A Sublogarithmic Approximation for Tollbooth Pricing on Cactus Graphs

(Algorytm aproksymacyjny dla problemu rogatek na kaktusach)

Andrzej Turko

Praca licencjacka

Promotor: prof. Jarosław Byrka

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

18 czerwca 2021

Abstract

We study a special case of the envy-free pricing problem, in which each buyer wishes to buy a shortest path connecting her individual pair of vertices in a network owned by a single vendor. The vendor sets the prices of individual edges with the aim of maximizing the total revenue generated by all buyers. Each customer buys a path as long as its cost does not exceed her individual budget. In this case, the revenue generated by her equals the sum of prices of edges along this path. Otherwise, it is zero. We consider the unlimited supply setting, where each edge can be sold to arbitrarily many customers. The problem is to find a price assignment which maximizes vendor's revenue.

In the tollbooth problem we assume that the network is a tree. In this work we consider a slightly more general model and do not require the network to be acyclic. Instead, each edge can belong to at most one simple cycle. Our result is a polynomial time $\mathcal{O}\left(\frac{\log m}{\log \log m}\right)$-approximation algorithm for this problem. It is a generalization of a previous result for the original tollbooth problem with the same approximation guarantee [4].

Streszczenie

Niniejsza praca poświęcona jest następującemu problemowi: dany jest zbiór potencjalnych nabywców, z których każdy chciałby kupić ścieżkę pomiędzy ustalonymi dwoma wierzchołkami w pewnej sieci. Sprzedawca, który jest właścicielem całej sieci, ustala ceny jej poszczególnych krawędzi. Każdy klient wybiera najtańszą ścieżkę łączącą jego wybrane dwa wierzchołki. Jeśli jej koszt nie przekracza jego budżetu, kupuje ją, generując przychód sprzedawcy równy sumie cen krawędzi na tej ścieżce. Rozważamy wariant z nieograniczoną podażą, to znaczy każda krawędź może być sprzedana dowolnie wielu nabywcom. Problem polega na znalezieniu cen krawędzi, które maksymalizują przychód sprzedawcy.

Problem rogatek dotyczy powyższego scenariusza z dodatkowym założeniem, że sieć jest drzewem. W tej pracy rozpatrujemy ogólniejszą sytuację. Zakładamy jedynie, że każda krawędź leży na co najwyżej jednym cyklu prostym. Prezentujemy wielomianowy algorytm, który znajduje ceny krawędzi generujące dochód mniejszy od optymalnego $\mathcal{O}\left(\frac{\log m}{\log \log m}\right)$ razy. Ten wynik jest uogólnieniem znanego już algorytmu rozwiązującego klasyczny problem rogatek, który zapewnia takie samo ograniczenie dolne na wysokość uzyskanych przychodów [4].

Contents

Chapter 1

Introduction

Picture a vendor offering a number of various goods to customers, who have different preferences regarding them. Adjusting the prices of goods to match the profile of potential buyers is a natural step to consider in order to maximize revenue. Especially since the ever-increasing amount of customer data being collected makes it easier to measure buyers' valuations with greater precision.

The problem of maximizing revenue by setting optimal prices has been widely studied in various settings. This work discusses the problem of envy-free pricing for revenue maximization. In general, this problem can be modeled as a two phase game. In the first step, vendor assigns prices to the offered goods. Then, each buyer purchases her most preferred subset of goods based on given prices and her own preferences. Every buyer aims to maximize her utility, and the seller aims to maximize the total price paid by customers. The problem is to find an optimal strategy for the vendor.

More precisely, an instance of the envy-free pricing problem consists of m goods and n buyers. Each buyer is defined by a function which assigns a non-negative valuation to every subset of the goods. It is assumed that the valuation of an empty set for each customer equals zero. A solution to the problem is formed by non-negative prices of goods and an envy-free allocation of goods to the buyers. Utility of a buyer from a set of goods equals her valuation of this set minus the total price of its elements. An allocation is envy-free when no buyer would like to change her assigned set of goods. In other words, the set assigned to her must maximize her utility. In this work we focus on the unlimited supply setting, where each one of the m goods can be sold to arbitrarily many buyers. Such goods may be thought of as intellectual property or access to infrastructure. Sometimes the limited supply setting is also considered, where each good is available only in a certain number of copies. In that case, the solution must not only satisfy the envy-freeness constraints, but also the number of buyers any good is allocated to must not exceed its supply.

In this work we study a particular case of the envy-free pricing with unlimited supply, where the goods can be modeled by edges in a graph and buyers wish to purchase cheapest paths. More precisely, each buyer has equal positive valuations for paths connecting a certain pair of vertices and zero valuation for all the other sets of goods. Such a problem may be used to model a situation where the vendor is an owner of a road network and buyers are drivers wishing to travel from one city to another. Guruswami et al. [6] have defined and studied two subcases of this scenario: the tollbooth and highway problems. In the former the underlying graph is a tree and in the latter it is a path.

Envy-free pricing for revenue maximization is a computationally hard problem. Even the tollbooth and highway problems have been shown to be NP-hard. Thus, the main focus of works in this area is on approximation algorithms.

1.1 Our result

We consider the tollbooth problem on cactus graphs, a generalization of the tollbooth problem with unlimited supply. Instead of a tree, the underlying graph is a cactus, i.e. its every edge belongs to at most one simple cycle. Our contribution is a polynomial approximation algorithm, which is a generalization of a similar result by Iftah Gamzu and Danny Segev [4] for the classical tollbooth problem. The main difference between the two models is that, unlike in a tree, in a cactus there can be multiple simple paths connecting a single pair of vertices. Thus, each buyer can be interested in purchasing multiple sets of goods, i.e. is not single-minded.

Our algorithm achieves an approximation ratio on revenue of $\mathcal{O}\left(\frac{\log m}{\log \log m}\right)$, which matches the guarantee given by [4]. To the best of our knowledge, no algorithms giving sublogarithmic guarantees on revenue in generalizations of the tollbooth problem are known.

1.2 Related work

The problem of envy-free pricing for revenue maximization has been studied in various settings. Among the classes of buyers' valuations, which have been considered, we are going to survey mostly results for single-minded buyers, a model where each buyer has positive valuation for exactly one set of goods. Although the tollbooth problem on cactus does not fall into this category, this work is strongly related to the classical tollbooth problem, where buyers are single-minded.

Guruswami et al. [6] have defined the single-minded buyers setting and presented a polynomial $\mathcal{O}(\log m + \log n)$-approximation algorithm for the variant with unlimited supply. Also for the unlimited supply setting, Balcan, Blum and Mansour [1] have shown that a logarithmic guarantee on expected revenue can be achieved by

randomly setting a single price to all the goods. That result holds for buyers with arbitrary valuations.

In the unlimited supply setting with single-minded buyers, there are stronger results for cases with additional assumptions about buyers' valuations. For the tollbooth problem, Gamzu and Segev [4] have achieved an $\mathcal{O}\left(\frac{\log m}{\log\log m}\right)$-approximation of revenue with a polynomial algorithm. For the highway problem, Gradoni and Rothvoß [5] have designed a polynomial time approximation scheme. Those two problems have been used to show impossibility results for envy-free pricing with single-minded buyers. Guruswami et al. [6] have proven that the tollbooth problem is NP-hard. This was followed by a result from Briest and Krysta [2], who showed the same for the highway problem.

Of course, those impossibility results hold for limited supply as well. In that setting there also are several approximation results. Cheung and Swamy [3] have designed an $\mathcal{O}\left(\sqrt{m}\log u_{max}\right)$-approximation algorithm for the general envy-free pricing problem with single-minded buyers (u_{max} denotes the maximal number of copies of a single good). In the tollbooth and highway problems they have obtained approximation ratio of $\mathcal{O}\left(\log u_{max}\right)$.

1.3 Model and preliminaries

Let us consider an instance of tollbooth problem on cactus graphs with m goods and n buyers. Its description consists of a simple graph G with m edges such that no edge lies on two simple cycles and a set B of buyers. Each customer $i \in B$ is described by a pair of vertices in this graph, denoted u_i and v_i, and her budget $b_i > 0$. For each subset of edges S, her valuation is defined in the following way:

$$f_i(S) = \begin{cases} b_i, & \text{if } S \text{ consists of edges along a } u_i\text{-}v_i \text{ path} \\ 0, & \text{otherwise} \end{cases}$$

A solution to this problem is a real vector p assigning non-negative prices to the edges of the graph. Let us treat the prices as lengths of edges and let d_i denote the distance between v_i and u_i. If $b_i \geq d_i$ i-th buyer purchases all edges along a shortest path between v_i and u_i. Otherwise, she would buy nothing. Such an allocation is envy-free. Note that in the case of numerous shortest u_i-v_i paths, choosing either one does not change vendor's revenue.

In this work we present an algorithm for finding such prices that the above-mentioned way of allocating goods to buyers results in revenue $\mathcal{O}\left(\frac{\log m}{\log\log m}\right)$ times smaller than optimal. The following theorem formalizes our result.

Theorem 1. *There exists a polynomial time approximation algorithm for the tollbooth problem on cactus graphs with unlimited supply which achieves an approximation guarantee for revenue of* $\mathcal{O}\left(\frac{\log m}{\log\log m}\right)$.

1.4 Interesting techniques

In this section we describe the general idea behind the algorithm and highlight the most important techniques.

First, the buyers are split into $\mathcal{O}\left(\frac{\log m}{\log \log m}\right)$ subsets, which define separate instances of the problem. Each of the subproblems is processed independently. Its additional properties, which are ensured by the construction, allow the algorithm to calculate prices generating revenue within a constant factor of the optimal revenue. This is enough to obtain the desired bound on the global revenue, because the supply of goods is unlimited and, thus, the revenue of a solution to a subproblem does not decrease when applied to the initial instance with all the buyers.

For each subproblem the algorithm constructs a subgraph of G, called the skeleton, so that all paths desired by buyers in the given instance enter and leave the skeleton exactly once. This way, each such path is split into three parts, one of which is in the skeleton and the other two are not. Revenue generated by the former is controlled by setting appropriate prices of the skeleton's edges. The latter are processed in groups entering or leaving the skeleton through the same vertex.

The subcase of the tollbooth problem where all customers want to buy paths starting in a single vertex has been solved using dynamic programming in [6]. That solution, which is employed by the algorithm for the original tollbooth problem, is based on the uniqueness of paths in trees. Cactus graphs have a weaker property: all simple paths between a fixed pair of vertices pass through the same biconnected components. Thanks to this structure some dynamic programming algorithms, which were originally designed for trees, can, after several modifications, be applied to cactus graphs. Typically, such a dynamic programming would calculate partial results for each subtree, where the result for a subtree of a given vertex is a function of results for subtrees of its children. Intuitively, a cactus can be thought of as a tree, in which some vertices have expanded to simple cycles. Thus, a corresponding dynamic programming for a cactus would calculate partial results for the subtree of each biconnected component. Subtrees of individual edges would be very similar to the subtrees of vertices in a tree. Subtrees of simple cycles, however, would have to be processed differently since the transitions in the dynamic programming are usually defined for individual vertices with unique ancestors. In some applications, where the solutions are based only on the shortest paths to the root, one edge in a cycle would be unused. By guessing this edge and removing it from the graph, the cycle can be transformed to a path, which can be processed using the original algorithm. Thanks to the tree-like structure of biconnected components of a cactus, it is often enough to guess the unused edge in only one cycle at a time. This allows to preserve the polynomial running time. We believe that this technique has a wide range of applications in generalizing algorithms for a tree to a cactus. Its specific usage for the tollbooth problem is described in Section 2.2.1.

The skeleton is split into several parts with simple structure. Buyers forming each subproblem are chosen in such a way, that each of them wishes to buy paths with both endpoints in the same part. First, near-optimal solutions are found for each part individually. Then, the global solution is constructed by merging them. In the first phase, for each part, the algorithm iterates over many possible ways of pricing its edges. In the second, the algorithm uses the tree-like structure of the graph to resolve dependencies between particular parts.

Those dependencies originate from another fundamental difference between cactus graphs and trees. A connected subgraph of a cactus does not necessarily contain all paths between its vertices. Thus, even if a buyer wishes to buy a path starting and ending in a particular connected subgraph, she may choose to purchase a path not contained inside it. In order to avoid this, cycles of the cactus must not be split into multiple parts. However, splitting them is vital to maintaining the polynomial running time. Thus, those additional paths also need to be considered when processing an individual part of the skeleton. The structure of a cactus guarantees them to be disjoint, so each such path can be treated as a single edge. This allows the algorithm to handle both phases efficiently.

Another technique used during both stages of processing the skeleton is rounding the prices. The main idea is that because revenue with the prices two times lower is at least half of the original revenue, considering prices in form $c \cdot 2^k$ for a certain constant c is enough to obtain a constant factor approximation of revenue. This bounds the number of possibilities the algorithm processes in the first phase. Furthermore, a similar observation allowed to efficiently implement the second phase. The main usage of this technique by our algorithm is presented in Lemma 5.1.

Chapter 2

Graph decompsition

Our algorithm employs a recursive decomposition of the cactus. At each step a subset of buyers is chosen based on the partition of the graph G. For each constructed subproblem prices achieving a constant factor approximation of optimal revenue can be found in polynomial time. On the next level of decomposition each fragment of the graph is split into smaller parts. This process is repeated until the subproblem becomes trivial.

Let L be the number of levels in the decomposition process. Each buyer will be processed at exactly one of them. Since optimal revenue is a subadditive function of subsets of buyers, the optimal revenue in the whole problem does not exceed the sum of optimal revenues for disjoint subsets of buyers. Thus, revenue obtained in one of the subproblems will approximate the optimal one with a ratio of at most $\mathcal{O}(L)$. This chapter defines a decomposition with bounded number of levels. This process, along with classification of buyers, creates subproblems solved by the algorithm for a single partition, which is described in Chapters 3, 4 and 5.

2.1 Tree of biconnected components

The algorithm relies on the structure of biconnected components of the cactus. An arbitrarily chosen vertex of graph G will be fixed as its root for the duration of the whole algorithm. A vertex or edge is said to be above another one, when it is closer to the root. Note that every biconnected component of a cactus is either a single edge or a simple cycle. Thus, it has a single topmost vertex and at most two topmost edges, which are exactly those adjacent to the topmost vertex.

Definition 2.1. *For each cycle in G, its two edges closest to the root of G, i.e. topmost edges, form a* ***pair of associated edges****. Note that every edge belongs to at most one such pair.*

Although each vertex can be a topmost vertex in arbitrarily many biconnected components, it can belong to at most one without being its topmost vertex. Furthermore, all vertices except for the root belong to exactly one such biconnected component. Let us call it the main component of this vertex. For the root it is a special component, consisting only of itself (a single vertex). Our algorithm uses the tree of biconnected components rooted in this special component. Every other component is a child of the main component of its topmost vertex. Note that such a tree is unique.

