A Scalable Work Function Algorithm for the k-Server Problem

Sharath Raghvendra
Department of Computer Science, Virginia Tech, Blacksburg, USA.

Rachita Sowle
Department of Computer Science, Virginia Tech, Blacksburg, USA.

Abstract

We provide a novel implementation of the classical Work Function Algorithm (WFA) for the k-server problem. In our implementation, processing a request takes $O(n^2 + k^2)$ time per request; where n is the total number of requests and k is the total number of servers. All prior implementations take $\Omega(kn^2 + k^3)$ time per request. Previous approaches process a request by solving a min-cost flow problem. Instead, we show that processing a request can be reduced to an execution of the Dijkstra's shortest-path algorithm on a carefully computed weighted graph leading to the speed-up.

2012 ACM Subject Classification Theory of computation → K-server algorithms

Keywords and phrases K-server, Work Function Algorithm, Minimum-cost Flow

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.31

Funding Work on this paper was supported by NSF under grant CCF 1909171

1 Introduction

In several applications such as emergency response, grocery delivery or virtual memory management, a new request has to be irrevocably assigned to a service provider in real-time. The k-server problem is a simplified abstraction of this problem. In this paper, we present a new implementation of the classical Work Function Algorithm for the k-server problem. We begin by introducing the k-server problem.

Problem Statement: Consider a vertex set V and a weighted complete graph where each edge $(u, v) \in V \times V$ has a cost $d(u, v)$. We assume that $d(\cdot, \cdot)$ is a metric. Given any two multi-sets A and B of points with $A, B \subseteq V$ and $|A| = |B|$, we use $d(A, B)$ to denote the minimum-cost bipartite matching of the points in A to points in B under the distance $d(\cdot, \cdot)$. For an integer $k > 0$, we are given k identical servers and their initial locations, also called the *initial configuration* $\mathcal{C}^0 = \{s_1^0, \ldots s_k^0\}$ in the metric space. A *configuration* is simply any multi-set $\mathcal{C} \subset V$, with $|\mathcal{C}| = k$. We use configurations to denote the locations of the k servers. For any request r_i, a configuration $\mathcal{C}$ *serves* r_i if the location of r_i is contained in the multi-set $\mathcal{C}$. In other words, a server $s \in \mathcal{C}$ that is co-located with r_i serves r_i at zero cost. We are also given a sequence of n requests $R = \langle r_1, \ldots, r_n \rangle$ that arrive over time with r_i arriving at time $t = i$. After r_i arrives, we move the servers to a configuration $\mathcal{C}^i = \{s_1^i, \ldots, s_k^i\}$ that serves r_i. The input to the k-server problem is simply the initial configuration $\mathcal{C}^0$ and the request sequence R.

A *valid* solution to the problem is any sequence of configurations $\sigma = \langle \mathcal{C}^0, \mathcal{C}^1 \ldots, \mathcal{C}^n, \mathcal{C}^{n+1} \rangle$ where $\forall 1 \le i \le n, \mathcal{C}^i$ serves request r_i. Note that, we set the *final configuration* $\mathcal{C}^{n+1}$ to be the same as $\mathcal{C}^n$ unless otherwise specified. Furthermore, we define the points within the final configuration as *anchor nodes*. The cost of σ is denoted by $w(\sigma)$ and $w(\sigma) = \sum_{i=0}^{n} d(\mathcal{C}^i, \mathcal{C}^{i+1})$. The *optimal solution*, denoted by $\sigma^*_{\mathcal{C}^0,R}$ is a valid solution with the smallest possible cost when the input is the initial configuration $\mathcal{C}^0$ and the request sequence is R. We denote $\sigma^*_{\mathcal{C}^0,R}$ as σ^* when the request sequence R and the initial configuration $\mathcal{C}^0$ are obvious from the context. Let R_i be the sequence of first i requests, i.e., $R_i = \langle r_1, \ldots, r_i \rangle$. We use σ_i^* to denote $\sigma^*_{\mathcal{C}^0, R_i}$. Suppose, in addition to the initial configuration $\mathcal{C}^0$ and the request sequence R_i, the problem also specifies a final configuration $\mathcal{C}$, then $\sigma_i^*(\mathcal{C})$ denotes the minimum-cost solution that places the k servers in the initial configuration $\mathcal{C}^0$ after which it serves the i requests, and ends with $\mathcal{C}$ as the final configuration.

