta \leq w_{3'}. \tag{6}$$

Under these constraints, we still need to decide the values of $w_{3'}, \gamma$ and δ such that the time bound derived by the worst recurrences in our algorithm will be minimized. In (6), $2\Delta_3 \geq \gamma$ is important, because it will be used to satisfy the measure condition (ii). The other constraints in (6) are mainly used to simplify some arguments and they will not become the bottleneck in our analysis. Next, we first describe our simple algorithm.

8 The algorithm

A block is called a *normal block* if it is none of trivial, reducible and 2-pendent critical. A normal block is *minimal* if no subgraph of it is a normal block along any circuit. Note that when F is not empty, each U-component has at least one nontrivial circuit. Our recursive algorithm for forced TSP only contains two main steps:
1. First apply the reduction rules to a given instance until it becomes a reduced one; and
2. Then take any U-component H that is neither trivial nor a 4-cycle (if no such U-component H, then the instance is polynomially solvable by Lemma 7), and branch on a nontrivial circuit $\mathcal{C}$ in H, where $\mathcal{C}$ is chosen so that
(1) no normal block appears along $\mathcal{C}$ (i.e., $\mathcal{C}$ has only trivial and 2-pendent critical blocks) if this kind of circuits exist; and
(2) a minimal normal block B_1 in H appears along $\mathcal{C}$ otherwise.

We use μ as the measure to analyze the size of the search tree in our algorithm. It is easy to see that after applying the reduction rules on a 2-edge-connected instance, the resulting instance remains 2-edge-connected. By this observation and Lemma 8, we can guarantee that an input instance is always 2-edge-connected. Our analysis is based on this.

8.1 Basic properties of the measure

Before analyzing the time bound on our algorithm, we first give basic properties of the measure μ.

We show that the measure will not increase after applying any reduction operation in an 2-edge-connected instance. Since an input instance is 2-edge-connected, there is no eliminable edge. In fact, we always deal with eliminable edges in circuit procedures. For reducible edges, we deal with them during a process of a reducible circuit (including the reducible edges to F and dealing with the resulting eliminable edges). We will show that μ never increases after processing a circuit. The measure μ will not increase after deleting any unforced parallel edge. The following lemma also shows that applying the 3/4-cut reduction does not increase μ.

Lemma 9. *For a given instance,*
(i) *applying the* 3*-cut reduction does not increase the measure* μ*; and*
(ii) *applying the* 4*-cut reduction on a* U*-component* X *decreases the measure* μ *by* $w(X) + c(X)$.

Proof. It is easy to observe (i). Next we prove (ii). In the case where the resulting graph consists of four single vertices or a pair of forced edges after applying the 4-cut reduction, the whole component X is eliminated, decreasing μ by $w(X) + c(X)$ and then the lemma holds. Otherwise the resulting component, say X' is a 4-cycle, where $w(X') + c(X') = 0$ according to our setting on the component-weight of 4-cycles, and μ again decreases by $w(X) + c(X)$. ∎

Next we consider how much amount of measure decreases by processing a circuit. We consider that the measure μ becomes zero whenever we find an instance infeasible by Lemma 3. After processing a circuit $\mathcal{C} = \{e_i = u_iv_i \mid 1 \leq i \leq p\}$ in a U-component H, each block B_i along $\mathcal{C}$ becomes a new U-component, which we denote by $\bar{B}_i$. We define the *direct benefit* $\beta'(B_i)$ from B_i to be the decrease in vertex-weight of the endpoints v_i and u_{i+1} of B_i minus the component-weight $c(\bar{B}_i)$ in the new instance after the circuit procedure. Immediately after the procedure, the measure μ decreases by $w(H) + c(H) - \sum_i (w(\bar{B}_i) + c(\bar{B}_i)) = c(H) + \sum_i \beta'(B_i)$. After the circuit procedure, we see that the vertex-weights of endpoints of each non-reducible and nontrivial block B_i decreases by $\varDelta_3$ and $\varDelta_3$ (or w_3 and w_3) respectively if B_i is even, and by $\varDelta_3$ and w_3 (or w_3 and $\varDelta_3$) respectively if B_i is odd. Summarizing these, the direct benefit $\beta'(B)$ from a block B is given by

$$\beta'(B) = \begin{cases} 0 & \text{if } B \text{ is reducible,} \\ w_{3'} & \text{if } B \text{ is trivial,} \\ w_3 + \varDelta_3 - \delta & \text{if } B \text{ is odd and nontrivial,} \\ 2w_3 - \delta & \text{if } B \text{ is even and non-reducible, and } \mathrm{cut}_U(B) \text{ is deleted,} \\ 2\varDelta_3 - \gamma & \text{if } B \text{ is 2-pendent critical, and } \mathrm{cut}_U(B) \text{ is included in } F, \\ w(B) & \text{if } B \text{ is a 2-pendent 4-cycle, and } \mathrm{cut}_U(B) \text{ is included in } F, \\ 2\varDelta_3 - \delta & \text{otherwise (i.e., } B \text{ is even, non-reducible but not} \\ & \quad \text{a 2-pendent critical } U\text{-graph or a 2-pendent 4-cycle,} \\ & \quad \text{and } \mathrm{cut}_U(B) \text{ is included to } F). \end{cases} \tag{7}$$