Definition 2.2. *A **subtree graph of a component** C is a graph consisting of all the edges and vertices belonging to any descendant of C (inclusive) in the tree of biconnected components. The subtree graph of the root component is the whole graph G.*

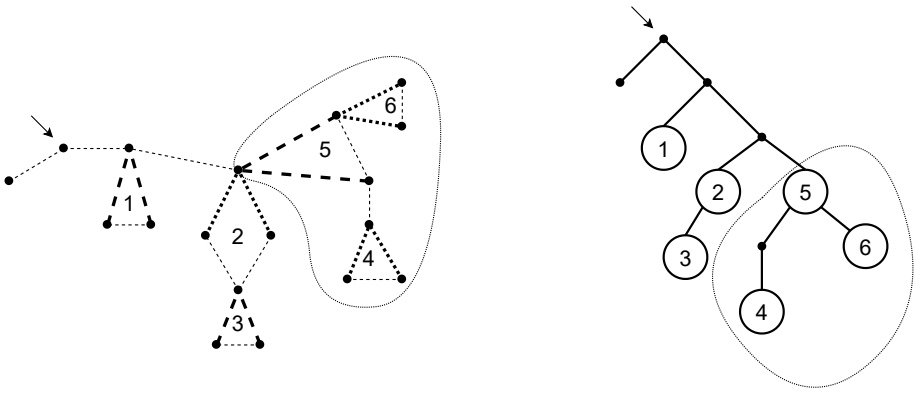


Figure 2.1: An example cactus graph with marked pairs of associated edges and its tree of biconnected components. The arrows indicate respectively the root vertex and the root component. On both drawings the subtree graph of the cycle number 5 is marked out.

2.2 Balanced decomposition

Decompositions $D_1, D_2, \ldots D_L$ of G are defined recursively. Each of them is a family of edge-disjoint subgraphs, called fragments, which cover the graph G. In D_1 the whole graph G forms a single fragment, and in D_L each fragment consists of at most two edges. For all $i < L$ each fragment in the partition D_i is split into a number of subgraphs, which become fragments in D_{i+1}.

Definition 2.3. *A vertex which belongs to multiple fragments in partition D_{i+1} is called a **border vertex** of i-th level. Furthermore, every vertex of G is considered to be a border vertex of L-th level.*

It follows from the recursive structure of decomposition, that a border vertex of i-th level is also a border vertex of $(i+1)$-th level.

Lemma 2.1. *Consider a family of decompositions $D_1, D_2, \ldots D_L$ of a graph G satisfying the following invariants for each valid i:*

1. *Each fragment in D_i is split into $\mathcal{O}(k)$ fragments in D_{i+1}.*
2. *The maximal number of edges in a fragment forming D_{i+2} is $\Omega(k)$ times smaller than in D_i.*
3. *Each fragment forming D_i contains at most $\mathcal{O}(k)$ border vertices of i-th level.*
4. *Each pair of associated edges belongs to the same fragment of D_i.*
5. *All fragments forming D_i are connected subgraphs of G.*

If G is a cactus graph, such a family can be found for any positive k in polynomial time.

The second invariant ensures that the number of levels will be bounded by $\mathcal{O}(\log_k m)$. By fixing $k = \left\lceil \log^{\frac{1}{2}} m \right\rceil$ we achieve a $\mathcal{O}\left(\frac{\log m}{\log \log m}\right)$ bound on L.

There are two procedures for obtaining D_{i+1} from D_i. By applying them interchangeably the decompositions from Lemma 2.1 are created.

2.2.1 Reducing the number of edges

We devise a procedure for splitting each fragment in such a way that the size of subparts in bounded. As all fragments are treated in the same way, we describe an algorithm which for one of them, denoted $F = \langle V, E \rangle$, finds a decomposition satisfying the following conditions:

- The number of subparts is between $\left\lfloor \frac{k}{4} \right\rfloor$ and k.
- The number of edges in each subpart is at most $4 \cdot \left\lceil \frac{|E|}{k} \right\rceil$.
- Associated edges belong to the same subpart.

The main idea is to partition F into connected subgraphs with the number of edges between $\left\lceil \frac{|E|}{k} \right\rceil$ and $4 \cdot \left\lceil \frac{|E|}{k} \right\rceil$. We allow a single exception, one of the subparts may have fewer edges. From this the bound on the number of subparts follows immediately.

All cycles in F contain a pair of associated of edges. By erasing one of them from each cycle, we can obtain a tree. Let us call this tree T. The algorithm first greedily

partitions the edges of T. Then adds edges from $F \setminus T$ into respective subparts, so that the fourth invariant from Lemma 2.1 is preserved.

The algorithm for partitioning the tree also needs to deal with pairs of associated edges, which may still be present in T. This is because the cycles of G, on which the associated edges lie, do not have to be fully contained in F so it is possible that a pair of associated edges is not a part of any cycle in F. Hence, both its edges may belong to T. Let us root T in one of its topmost vertices (there can be two of them). This way every pair of associated edges present in T shares their upper endpoint. Using this property the algorithm makes sure they end up in the same subpart.

The algorithm for partitioning T maintains two kinds of subparts: open, to which edges of T can still be added and closed, to which they cannot. This approach yields the following procedure for partitioning edges from F:

1. From each pair of associated edges which is lying on a cycle in F erase an arbitrarily chosen edge. Let us denote the resulting tree by T.
2. Root T in one of its topmost vertices and process every vertex v in the order of non-increasing depths:
   (a) For every child u of v in T: If there is an open subpart containing u, add the (u, v) edge to it. Otherwise, initialize a new open subpart consisting only of (u, v).
   (b) Merge all pairs of subparts from the previous step containing edges associated with each other.
   (c) Mark each open subpart as closed if it contains at least $\left\lceil \frac{|E|}{k} \right\rceil$ edges.
   (d) As long as there are at least two open subparts, merge them. If their size reached $\left\lceil \frac{|E|}{k} \right\rceil$, mark the resulting subpart as closed.
3. Add each edge from $F \setminus T$ to the subpart containing the one associated with it.

After v has been processed in the second step there is at most one open subpart in the subtree of v and it contains strictly less than $\left\lceil \frac{|E|}{k} \right\rceil$ edges. Hence, open subparts created in the step 2a contain at most $\left\lceil \frac{|E|}{k} \right\rceil$ of them. This results in the size of closed subparts created in the second step being between $\left\lceil \frac{|E|}{k} \right\rceil$ and $2\left\lceil \frac{|E|}{k} \right\rceil$. When edges from $F \setminus T$ are added, the subparts created so far can at most double their size because each edge is associated with at most one other edge. Thus, each subpart can contain at most $4\left\lceil \frac{|E|}{k} \right\rceil$ edges. Furthermore, after the algorithm concludes, there may be only one subpart with fewer than $\left\lceil \frac{|E|}{k} \right\rceil$ edges.

2.2.2 Reducing the number of border vertices

Like previously, we restrict our attention to a single fragment F on a particular level of decomposition, denoted i. We assume that F contains at most $18k$ border

vertices from $(i-1)$-th level and define a procedure for splitting it into at most three connected subparts in such a way that:

- Each subpart contains at most $14k$ border vertices of i-th level.
- F contains at most $20k$ border vertices of i-th level.
- Edges associated with each other are in the same subpart.

Let b be the number of border vertices of $(i-1)$-th level in F. First, F is rooted in one of its vertices and a local tree of its biconnected components is created. Although the tree is constructed locally (as if F was the whole graph G), pairs of associated edges are still defined globally (Definition 2.1 still refers to the global root of G). Then, the lowest component containing more than $\lfloor \frac{b}{2} \rfloor$ border vertices in its subtree graph is found in the local tree. Since the subtree of the root component is the whole fragment with b border vertices, such a connected component is guaranteed to exist. We will call it the pivot component. If it is not the root component, it can be either a single edge or a cycle. If it is, we handle it as in the case of an edge, where the root vertex is considered to be the lower vertex.

The case of a single edge: Consider the connected components of F resulting from removing the lower vertex v of the pivot component. Each such subgraph has exactly one edge adjacent to v and contains at most $\lceil \frac{b}{2} \rceil + 1$ border vertices on $(i-1)$-level (we include v in all these subgraphs). This partition can, however, split pairs of associated edges. Let edges (v, u_1) and (v, u_2) be associated and belong to distinct subparts. If the number of border vertices of level $i-1$ contained in the union of those two subparts is at most $\lceil \frac{b}{2} \rceil$, the two subparts are merged. Otherwise, edge (v, u_1) is transferred to the subpart of (v, u_2). In this case the subpart of u_2 gained only one additional vertex and the subpart of u_1 is still connected, as the edge (v, u_1) must have been its only edge adjacent to v.

In the partition defined this way every subpart (fragment on $(i+1)$-th level of decomposition) contains at most $\lceil \frac{b}{2} \rceil + 2$ border vertices of $(i-1)$-th level. Furthermore, at most two new border vertices are created in F on i-th level (one is v and the other one results from transferring the associated edge to another subpart, which can happen only once).

However, such a partition may result in too many fragments on $(i+1)$-th level contained in F. Thus, as long as there are two subparts whose union contains no more than $\lceil \frac{b}{2} \rceil + 2$ border vertices of $(i-1)$-level they are merged. Afterwords, there can be at most three subparts in the partition.

The case of a cycle: Let us denote the cycle being the pivot component as C. Consider connected components of F resulting from removing all the edges of C.

If no such component has more than $\lfloor \frac{b}{2} \rfloor$ border vertices on the $(i-1)$-level, we can choose a number of consecutive vertices on C such that their connected components contain between $\lfloor \frac{b}{3} \rfloor$ and $\lceil \frac{2b}{3} \rceil$ border vertices of $(i-1)$-level in total. Let us paint all the vertices in their connected components black and the rest of the vertices in F white. All the edges between pairs of two black and two white vertices are painted respectively black and white. Note that both the black and white subgraphs are connected. All white edges will form the first subpart and black ones the second. The edges between vertices of different colors can be assigned to either one. It is possible not to split pairs of associated edges since they share a vertex and thus cannot be both painted and have distinct colors.

Each subpart contains at most $\lceil \frac{2b}{3} \rceil$ border vertices of level $(i-1)$. The new border vertices in F on i-th level can be created only by the edges connecting vertices with distinct colors. From the construction it follows that there are exactly two such edges and hence at most two new border vertices in F on i-th level.

If one of the connected components resulting from removing edges of C contains at least $\lfloor \frac{b}{2} \rfloor + 1$ border vertices of the $(i-1)$-level, we implement a different approach. Let v be the vertex from C belonging to this component. First, we add all edges from C back to the graph, then remove v from it. Note that each resulting connected component contains at most $\lceil \frac{b}{2} \rceil$ vertices. Thus, we can use the previously described procedure for the case of a single edge, where v is treated like the lower vertex.

Regardless of whether the pivot component is a cycle or a single edge, the above algorithm results in at most two new border vertices in F on i-th level of decomposition. Furthermore, each subpart contains at most $\lceil \frac{2b}{3} \rceil + 2$ border vertices of i-th level. With $b \leq 18k$ this gives an upper bound of $14k$ on their number in a single subpart of F.

2.2.3 Recursive decomposition

Using subprocedures from Sections 2.2.1 and 2.2.2 we can define the recursive decomposition. Each fragment of D_i is split into subparts. For odd i the procedure of reducing the number of edges is used. For even i reducing the number of border vertices is applied. Then, the subparts become fragments of D_{i+1} and the process continues until all fragments consist of at most two edges.

Recall that both kinds of steps create $\mathcal{O}(k)$ connected subparts from each fragment and do not split associated edges. Reducing the number of edges every second step of the decomposition guarantees that the second invariant holds too. The following lemmas imply that our construction satisfies the third invariant and thus conclude the proof of Lemma 2.1.

Lemma 2.2. *Consider a fragment F split into s edge-disjoint connected subparts in such a way, that the topmost edges of each biconnected component belong to the same subpart. Then, at most $2s-2$ vertices belong to more than one subpart.*

Proof. Consider a cycle C in F, whose edges belong to multiple subparts. Since the topmost edges of C must belong to the same subpart, at least one of them does not contain any topmost edge. Let us denote it by S. In such a case, the topmost vertex of C does not belong to S, so S must be fully contained in the subtree graph of C. Thus, for fixed S there can be only one such cycle C. Furthermore, if S contains a topmost vertex of F, no such cycle exists. Hence, there are at most $s-1$ cycles whose edges belong to multiple subparts.

Let us transform F into a tree T by removing one of the topmost edges from each cycle. Consider a cycle C and let S be the subpart containing its topmost edges. If all edges in C belong to S, this subpart is still connected. Otherwise, let v be the topmost vertex and u be the other endpoint of the erased edge. If u is incident to another edge of S, S no longer is connected. Let us denote the two connected components of S as S_1 and S_2. This can happen in only $q \leq s-1$ cycles. Otherwise, the remaining edges of S still form a connected subgraph.

In the tree T there are $s+q$ connected subparts, so there are $s+q-1$ vertices shared by them. Let us check how this changes when a previously removed (u,v) edge is added to the cycle C. If all edges in C belong to the same subpart (S), no vertex is added to S. Let us consider the opposite case. If u has been incident to another edge in S, the subparts S_1 and S_2 are in fact a single subpart S, which contained u and v even before adding the (u,v) edge. Otherwise, u is added to S and shared with the subparts it previously belonged to. This can happen at most $s-1-q$ times.

Summarizing, at most $2s-2$ vertices of F belong to multiple subparts contained in F. □

Lemma 2.3. *For all i each fragment of D_i contains at most $20k$ border vertices of i-th level. Among them, at most $18k$ are border vertices of the previous level.*

Proof. We show the above lemma using induction over even levels of decomposition. In D_3 there are at most $3k$ fragments in total, so there are only $6k-2$ border vertices on the second level of decomposition and the lemma holds for $i \leq 2$.

Let us consider a decomposition D_i for even i, for which Lemma 2.3 holds. Since D_{i+1} is created using the procedure from Section 2.2.2, each fragment in D_{i+1} contains at most $14k$ border vertices on i-th level. Because the odd step splits each fragment into at most k subparts, by Lemma 2.2 there are at most $16k-2$ border vertices on level $(i+1)$ in each fragment of D_{i+1}.

Naturally, the same bound holds for fragments in D_{i+2}. Since each fragment in D_{i+2} is split into at most three fragments of D_{i+3}, it can contain at most $16k + 2$ border vertices of $(i + 2)$-th level. Thus, Lemma 2.3 holds for D_{i+1} and D_{i+2}, which concludes the proof of the inductive step. □

2.3 Classification of buyers

A pair of vertices u and v is said to be connected in a decomposition D_i if there exists a path from u to v fully contained in a single fragment from D_i. The buyer represented by vertices u, v and budget b will be processed at the last level, where his vertices are connected. This criterion unambiguously assigns every buyer to a single decomposition.