In the k-server problem, when a request r_i arrives, one has to immediately and irrevocably commit to a configuration $\mathcal{C}^i$ that serves request r_i. For any algorithm $\mathcal{A}$, let $\sigma_{\mathcal{A}} = \sigma_{\mathcal{A}, \mathcal{C}^0, R}$ be the sequence of configurations that $\mathcal{A}$ chooses for an input request sequence R. Then, for a constant $\alpha > 0$, we say that $\mathcal{A}$ has a competitive ratio of α if there exist another constant $\beta > 0$ such that, over all possible request sequences R, $w(\sigma_{\mathcal{A}, \mathcal{C}^0, R}) \leq \alpha w(\sigma^*_{\mathcal{C}^0, R}) + \beta$.

Next, we describe the work function algorithm for the k-server problem.

Work Function Algorithm: Given a request r_i, the work function algorithm chooses a configuration $\mathcal{C}^i$ that serves request r_i as follows:

$$\mathcal{C}^i = \underset{\mathcal{C}}{\operatorname{argmin}}(w(\sigma_i^*(\mathcal{C})) + d(\mathcal{C}^{i-1}, \mathcal{C})). \tag{1}$$

Note that the minimum is over all possible configurations, i.e., every multi-set of size k. However, work function algorithm can be shown to be a lazy algorithm; see [8, 15], i.e., the configuration $\mathcal{C}^i$ that minimizes (1) is obtained by choosing one server s^* in $\mathcal{C}^{i-1}$ to serve request r_i where s^* is given by

$$s^* = \underset{s \in \mathcal{C}^{i-1}, \mathcal{C} = \mathcal{C}_{i-1} \setminus \{s\} \cup \{r_i\}}{\operatorname{argmin}} \left(w(\sigma_i^*(\mathcal{C})) + d(s, r_i)\right). \tag{2}$$

Prior Work: The k-server problem is central to the theory of online algorithms. For a survey of the problem, see [7]. The problem was first posted by Manasse *et al.* [11] who established a lower bound that as long as a metric space has $k+1$ points, no deterministic algorithm can achieve a competitive ratio better than k. They also showed the competitive ratio of 2 for the 2-server problem. With this as evidence, they conjectured that in fact there is a k-competitive algorithm for this problem for any metric space. This conjecture is the celebrated *k-server conjecture*. Since then, the k-server conjecture has also been shown to be true for the line metric (1-dimensional Euclidean space) [2] and the tree metric [3]. It was shown that the Work Function Algorithm achieves a competitive ratio of $2k - 1$ on any metric space [8]. There has not been any significant progress on this conjecture since then. On the other hand, there has been substantial work on the randomized version of the k-server conjecture; see for instance Bubeck et al. [1] and Lee [10].

The analysis of the WFA in [8] was based on an exponential time dynamic programming implementation which processes the ith request r_i by solving equation (2).

The problem of finding the offline optimal solution for the k-server problem can be carefully modelled as a minimum-cost flow problem [2] where each edge has a unit capacity. Every unit of flow corresponds to a path taken by a server. In this flow network, every request is represented by two nodes connected by an edge of weight $-\infty$. This forces any minimum-cost flow to visit every request. The optimal solution to this flow network of $2n + k + 2$ nodes can be found in $O(n^2 k)$ time.

For the online case, processing the ith request r_i requires solving equation (2). Similar to the offline case, evaluating (2) explicitly can be modelled as computing k distinct minimum-cost flow values, each of which can take $\Theta((i+k)^2 k)$ time [2]. This observation leads to an $O((i+k)^2 k^2)$ time algorithm for the WFA [16]. Using clever observations, evaluation of (2) can be reduced to computation of a single minimum-cost flow which takes $O(k(i+k)^2)$ time [15].