By (6) and (7), we have that $\beta'(B_i) \geq 0$ for any type of block B_i, which implies that the decrease $c(H) + \sum_i \beta'(B_i) \geq c(H) \geq 0$ (where H is not a 4-cycle) is in fact nonnegative, i.e., the measure μ never increases by processing a circuit.

After processing a circuit $\mathcal{C}$, a reduction operation may be applicable to some U-components $\bar{B}_i$ and we can decrease μ more by reducing them. The *indirect benefit* $\beta''(B)$ from a block B is defined as the amount of μ decreased by applying reduction rules on the U-component $\bar{B}$ after processing the circuit. Since we have shown that μ never increases by applying reduction rules, we know that $\beta''(B)$ is always nonnegative. The *total benefit* (*benefit*, for short) from a block B is

$$\beta(B) = \beta'(B) + \beta''(B).$$

Lemma 10. *After processing a circuit $\mathcal{C}$ in a 2-edge-connected U-component H (not necessary being reduced) and applying reduction rules until the instance becomes a reduced one, the measure μ decreases by*

$$c(H) + \sum_i \beta(B_i),$$

where B_i are the blocks along circuit $\mathcal{C}$.

The indirect benefit from a block depends on the structure of the block. In our algorithm, we hope that the indirect benefit is as large as possible. Here we prove some lower bounds on it for some special cases.

Lemma 11. *Let H be a U-component containing no induced triangle and $\mathcal{C}'$ be a reducible circuit in it such that there is exactly one reducible block along $\mathcal{C}'$. The measure μ decreases by at least $2\mathit{\Delta}_3$ by processing the reducible circuit $\mathcal{C}'$ and applying reduction rules.*

Proof. By assumption, the reducible circuit has at least two blocks, one reducible block B_1 and one non-reducible block B_2. (1) Assume that every other block than B_1 along $\mathcal{C}'$ is trivial. Then H should be a cycle of length $k \geq 5$ (if H is a 4-cycle, then there are three odd blocks along $\mathcal{C}'$ and we find the instance infeasible by Lemma 3). There are $k-1 \geq 4$ trivial blocks along $\mathcal{C}'$. After processing the circuit, the whole component H will be eliminated decreasing μ by at least $c(H) + w(H) \geq \delta + 4w_{3'} \geq 2\mathit{\Delta}_3$ (by (6)). (2) Otherwise, i.e., there is a block B_2 of more than one vertex (B_2 is not trivial or reducible): (2-i) $\mathrm{cut}_U(B_2)$ is deleted in the circuit procedure: Then μ decreases by at least $c(H) + \beta'(B_2) = \delta + (2w_3 - \delta) = 2w_3 \geq 2\mathit{\Delta}_3$ by (7). Hence assume that $\mathrm{cut}_U(B_2)$ is included to F in (2). (2-ii) B_2 is not 2-pendent critical: Then the measure μ decreases by at least $c(H)+\beta'(B_2) = \delta+(2\mathit{\Delta}_3-\delta) = 2\mathit{\Delta}_3$. The remaining case is that B_2 is 2-pendent critical and $\mathrm{cut}_U(B_2)$ is included to F. (2-iii) there are only two blocks B_1 and B_2 along $\mathcal{C}'$: By processing the circuit $\mathcal{C}'$, the two edges in $\{xz, yz\} = \mathrm{cut}_U(B_2)$ become forced edges. Since they are incident on the single vertex z in B_1, we can replace xz and yz with a single forced edge xy and then B_2 becomes a 0-pendent 6-cycle or extension of a 6-cycle with only one forced chord xy. After the circuit procedure of $\mathcal{C}'$, we can apply 4-cut reduction to the 0-pendent 6-cycle B_2 (by Lemma 6) and then this decreases μ by $c(H) + \beta(B_2) = \delta + w(B_2) > 2\mathit{\Delta}_3$ (by Lemma 9(ii)). (2-iv) there is a pair of two trivial blocks B_3 and B_4 or a nontrivial and non-reducible block B_5 along $\mathcal{C}'$: Now we can decrease μ by at least $c(H)+\beta(B_2)+\sum_{i\neq 2}\beta(B_i) \geq \delta+(2\mathit{\Delta}_3-\gamma)+\min\{2w_{3'},(2\mathit{\Delta}_3-\delta)\} \geq 2\mathit{\Delta}_3$ (by (6)).