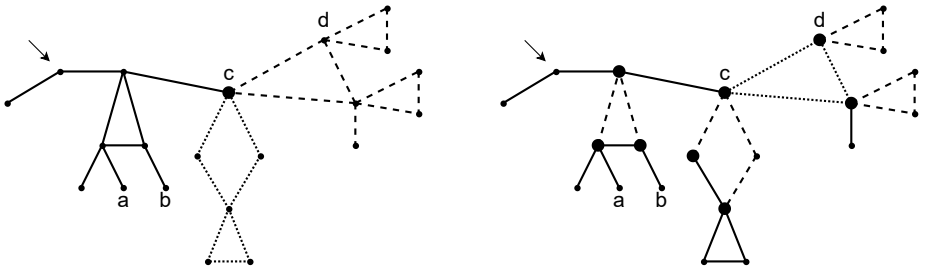


Figure 2.2: Two levels of a recursive decomposition, which satisfy Lemma 2.1, but do not necessarily result from the construction described in Section 2.2. Fragments from levels j and $j + 1$ are marked respectively on the left and right hand side. Highlighted vertices are shared between the fragments and the ones on the right hand side are border vertices on j-th [sic] level. A buyer wishing to purchase an a-b path would not be processed on j-th level, but a buyer interested in a-c paths would. The pair of vertices c and d is connected in at all levels of decomposition, so corresponding customers would be assigned to its last level.

Remark 2.1. *Consider a buyer wishing to purchase a u-v path assigned to i-th level of the recursive decomposition. Then, every u-v path in the whole graph contains a border vertex on the i-th level.*

Proof. Since for D_L each vertex is a border vertex, we consider D_i for $i < l$. In such a case, no u-v path can be contained inside a single subpart (fragment of D_{i+1}). Otherwise, the buyer would be assigned to a lower level of decomposition. Since every u-v path contains edges from distinct fragments of D_{i+1}, one of its vertices must be shared between them and be a border vertex on i-th level. □

Chapter 3

Algorithm for a single decomposition

In the previous chapter buyers have been divided into subsets by a recursive graph decomposition. Now we focus on a single (j-th) level of decomposition. By exploiting its properties we prove the following lemma:

Lemma 3.1. *Let B_j be the set of buyers assigned to D_j from Lemma 2.1. There exists a polynomial time algorithm for the instance of the tollbooth problem on cactus graphs defined by buyers from B_j which gives a constant factor approximation guarantee on revenue.*

Recursive decomposition of the graph was used to create subproblems by partitioning the buyers. The main idea behind the algorithm for a single decomposition is to split the paths desired by buyers into smaller sections and handle them separately. In this chapter we define a partitioning of those paths.

3.1 The skeleton

Definition 3.1. ***Skeleton on j-th level****, denoted SK_j, is a minimal subgraph of G containing all simple paths between border vertices of j-th level. Equivalently, an edge belongs to the skeleton, i.e. is a* ***skeleton edge****, if and only if a simple path connecting two border vertices passes through it. A vertex adjacent to a skeleton edge is called a* ***skeleton vertex****.*

Note that, by the definition of a border vertex, SK_{j+1} is always a superset of SK_j and $\mathrm{SK}_L = G$.

Definition 3.2. *A* ***non-skeleton component on j-th level*** *is a maximal connected subgraph of a fragment from D_{j+1} containing no edges from SK_j.*

Lemma 3.2. *Every simple path connecting two skeleton vertices passes only though skeleton edges.*

Proof. We assume that there are at least two border vertices of j-th level. Otherwise, the lemma is trivially true. The proof is by contradiction. Let us say there is a simple path between two skeleton vertices passing though a number of non-skeleton edges. Consider a maximal part of this path consisting only of non-skeleton edges. It must start and end at skeleton vertices, let us denote them by u and v. From the construction of SK_j it follows that there exists another u-v path fully contained in the skeleton. Those two paths are of course disjoint, so u and v lie on a simple cycle. Consider an arbitrary skeleton edge on this simple cycle and a path between two border vertices, s and t, which passes through it. Let u' be such vertex on this s-t path that lies on the cycle and is the closest one to u. Similarly, let v' be the one closest to v. Without loss of generality we assume that u' appears before v' on the s-t path. Consider the following path:

- From s to u' along the original s-t path.
- From u' to u along the skeleton u-v path.
- From u to v along the non-skeleton u-v path.
- From v to v' along the skeleton u-v path.
- From v' to t along the original s-t path.

It follows from the choice of u' and v' that the above path has no cycles (is simple). This leads to contradiction and concludes the proof. □

Corollary 3.2.1. *Each non-skeleton component contains exactly one skeleton vertex.*

Definition 3.3. *Let us define a* ***skeleton representative*** *of a vertex v on j-th level denoted by* $\mathrm{repr}_j(v)$*. If v is a skeleton vertex in D_j, then* $\mathrm{repr}_j(v) = v$*. Otherwise, the representative of v is the unique skeleton vertex in the non-skeleton component on the j-th level containing v.*

Consider a buyer i from B_j wishing to buy the cheapest u_i-v_i path. Recall from Chapter 2 that each u_i-v_i path contains at least one border vertex of j-th level. By Corollary 3.2.1 each path from a vertex s to any skeleton vertex passes through $\mathrm{repr}_j(s)$. Hence, each path from u_i to v_i contains vertices u_i, $\mathrm{repr}_j(u_i)$, $\mathrm{repr}_j(v_i)$, v_i. Although some of those four vertices may be equal, they are guaranteed to appear in this order. This allows us to split every such path into three parts (some of which may be empty):

- First **non-skeleton section** – a simple path from u_i to $\mathrm{repr}_j(u_i)$, which contains no skeleton edges.

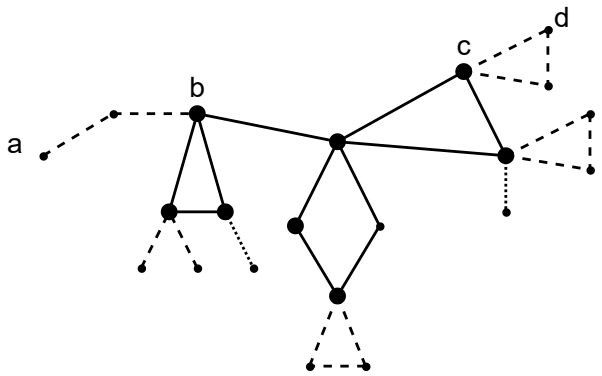


Figure 3.1: A cactus from Figure 2.3 with highlighted border vertices and the skeleton on j-th level. Each connected group of edges dotted in a same style forms a single non-skeleton component. Each a-d path is split into a skeleton section (a b-c path) and two non-skeleton sections: a-b and c-d paths.

- A **skeleton section** – a simple path from $\text{repr}_j(u_i)$ to $\text{repr}_j(v_i)$. By Lemma 3.2, it passes only through skeleton edges.
- Second **non-skeleton section** – a simple path from $\text{repr}_j(v_i)$ to v_i. Similarly to the first one, it does not contain skeleton edges.

3.2 The skeleton and non-skeleton edges

For each level of decomposition $j < L$ the algorithm handles two subproblems. Since $\text{SK}_L = G$, on the last level only the skeleton subproblem is processed.

The skeleton subproblem: Consider a buyer $i \in B_j$ wishing to purchase the cheapest u_i-v_i path. In this subproblem they will buy the cheapest path between $\text{repr}_j(u_i)$ and $\text{repr}_j(v_i)$ as long as its cost does not exceed b_i (her original budget). Such a situation may be achieved by setting the price of all non-skeleton edges to zero.

The non-skeleton subproblem: In this case, we set the prices of all the skeleton edges to zero. Each buyer $i \in B_j$ will purchase the cheapest paths from u_i to $\text{repr}_j(u_i)$ and from v_i to $\text{repr}_j(v_i)$ if their total cost does not exceed b_i.

Let us introduce additional notation:

- Let OPT_j be the maximal revenue obtained by any price vector and envy-free assignment of paths to the buyers from B_j.
- Let SKOPT_j and NSKOPT_j be the maximal revenues for the skeleton and non-skeleton subproblem respectively.

Note that any envy-free solution for the whole graph immediately yields envy-free solutions for both subproblems. Thus, $\mathrm{SKOPT}_j + \mathrm{NSKOPT}_j \geq \mathrm{OPT}_j$.

The algorithm solves both subproblems independently. Then, the computed solutions are compared and the one with greater revenue is chosen. Chapters 5 and 4 describe polynomial time approximation algorithms for the skeleton and non-skeleton subproblem respectively.

Chapter 4

Non-skeleton edges

This chapter describes an algorithm for solving the non-skeleton subproblem on j-th level. Let B_j be the set of buyers assigned to this level. The prices of all edges in the skeleton on j-th level are set to zero. The focus of this chapter is on pricing the non-skeleton edges and maximizing revenue generated by non-skeleton sections of paths allocated to buyers from B_j. Recall that on the last level of decomposition the skeleton contains the whole graph G. Thus, we assume that $j < L$. Let the optimal revenue for the given instance of the non-skeleton subproblem be denoted by NSKOPT_j. The algorithm presented in this chapter finds prices generating at least $\frac{\mathrm{NSKOPT}_j}{4}$ revenue.

4.1 The rooted case

Before describing the method for pricing non-skeleton edges, let us discuss an easier problem, whose solution is a subprocedure used by the final algorithm.

Definition 4.1. *Consider an instance of the tollbooth problem on cactus graphs defined by a cactus H and a set of buyers B_H. We will say that it is a **rooted instance** if there exists a vertex in H, called root, which is an endpoint of every path desired by the buyers.*

Definition 4.2. *Consider a buyer $i \in B_H$ in a rooted instance, who wishes to purchase a cheapest u_i-v_i path. Her **destination vertex** is the one of vertices u_i or v_i that is not the root.*

Lemma 4.1. *Any rooted instance of the tollbooth problem on cactus graphs can be solved in polynomial time. It is also true if we admit only those price assignments, under which the distances from the root to some vertices are equal to arbitrarily fixed constants.*

Regardless of the prices, in every envy-free allocation each buyer is assigned a shortest path from her destination vertex to the root as long as its cost does not

exceed her budget. Thus, presenting a polynomial algorithm for finding optimal prices is sufficient to prove the above lemma.

The algorithm is based on dynamic programming, whose subproblems mimic the structure of the tree of biconnected components of H rooted in r – the root from Definition 4.1. For each biconnected component C it calculates values $dp_{C,d}$, which are defined in the following way: Let us assume that the distance, i.e. cost of a cheapest path, from r to C (its depth) equals d. Then $dp_{C,d}$ equals the maximum revenue generated by buyers whose destination vertices belong to the subtree graph of C (excluding its topmost vertex). Note that the distance from C to the root is in fact the distance between r and the topmost vertex of C. The following lemma allows our algorithm to consider only polynomially many values d.

Lemma 4.2. *For any rooted instance of the tollbooth problem on cactus graphs there exists an optimal solution, such that the distance from each vertex to the root belongs to the set D containing zero and buyers' budgets:*

$$\mathcal{D} = \{0\} \cup \{b_i \mid i \in B_H\}$$

Proof. Let us call a price assignment regular if it satisfies the condition from the above lemma. Consider optimal prices that are not regular. We prove the lemma by showing the existence of a regular price assignment which generates at least as much revenue.

By T let us denote a shortest-path tree of H rooted in r, which is an acyclic subgraph of H containing a shortest path from every vertex to the root. Let us increase the price of all edges in $H \setminus T$ to the maximal budget $b_{max} = \max B_H$. While it does not change the distances from vertices to the root, it guarantees that each vertex has a shortest path to the root contained in T under any regular prices.

Let us consider a vertex v and let d_v be the distance from v to r under initial prices. We set d'_v to the smallest element of $\mathcal{D}$ greater or equal than d_v or to b_{max} if no such element exists. Note that if for any two vertices u and v $d_u \geq d_v$, then $d'_u \geq d'_v$. Hence, the values of d'_v are non-decreasing on the paths from r to every leaf in T. Thus, there exist prices for edges in T such that the distance from r to v equals d'_v for every v.

Consider a vertex v and customers whose destination vertex is v. If $d'_v > d_v$, none of them has a budget from $[d_v, d'_v)$. Hence, they now generate no less revenue than under the original prices. If $d'_v < d_v$, then $d_v > b_{max}$, so in this case no buyer could have been allocated a v-r path under the original prices. Thus, the regular prices result in at least as much revenue as optimal ones. □

In Chapter 5 the algorithm needs to find optimal prices given constraints on the distance from certain vertices to the root. The following corollary allows for handling such cases.

Corollary 4.2.1. *Consider a rooted instance of the tollbooth problem on cactus graphs and a subset S of vertices of H such that for each $v \in S$ required depth e_v of this vertex is given. Prices of edges in H are said to be feasible if the cost of a cheapest r-v path equals e_v for each $v \in S$. Let us assume, that there is at least one such price assignment. Then, there exists a feasible price assignment maximizing revenue for which the distance from r to each vertex $v \notin S$ belongs to the set $\mathcal{D}'$:*

$$\mathcal{D}' = \{0\} \cup \{b_i \mid i \in B_H\} \cup \{e_i \mid i \in S\}$$

The proof of this corollary follows immediately from the proof of Lemma 4.2. Note that all buyers with destination vertices in S generate the same revenue under all feasible price assignments. The argument for vertices outside S remains the same.

4.1.1 Dynamic programming

The input of the procedure consists of a graph H, buyers B_H and a set of constraints S (possibly empty) from Corollary 4.2.1. For each vertex v we define a set of its possible depths $\mathcal{D}_v$ in the following way:

$$\mathcal{D}_v = \begin{cases} \{e_v\}, & v \in S \\ \{0\} \cup \{b_i \mid i \in B_H\} \cup \{e_u \mid u \in S\}, & v \notin S \end{cases}$$

By Corollary 4.2.1, in order to find optimal prices which satisfy the constraints it is sufficient to consider only such price assignments that the costs of a cheapest r-v path belongs to $\mathcal{D}_v$ for each $v \in H$. Thus, for each biconnected component C the algorithm calculates the values of $dp_{C,d}$ for every $d \in \mathcal{D}_v$ where v is the topmost vertex of C. It is possible that some values of d inevitably lead to violation of the constraints on depths of vertices from S. In such a case we set $dp_{C,d} = -\infty$. For simplicity, we also assume that $dp_{C,d} = -\infty$ for each $d \notin \mathcal{D}_v$.

The biconnected components of H are processed bottom up based on the structure of the tree of biconnected components. The algorithm handles biconnected components differently depending on whether they consist of a single edge or a cycle. The root component R, which is the root of the tree of biconnected components, contains the whole H in its subtree graph and is treated in yet another way. Let us introduce useful notation:

- $cnt_{v,x}$ – the number of buyers, whose budgets are at least x and whose destination vertex is v.
- $\mathcal{C}_v$ – the set of all biconnected components whose topmost vertex is v.