Rudec and Manger [17] presented an alternative approach for computing an offline solution to the k-server problem. Instead of creating a flow network with edges of $-\infty$ cost, they define a graph in the original metric space and define the notion of *regular flow* to be any flow in which each request is served by at least one server. One can move from any regular flow to another one using the so-called *up-down cycles*. Upon adding a new request, they show that finding the minimum-cost regular flow can be done by finding the most negative up-down cycle which they accomplish by conducting an exhaustive search. They argue that there is empirical benefit to this approach despite the worst-case execution time of this algorithm being slower than that of [15]. They also extend this approach to the work function algorithm.

Our Results and Approach: In this paper, we present a new implementation of the classical work function algorithm. Similar to Rudec and Manger [17], after processing each request, we maintain a valid solution that serves all the requests seen so far. Moreover, our algorithm processes the ith request by executing a single Dijkstra's shortest path search on a weighted graph in $O((i+k)^2)$ time which is faster than previous methods that take $\Omega(k(i+k)^2)$ time [15, 17].

For every server $s \in \mathcal{C}^{i-1}$, and $\mathcal{C} = \mathcal{C}^{i-1} \setminus \{s\} \cup \{r_i\}$, our algorithm computes (2) explicitly and then computes the minimum across all k choices of s. We show that the symmetric difference between $\sigma = \sigma^*_{i-1}(\mathcal{C}^{i-1})$ and $\sigma' = \sigma^*_i(\mathcal{C})$ is a trail[1] T whose edges alternate between those in σ and σ'. We refer to this as an *augmenting trail* and define its net-cost to be $w(\sigma') - w(\sigma)$. In order to find $w(\sigma')$, we simply have to identify the minimum net-cost augmenting trail that starts at r_i and ends at s. Our augmenting trails can be seen as a variant of the up-down cycles maintained by Rudec and Manger [17]. However, instead of conducting an exhaustive search, we describe an efficient algorithm (similar to the Kuhn-Munkres algorithm [9]) to find this minimum net-cost augmenting trail.

Using a graph search algorithm to find a minimum net-cost augmenting trail in the residual graph can be difficult since these algorithms find simple paths and not trails. In order to assist in the search of an augmenting trail, we define a weighted graph which we refer to as the *alternating graph*. Any augmenting trail in the residual graph maps to a directed path in the alternating graph and every directed path in the alternating graph corresponds to an alternating trail in the residual graph (See Lemma 2 and Figure 1).

Critically, we also store a set of weights on the vertices of the alternating graph. These weights satisfy a set of feasibility constraints, one for each edge in the alternating graph. Vertex weights have been used to speed-up computation for a shortest path in a graph with negative edge weights, for instance, in Johnson's algorithm [4]. These weights allow us to reduce the problem of finding minimum net-cost augmenting trail from r_i to every server $s \in \mathcal{C}^{i-1}$ to a single execution of Dijkstra's search procedure. Consequently, one can find the optimal choice in (2) in $O((i+k)^2)$ time. After the optimal choice is identified, we augment the solution to serve request r_i. This may create many new edges, delete existing edges and also change the cost of some edges in the alternating graph. Somewhat surprisingly, despite the many updates to the alternating graph, we show that the vertex weights maintained by our algorithm continue to satisfy the feasibility constraints for all edges.

Significance of our Result: Minimum net-cost paths have been central to the design of algorithms for the closely related online minimum metric bipartite matching (OMBM) problem [6, 5, 13]. Unlike in the k-server problem where a server can serve any number of requests, in the OMBM problem, a server can serve no more than one request. Recent

[1] Recollect that a trail is a (possibly non-simple) path that does not repeat edges

analysis of algorithms for the OMBM rely on the behavior of the vertex weights (also called *dual weights*) maintained while computing the minimum net-cost paths [12, 14]. Similarly, we hope that our formulation of the WFA as computing a minimum net-cost augmenting trail as well as the use of vertex weights can shed light into the dynamics of the WFA leading to an improved analysis.