In any case, the measure μ decreases by at least $2\mathit{\Delta}_3$. ∎

Lemma 12. *In the circuit procedure for a circuit $\mathcal{C}$ in a reduced instance, the indirect benefit from a block B along $\mathcal{C}$ satisfies*

$$\beta''(B) \geq \begin{cases} 2\mathit{\Delta}_3 & \text{if } B \text{ is odd and nontrivial,} & \text{(i)} \\ w(B) - \beta'(B) & \text{if } B \text{ is a 2-pendent cycle or critical graph,} & \\ & \quad \text{and } \mathrm{cut}_U(B) \text{ is deleted,} & \text{(ii)} \\ \delta & \text{if } B \text{ is even but not reducible or a 2-pendent} & \\ & \quad \text{cycle, and } \mathrm{cut}_U(B) \text{ is deleted,} & \text{(iii)} \\ 0 & \text{otherwise.} & \text{(iv)} \end{cases}$$

Proof. We will use $\bar{B}$ to denote the U-component resulting from B after the circuit procession. Case (i): Since B is odd, only one edge in $\mathrm{cut}_U(B)$ is deleted (the other one is included to F) in the circuit procession. Then there is exactly one vertex of degree 2 in $\bar{B}$, which is the only reducible block along a circuit $\mathcal{C}'$ in $\bar{B}$. Since the original instance is reduced and contains no triangle, we know that circuit $\mathcal{C}'$ satisfies the condition in Lemma 11. By processing $\mathcal{C}'$ in $\bar{B}$, we can decrease μ by at least $2\Delta_3$ by Lemma 11. Then we get $\beta''(B) \geq 2\Delta_3$. For case (ii), if B is a 2-pendent cycle, we can reduce the whole U-component $\bar{B}$ after the circuit procedure, and then we have $\beta''(B) = c(\bar{B}) + w(\bar{B})$. Otherwise B in (ii) is a 2-pendent critical graph and the 4-cut reduction can be applied to $\bar{B}$ by Lemma 6 since the two end vertices of edges in $\mathrm{cut}_U(B)$ will be of degree 2 in $\bar{B}$ after $\mathrm{cut}_U(B)$ is deleted. Then we still have $\beta''(B) = c(\bar{B}) + w(\bar{B})$ by Lemma 9(ii). Note that $\beta'(B) = w(B) - w(\bar{B}) - c(\bar{B})$ (by the definition of $\beta'(B)$). We get $\beta''(B) = w(B) - \beta'(B)$. For Case (iii), we will get at least one reducible circuit $\mathcal{C}'$ in $\bar{B}$. Note that $\bar{B}$ has some vertex of degree 2 and cannot be a critical graph. Hence $c(\bar{B}) = \delta$. By processing the reducible circuit $\mathcal{C}'$, we can decrease μ by at least $c(\bar{B}) + \sum_i \beta(B'_i) \geq c(\bar{B}) = \delta$, where B'_i are the blocks along $\mathcal{C}'$. The inequality in (iv) holds since we have proved that μ will never increase after applying reduction rules. ∎

9 The Analysis

Now we are ready to analyze our algorithm. In the algorithm, branching on a circuit generates two instances (G_1, F_1) and (G_2, F_2). By Lemma 10, we get branch vector

$$(c(H) + \sum_i \beta_1(B_i); c(H) + \sum_i \beta_2(B_i)),$$

where $\beta_j(B)$, $\beta'_j(B)$ and $\beta''_j(B)$ denote the functions $\beta(B)$, $\beta'(B)$ and $\beta''(B)$ evaluated in (G_j, F_j), $j = 1, 2$ for clarifying how branch vectors are derived in the subsequent analysis. We consider this branch vector for different cases. First we analyze the easy case where the chosen circuit $\mathcal{C}$ has no normal block. Then we analyze the somewhat complicated case where there is a minimal normal block along $\mathcal{C}$.

9.1 Circuits with only trivial and 2-pendent critical blocks

In this subsection, we assume that the chosen circuit $\mathcal{C}$ in a U-component H (not a 4-cycle) has only trivial and 2-pendent critical blocks. We consider the following three cases.