The case of a single edge: Let us denote the lower vertex of the considered biconnected component C as v and the upper as u. Note that the subtree graph of

C consists of the (u, v) edge and subtree graphs of biconnected components from $\mathcal{C}_v$, which are edge disjoint and share only the topmost vertex. All simple paths to the root from vertices contained in the latter pass though v. Furthermore, each simple path from v to the root must contain u. Basing on those observations, the algorithm calculates $dp_{C,d}$ for each $d \in \mathcal{D}_u$ according to the following formula:

$$dp_{C,d} = \max_{d' \in \mathcal{D}_v \ ; \ d' \geq d} \left(cnt_{d',v} \cdot d' + \sum_{C' \in \mathcal{C}_v} dp_{C',d'} \right)$$

The optimal value of d' is stored along with $dp_{C,d}$ in order to find the prices after calculating optimal revenue ($(d' - d)$ is the price of the (u, v) edge).

The case of the root component: The root must have depth 0, hence for this component only a single value ($dp_{R,0}$) is calculated. It is equal to the optimal revenue satisfying the constraints on depths of vertices in S. The algorithm calculates $dp_{R,0}$ as follows:

$$dp_{R,0} = \sum_{C' \in \mathcal{C}_r} dp_{C',0}$$

The case of a cycle: Let us denote the considered cycle by C, its topmost vertex as v and its subtree graph by G_C. G_C consists of C itself and the subtree graphs of components from $\mathcal{C}_u$ for all $u \in C \setminus \{v\}$.

Let us examine how different prices of edges in C affect the shortest paths from vertices of G_C to the root. First of all, they all pass through v. Next, consider a vertex s belonging to a subtree component of $C' \in \mathcal{C}_u$ for $u \in C \setminus \{v\}$. It follows from the structure of the cactus, that any simple s-v path passes though u. Prices assigned to edges of C only influence the shortest u-v paths, which are always fully contained in C. Thus, prices of edges from $G_C \setminus C$ do not influence the dephts of vertices on C. However, depths of those vertices influence the optimal prices of edges in $G_C \setminus C$.

Consider any prices assigned to the edges in C. Let T be a shortest-path tree of C rooted in v. Exactly one edge from C does not belong to T, we will say that it is unused. After removing this edge, the cost of a cheapest path from any vertex in G_C to v does not change. Thus, its price can be set to $b_{max} + 1$ without impacting the revenue ($b_{max} = \max\{b_i \mid i \in B_H\}$).

The algorithm iterates over all edges in C fixing the current one, denoted e, to be the unused edge. In this step the algorithm finds an optimal solution among those price assignments which result in e being an unused edge in C. First, the price of e is set to $b_{max} + 1$. This effectively removes e from the graph, as no buyer will ever purchase a path containing it.

Let us consider the graph G_C without e for a moment. The other edges of C do not form a single biconnected component anymore. Instead, $C \setminus \{e\}$ is a path

and each one of its edges is a biconnected component on its own. Furthermore, $G_C \setminus \{e\}$ is formed by this path and subtree graphs belonging to components from $\mathcal{C}_u$ for $u \in C \setminus \{v\}$. Note that the optimal solutions for the latter have already been calculated for every valid depth of their topmost vertices.

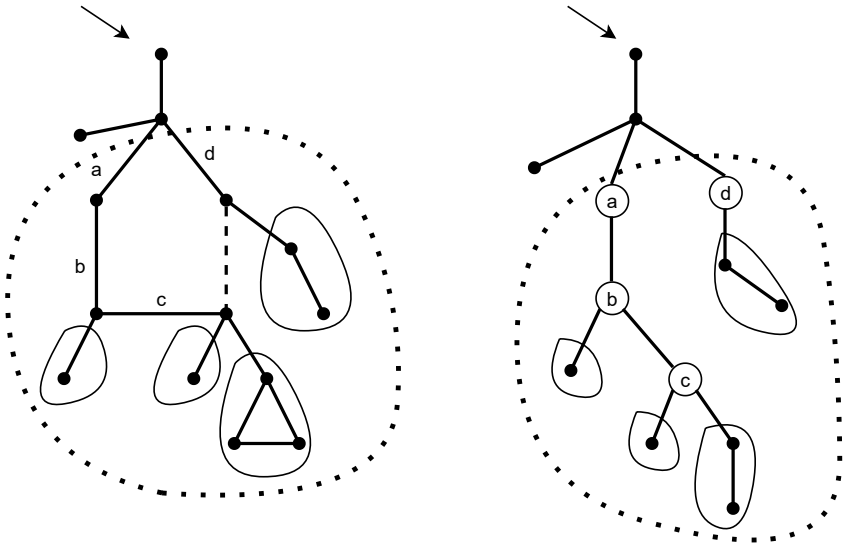


Figure 4.1: Processing a cycle for a fixed unused edge. The dotted line marks out its subtree graph on both the cactus graph (left) and the temporary tree of biconnected components (right). Solid lines enclose subproblems which already have been solved. The algorithm calculates the values of dp^e for biconnected components c, b, a and d (in this order).

Erasing e from the graph is also reflected in the tree of biconnected components. The single biconnected component of C is replaced by multiple components, each one formed by an individual edge in $C \setminus \{e\}$. However, all the children of C in the original tree remain the same after removing e. Each one belonging to $\mathcal{C}_u$ for $u \in C \setminus \{v\}$ is a child of the unique edge for which u is the lower vertex. Note that the already calculated values of dp for those components are still valid in the new tree.

Let C' be a biconnected component formed by an edge in $C \setminus \{e\}$. We define the values of $dp^e_{C',d}$ the same way as the original dp, but based on the temporary tree of biconnected components. The algorithm processes these new biconnected components in the bottom-up order from the temporary tree and calculates dp^e for each of them. Let C' be the current biconnected component formed by an s-t edge and let t be the topmost vertex. For each value of $d \in D_t$ the algorithm must calculate $dp^e_{C',d}$. It is of course equal to the revenue generated by buyers with destination somewhere in the subtree graph of a component whose topmost vertex is s. Such biconnected components are children of C' in the tree. Each of them either is an edge from $C \setminus \{e\}$ or it must have been a child of C in the original tree of biconnected components. Either way, the values of dp^e or dp have already been calculated for them for each valid depth of s. Hence, the problem of calculating

$dp^e_{C',d}$ is the same as calculating $dp_{C'',d}$ for any biconnected component C'' in H formed by a single edge.

Let C_1 and C_2 be the biconnected components formed by edges of C which are adjacent to v. If e, the unused edge, happens to be adjacent to v, there is, in fact, only one such component. In this case C_2 is just a placeholder with an empty subtree graph and $dp^e C_2, d$ equal zero for all $d \in \mathbb{R}$. Note that the union of subtree graphs of C_1 and C_2 in the temporary tree contains the same vertices and edges (except for e) as G_C. Furthermore, the two subtree graphs can only share one vertex: v. Thus, if we admit only such solutions, where e is unused, then $dp_{C,d} = dp^e_{C_1,d} + dp^e_{C_2,d}$. Since for every possible price assignment there exists an optimal allocation, where one edge of C is unused, it is enough to iterate over all possible edges $e \in C$:

$$dp_{C,d} = \max_{e \in C} \left(dp^e_{C_1,d} + dp^e_{C_2,d} \right)$$

Using this formula the algorithm computes $dp_{C,d}$ for every $d \in \mathcal{D}_v$. Like previously, respective price assignments to edges of C are stored along with the results.

Let us summarize how the procedure calculating $dp_{C,d}$ works. The algorithm iterates over all possible edges $e \in C$. For every possibility, new biconnected components C' are created, one for each edge in $C \setminus \{e\}$. Then, the subprocedure for the case of a single edge finds all values of $dp^e_{C',d}$, which is done in polynomial time. Thus, the running time of the whole procedure is also polynomial.

Since all the above procedures run in polynomial time and each biconnected component is processed only once, the solution is found in polynomial time. Prices obtaining the computed maximal revenue can be easily calculated using additional information stored alongside the values of $dp_{C,d}$. This proves Lemma 4.1.

4.2 The non-skeleton subproblem

In order to solve the non-skeleton subproblem the algorithm utilizes a special structure of non-skeleton components on j-th level. For each of them, a rooted instance of the tollbooth problem on cactus graphs is created. Those subproblems are solved by the procedure described in Section 4.1. The price assignments for individual non-skeleton components are merged by a probabilistic procedure which can, however, be derandomized.

4.2.1 Non-skeleton components

Recall that each buyer $i \in B_j$ is defined by a triple (u_i, v_i, b_i), which means that she has a valuation of b_i for all u_i-v_i paths. Since the skeleton edges are given away for free, the algorithm only processes respective non-skeleton sections, which are modeled

by two independent copies of i-th buyer. If $u_i \neq \text{repr}_j(u_i)$, u_i is not a skeleton vertex on the j-th level and thus belongs to exactly one non-skeleton component, denoted C. In this case a buyer defined by the triple $\big(u_i, \text{repr}_j(u_i), b_i\big)$ is added to the instance associated with C. If $v_i \neq \text{repr}_j(v_i)$, we add a buyer $\big(v_i, \text{repr}_j(v_i), b_i\big)$ to the instance associated with the respective non-skeleton component. Since $\text{repr}_j(s)$ is the same for all vertices s within a single non-skeleton component (Corollary 3.2.1), all subproblems defined this way will indeed be rooted instances.

Remark 4.1. *For $i \in B_j$ such that $u_i \neq \text{repr}_j(u_i)$ and $v_i \neq \text{repr}_j(v_i)$ let C_1 and C_2 be the non-skeleton components containing respectively u_i and v_i. Then, C_1 and C_2 are contained in a single fragment of D_j, but they belong to different fragments of D_{j+1}.*

Proof. By Definition 3.2 each skeleton component on the j-th level is fully contained within a fragment in D_{j+1}. Recall that in order for i-th buyer to be processed on level j (for $j < L$), u_i and v_i must belong to the same fragment in D_j, but no path between them can be contained within a fragment in D_{j+1}. Since $u_i \in C_1$ and $v_i \in C_2$, this concludes the proof. □

4.2.2 The randomized algorithm

Let us begin by introducing additional definitions:

- A price vector p is said to be feasible for the non-skeleton subproblem if it assigns zero to all skeleton edges.
- For any feasible prices p let $\text{rev}_j(p)$ be the revenue generated by buyers from B_j under prices p.
- For a non-skeleton component C and feasible prices p by $\text{rev}_{j,C}(p)$ let us denote the part of $\text{rev}_j(p)$ resulting from selling edges from C. It is well-defined, i.e. is the same regardless of which shortest path between u_i and v_i is assigned to i-th buyer for all $i \in B_j$. For a fragment F from D_j or D_{j+1} let us define $\text{rev}_{j,F}(p)$ in a similar way. Since each such F is a union of a subgraph of the skeleton and non-skeleton components, $\text{rev}_{j,F}(p)$ is also well-defined.
- For any subgraph S of G, by $B_{j,S}$ let us denote the set of such buyers $i \in B_j$ that u_i or v_i is not a skeleton vertex and belongs to S.

By Remark 4.1, for each $H \in D_j$ and any feasible prices p only buyers from $B_{j,H}$ contribute to $\text{rev}_{j,H}(p)$, which depends solely on the prices of edges in H. Thus, all fragments $H \in D_j$ can be processed separately, each with the respective set of buyers – $B_{j,H}$. Here follows a description of an algorithm for handling a single fragment $H \in D_j$.

The algorithm handles all subparts of H, i.e. fragments from D_{j+1} contained in H, individually. First, each of them is equiprobably and independently colored either white or black. In white subparts the price of all edges is set to zero. For each non-skeleton component contained in a black subpart, the associated rooted instance is solved using the procedure presented in section 4.1. Edges of this non-skeleton component are priced according to the obtained optimal solution.

Lemma 4.3. *Let p be the price vector found by the above randomized algorithm and q be any price vector feasible for the non-skeleton subproblem. Then, the following inequality holds:*

$$\mathbb{E}\left[\mathrm{rev}_{j,H}(p)\right] \geq \frac{1}{4}\,\mathrm{rev}_{j,H}(q)$$

Proof. Consider a non-skeleton component C and a fragment F in D_{j+1} containing C. Assume that F has been painted black and all the other subparts of H have been painted white. Then, by Remmark 4.1 each buyer $i \in B_{j,C}$ has a budget of b_i for purchasing a path in C from her destination vertex to its skeleton representative because her other non-skeleton section is either empty or in a white subpart. Note that this exactly reflects the situation modeled by the rooted instance associated with C. Since only buyers from $B_{j,C}$ contribute to $\mathrm{rev}_{j,C}$, p maximizes $\mathrm{rev}_{j,C}$.

$$\mathrm{rev}_{j,C}(p) \geq \mathrm{rev}_{j,C}(q)$$

In reality, however, the other fragments from D_{j+1} contained in H also can be black. Nevertheless, each of them is white independently of the color of F with probability $\frac{1}{2}$. Thus each buyer $i \in B_{j,C}$ can spend up to b_i on her non-skeleton section in C with probability at least $\frac{1}{2}$.

$$\mathbb{E}\left[\mathrm{rev}_{j,C}(p) \mid \text{F is black}\right] \geq \frac{1}{2}\,\mathrm{rev}_{j,C}(q)$$

$$\mathbb{E}\left[\mathrm{rev}_{j,C}(p)\right] \geq \frac{1}{4}\,\mathrm{rev}_{j,C}(q)$$

By summing the above inequality over all non-skeleton components in H, we obtain the desired lower bound:

$$\mathbb{E}\left[\mathrm{rev}_{j,H}(p)\right] \geq \frac{1}{4}\,\mathrm{rev}_{j,H}(q)$$

□

4.2.3 Derandomization

Corollary 4.3.1. *There exists a deterministic polynomial algorithm which for a non-skeleton subproblem on j-th level finds prices achieving at least $\frac{\mathrm{NSKOPT}_j}{4}$ revenue.*

Proof. Let us begin the proof by derandomizing the procedure from Lemma 4.3 for pricing edges in a single fragment $H \in D_j$. Note that in at least one of its outcomes the computed prices generate at least as must revenue as the expected value. Each possible outcome is determined by assigning one of two colors for each fragment $F \in D_{j+1}$ contained in H. Since the number of them is $\mathcal{O}(k)$, there are only 2^{ck} possibilities (for a constant c). As $k = \left\lceil \log^{\frac{1}{2}} m \right\rceil$, this number can be bounded by a polynomial in m. The algorithm iterates over all possible color assignments and chooses the solution maximizing revenue in H. Prices p found this way achieve revenue not smaller than the expected revenue achieved by the randomized procedure. Thus, for any feasible prices q:

$$\mathrm{rev}_{j,H}(p) \geq \frac{1}{4}\,\mathrm{rev}_{j,H}(q) \tag{1}$$

The algorithm performs this derandomized procedure for each fragment of D_j and prices the edges accordingly. Let us denote the resulting price assignment by p_{full}. Recall that the procedure from Section 4.2.2 does not make any assumptions on prices of edges outside the single part H. Thus, Inequality 1 holds for p_{full}, too. By taking $q = p_{opt}$, an optimal solution to the non-skeleton subproblem on j-th level, we obtain the desired upper bound on generated revenue.

$$\sum_{H \in D_j} \mathrm{rev}_{j,H}(p_{full}) \geq \sum_{H \in D_j} \frac{1}{4}\,\mathrm{rev}_{j,H}(p_{opt})$$

$$\mathrm{rev}_j(p_{full}) \geq \frac{1}{4}\,\mathrm{rev}_j(p_{opt}) = \frac{1}{4}\mathrm{NSKOPT}_j$$

□

Chapter 5

Skeleton edges

This chapter describes an algorithm solving the skeleton subproblem on a single, j-th level of decomposition. In this case, all buyers assigned to j-th level, denoted B_j, wish to buy paths connecting their pairs of vertices with the cheapest skeleton sections. Non-skeleton sections do not influence the envy-freeness of a solution because they are given away for free. Hence, a buyer $i \in B_j$ wishing to buy an u_i-v_i path in the original problem can be thought of as a buyer with the same budget wishing to buy a shortest $\text{repr}_j(u_i)$-$\text{repr}_j(v_i)$ path.