2 Preliminaries

Recollect that a valid solution is provided by a sequence of configurations $\sigma = \langle \mathcal{C}^0, \mathcal{C}^1, \dots, \mathcal{C}^n, \mathcal{C}^{n+1}\rangle$. However, lazy valid solutions can also be represented as a set of k paths $\{\Gamma_1, \dots, \Gamma_k\}$ taken by each of the k servers. More precisely, for any $1 \le i \le n$ and $1 \le j \le k$, these paths satisfy the following properties:

(P1) For each server s_j, its path Γ_j starts at the location of s_j in the initial configuration. After the first vertex, Γ_j consists of a sequence of requests served by s_j in increasing order of their arrival time. Finally, the last vertex of Γ_j is the location of s_j in the final configuration.

(P2) Every request in R_i participates in exactly one of the k paths.

Furthermore, it can be shown that any set of paths $\{\Gamma_1, \dots, \Gamma_k\}$ that satisfies (P1) and (P2) will be a valid solution .

The work function algorithm is a lazy algorithm. Therefore, we can represent the solution it produces as k paths satisfying (P1) and (P2). Next, we introduce the notations that are needed to describe an efficient implementation of the work function algorithm.

Notations: Throughout the rest of the paper, we consider directed graphs. We assume that any edge (u, v) is directed from u to v unless otherwise stated. Let σ_i be a valid solution to the first i requests of the k-server problem. Let $\{\Gamma_1, \Gamma_2, \dots, \Gamma_k\}$ be the set of k paths satisfying (P1) and (P2). For $1 \le j \le k$, and for any vertex v on a path Γ_j where v is not an anchor node, let $f(v)$ denote the location of the next vertex on the path. Similarly, for any vertex v on a path Γ_j where v is not a location in the initial configuration, let $p(v)$ denote the vertex that precedes v in the path Γ_j. In our algorithm, for any given valid solution σ_i, we create a directed graph G_i called the *residual graph* as follows. The vertex set V_i of G_i contains all vertices participating in any of the k paths, i.e., $V_i = \bigcup_{l=1}^{k} \left(\bigcup_{v \in \Gamma_l} v \right)$. There are two types of edges in the edge set E_i of G_i

- *Forward edges:* For each of the k paths and any vertex $v \in V$ where v is not an anchor node, we add a directed edge from v to $f(v)$ denoting that the server at v moves to $f(v)$.
- *Backward edges:* For every request r_j, we add a backward edge to $r_{j'}$ provided $j' < j$ and $j \ne f(j')$. This edge is directed from r_j to $r_{j'}$. We also add a backward edge directed from r_j to the k vertices of the initial configuration.

We refer to the residual graph with respect to σ_i as G_i. The k paths $\{\Gamma_1, \dots, \Gamma_k\}$ are represented as k directed paths consisting of all the forward edges in G_i. The backward edges, on the other hand, are not in the solution.

The set of all forward edges of a residual graph G_i corresponds to a valid solution if and only if

(Q1) The forward edges are directed from an earlier request to a later request, and,

(Q2) Every request r has exactly one incoming forward edge and one outgoing forward edge, every vertex from the initial configuration has one outgoing forward edge and every anchor node has one incoming forward edge.

One can prove this by showing their equivalence to (P1) and (P2) . Next, we define alternating and augmenting trails that play a critical role in processing a request.

Alternating Trails: Recollect that, in graph theory, a *trail* T is a path that is not necessarily a simple path but it does not repeat edges. We define an *alternating trail* T in G_i as a directed trail that alternates between forward and backward edges and ends at an anchor node.