Case 1. All blocks along $\mathcal{C}$ are trivial blocks: Now H should be a cycle of even length $l \geq 6$. By (7) and Lemma 10, we know that in each branch we can decrease μ by at least

$$c(H) + \sum_i \beta(B_i) \geq \begin{cases} \gamma + 6w_{3'} & \text{if } l = 6 \\ \delta + 8w_{3'} & \text{if } l \geq 8. \end{cases}$$

Then we get branch vectors

$$[6w_{3'} + \gamma]_2 \tag{8}$$

for $l = 6$ and $[8w_{3'} + \delta]_2$ for $l \geq 8$, which is covered by (8). Note that (8) is tight for a circuit of 6-cycle in Fig. 2(a).

Case 2. There is only one 2-pendent critical block B_1 along $\mathcal{C}$: Since B_1 is even, the number $r > 0$ of trivial blocks (odd blocks) along $\mathcal{C}$ is also even by the parity condition. Note that if $r = 2$ and B_1 is 2-pendent 6-cycle then H is a critical graph (an extension of a 6-cycle) and $c(H) = \gamma$. Otherwise H is not critical and $c(H) = \delta$. Then in the branch where $\mathrm{cut}_U(B_1)$ is deleted, the decrease $c(H) + \sum_i \beta(B_i)$ of μ is at least $c(H) + w(B_2) + rw_{3'} \geq \min\{\gamma + (2w_3 + 4w_{3'}) + 2w_{3'}, \delta + (4w_3 + 4w_{3'}) + 2w_{3'}, \delta + (2w_3 + 4w_{3'}) + 4w_{3'}\} =$

$$\gamma + 2w_3 + 6w_{3'}.$$

In the branch where $\mathrm{cut}_U(B_1)$ is included to F, the decrease $c(H) + \sum_i \beta(B_i)$ of μ is at least

$$\min\{\gamma, \delta\} + (2\Delta_3 - \gamma) + 2w_{3'} = \delta + 2w_3 - \gamma.$$

This gives branch vector

$$(\gamma + 2w_3 + 6w_{3'}; \delta + 2w_3 - \gamma). \tag{9}$$

Case 3. There are at least two 2-pendent critical blocks B_1 and B_2 along $\mathcal{C}$: If $\mathrm{cut}_U(B_2)$ is included to F in the branch where $\mathrm{cut}_U(B_1)$ is deleted, then we can get branch vector $[c(H)+\beta(B_1)+\beta(B_2)]_2 = [\delta+(2w_3+4w_{3'})+(2\Delta_3-\gamma)]_2 = [\delta + 4w_3 + 2w_{3'} - \gamma]_2$, which is covered by (8). On the other hand, if $\mathrm{cut}_U(B_2)$ is also deleted in the branch where $\mathrm{cut}_U(B_1)$ is deleted, we get branch vector

$$(\delta + 2(2\Delta_3 - \gamma); \delta + 2(2w_3 + 4w_{3'})). \tag{10}$$

9.2 Circuits with a minimal normal block

In this subsection, we assume that the chosen circuit $\mathcal{C}$ in a U-component H has a minimal normal block B_1. Now $c(H) = \delta$ always holds. We distinguish several cases to analyze the branch vectors.

Case 1. Block B_1 is odd: Circuit $\mathcal{C}$ has another odd block B_2. By (7) we have $\beta(B_2) \geq \beta'(B_2) = \min\{w_{3'}, w_3 + \Delta_3 - \delta\} = w_{3'}$ in each branch. For B_1, we have that $\beta'(B_1) = w_3 + \Delta_3 - \delta$ and $\beta''(B_1) \geq 2\Delta_3$ by Lemma 12. In each branch, μ decreases by at least $c(H) + \sum_i \beta(B_i) \geq \delta + \beta'(B_1) + \beta''(B_1) + \beta(B_2) \geq \delta + (w_3 + \Delta_3 - \delta) + 2\Delta_3 + w_{3'} = 4w_3 - 2w_{3'}$. Therefore, we can get branch vector

$$[4w_3 - 2w_{3'}]_2. \tag{11}$$

Note that (11) is tight for a circuit of chain of length 2 in Fig. 2(b).

Case 2. Block B_1 is even: In the branch where $\mathrm{cut}_U(B_1)$ is included to F, we have that $\beta'_1(B_1) = 2\Delta_3 - \delta$. In the other branch, we have that $\beta'_2(B_1) = 2w_3 - \delta$. By branching on $\mathcal{C}$, we get branch vector

$$(\delta + \beta'_1(B_1) + \beta''_1(B_1) + \xi_1; \delta + \beta'_2(B_1) + \beta''_2(B_1) + \xi_2), \tag{12}$$

where $\xi_j = \sum_{i \neq 1} \beta_j(B_i) \geq 0$ ($j \in \{1,2\}$), $\beta'_1(B_1) = 2\Delta_3 - \delta$ and $\beta'_2(B_1) = 2w_3 - \delta$.