Let SKOPT_j be the maximum revenue generated by customers from B_j under the condition that the price of all non-skeleton edges, that is those outside SK_j, equals zero. This chapter describes a polynomial time algorithm which finds prices for edges in SK_j generating at least $\frac{\text{SKOPT}_j}{2048}$ revenue.

5.1 Decomposing the skeleton

Before describing the algorithm itself, let us explore important properties of the skeleton subproblem.

Remark 5.1. *For each buyer $i \in B_j$, there exists a fragment in D_j containing the skeleton representatives of both u_i and v_i. Furthermore, every path between the skeleton representatives contains a border vertex of j-th level.*

This property is true for u_i and v_i, which follows from the way buyers are assigned to levels of decomposition. It also holds for their representatives because by Definition 3.3 each vertex and its representative on j-th level belong to the same fragment of D_{j+1} (for $j = L$ they are the same vertex). In order to take advantage of this property, it is convenient to decompose SK_j into smaller subgraphs.

Consider a process of compressing SK_j by applying the following operations:

1. Let there be two edges: (u, v), (v, w) such that v is not a border vertex and has degree equal two. Merge them into a single edge (u, w) and erase v from the graph.

2. For any two vertices u and v, if there are parallel (u, v) edges, merge them.

The process concludes when none of the above operations can be executed anymore.

Definition 5.1. *A **segment on j-th level** is a subgraph of SK_j which is contracted into a single edge by the above procedure. The **endpoints of a segment** are the endpoints of the corresponding edge in the compressed graph. The **cost** or **length** of a segment is the cost of a cheapest path between its endpoints which is fully contained within the segment.*

Note that the only vertices shared by segments are their endpoints. Furthermore, every border vertex on j-th level is an endpoint of a segment because border vertices are never erased from SK_j during the compression.

Definition 5.2. *The **skeleton of a fragment** $F \in D_j$ is the minimal subgraph of G containing all simple paths between skeleton vertices from F. It is denoted $\mathrm{SK}_j(F)$.*

Recall that by Lemma 3.2 all simple paths between skeleton vertices are contained in the skeleton. Hence, $\mathrm{SK}_j(F) \subseteq \mathrm{SK}_j$.

Definition 5.3. *Let F be a fragment in D_j. A segment which is contained in F is said to be an **inner segment** of $\mathrm{SK}_j(F)$. A simple path which connects two skeleton vertices of F and contains no edges from F is called an **outer segment** of $\mathrm{SK}_j(F)$.*

Each outer segment starts and ends in a border vertex. Thus, it consists of several whole segments, i.e. traversed from one endpoint to another.

Remark 5.2. *Inner and outer segments form an edge-disjoint partition of $\mathrm{SK}_j(F)$.*

Proof. Let us begin the proof by showing that inner segments form an edge-disjoint partition of $\mathrm{SK}_j(F) \cap F$, that is all skeleton edges inside F. Since every border vertex is an endpoint of a segment, each segment is fully contained within a certain fragment in the current decomposition. Thus, as segments form a disjoint cover of SK_j, the inner segments of the skeleton of F form a disjoint cover of $\mathrm{SK}_j(F) \cap F$.

By definition, each edge in $\mathrm{SK}_j(F) \setminus F$ belongs to an outer segment of $\mathrm{SK}_j(F)$. Note that every outer segment defines a simple cycle in $\mathrm{SK}_j(F)$ consisting of two non-empty paths: the outer segment itself and a path in F connecting its endpoints. Thus, if an edge belonged to two distinct outer segments, it would lie on two simple cycles, which leads to contradiction. □

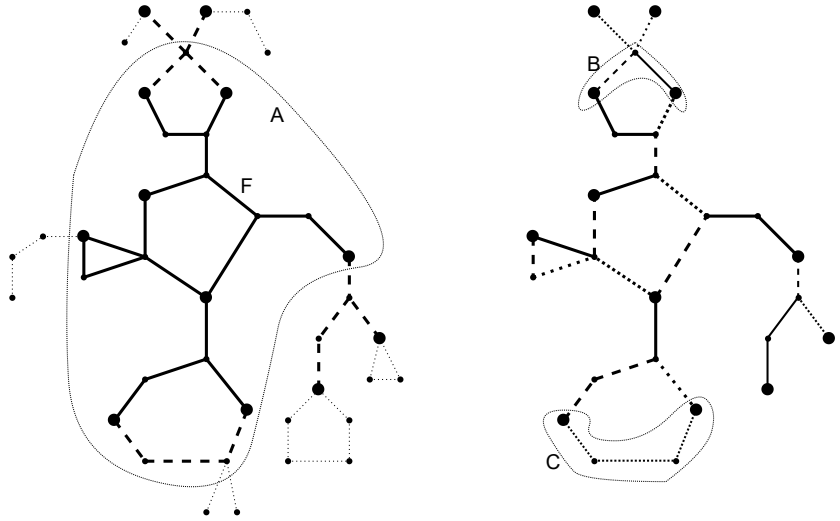


Figure 5.1: An example cactus graph with its border vertices and skeleton on j-th level highlighted. Non-skeleton edges are marked by light dotted lines on the left and omitted on the right. Skeleton edges are grouped on the left by the fragments they belong to and on the right by segments on j-th level. Edges from the fragment $F \in D_j$ are marked by solid black lines. B and C are the outer segments of its skeleton, denoted A.

5.2 Rounding

If a simple path neither starts nor ends in a given segment, it will either traverse this segment from one endpoint to another or not at all. Thus, for such paths it is sufficient to consider the cost of the segment and not the prices of individual edges. Let us formalize this observation.

Definition 5.4. *A buyer $i \in B_j$ is said to be **involved** in a segment on j-th level if either $\mathrm{repr}_j(u_i)$ or $\mathrm{repr}_j(v_i)$ belongs to this segment and is not its endpoint.*

Remark 5.3. *For fixed costs of segments, changing prices of individual edges inside a segment influences only the involved buyers.*

Using this observation, the algorithm could guess the costs of segments and then distribute those costs among individual edges in a way that would maximize the revenue of involved buyers. However, such a naive approach cannot be used by a deterministic algorithm because there are infinitely many combinations of segments' costs. The following lemma allows for considering only finitely many of them.

Lemma 5.1. ***(rounding)*** *There exists a price assignment obtaining revenue of at least $\frac{\mathrm{SKOPT}_j}{4}$ such that each segment's length belongs to the following set:*

$$P = \left\{ \frac{mb_{max}}{2^t} \;\middle|\; t \in \left\{0, 1, \ldots, \left\lceil \log\left(1024 \cdot m^2 \cdot |B_j|\right) \right\rceil\right\} \right\} \cup \{0\}$$

Here b_{max} is the greatest budget of buyers in B_j and m is the number of edges in the whole cactus graph G.

Proof. Consider any prices generating revenue SKOPT_s. If a price of any edge is greater than b_{max}, it can be lowered to b_{max} without loss of revenue. After such a modification, the length of any path is at most $m \cdot b_{max}$. We further round the prices down so that the price of each segment belongs to P. Let us consider a segment s with cost p_s and let u, v be its endpoints. We choose a u-v path of length p_s fully contained within s and set a maximal such $x \in [0, 1]$ that $p_s \cdot x \in P$. Note that either $x > \frac{1}{2}$ or $x = 0$, which implies $p_s < \frac{b_{max}}{1024 \cdot m \cdot |B_j|}$. Next we multiply the cost of the edges along the chosen path by x. By applying this procedure to all segments in the graph, we ensure that segments' lengths belong to P. Let us fix any path, which initially has length d. After rounding the prices down, its cost is at least $\frac{d}{2} - \frac{b_{max}}{1024 \cdot |B_j|}$. As a consequence, the cost of a cheapest path desired by a buyer, which initially was equal d, will be between $\frac{d}{2} - \frac{b_{max}}{1024 \cdot |B_j|}$ and d after the rounding. Summing these inequalities for all buyers results in a lower bound on the revenue of $\frac{\mathrm{SKOPT}_j}{2} - \frac{b_{max}}{1024}$. As $\mathrm{SKOPT}_j \geq b_{max}$, this concludes the proof.

□

5.3 Pricing strategies

The number of segments can potentially be large. In particular, there are m segments on L-th level. Thus, the algorithm can not explicitly iterate over all combinations of their rounded costs. Instead, it uses the structure of paths desired by customers in B_j to handle each fragment in the current decomposition separately.

Recall that for each buyer $i \in B_j$ there exists at least one fragment in D_j such that both $\mathrm{repr}_j(u_i)$ and $\mathrm{repr}_j(v_i)$ belong to it (Remark 5.1). The algorithm assigns each customer from B_j to one of such fragments. Let us fix a fragment $F \in D_j$ and denote the set of buyers assigned to it by $B_{j,F}$. By definition, the skeleton of F contains the skeleton sections of all paths desired by buyers from $B_{j,F}$. Thus, revenue generated by those buyers in the skeleton subproblem under any prices p, denoted $\mathrm{rev}_{j,F}(p)$, depends only on the costs of edges in $\mathrm{SK}_j(F)$.

Having said that, particular fragments in D_j cannot be processed completely independently, as their skeletons overlap. More specifically, each outer segment of $\mathrm{SK}_j(F)$ consist of several inner segments from different fragments in D_j. However, simple paths starting and ending in F traverse the outer segments only as a whole, from one endpoint to another. Thus, the costs of individual edges, or even segments, forming the outer segments of $\mathrm{SK}_j(F)$ do not influence $\mathrm{rev}_{j,F}(p)$. Only the total costs of outer segments themselves do. An outer segment's length is a sum of costs of several inner segments from different fragments. Thus, even if considering only

price assignments satisfying the rounding lemma (5.1), it may not belong to P. Because of this, they need to be treated differently than inner segments.

Definition 5.5. *For a fragment $F \in D_j$ a **pricing strategy** s for its skeleton edges is defined by:*

- *A single number $p_{s,i} \in P$ for each inner segment i of $\mathrm{SK}_j(F)$.*
- *Two consecutive numbers from $l_{s,o}, r_{s,o} \in P'$ for each outer segment o of $\mathrm{SK}_j(F)$. P' is defined as follows:*

$$P' = \left\{ \frac{m^2 b_{max}}{2^k} \;\middle|\; k \in \left\{0, 1, \ldots, \left\lceil \log\left(1024 \cdot m^3 \cdot |B_j|\right) \right\rceil\right\} \right\} \cup \{0\}$$

Alternatively, $l_{s,o}$ and $r_{s,o}$ can be both equal to zero.

*Prices of edges in inner segments of $\mathrm{SK}_j(F)$ **implement** the strategy s if the length of each segment i in F equals $p_{s,i}$. A global price assignment **implements** this strategy if additionally the length of each outer segment o of $\mathrm{SK}_j(F)$ is in the interval $(l_{s,o}, r_{s,o}]$ or equals zero if $l_{s,o} = r_{s,o} = 0$. A combination of pricing strategies is **valid** if it's not contradictory, i.e. the costs assigned to inner segments satisfy the requirements given on the lengths of outer segments.*

On the high level, the algorithm for solving the skeleton subproblem works in two phases. In the first phase, described in Section 5.4, for each fragment from D_j and each pricing strategy for it near-optimal prices implementing that strategy are found. Then, using dynamic programming the algorithm constructs a valid combination of strategies which achieves high overall revenue.

5.3.1 Approximating revenue

For each fragment $F \in D_j$, the algorithm iterates over all possible strategies. For each of them it calculates near-optimal prices of edges in inner segments of $\mathrm{SK}_j(F)$ without any additional assumptions about prices of edges in $\mathrm{SK}_j(F) \setminus F$. However, if instead of the exact lengths of outer segments only intervals of their possible values are known, it is impossible to calculate the revenue generated by buyers from $B_{j,F}$. It is possible to provide an approximation, though.

Definition 5.6. *Consider a fragment $F \in D_j$, pricing strategy s for it and any prices implementing s, denoted p. The **approximate revenue** generated by those prices, denoted $\mathrm{rev}_{j,F,s}(p)$, is the revenue generated by customers from $B_{j,F}$ under the following assumptions:*

- *The skeleton edges of F are priced according to p.*
- *Each outer segment o in $\mathrm{SK}_j(F)$ has length $r_{s,o}$.*

The following lemma allows the algorithm to focus on maximizing the values of approximate revenue calculated locally for each fragment and pricing strategy.

Lemma 5.2. *Consider a valid combination of pricing strategies and denote the strategy for a fragment $F \in D_j$ as s_F. Let p be any assignment of prices to all skeleton edges which implements that combination of strategies. Then, the inequality*

$$\sum_{F \in D_j} \mathrm{rev}_{j,F,s_F}(p) \geq \frac{b_{max}}{512}$$

implies

$$\mathrm{rev}(p) \geq \frac{1}{4} \sum_{F \in D_j} \mathrm{rev}_{j,F,s_F}(p)$$

Proof. Consider a fragment $F \in D_j$ and a buyer i from $B_{j,F}$, who desires paths connecting u_i and v_i. We fix any such path and by d, d' denote its length given respectively the prices p or prices p modified in such a way, that the length of each outer segment o of $\mathrm{SK}_j(F)$ equals $r_{s_F,o}$. If $l_{s_F,o} = 0$, then $r_{s_F,o} \leq \frac{b_{max}}{1024 \cdot m \cdot |B_j|}$. Thus, due to decreasing the cost of each outer segment o from $r_{s_F,o}$ to its original cost, the path's length can decrease by at most $\frac{b_{max}}{1024|B_j|}$. Otherwise, $r_{s_F,o} = 2l_{s_F,o}$ so strategy s_F assumes each outer segment o of $\mathrm{SK}_j(F)$ to be at most two times longer than it is under prices p. Hence, $d \geq \frac{d'}{2} - \frac{b_{max}}{1024|B_j|}$. Of course, $d \leq d'$.