When a new request r_{i+1} arrives, we include the request r_{i+1} and extend the residual graph to create an *extended* graph G^0_{i+1} from G_i as follows. The new vertex set V_{i+1} is $V_i \cup \{r_{i+1}\}$. The edges incident on r_{i+1} are as follows: for each vertex $v \in V_i$, if v is not an anchor node, we add a backward edge directed from r_{i+1} to v. Figure 1(a) shows an example of an extended graph where $r_{i+1} = r_6$ with i_1, i_2, i_3, i_4 being the nodes in initial configuration and a_1, a_2, a_3, a_4 are the anchor nodes. Any alternating trail T in the extended graph G^0_{i+1} that starts at r_{i+1} is an *augmenting trail.* Every edge going out of r_{i+1} in the extended graph G^0_{i+1} is a backward edge. Therefore, an augmenting trail T starts with a backward edge and ends at an anchor node. For example, in Figure 1(a), $\langle r_6, i_1, r_2, i_2, r_3, r_2, r_4, r_1, a_3\rangle$ is an augmenting trail.

Alternating Graph: Finding augmenting trails can be tricky. Typical graph search algorithms only find paths and not trails. In order to assist us in finding an augmenting trail efficiently, we define a different directed graph called the *alternating graph* for G^0_{i+1} and denote it by $\mathcal{G}^0_{i+1}(\mathcal{V}^0_{i+1}, \mathcal{E}^0_{i+1})$. Every directed simple path in this alternating graph $\mathcal{G}^0_{i+1}$ maps to a unique alternating trail in G^0_{i+1} and every augmenting trail T in G^0_{i+1} maps to a unique simple path in the alternating graph $\mathcal{G}^0_{i+1}$ which we refer to as the *augmenting path* (Lemma 2).

Thus, finding augmenting trails in G^0_{i+1} reduces to finding augmenting paths in $\mathcal{G}^0_{i+1}$ which can be done via graph search algorithms. We describe the alternating graph next. The vertex set $\mathcal{V}^0_{i+1}$ of the alternating graph is the same as that of G^0_{i+1}, i.e., $\mathcal{V}^0_{i+1} = V_{i+1}$. The edge set of the alternating graph, $\mathcal{E}^0_{i+1}$, is defined as follows. For every vertex v, if v has a backward edge to a node v', then we add a directed edge from v to $f(v')$ in $\mathcal{E}^0_{i+1}$. Figure 1(a) is an extended graph and Figure 1(b) is its alternating graph. For any directed edge $(v, f(v'))$ in $\mathcal{G}^0_{i+1}$, denoted by $\textsc{Proj}(v, f(v'))$ is the backward edge (v, v') concatenated with the forward edge $(v', f(v'))$, i.e., $\textsc{Proj}(v, f(v')) = \langle (v, v'), (v', f(v'))\rangle$. For example, the projection of an edge (r_3, r_4) (Figure 1(b)) in the alternating graph consists of the edges $\langle (r_3, r_2), (r_2, r_4)\rangle$ (Figure 1(a)) of the residual graph. For any path P in the alternating graph, its projection is simply the concatenation of the projection of the individual edges. The highlighted augmenting path $\langle r_6, r_2, r_3, r_4, a_3\rangle$ (Figure 1(b)) when projected gives the highlighted augmenting trail $\langle r_6, i_1, r_2, i_2, r_3, r_2, r_4, r_1, a_3\rangle$ (Figure 1(a)). The construction of alternating graph and the definition of projection will also extend to the residual graph G_{i+1} in a straight-forward way. The alternating graph for G_{i+1} will be referred to as $\mathcal{G}_{i+1}(\mathcal{V}_{i+1}, \mathcal{E}_{i+1})$.