In what follows, we derive some lower bounds on ξ_1, ξ_2, $\beta''_1(B_1)$ and $\beta''_2(B_1)$ by examining the structure of B_1.

Let $\mathrm{cut}_U(B_1) = \{xv, yu\}$, where x and y are in B_1. Let (G_1, F_1) and (G_2, F_2) be the two resulting instances after branching and processing the circuit $\mathcal{C}$, where (G_1, F_1) corresponds to the branch where $\mathrm{cut}_U(B_1)$ is included to F and (G_2, F_2) corresponds to the branch where $\mathrm{cut}_U(B_1)$ is deleted. Let x_1x and x_2x (resp., y_1y and y_2y) be the two unforced edges incident on x (resp., y) in (G_1, F_1). Note that x_1x and x_2x (resp., y_1y and y_2y) will be in the same circuit $\mathcal{C}_x$ (resp., $\mathcal{C}_y$) in (G_1, F_1), since x (resp., y) is a forced vertex now. See Fig. 4 for illustrations of the structure of the edges incident to x and y.

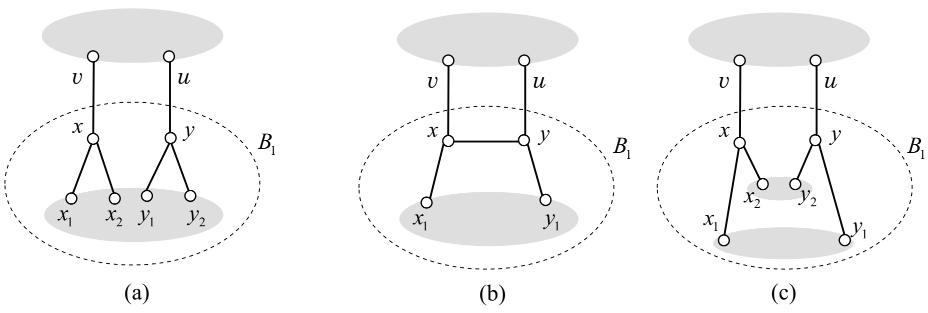

Fig. 4. Illustrations of block B_1: (a) $\mathcal{C}_x \neq \mathcal{C}_y$; (b) $\mathcal{C}_x = \mathcal{C}_y$ and x and y are adjacent; (c) $\mathcal{C}_x = \mathcal{C}_y$ and x and y are not adjacent.

Case 2.1. Block B_1 is even and $\mathcal{C}_x$ and $\mathcal{C}_y$ are two different circuits in (G_1, F_1) (see Fig. 4(a)): Now $\mathcal{C}_x$ and $\mathcal{C}_y$ are two different circuits also in (G_2, F_2). In the branch where $\mathrm{cut}_U(B_1)$ is deleted, we have $\beta''_2(B_1) \geq 4\Delta_3$ (by applying Lemma 11 to $\mathcal{C}_x$ and $\mathcal{C}_y$). Next we consider ξ_1 and ξ_2 in (12).

Case 2.1.1. There are at least two odd blocks B_2 and B_3 along $\mathcal{C}$: By (7), $\beta_1(B_i) \geq w_{3'}$ and $\beta_2(B_i) \geq w_{3'}$ ($i \in \{2,3\}$). Then ξ_1 and $\xi_2 \geq 2w_{3'}$. By (12), we get branch vector

$$(\delta + (2\Delta_3 - \delta) + 2w_{3'}; \delta + (2w_3 - \delta) + 4\Delta_3 + 2w_{3'}) = (2w_3; 6w_3 - 2w_{3'}). \quad (13)$$

Case 2.1.2. There is no odd block along $\mathcal{C}$: Let B_2 be the block along $\mathcal{C}$ containing vertex v. Then B_2 is an even block such that $\mathrm{cut}_U(B_2)$ is also included to F in the branch where $\mathrm{cut}_U(B_1)$ is included to F. We have that $\xi_1, \xi_2 \geq \beta(B_2)$. If B_2 is not a 2-pendent critical block, then $\beta'_1(B_2) = 2\Delta_3 - \delta$, $\beta'_2(B_2) = 2w_3 - \delta$ and $\beta''_2(B_2) \geq \delta$ (by Lemma 12), which imply $\xi_1 \geq 2\Delta_3 - \delta$ and $\xi_2 \geq (2w_3 - \delta) + \delta = 2w_3$. By (12), we get branch vector

$$(\delta + 2(2\Delta_3 - \delta); \delta + (2w_3 - \delta) + 4\Delta_3 + 2w_3) = (4\Delta_3 - \delta; 4w_3 + 4\Delta_3). \quad (14)$$