Since the above inequalities hold for all u_i-v_i paths, it is also true for the distance between u_i and v_i. Hence, every buyer contributing to $\mathrm{rev}_{j,F,s_F}(p)$ also contributes to $\mathrm{rev}_{j,F}(p)$. By summing the inequality for all those buyers, we obtain:

$$\mathrm{rev}_{j,F}(p) \geq \frac{\mathrm{rev}_{j,F,s_F}(p)}{2} - |B_{j,F}| \cdot \frac{b_{max}}{1024|B_j|}$$

When applying to all parts of the current decomposition, we have:

$$\mathrm{rev}_j(p) \geq \frac{1}{2} \sum_{F \in D_j} \mathrm{rev}_{j,F,s_F}(p) - \frac{b_{max}}{1024}$$

As $\sum_{F \in D_j} \mathrm{rev}_{j,F,s_F}(p) \geq \frac{b_{max}}{512}$, this concludes the proof.

□

5.3.2 Bounding the number of pricing strategies

Since for each fragment $F \in D_j$ the algorithm iterates over all possible pricing strategies, it is vital to make sure that their number is polynomial. This is achieved by the following lemma.

Lemma 5.3. *For each fragment $F \in D_j$, there are $\mathcal{O}(k)$ inner and outer segments in its skeleton.*

Proof. Let us consider a graph resulting from converting each inner and outer segment from $\mathrm{SK}_j(F)$ into a single edge, let us denote it by H. We will prove that H contains $\mathcal{O}(k)$ vertices, which is sufficient, since H, being a cactus, can have at most twice as many edges as vertices.

Like in $\mathrm{SK}_j(F)$, all edges of H must lie on a path between border vertices on j-th level. If it was not the case, no edge of the corresponding segment would lie on such a path in $\mathrm{SK}_j(F)$ and hence the whole segment would not belong to the skeleton on j-th level.

Consider a tree of biconnected components of H rooted in a vertex denoted r. Each one of its leaves may be either a cycle or a single edge. Either way, at least one of its non-topmost (relative to r) vertices must be a border one. If it was not the case, edges of that biconnected component would not lie on any simple path connecting border vertices. Since a vertex can be non-topmost in only one biconnected component, this results in an upper bound of $\mathcal{O}(k)$ on the number of leaf components in the tree.

Let us define a coloring of biconnected components in H. A biconnected component is said to be black either if it has at least two children in the rooted tree of biconnected components or one of its non-topmost vertices is a border vertex. Otherwise, it is colored red. Since the numbers of border vertices and leaf components are bounded, there are only $\mathcal{O}(k)$ black components.

Note that a red component must be single edge in H. It's because only its topmost vertex and its child's topmost vertex can have degree greater than two. Any other vertex in a red component, having degree two and not being a border vertex, would be removed during the compression, so would not be an endpoint of a segment.

Let us say that, hypothetically, a red component is a child of another red component. Then the corresponding biconnected components could be merged using the first operation of the compressing procedure from Definition 5.1. Hence, every red biconnected component can have only black children. Thus, there can be only as many non-leaf red components as there are black components. From this follows that there are $\mathcal{O}(k)$ biconnected components in H.

Note that every vertex in H with degree greater than two is a topmost vertex of a biconnected component. Hence, each vertex in H is either a topmost vertex of a biconnected component or a border vertex. There are only $\mathcal{O}(k)$ vertices and thus $\mathcal{O}(k)$ edges in H. Since each inner or outer segment in $\mathrm{SK}_j(F)$ corresponds to an edge in H, there are only $\mathcal{O}(k)$ of them in the skeleton of F.

□

Corollary 5.3.1. *For each fragment on j-th level of decomposition, the number of pricing strategies is polynomial in m and n.*

Proof. Let us fix a single fragment $F \in D_j$. A pricing strategy assigns one of $|P|$ possible values to each inner segment of $\mathrm{SK}_j(F)$ and one of $|P'|$ intervals to each outer segment. Note that $|P| \leq |P'|$ and $|P'|$ is $\mathcal{O}(\log mn)$ ($n = |B|$, so $n \geq |B_j|$). Thus, there exist such constants c_1 and c_2 that the number of strategies can be bounded by $(c_1 \cdot \log mn)^{c_2 k}$, which can be bounded by a polynomial in n and m:

$$(c_1 \cdot \log mn)^{c_2 k} \leq 2^{(\log c_1 \cdot \log\log mn) \cdot 2c_2\sqrt{\log m}} \leq 2^{2c_2 \log c_1 \cdot \log nm} = (nm)^{2c_2 \log c_1}$$

□

5.4 Solution for a single fragment

In the first phase the algorithm iterates over all possible pricing strategies for each fragment in the current decomposition. Let us fix a single fragment $F \in D_j$ and a pricing strategy s_F. This section presents a polynomial procedure for finding prices of skeleton edges in F which implement the strategy s_F and achieve high approximate revenue. More precisely, let SKOPT_{j,F,s_F} be the maximum value of $\mathrm{rev}_{j,F,s_F}(p)$ among p implementing s_F. Then, the prices found using the method presented here achieve at least $\frac{\mathrm{SKOPT}_{j,F,s_F}}{32}$ approximate revenue.

For ease of presentation we describe a randomized procedure, which will be later derandomized. It processes each segment contained in F individually and independently. Let us fix a single inner segment S of $\mathrm{SK}_j(F)$ and introduce the following notation:

- The subset of buyers from $B_{j,F}$ involved in S is denoted as B_S.
- Endpoints of S are l and r.
- The cost of S given by s_F is denoted c.

For a buyer $i \in B_S$ wishing to purchase the cheapest $\mathrm{repr}_j(u_i)$-$\mathrm{repr}_j(v_i)$ path we will assume without loss of generality that $\mathrm{repr}_j(u_i)$ lies strictly inside S (is not an endpoint). On the other hand, $\mathrm{repr}_j(v_i)$ either lies strictly inside another segment or is an endpoint of a segment (possibly of S). In the former case we set v_i' and v_i'' to be the endpoints of the segment containing $\mathrm{repr}_j(v_i)$, in the latter $\mathrm{repr}_j(v_i) = v_i' = v_i''$. The pricing strategy for F is fixed, and we are interested only in approximate revenue. Thus, in this section costs of paths are always calculated with the assumption that the length of each outer segment o of $\mathrm{SK}_j(F)$ equals $r_{s_F,o}$. This way, distances between endpoints of inner and outer segments in $\mathrm{SK}_j(F)$ are determined by s_F. This metric is denoted dist_{s_F}.

Let B'_S be any subset of B_S. By $\mathrm{CONTRIB}_{S,B'_S,s_F}$ let us denote the maximum cost paid by buyers from B'_S for the edges of S in any envy-free solution based on

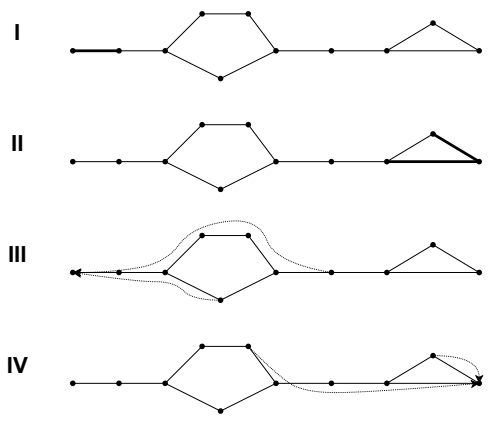
prices implementing s_F. In other words, $\text{CONTRIB}_{S,B'_S,s_F}$ is the maximum possible contribution towards rev_{j,F,s_F} resulting from buyers of B'_S purchasing edges in S.

Now follows the description of two subprocedures. The first one handles segments S which contain cycles and the other the ones which do not.

5.4.1 Cyclic segments

Let S be a segment containing a cycle. Since the whole graph is a cactus, there can be no path from l to r bypassing edges of S. Otherwise, edges on the cycle inside S would lie on two simple cycles. Thus, B_S can be split into two disjoint sets:

- $B_{S,l}$ – such buyers i, that every $\text{repr}_j(u_i)$-$\text{repr}_j(v_i)$ path contains l.
- $B_{S,r}$ – such buyers i, that every $\text{repr}_j(u_i)$-$\text{repr}_j(v_i)$ path contains r.



The algorithm chooses one of the following four options uniformly at random.

First option: Set the price of all edges neighboring the vertex l to c and the price of all the others to 0. This way each buyer from $B_{S,l}$ must pay c for leaving the segment S through the l. Buyers from $B_{S,r}$ pay nothing for edges inside S because leaving the segment through r is free.

Second option: The same as the previous one, but this time edges adjacent to r have price c. Leaving S is free for $B_{S,l}$ and costs c for BS, r.

Third option: In this case the edges are priced only with buyers from $B_{S,l}$ in mind. Since for all $i \in B_{S,l}$ every $\text{repr}_j(u_i)$-$\text{repr}_j(v_i)$ path passes through l, we focus on the revenue generated by the respective $\text{repr}_j(u_i)$-l paths. For a buyer $i \in B_{S,l}$ we define:

$$b'_i = b_i - \min\left\{\text{dist}_{s_F}(l, v'_i), \text{dist}_{s_F}(l, v''_i)\right\}$$

Note that the segment S and buyers from $B_{S,l}$ with their $\text{repr}_j(u_i)$-l paths and reduced budgets b'_i form a rooted instance of the tollbooth problem on cactus (definition 4.1). Recall that the procedure for solving such subproblems presented

in Section 4.1 allowed for predefining distances from the root to some vertices. By adding a constraint that the depth of r must equal c, we restrict ourselves only to those prices in S, which implement s_F. Using the procedure described in Section 4.1 the algorithm finds an optimal solution for this artificial rooted instance and prices the edges in S accordingly.

Lemma 5.4. *Let us assume that the third option was chosen in S. Then, the expected approximate revenue under s_F generated by buyers from $B_{S,l}$ purchasing edges from S is at least $\frac{1}{4}\text{CONTRIB}_{S,B_{S,l},s_F}$*

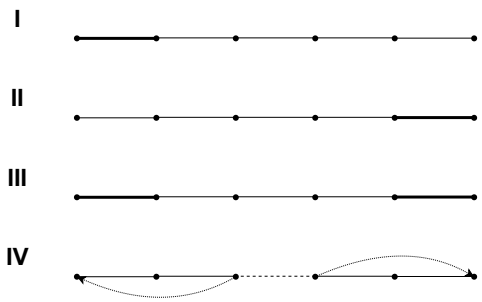
Proof. Note that a buyer $i \in B_{S,l}$ cannot pay more than b'_i for edges in S under any prices implementing s_F. Thus, revenue achieved in the artificial rooted instance is not smaller than $\text{CONTRIB}_{S,B_{S,l},s_F}$.

With probability of at least one-fourth a buyer $i \in B_{S,l}$ is able to afford to pay b'_i for a $\text{repr}_j(u_i)$-l path, as the first two options in the segment of $\text{repr}_j(v_i)$ (regardless whether it's cyclic or not) guarantee her a free path from $\text{repr}_j(v_i)$ to v'_i or v''_i. Thus, the expected revenue is at least one-fourth of the revenue from the hypothetical rooted instance, which concludes the proof. □

Fourth option: This case is analogous to the previous one, but the algorithm optimizes approximate revenue generated in S by $B_{S,r}$. Of course, for this case an analog of Lemma 5.4 also holds.

5.4.2 Acyclic segments

If the inner segment S does not contain any cycles, it must be a path. There are four possible ways of assigning prices to individual edges of S which are chosen uniformly at random. All of them guarantee the length of S to be equal c.



First option: The same as in the case of a cyclic segment, cost c is assigned to the edge in S adjacent to l.

Second option: Price of the edge adjacent to r is set to c, all the other edges are given away for free.

Third option: The price of the first and last edge of the path is set to $\frac{c}{2}$. However, if S consists of a single edge, its price equals c. This way each buyer from B_S must pay $\frac{c}{2}$ for edges in the segment S (if only one edge belongs to S, B_S is empty).

Similarly as in the cyclic case, let us define the upper bound on i-th buyer's budget for edges in S:

$$b'_i = b_i - \min\left\{\mathrm{dist}_{s_F}(l, v'_i), \mathrm{dist}_{s_F}(l, v''_i), \mathrm{dist}_{s_F}(r, v'_i), \mathrm{dist}_{s_F}(r, v''_i)\right\}$$

Let $B'_S \subseteq B_s$ consist of those buyers i whose b'_i is at least $\frac{c}{2}$.

Lemma 5.5. *Let us assume that the third option was chosen in S. Then, the expected approximate revenue under s_F generated in S by buyers from B'_S is at least $\frac{1}{8}\mathrm{CONTRIB}_{S,B'_S,s_F}$.*

Proof. With probability at least $\frac{1}{4}$ $\mathrm{dist}_{s_F}(v''_i, \mathrm{repr}_j(v_i)) = 0$, the same holds for $\mathrm{dist}_{s_F}(v'_i, \mathrm{repr}_j(v_i))$. Thus, for each buyer $i \in B'_s$ with probability at least $\frac{1}{4}$ there exists a $\mathrm{repr}_j(u_i)$-$\mathrm{repr}_j(v_i)$ path of length at most b_i. In such a case, she would pay $\frac{c}{2}$ for edges in S. Since S is a path connecting l and r, it is impossible for a buyer to pay more than c for a simple path within S under any prices implementing s_F. Thus, each customer from B'_s with probability at least $\frac{1}{4}$ pays at least half as much for edges in s as they would given any other price assignment. Hence, the expected contribution of B'_S towards rev_{j,F,s_F} generated by purchasing edges from S is at most eight times smaller than its maximal possible value.

□

Fourth option: In this case let us restrict our attention to $B''_s = B_s \backslash B'_s$, i.e. buyers whose b'_i is strictly smaller than $\frac{p_s}{2}$. Consider any prices of edges in S implementing s_F. Note that the subsets of vertices in S reachable from l and r through a path in S of length at most $\frac{c}{2}$ are disjoint. Since S is a path, there is exactly one edge in S connecting these sets. Let us call it the **pivot edge**. Note that no buyer with $b'_i < \frac{c}{2}$ would purchase a pivot edge. Thus, removal of the pivot edge does not influence revenue generated by B''_s.

Let us fix an edge e in S and consider only such price assignments, which result in e being the pivot edge. B''_S is split into disjoint sets: $B''_{S,l}$ and $B''_{S,r}$, the buyers i such that $\mathrm{repr}_j(u_i)$ is on the same side of the pivot edge as l and r respectively. Each subset of buyers is processed independently in its part of the segment.

The algorithm constructs a rooted instance of the tollbooth problem on cactus using $B''_{S,l}$. Each buyer $i \in B''_{S,l}$ wishes to buy a $\mathrm{repr}_j(u_i)$-l path and has the budget of b''_i defined in the following way:

$$b''_i = b_i - \min\left\{dist(l, v'_i), dist(l, v''_i)\right\}$$

Such a problem is solved by the dynamic programming algorithm described in Section 4.1. Edges between l and the pivot edge are priced according to this solution.

The costs of edges on a path from the pivot edge to r are set in an analogous way. The price of e is set in such a way, that the length of S would be equal to c. Such a price assignment is constructed for each possible pivot edge. The algorithm chooses the one maximizing the sum of revenues achieved in the two rooted instances.