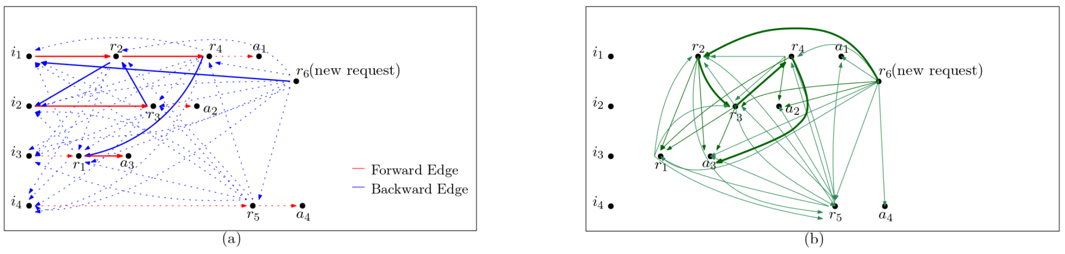


Figure 1 Example of an (a) Extended graph G^0_6 and (b) its alternating graph $\mathcal{G}^0_6$

▶ **Lemma 1.** *For any two edges (u, v) and (u', v') in $\mathcal{G}_i^0$, their projections are edge-disjoint if and only if the head of both of the edges are distinct, i.e., $v \neq v'$.*

Proof. The projection of (u, v) is $\langle(u, p(v)), (p(v), v)\rangle$ and the projection of (u', v') is $\langle(u', p(v')), (p(v'), v')\rangle$. Note that for every vertex s, there is a unique previous vertex $p(s)$. Therefore, if these projections are edge-disjoint and $(p(v), v)$ and $(p(v'), v')$ are distinct edges, then v and v' must be distinct points, i.e., $v \neq v'$. If $v \neq v'$, then $p(v) \neq p(v')$. Therefore, the forward edges $(p(v), v)$ and $(p(v'), v')$ are two vertex-disjoint edges implying that the projections of (u, v) and (u', v') are edge-disjoint. ◀

▶ **Lemma 2.** *For every directed simple path P in the alternating graph $\mathcal{G}_i^0$ that ends at an anchor node, its projection $T = \textsc{Proj}(P)$ is an alternating trail. Furthermore, for every alternating trail T in G_i^0 where the first edge of T is a backward edge, there is a directed simple path P in $\mathcal{G}_i^0$ such that $\textsc{Proj}(P) = T$.*

Proof. The in-degree of any vertex on a simple directed path is at most one. Therefore, for any two edges (u, v) and (u', v') on a simple directed path P, $v \neq v'$. From Lemma 1, the projections of (u, v) and (u', v') are edge-disjoint. Therefore, the projection T of P which is simply the concatenation of projections of all the edges of P will be a path that does not repeat any edges, i.e., T is a trail. By construction, T starts with a backward edge, alternates between backward and forward edges, and ends at an anchor node, i.e., T is an alternating trail.

For any alternating trail T in G_i^0, let the backward edges be $(u_1, v_1), (u_2, v_2), \ldots, (u_j, v_j)$ in the order in which they appear on the trail. Similarly let the forward edges be $(v_1, f(v_1)), (v_2, f(v_2)), \ldots, (v_j, f(v_j))$, i.e., $T = \langle(u_1, v_1), (v_1, f(v_1)), (u_2, v_2), (v_2, f(v_2)), \ldots, (u_j, v_j), (v_j, f(v_j))\rangle$. By our assumption, the first edge of the alternating trail (u_1, v_1) must be a backward edge, and, $f(v_j)$ must be an anchor node and for $1 \leq t < j$, $f(v_t) = u_{t+1}$. For each $1 \leq t \leq j$, the pair of edges $(u_t, v_t)(v_t, f(v_t))$ is represented by a unique directed edge $(u_t, f(v_t))$ in $\mathcal{G}_i^0$. We can therefore *lift* T to a path P in $\mathcal{G}_i^0$ by simply replacing successive pairs $(u_t, v_t)(v_t, f(v_t))$ with $(u_t, f(v_t))$. The resulting sequence of edges is $P = \langle(u_1, f(v_1)), (u_2, f(v_2)), \ldots, (u_j, f(v_j))\rangle$. Since for $1 \leq t < j$, $f(v_t) = u_{t+1}$, P is precisely the directed path $\langle u_1, u_2, \ldots, u_j, f(v_j)\rangle$. Furthermore, since T is a trail and does not repeat any edges, for any two edges (u, v) and (u', v') in P its projections will be edge-disjoint. Therefore, from Lemma 1, $v \neq v'$ and so, P is a simple path. ◀