Otherwise B_2 is a 2-pendent critical block. Then $\beta'_1(B_2) = 2\varDelta_3 - \gamma$ and $\beta_2(B_2) = \beta'_2(B_2) + \beta''_2(B_2) = w(B_2) \geq 2w_3 + 4w_{3'}$ (by (7) and Lemma 12), which imply $\xi_1 \geq 2\varDelta_3 - \gamma$ and $\xi_2 \geq 2w_3 + 4w_{3'}$. We get branch vector $(\delta + (2\varDelta_3 - \delta) + (2\varDelta_3 - \gamma); \delta + (2w_3 - \delta) + 4\varDelta_3 + (2w_3 + 4w_{3'}))$, i.e.,

$$(4\varDelta_3 - \gamma; 8w_3). \tag{15}$$

Case 2.2. Block B_1 is even and $\mathcal{C}_x = \mathcal{C}_y$ in (G_1, F_1) (see Fig. 4(b),(c)): We first evaluate ξ_1 and ξ_2 in (12). If all blocks other than B_1 along $\mathcal{C}$ are 2-pendent critical, then we have that $\xi_1 \geq 2\varDelta_3 - \gamma$ and $\xi_2 = \sum_{i \neq 1} w(B_i) \geq w(B_2) \geq 2w_3 + 4w_{3'}$. Otherwise, $\xi_1 \geq \min\{2w_{3'}, w_3 + \varDelta_3 - \delta, 2\varDelta_3 - \delta\} = 2w_{3'}$ and $\xi_2 \geq \min\{2w_{3'}, w_3 + \varDelta_3 - \delta, 2w_3 - \delta\} = 2w_{3'}$. We have the following two choices for (ξ_1, ξ_2):

$$(\xi_1, \xi_2) = (2\varDelta_3 - \gamma, 2w_3 + 4w_{3'}) \quad \text{and} \quad (2w_{3'}, 2w_{3'}). \tag{16}$$

Next, we consider $\beta''_1(B_1)$, $\beta''_2(B_1)$ and others. We look at the circuit $\mathcal{C}_x$ in (G_1, F_1). Except blocks $\{x\}$ and $\{y\}$, there are some other blocks along $\mathcal{C}_x$. Note that each block B' along $\mathcal{C}_x$ should be a trivial or 2-pendent critical block since B_1 is a minimal normal block. We distinguish three cases by considering the number of critical blocks along $\mathcal{C}_x$.

Case 2.2.1. B_1 is a 2-pendent cycle of length ℓ (all blocks along $\mathcal{C}_x$ are trivial in (G_1, F_1)): Then ℓ is an even integer with $\ell = 4$ or $\ell \geq 8$ (since B_1 is even and non-critical). When $\ell = 4$, in both branches, we have $\beta(B_1) = w(B_1) = 2w_3 + 2w_{3'}$. Then we can branch with a branch vector $[c(H) + \beta(B_1)]_2 = [\delta + 2w_3 + 2w_{3'}]_2$ covered by (8). Next we assume that $\ell \geq 8$. Now we may get only $\beta'_1(B_1) = 2\varDelta_3 - \delta$ and $\beta''_1(B_1) \geq 0$ instead of $\beta_1(B_1) = w(B_1)$. But it holds that $\beta_2(B_1) = w(B_1) = 2w_3 + (\ell - 2)w_{3'} \geq 2w_3 + 6w_{3'}$. We get branch vector

$$(\delta + (2\varDelta_3 - \delta) + \xi_1; \delta + (2w_3 + 6w_{3'}) + \xi_2). \tag{17}$$

Case 2.2.2. There are only three blocks $\{x\}$, $\{y\}$ and B' along $\mathcal{C}_x$ in (G_1, F_1), where B' is a 2-pendent critical block: Now x and y are adjacent (see Fig. 4(b)). We assume that $x_2 = y$, $y_2 = x$, and $x_1, y_1 \in B_1$. We look at (G_2, F_2) wherein x and y are degree-2 vertices. After including edges xy, xx_1 and yy_1 to F, B' becomes a 4-cut reducible graph by Lemma 6. Then μ decreases by $w(B')$ after applying the 4-cut reduction. Then we know that $\beta_2(B_1) = w(B_1) = w(x) + w(y) + w(B') \geq 4w_3 + 4w_{3'}$. Therefore, we get branch vector

$$(\delta + (2\varDelta_3 - \delta) + \xi_1; \delta + (4w_3 + 4w_{3'}) + \xi_2),$$

which is covered by (17).