Lemma 5.6. *Let us assume that the fourth option was chosen in S. Then, the expected approximate revenue under s_F generated by buyers from B''_s by purchasing edges from S is at least $\frac{1}{4}\text{CONTRIB}_{S,B''_S,s_F}$.*

Proof. It is sufficient to prove the lemma for a fixed pivot edge because the algorithm iterates over all possibilities. We will first show that the edge designated to be a pivot edge, denoted e, indeed is one. Then, we will prove that the found prices result in high approximate revenue compared to other solutions, in which no buyer purchases e.

Since the budgets of buyers in the rooted instances are smaller than $\frac{c}{2}$, the distances from both l and r to the pivot edge must be too. Thus, the price of e is positive, and it indeed divides the vertices within the distance of $\frac{p}{2}$ from l and r.

Note that no customer $i \in B''_S$ can pay more than b''_i for edges inside S under any prices implementing s_F as long as she does not buy e. Thus, in this case the revenue achieved in the two rooted instances by optimal solutions is not smaller than the maximal approximate revenue generated by B''_S in S.

Thanks to the first two options, each customer $i \in B''_S$ with probability at least $\frac{1}{4}$ pays nothing for edges in the other segment she is involved in, so she has a budget of b''_i to pay for edges in S. Thus, the expected approximate revenue in S from B''_S is at most four times smaller than the total revenue obtained in the two rooted instances, which concludes the proof. □

5.4.3 Analysis of the expected value

Lemma 5.7. *Let p and q be two price assignments implementing s_F such that q maximizes rev_{j,F,s_F} and p is the result of the randomized algorithm for pricing the skeleton edges of a single fragment. Then, the following inequality holds:*

$$\mathbb{E}\left[\text{rev}_{j,F,s_F}(p)\right] \geq \frac{1}{32}\,\text{rev}_{j,F,s_F}(q)$$

Proof. Let us fix any two assignments of paths to the buyers such that each buyer $i \in B_{j,F}$ is assigned a u_i-v_i path, which is the cheapest under prices p or q respectively. Of course, only as long as its cost does not exceed b_i. Furthermore, each outer segment o of $\text{SK}_j(F)$ is assumed to have length equal $r_{s_F,o}$.

By $\text{inv}_{j,S,s_F}(p)$ let us denote the total cost paid by buyers from B_S for edges in the inner segment S and by $\text{inv}_{j,F,s_F}(p)$ its sum over all inner segments of $\text{SK}_j(F)$. The total cost paid by buyers from $B_{j,F}$ in segments they are not involved in is

denoted $\text{full}_{j,F,s_F}(p)$. Note that this includes the whole revenue generated by $B_{j,F}$ in outer segments of $\text{SK}_j(F)$. Obviously $\text{rev}_{j,F,s_F}(p) = \text{inv}_{j,F,s_F}(p) + \text{full}_{j,F,s_F}(p)$. The same equality holds for q, for which both values are defined in the same way. We will prove the lemma by separately showing lower bounds for revenue generated by selling parts of segments to involved buyers and selling whole segments to all others.

Selling whole segments: Consider any buyer $i \in B_{j,F}$ buying a $\text{repr}_j(u_i)$-$\text{repr}_j(v_i)$ path under the prices q. By u'_i and v'_i let us denote respectively the first and last endpoint of a segment on this path. Obviously, she contributes $\text{dist}_{s_F}(u'_i, v'_i)$ towards $\text{full}_{j,F,s_F}(q)$. Let u''_i and v''_i be the other endpoints of segments she is involved in. If she is not involved in the given segment, for example because $\text{repr}_j(u_i) = u'_i$, we set $u'_i = u''_i$ (or $v'_i = v''_i$).

Note that the buyer i cannot be 'doubly' involved in a segment. In other words, if $\text{repr}_j(u_i)$ and $\text{repr}_j(v_i)$ are not endpoints of segments, they must belong to different ones. Thus, there is such a combination of the first two options for the segments she is involved in, that there exist free $\text{repr}_j(u_i)$-u'_i and $\text{repr}_j(v_i)$-v'_i paths in those segments. In that case $\text{repr}_j(v_i)$-v''_i and $\text{repr}_j(u_i)$-u''_i paths in those segments are not cheaper under the prices p than they are under prices q. Thus, under the prices p a cheapest $\text{repr}_j(u_i)$-$\text{repr}_j(v_i)$ path has length $\text{dist}_{s_F}(u'_i, v'_i)$ and leads through v'_i and u'_i. Although other shortest $\text{repr}_j(u_i)$-$\text{repr}_j(v_i)$ paths can exist, their cost must also include paying $\text{dist}_{s_F}(u'_i, v'_i)$ for traversing whole segments. Thus, i contributes $\text{dist}_{s_F}(u'_i, v'_i)$ towards $\text{full}_{j,F,s_F}(p)$.

Options for all the inner segments of $\text{SK}_j(F)$ are chosen independently and each one with probability $\frac{1}{4}$. Hence, the situation described above occurs with probability at least $\frac{1}{16}$, which yields the following inequality:

$$\mathbb{E}\left[\text{full}_{j,F,s_F}(p)\right] \geq \frac{1}{16}\,\text{full}_{j,F,s_F}(q) \tag{2}$$

Selling parts of segments: Let us consider a single inner segment S of $\text{SK}_j(F)$ and let B_S be the set of buyers involved in S.

If S contains at least one cycle, let us split B_S into two disjoint sets: $B_{S,l}$ and $B_{S,r}$, which are defined in Section 5.4.1. If the third option is chosen, Lemma 5.4 guarantees that the expected total cost paid by $B_{S,l}$ for edges in S is at most four times smaller than it is under prices q. If the fourth option is chosen, the same holds for $B_{S,r}$. Since both options are chosen with probabilities equal $\frac{1}{4}$, this reasoning yields the following bound:

$$\mathbb{E}\left[\text{inv}_{j,S,s_F}(p) \geq \frac{1}{16}\,\text{inv}_{j,S,s_F}(q)\right] \tag{3}$$

If S contains no cycles, its prices were constructed using the procedure from Section 5.4.2. Let us consider B'_S and B''_S separately. By Lemma 5.5, choosing the

third option guarantees the expected approximate revenue from B'_S in S under prices p to be at most eight times smaller than under prices q. If the fourth option was chosen, by Lemma 5.6 B''_S contributes towards $\mathbb{E}\left[\mathrm{inv}_{j,S,s_F}(p)\right]$ at least one-fourth of what it contributes towards $inv_{j,S,s_F}(q)$. Both options are chosen with probability $\frac{1}{4}$ so this results in the following lower bound:

$$\mathbb{E}\left[\mathrm{inv}_{j,S,s_F}(p)\right] \geq \frac{1}{32}\,\mathrm{inv}_{j,S,s_F}(q) \tag{4}$$

Applying Inequalities 3 and 4 to all inner segments of $\mathrm{SK}_j(F)$ and combining them with 2 concludes the proof:

$$\mathbb{E}\left[\mathrm{rev}_{j,F,s_F}(p)\right] \geq \frac{1}{32}\,\mathrm{rev}_{j,F,s_F}(q)$$

□

5.4.4 Derandomization

The above algorithm can be derandomized yielding the same lower bound on $\mathrm{rev}_{j,F,s_F}(p)$ as on its expected value. There must exist such a price assignment p', that $p = p'$ with non-zero probability and $\mathrm{rev}_{j,F,s_F}(p') \geq \mathbb{E}\left[\mathrm{rev}_{j,F,s_F}(p)\right]$. It turns out that the number of possible outcomes of the randomized algorithm can be bounded by a polynomial. There are only four possibilities for each inner segment, of which there are only $\mathcal{O}(k)$. Hence, there are only $4^{\mathcal{O}(\sqrt{\log m})}$ combinations. Thus, the algorithm can iterate over all possible combinations and find corresponding prices p'. Note that when the price assignment p' is known, $\mathrm{rev}_{j,F,s_F}(p')$ can be found simply by calculating distances between $\mathrm{repr}_j(u_i)$ and $\mathrm{repr}_j(v_i)$ for all $i \in B_{j,F}$. The algorithm chooses the prices p' maximizing $\mathrm{rev}_{j,F,s_F}(p')$, which must satisfy the bound from Lemma 5.7. Let us formalize this observation:

Corollary 5.7.1. *Consider a single fragment $F \in D_j$ and a fixed strategy s_F. Let q be a price assignment implementing s_F which maximizes rev_{j,F,s_F}. There exists a deterministic polynomial procedure which finds prices p implementing s_F such that the following inequality holds:*

$$\mathrm{rev}_{j,F,s_F}(p) \geq \frac{1}{32}\,\mathrm{rev}_{j,F,s_F}(q)$$

5.5 Solution for the whole graph

After applying the procedure from the previous section to each fragment $F \in D_j$ and every pricing strategy, the algorithm merges the resulting partial solutions into a global price assignment for SK_j. More formally, for each fragment $F \in D_j$ there is polynomially many assignments of prices to its skeleton edges. Each one implements a particular pricing strategy for F. This way every valid combination of pricing

strategies for all fragments of D_j defines a global price assignment for SK_j. The procedure presented in this section finds a valid combination maximizing total approximate revenue obtained by respective price assignments in particular fragments. In other words, it finds a valid combination of pricing strategies which maximizes the global score.

Definition 5.7. *Let F be a fragment in D_j, s a pricing strategy for F and p the prices constructed by the procedure described in Section 5.4 for F and s. Then, the* ***score*** *of s equals* $\mathrm{rev}_{j,F,s}(p)$.

The ***score*** *of a valid combination of pricing strategies for a set of fragments is defined as a sum of respective scores.*

Before describing the algorithm, let us discuss the dependencies between skeletons of particular fragments in j-th level of decomposition. Consider a fragment $F \in D_j$. Inner segments of $\mathrm{SK}_j(F)$ by definition contain only skeleton edges from F and are guaranteed not to intersect with inner segments in skeletons of other fragments. Thus, it is enough to consider dependencies caused by outer segments. Let o be an outer segment in $\mathrm{SK}_j(F)$. By definition, o is a path between two skeleton vertices u and v in F consisting of edges from outside of F. Furthermore, by Lemma 3.2, there must exist a u-v path consiststing of skeleton edges fully contained in F. Thus, u and v lie on a simple cycle C, which is contained in SK_j. Note that it's length is a sum of lengths of o and of inner segments of $\mathrm{SK}_j(F)$ contained in C because they form the second u-v path. Thus, by giving bounds for the length of o and setting the costs of inner segments, a pricing strategy for F in fact imposes certain constraints on the length of C. Using this observation the algorithm ensures that the constructed combination of strategies is valid by controlling lengths of cycles which are shared between fragments.

5.5.1 The dynamic programming

The subproblems of the dynamic programming are subsets of fragments in D_j. They are constructed using the tree of biconnected components of G rooted in vertex r, which is defined in Section 2.1. The algorithm iterates over all biconnected components in the bottom-up order defined by the tree. If certain criteria are met, the algorithm constructs and solves the subproblems using previously calculated solutions.

Now we present the method used to process each biconnected component. Let C be the current biconnected component.

- If C is a cycle and its edges are split into multiple fragments from D_j, an **open subproblem** is created. It consists of all the fragments in D_j contained in the subtree graph of C except for the fragment containing the two topmost edges of C. For an open subproblem multiple solutions are found.

- Let v be the topmost vertex of C. If there exists a biconnected component, which has not been processed yet and whose topmost vertex is v, the algorithm solves no subproblems. Otherwise, a separate **closed subproblem** is created for each fragment F whose topmost vertex is v. Let $\mathcal{C}_F$ be the set of all biconnected components whose topmost vertex is v and whose topmost edges belong to F. A closed subproblem for F is formed by all fragments in the subtree graphs of components from $\mathcal{C}_F$ (including F). It is solved by the procedure described in Section 5.5.3. That procedure finds exactly one solution for this subproblem, a valid combination of strategies maximizing the score.

Note that while processing a cycle the algorithm may create both closed and open subproblems. In such a case, the open subproblem is processed first and then the closed ones. It is worth mentioning that the closed subproblems created in the second step are always disjoint. The reason for this is that the topmost edges of any biconnected component always belong to the same fragment.

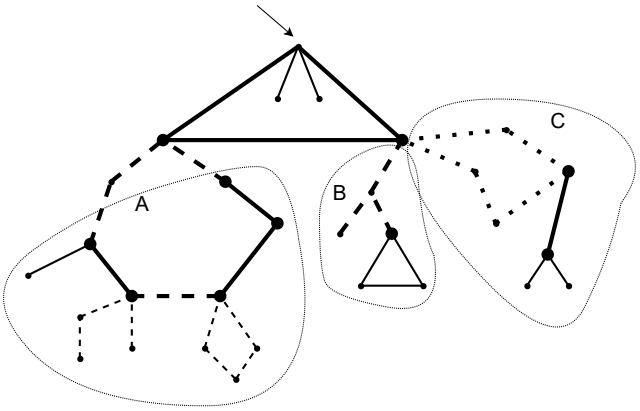


Figure 5.2: Example structure of the dynamic programming. Connected sets of edges marked using the same style belong to the same fragments. Those marked by bold lines are skeleton edges. The arrow points to the root of the tree of biconnected components. A is an open subproblem, whereas B and C are closed ones.

Remark 5.4. *Let S be a closed subproblem. Then, merging any two valid combinations of pricing strategies for segments in S and in $D_j \setminus S$ results in a valid global combination of strategies.*

Proof. A subgraph of G corresponding to S is a union of subtree graphs of biconnected components. Thus, it contains only whole biconnected components and no cycle is shared between a fragment from S and a fragment from $D_j \setminus S$. Because of that, the combinations of strategies for those two sets of fragments can be chosen independently of each other. □

At the end of the procedure, the root of the tree of biconnected components will be processed as v. The resulting closed subproblems will form a disjoint cover of D_j. Together, their solutions form the score-maximizing valid combination of pricing strategies.

Definition 5.8. *Let S be a subproblem (a subset of D_j) processed by the algorithm as a result of the above procedure. A **child subproblem** of S is such a maximal strict subset of S that also is a subproblem processed by the algorithm.*

It follows from the structure of the tree of biconnected components, that child subproblems of S must be disjoint.

5.5.2 Open subproblems

Let us denote the current subproblem as S and the subtree graph of C as G_C. Furthermore, let F_0 be the fragment containing both topmost edges of C. Existence of such a fragment is guaranteed by one of the invariants of the decomposition. Note that F_0 is the only fragment which may contain edges both from G_C and outside it. It is because the only edges from G_C incident to the topmost vertex of C are the two topmost edges of C.

Recall that S is constituted by all the fragments of D_j present in G_C apart from F_0. Hence, C can be split into two parts: one formed by edges belonging to F_0 and the other by edges included in the subproblem. Let u and v be two vertices on C common for both of them. Obviously, there are two u-v paths: the **upper path** contained in F_0 and the **lower path** contained in fragments from S.