Augmentation: Consider any augmenting trail T in the extended graph G_{i+1}^0 that starts at r_{i+1} and ends at an anchor node a. We *augment* a valid solution σ_i (represented by the extended graph G_{i+1}^0) along an augmenting trail T to produce a solution σ_{i+1} and the residual graph G_{i+1} as follows:

To obtain the residual graph G_{i+1} from the extended graph G_{i+1}^0, we can simply reverse the direction of all the edges on the augmenting trail T and relabel the forward edges as backward and all backward edges as forward. Finally, we remove the incoming forward edge to the anchor node a and add a new forward edge from r_{i+1} to the anchor node a and update the location of a to that of r_{i+1}.

Equivalently, one can consider modifying the k paths $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_k\}$ of σ_i by removing all forward edges of T and adding all backward edges of T to obtain the k paths $\{\Gamma'_1, \Gamma'_2, \ldots, \Gamma'_k\}$ of σ_{i+1}. It can be shown that the solution σ_{i+1} is a valid solution that also serves request r_{i+1} . One can also generalize the augment operation for alternating trails and cycles. We refer to this generalized operation as the *flip* operation and use it in Section 4.2. .

We define the net-cost of an augmenting trail T to be

$$\Phi(T) = \sum_{(u,v) \text{ is backward}} d(u,v) - \sum_{(u,v) \text{ is forward}} d(u,v).$$

Note that the net-cost of an augmenting trail T with respect to σ_i is the change in the cost due to augmenting σ_i along T. Therefore, one can express net-cost as $\Phi(T) = w(\sigma_{i+1}) - w(\sigma_i)$.

For any edge (u, v) in the alternating graph, we set its cost $c(u, v) = d(u, p(v)) - d(p(v), v)$ and for any simple path P, let $c(P) = \sum_{(u,v)\in P} c(u, v)$ denote its *net-cost*. Note that the cost of any edge in the alternating graph can be negative. For any simple augmenting path P in the alternating graph $\mathcal{G}^0_{i+1}$, its net-cost $c(P)$ is simply the net-cost of its projection $\Phi(\textsc{Proj}(P))$.

▶ **Lemma 3.** *The net-cost of an augmenting path P in the alternating graph is equal to the net-cost of its projection.*

Proof. Given an augmenting path P in the alternating graph let the augmenting trail P' be its projection. Note that the set $B = \{(a, p(b)) \mid (a, b) \in P\}$ is the set of all backward edges of P'. Similarly, the set $F = \{(p(b), b) \mid (a, b) \in P\}$ is the set of all forward edges of P'. Therefore, the net-cost of P is $\sum_{(a,b)\in P} c(a, b) = \sum_{(a,b)\in P}(d(a, p(b)) - d(p(b), b)) = \sum_{(u,v)\in B} d(u, v) - \sum_{(u,v)\in F} d(u, v) = \Phi(P')$. ◀

Let $y(\cdot)$ be a weight associated with every vertex of the alternating graph. We say that any valid solution σ_i and the weight function $y(\cdot)$ is *feasible* if for any edge (a, b) directed from a to b

$$y(a) - y(b) \leq c(a, b). \tag{3}$$

We say that any edge satisfying this inequality is *feasible*. We define *slack* of any edge (a, b) directed from a to b to be $c(a, b) + y(b) - y(a)$ and denote it by $s(a, b)$. Given these notations, we are ready to describe our algorithm.