Case 2.2.3. B_1 is not a 2-pendent cycle and there are more than three blocks along $\mathcal{C}_x$ in (G_1, F_1): Then there is a nontrivial and nonreducible block B'_1 along $\mathcal{C}_x$ in (G_1, F_1), where B'_1 is a 2-pendent critical block since B_1 is a minimal normal block. For this case, we only get the following branch vector by branching on $\mathcal{C}$: $(\delta + (2\varDelta_3 - \delta) + \xi_1; \delta + (2w_3 - \delta) + \beta''_2(B_1) + \xi_2) =$

$$(2\varDelta_3 + \xi_1; 2w_3 + \beta''_2(B_1) + \xi_2). \tag{18}$$

In fact, the branch vector (18) in Case 2.2.3 can be the bottleneck in the analysis of our algorithm. However, in (G_1, F_1), circuit $\mathcal{C}_x$ is a circuit with only trivial and 2-pendent critical blocks. In our algorithm, circuit $\mathcal{C}_x$ will be one of the circuits for the next branching, and it will never be destroyed until we branch on it. In fact, branching on $\mathcal{C}_x$ proves a branch vector better than (18). For the purpose of analysis, we derive a branch vector for the three branches, i.e., the branching on $\mathcal{C}$ followed by the branching on $\mathcal{C}_x$ in (G_1, F_1) in the branch of including $\mathrm{cut}_U(B_1)$ into F.

In Case 2.2.3, we see that: either (a) $\mathcal{C}_x$ has at least two trivial blocks B'_2 and B'_3 different from $\{x\}$ and $\{y\}$ (since the number of odd blocks is even); or (b) all blocks other than $\{x\}$ and $\{y\}$ are 2-pendent critical.

Case (a): There is also a 2-pendent critical block B'_1 along $\mathcal{C}_x$ (since B_1 is not a 2-pendent cycle). In (G_2, F_2), we can see that $\beta''_2(B_1) \geq c(B_1)+w(B'_2)+w(B'_3) = \delta+2w_{3'}$ (since B'_2 and B'_3 are trivial blocks). In (G_1, F_1), by branching on $\mathcal{C}_x$, we can get branch vector $(c(B_1)+\beta_1(\{x\})+\beta_1(\{y\})+\sum_{i=1}^{3}\beta_1(B'_i); c(B_1)+\beta_2(\{x\})+\beta_2(\{y\})+\sum_{i=1}^{3}\beta_2(B'_i)) = (\delta+4w_{3'}+(2\Delta_3-\gamma); \delta+4w_{3'}+(2w_3+4w_{3'})) =$

$$(\delta+2w_3+2w_{3'}-\gamma; \delta+2w_3+8w_{3'}).$$

By combining it with (18) and taking $\beta''_2(B_1) = \delta+2w_{3'}$, we get branch vector

$$(2\Delta_3+\xi_1+(\delta+2w_3+2w_{3'}-\gamma); 2\Delta_3+\xi_1+(\delta+2w_3+8w_{3'}); 2w_3+\delta+2w_{3'}+\xi_2). \quad (19)$$

Case (b): There are at least two 2-pendent critical blocks B'_1 and B'_2 along $\mathcal{C}_x$ (since there are at least four blocks among $\mathcal{C}_x$). In (G_2, F_2), we may get only $\beta''_2(B_1) \geq \delta$. In (G_1, F_1), by branching on $\mathcal{C}_x$, at least we can get branch vector

$$(\delta+2w_{3'}+2(2\Delta_3-\gamma); \delta+2w_{3'}+2(2w_3+4w_{3'}).$$

By combining it with (18) and taking $\beta''_2(B_1) = \delta$, we get branch vector

$$(2\Delta_3+\xi_1+(\delta+4w_3-2w_{3'}-2\gamma); 2\Delta_3+\xi_1+(\delta+4w_3+10w_{3'}); 2w_3+\delta+\xi_2). \quad (20)$$

Finally, by replacing ξ_1 and ξ_2 in (17), (19) and (20) respectively with the bounds in (16), we get the following six branch vectors

$$(4\Delta_3-\gamma; \delta+4w_3+10w_{3'}), \quad (21)$$

$$(2w_3; \delta+2w_3+8w_{3'}), \quad (22)$$

$$(\delta+6w_3-2w_{3'}-2\gamma; \delta+6w_3+4w_{3'}-\gamma; \delta+4w_3+6w_{3'}), \quad (23)$$

$$(\delta+4w_3+2w_{3'}-\gamma; \delta+4w_3+8w_{3'}; \delta+2w_3+4w_{3'}), \quad (24)$$

$$(\delta+8w_3-6w_{3'}-3\gamma; \delta+8w_3+6w_{3'}-\gamma; \delta+4w_3+4w_{3'}), \quad (25)$$

and

$$(\delta+6w_3-2w_{3'}-2\gamma; \delta+6w_3+10w_{3'}; \delta+2w_3+3w_{3'}). \quad (26)$$