Note that no cycle in G is split between G_C and $G \setminus G_C$. Thus, only edges from C belong to skeletons of fragments in S as well as of fragments in $D_j \setminus S$. Hence, for fixed costs of the upper and lower paths, combinations of strategies implemented in S and in the rest of D_j are completely independent. Thus, the algorithm finds multiple optimal combinations of strategies for S, one for each pair of possible lengths of the upper and lower paths. By *up* and *low* we denote the currently processed lengths of respectively the former and the latter.

Remark 5.5. *Let $\mathcal{L}$ be the set of possible lengths of any simple path between border vertices under prices satisfying the rounding lemma (5.1). Then, there are only polynomially many elements of $\mathcal{L}$.*

Proof. Any simple path between two border vertices consists of several whole segments. Thus, its length must be a sum of elements of P from the rounding lemma. Because all positive elements of P are in the form $\frac{mb_{max}}{2^k}$, L consists of zero and multiples of $p_{min} = \min P \setminus \{0\}$ not greater than $m^2 b_{max}$. Since $p_{min} \geq \frac{b_{max}}{2048 \cdot m \cdot |B_j|}$, $|\mathcal{L}|$ is bounded by a polynomial. □

Let us discuss the structure of S. Consider those fragments in S, which contain some edges of C. Let us call them the **upper fragments**. They do not belong to any child problem of S because their topmost edges are contained in C. All the other fragments from S belong to child problems. The algorithm handles open and closed child subproblems differently.

Closed child subproblems: Recall that the combination of stragies for closed subproblems can be chosen independently of the rest of the graph. Since all child subproblems has already been processed, a score-maximizing combination of strategies for each of them is already known. The edges in such subproblems are priced according to those solutions. Resulting scores will be included in the score of each solution to S.

Open child subproblems: Let us consider an open child subproblem S' formed by a cycle C' and let $F'_0 \in D_j$ be the fragment containing the topmost edges of C'. F'_0 must be one of the upper fragments, because any subproblem containing F'_0 would also contain S' and the child subproblems of S are disjoint. Note that F'_0 is the only fragment in $D_j \setminus S'$ sharing a cycle with any fragment forming S'. Thus, for a fixed strategy applied to F'_0 a score-maximizing valid combination of strategies for S' is known. F'_0 contains the upper path of S'. The lower path of S' is an outer segment of $\mathrm{SK}_j(F)$. Hence, a pricing strategy for F'_0 determines the exact length of the former and gives constraints on the length of the latter.

For each strategy for F'_0 the algorithm finds a solution for S' with the biggest score among those computed for compatible upper and lower path lengths. This combination of strategies will be used to price edges within S' only if that particular pricing strategy is applied to F'_0. Thus, its score is added to the score of the strategy of F'_0.

The upper fragments: Pricing of edges in the closed child subproblems is already determined, and the solution to each open child subproblem is also known for a fixed strategy in a corresponding fragment of F'_0. This way we have reduced solving S to finding a valid score-maximizing combination of strategies for the upper segments.

Let us discuss the structure of the upper fragments. Each of them consists of several consecutive edges from C and possibly some more edges from $G_C \setminus C$. Since the topmost edges of each cycle always belong to the same fragment, the upper fragments cannot share a cycle other than C. Thus, they only dependencies between them stem from the fact that they all contribute to and depend on the length of C. The algorithm iterates over all possible lengths of the lower and upper paths, so we can consider the length of C to be fixed and equal $low+up$. Under this assumption it is already known which strategies for upper fragments would be applicable, i.e. their constrains on the length of respective outer segments in C would be satisfied. What remains is finding a score-maximizing combination of them which assigns such prices to individual inner segments forming the lower path, that the prior assumption on its length is true.

We denote the upper segments as $F_1, F_2, \ldots F_q$ in the order of their appearance on the lower path. Let $u_0, u_1, \ldots u_q$ be such vertices that $u_0 = u$, $u_q = v$ and for the other i the vertex u_i is shared by F_i and F_{i+1}. For each fragment F_i and for each $l \in L$

we are only interested in the strategy which makes the length of the u_{i-1}-u_i path equal l and is compatible with the length of C being equal $low + up$. Recall that the intervals, which define constrains on lengths of outer segments, are disjoint. Thus, there is at most one such strategy. Let $score_{up+low,i,l}$ denote its score, including the scores of corresponding strategies for the open subproblems depending on F_i. It may happen that there is no such strategy. In that case we set $score_{up+low,i,l} = -\infty$.

The algorithm ensures that the lower path has the previously assumed length using backpack packing. For each $l \in \mathcal{L}$ and $i \leq k$ let $dp_{up+low,i,l}$ be the maximum score of a combination of strategies for segments $F_1, F_2, \ldots F_i$, which sets the cost of the u_0-u_i path contained in the lower path to l. The algorithm calculates the values of dp using the following formula:

$$dp_{up+low,i,l} = \max_{d \in \mathcal{L}} score_{up+low,i,d} + dp_{up+low,i,l-d}$$

By definition $dp_{up+low,q,low}$ is the maximum score of any valid combination of strategies for fragments in S for the lower path of length low and upper of length up. Along with this value, the algorithm also stores an optimal combination of strategies.

Remark 5.6. *The above procedure processes a subproblem in polynomial time.*

Proof. Since up and low are chosen from the set $\mathcal{L}$, which has polynomially many elements, it is sufficient to show that for fixed values of up and low the procedure finds a solution in polynomial time. Note that processing each closed child subproblem is performed in constant time. Processing an open child subproblem requires checking only the pairs formed by one of the $|\mathcal{L}|^2$ solutions to the subproblem and a strategy for corresponding upper fragment. Hence, it is also done in polynomial time. In the last phase only polynomially many values of dp are computed, which concludes the proof. □

5.5.3 Closed subproblems

Let F be the fragment, for which the closed subproblem S was created. It follows from construction of the subproblems, that F is the only fragment in S not contained in any child subproblem.

Thus, any open child subproblem of S depends only on F. More precisely, F is the fragment containing the upper path of any open child subproblem of S. The algorithm processes open child subproblems as described in Section 5.5.2: for each strategy s for F a score-maximizing compatible solution to the child subproblem is found and its score is added to the score of s.

Edges inside closed child subproblems are simply priced according to previously computed optimal combination of strategies. The scores of those solutions are added to the score of every possible strategy for F.

In order to solve the closed subproblem the algorithm iterates over all possible strategies for F and chooses the one with greatest score (including the scores from child subproblems). An optimal combination of pricing strategies for fragments in S is also stored alongside the maximal score.

5.6 Summing up

The above algorithm constructed a price assignment to all skeleton edges. In this section we show that its revenue is indeed only a constant factor away from SKOPT_j.

First, we have restricted our attention to prices, for which costs of inner segments on j-th level belong to the set P. Let Q_j denote the set of all such price assignments. The rounding lemma (5.1) guarantees that one of such price assignments obtains at least $\frac{\mathrm{SKOPT}_j}{4}$ revenue. Then, the algorithm handled fragments in the current decomposition independently. For every fragment $F \in D_j$ and every strategy s_F the algorithm found prices p_{s_F}, which obtained near-optimal approximate revenue in F (rev_{j,F,s_F}). More precisely, by Corollary 5.7.1, for each such price assignment p_{s_F} the following holds:

$$\mathrm{rev}_{j,F,s_F}(p_{s_F}) \geq \max_{q \in Q_j} \frac{1}{32} \mathrm{rev}_{j,F,s_F}(q) \tag{5}$$

The procedure from Section 5.5 has constructed a global solution p based on the local price assignments p_{s_F} Note that each price assignment $q \in Q_j$ implements a unique valid combination of pricing strategies. By $s_{q,F}$ let us denote the strategy applied to $F \in D_j$ in that combination. Since the prices p originate from a score-maximizing combination of pricing strategies, the following immediately follows from inequality 5:

$$\sum_{F \in D_j} \mathrm{rev}_{j,F,s_{p,F}}(p) \geq \frac{1}{32} \max_{q \in Q_j} \sum_{F \in D_j} \mathrm{rev}_{j,F,s_{q,F}}(q) \tag{6}$$

By Lemma 5.2 high score (approximate revenue) of the combination of pricing strategies results in high revenue. We have already shown that the score of the constructed combination is close to optimal. However, we still must provide a lower bound for the maximal score.

Lemma 5.8. *There exists a price assignment $q \in Q_j$ which implements a combination of strategies with total approximate revenue at most 4 times smaller than the maximal revenue obtained by any prices in Q_j:*

$$\sum_{F \in D_j} \mathrm{rev}_{j,F,s_{q,F}}(q) \geq \frac{1}{4} \max_{q' \in Q_j} \mathrm{rev}_j(q')$$

Proof. The lemma is proven by constructing the prices q. Let $q' \in Q_j$ be a price assignment maximizing $\mathrm{rev}_j(q')$. Consider any segment S in on j-th level. We define the main edges of S as the ones lying on any shortest (under prices q') path connecting its endpoints and fully contained inside it. By p_{min} let us denote the smallest positive element of P from the rounding lemma (5.1). Prices q for edges of S are defined in the following way:

- If the length of S is greater than p_{min} (equivalently at least $2p_{min}$), the prices of main edges of S are two times smaller than in q'. Otherwise, they are set to zero.
- All the other edges in S have the same prices as in q'.

Note that the length of S under prices q belongs to the set P from Lemma 5.1.

Since $\mathrm{rev}_j(q') = \sum_{F \in D_j} \mathrm{rev}_{j,F}(q')$, it is enough to show the inequality for a fixed part $F \in D_j$. Consider an outer segment o in $\mathrm{SK}_j(F)$. Let d and d' be its lengths under prices q and q' respectively. If $d' < 2p_{min}$, then $d = 0$ and $r_{s_{F,q},o} = 0$, i.e. the strategy implemented in F assumes that the length of 0 is exactly zero. Otherwise, $l_{s_{F,q'},o} \geq \frac{d'}{2}$ and $d \leq \frac{d'}{2}$ so $d \leq l_{s_{F,q'},o}$. Hence, $r_{s_{F,q},o} \leq l_{s_{F,q'},o} < d'$. Thus, in both cases the strategy for F implemented by q assumes an upper bound for the length of o not exceeding the actual length of o under prices q'.

Note that using the upper bounds on lengths of outer segments instead of their actual lengths is the only difference between calculating approximate and actual revenue. Thus, the length of each simple path desired by a buyer from $B_{j,F}$ is not more expensive with regard to $\mathrm{rev}_{j,F,s_{F,q}}(q)$ than to $\mathrm{rev}_{j,F}(q')$. Hence, a buyer contributing to $\mathrm{rev}_{j,F}(q')$ also contributes to $\mathrm{rev}_{j,F,s_{F,q}}(q)$. Now we are going to show that she does not contribute much less.

When calculating $\mathrm{rev}_{j,F,s_{F,q}}(q)$ the cost of each path is never understated in comparison to $\mathrm{rev}_{j,F}(q)$. Thus, it is enough to show that under prices q the paths in $\mathrm{SK}_j(F)$ are not much shorter than under q'. Note that if the price assigned to an edge by q is zero, then its price in q' could have been at most p_{min}. Otherwise, it is at most two times cheaper under q then under q'. Hence, for each $e \in \mathrm{SK}_j(F)$, $q(e) \geq \frac{q'(e)}{2} - p_{min}$. Thus, the distance between any two vertices under prices q is not smaller than $\frac{d'}{2} - m \cdot p_{min}$, where d' is the distance between them under q'. Since by calculating approximate revenue the algorithm overstates the distances, this results in the following bound on approximate revenue:

$$\mathrm{rev}_{j,F,s_{F,q}}(q) \geq \frac{1}{2}\,\mathrm{rev}_{j,F}(q') - m \cdot |B_{j,F}| \cdot p_{min}$$

$$\sum_{F \in D_j} \mathrm{rev}_{j,F,s_{q,F}}(q) \geq \frac{1}{2}\,\mathrm{rev}_j(q') - m \cdot |B_j| \cdot \frac{b_{max}}{1024 \cdot m \cdot |B_j|}$$

Since $\mathrm{rev}_j(q') \geq b_{max}$, this concludes the proof. □

By combining Inequality 6 and Lemma 5.8, we achieve the following bound on the score of p:

$$\sum_{F \in D_j} \text{rev}_{j,F,s_{p,F}}(p) \geq \frac{1}{32} \max_{q \in Q_j} \sum_{F \in D_j} \text{rev}_{j,F,s_{q,F}}(q) \geq \frac{1}{4 \cdot 32} \max_{q' \in Q} \text{rev}_j(q')$$

The rounding lemma (5.1) guarantees that $max_{q \in Q_j} \text{rev}_j(q') \geq \frac{1}{4}\text{SKOPT}_j$, so:

$$\sum_{F \in D_j} \text{rev}_{j,F,s_{p,F}}(p) \geq \frac{1}{512}\text{SKOPT}_j$$

Obviously, $\text{SKOPT}_j \geq b_{max}$, so by Lemma 5.2:

$$\text{rev}_j(p) \geq \frac{1}{4} \sum_{F \in D_j} \text{rev}_{j,F,s_{p,F}}(p) \geq \frac{1}{2048}\text{SKOPT}_j$$

By proving this inequality we have shown that the polynomial algorithm for the skeleton subproblem achieves constant approximation ratio.

Chapter 6

Concluding remarks

In Chapters 4 and 5 we have presented polynomial time constant factor approximation algorithms for the non-skeleton and skeleton subproblems. Thus, we have shown that prices achieving at least a constant fraction of optimal revenue can be found in polynomial time for each of the L levels of decomposition (Lemma 3.1). Recall that by setting $k = \left\lceil \log^{\frac{1}{2}} m \right\rceil$ we ensure L to be $\mathcal{O}\left(\frac{\log m}{\log \log m}\right)$. Hence, our polynomial algorithm for the tollbooth problem on cactus graphs yields an $\mathcal{O}\left(\frac{\log m}{\log \log m}\right)$ approximation guarantee on revenue.

It remains an open question whether there exist polynomial time algorithms giving sublogarithmic guarantees on revenue for further generalizations of the tollbooth problem, for example for the cases where the underlying graphs are only assumed to have bounded treewidth.

References

[1] Balcan, M.-F., Blum, A., and Mansour, Y. Item pricing for revenue maximization. In *Proceedings of the 9th ACM Conference on Electronic Commerce* (New York, NY, USA, 2008), EC '08, Association for Computing Machinery, p. 50–59.

[2] Briest, P., and Krysta, P. Single-minded unlimited supply pricing on sparse instances. In *In Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms* (2006), pp. 1093–1102.

[3] Cheung, M., and Swamy, C. Approximation algorithms for single-minded envy-free profit-maximization problems with limited supply. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science* (2008), pp. 35–44.

[4] Gamzu, I., and Segev, D. A sublogarithmic approximation for highway and tollbooth pricing, 2010.

[5] Grandoni, F., and Rothvoss, T. Pricing on paths: A ptas for the highway problem. vol. 45, pp. 675–684.

[6] Guruswami, V., Hartline, J. D., Karlin, A. R., Kempe, D., Kenyon, C., and McSherry, F. On profit-maximizing envy-free pricing. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (USA, 2005), SODA '05, Society for Industrial and Applied Mathematics, p. 1164–1173.