9.3 Overall analysis

A quasiconvex program is obtained from (6) and 13 branch vectors (from (8) to (15) and from (21) to (26)) in our analysis. There is a general method to solve quasiconvex programs [4]. For our quasiconvex program, we observe a simple way to solve it. We look at (8) and (11). Note that $\min\{6w_{3'}+\gamma, 4w_3-2w_{3'}\}$ under the constraint $2\varDelta_3 \geq \gamma$ gets the maximum value at the time when $6w_{3'}+\gamma = 4w_3-2w_{3'}$ and $2\varDelta_3 = \gamma$. We get $w_{3'} = \frac{1}{3}$ and $\gamma = \frac{4}{3}$. With this setting, we can verify that when $\delta \in [1.2584, 1.2832]$, all branch vectors other than (8) and (11) in our quasiconvex program will not be the bottleneck. We get a time bound $O^*(\alpha^\mu)$ with $\alpha = 2^{\frac{3}{10}} < 1.2312$ by setting $w_{3'} = \frac{1}{3}, \gamma = \frac{4}{3}$ and $\delta \in [1.2584, 1.2832]$ for our problem. The bottlenecks in the analysis are (8), (11) and $2\varDelta_3 \geq \gamma$ in (6).

Theorem 1. *TSP in an n-vertex graph G with maximum degree 3 can be solved in $O^*(1.2312^n)$ time and polynomial space.*

10 Concluding Remarks

In this paper, we have presented an improved exact algorithm for TSP in degree-3 graphs. The basic operation in the algorithm is to process the edges in a circuit by either including an edge in the circuit to the solution or excluding it from the solution. The algorithm is analyzed by using the measure and conquer method and an amortization scheme over the cut-circuit structure of graphs, wherein we introduce not only weights of vertices but also weights of U-components to define the measure of an instance.

The idea of amortization schemes introducing weights on components may yield better bounds for other exact algorithms for graph problems if how reduction/branching procedures change the system of components is successfully analyzed.

References

1. Bjorklund, A.: Determinant sums for undirected Hamiltonicity. In: Proc. 51st Annual IEEE Symp. on Foundations of Computer Science (2010) 173-182
2. Bjorklund, A., Husfeldt, T., Kasaki, P. and Koivisto, M.: The travelling salesman problem in bounded degree graphs. In: ICALP 2008, LNCS 5125 (2008) 198-209
3. Dorn, F., Penninkx, E., Bodlaender, H. L., and Fomin, F. V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. Algorithmica 58(3) (2010) 790-810
4. Eppstein, D.: Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. ACM Trans. on Algorithms 2(4) (2006) 492-509
5. Eppstein, D.: The traveling salesman problem for cubic graphs. J. Graph Algorithms and Applications 11(1) (2007) 61-81
6. Fomin, F., Grandoni, F., Kratsch, D. Measure and Conquer: Domination - A Case Study, *ICALP* Springer-Verlag LNCS 3580 (2005), pp. 191–203.

7. Fomin, F. V., Kratsch, D.: Exact Exponential Algorithms, Springer (2010)
8. Gebauer, H.: Finding and enumerating Hamilton cycles in 4-regular graphs. Theoretical Computer Science 412(35) (2011) 4579-4591
9. Iwama, K. and Nakashima, T.: An improved exact algorithm for cubic graph TSP. In: COCOON 2007. LNCS 4598 (2007) 108-117
10. Nagamochi, H., Ibaraki, T.: A linear time algorithm for computing 3-edge-connected components in multigraphs, J. of Japan Society for Industrial and Applied Mathematics, 9(2) (1992) 163-180
11. Nagamochi, H., Ibaraki, T.: Algorithmic Aspects of Graph Connectivities, Encyclopedia of Mathematics and Its Applications, Cambridge University Press (2008)
12. Woeginger, G. J.: Exact algorithms for NP-hard problems: A survey. In: Combinatorial Optimization. LNCS 2570 (2003) 185-207
13. Xiao, M. and Nagamochi, H.: An improved exact algorithm for TSP in degree-4 graphs. In: COCOON 2012. LNCS 7434 (2012) 74-85