3 The Algorithm

After processing i requests, our algorithm will maintain a feasible valid solution $\sigma = \sigma_i$. We refer to this as the *offline* solution. Initially, the weight $y(v)$ for every vertex $v \in \mathcal{C}^0$ is set to 0 and the offline solution σ is empty. For $i \geq 0$, using the alternating graph $\mathcal{G}^0_{i+1}$, our algorithm will identify an appropriate augmenting trail T in the extended graph G^0_{i+1}. Recollect that T ends at an anchor node a. The algorithm then moves the server located at a and that served request $p(a)$ to serve request r_{i+1}. The offline solution σ is updated by augmenting σ_i along T leading to a valid solution $\sigma = \sigma_{i+1}$. The algorithm consists of four steps[2]:

(1) *Augmenting path search:* Let G' be identical to this alternating graph $\mathcal{G}^0_{i+1}$ except the cost of any edge (a, b) is replaced by its slack $s(a, b)$. Note that G' is a graph with only non negative edge-costs. The algorithm executes Dijkstra's algorithm on G' with r_{i+1} as the source. Dijkstra's algorithm returns the shortest path from r_{i+1} to every other vertex in $\mathcal{V}^0_{i+1}$. Let, for any vertex $v \in \mathcal{V}^0_{i+1}$, ℓ_v be its shortest path cost as returned by Dijkstra's algorithm from the source r_{i+1}.

[2] An implementation of this algorithm is available here: `https://github.com/RachitaS/ScalableWorkFunction_Public`

(2) *Determine net-cost:* Next, we compute the minimum net-cost augmenting path from r_{i+1} to each of the k anchor nodes $\{a_1, \ldots, a_k\}$. For any anchor node $a_j \in \{a_1, \ldots, a_k\}$, we set the minimum net-cost to be $\Phi_j = \ell_{a_j} - y(a_j)$ and the path P_j corresponding to this net-cost is the shortest path from r_{i+1} to a_j in G' as returned by Dijkstra's algorithm (Step 1).

(3) *Choose server:* Let $a_m = \arg\min_{a_j \in \{a_1,\ldots,a_k\}}(d(a_j, r_{i+1}) + \Phi_j)$ and let s be the server located at a_m with $p(a_m)$ as its last served request. We assign s to serve request r_{i+1}.

(4) *Update offline solution:* We update the offline solution as follows: **(a)** For any vertex $v \in V_{i+1}$, if $\ell_v < \ell_{a_m}$, we set its weight $y(v) \leftarrow y(v) + \ell_{a_m} - \ell_v$. **(b)** After updating the weights, we augment σ_i along P_{a_m} to obtain $\sigma = \sigma_{i+1}$ and update the edges of the alternating graph to reflect the new solution σ. We also set $y(a_m) \leftarrow 0$.

Our algorithm maintains the following two invariants at all times:

(I1): The offline solution σ along with the weights $y(\cdot)$ is a valid and feasible solution, and,
(I2): Let $\mathcal{C}^i$ be the final configuration of σ_i. Then, $\sigma_i = \sigma_i^*(\mathcal{C}^i)$. Furthermore, for every anchor node $a_j \in \{a_1, \ldots, a_k\}$, let $\mathcal{C}_j^{i+1} = \mathcal{C}^i \setminus \{a_j\} \cup \{r_{i+1}\}$. Then $\Phi_j = w(\sigma_{i+1}^*(\mathcal{C}_j^{i+1})) - w(\sigma_i^*(\mathcal{C}^i))$.
The proofs of these invariants are given in Section 4. Note that, after each request is processed the set of edges in the alternating graph can change substantially. Despite this, our weight updates guarantee that every newly added edge in the alternating graph continues to be feasible (Section 4.1).

Efficiency Note that the $|V_{i+1}| = i + 1 + 2k$ and $|E_{i+1}| = O((i+k)^2)$. The extended graph and alternating graphs also have identical bounds. Step 1 of the algorithm requires computation of G' and an execution of Dijkstra's algorithm on G' which takes $O((i+k)^2))$ time. Step 2 of the algorithm requires constant time computation for each of the anchor nodes and therefore takes $O(k)$ time. The paths P_j computed in Step 2 is can be compactly represented using the shortest path tree that is returned by Dijkstra's algorithm. Therefore, computing P_j does not require any additional time. Choosing the server in Step 3 can be performed by simply accessing the cost between the r_{i+1} and each of the k servers and computing the one that minimizes $\Phi_j + d(a_j, r_{i+1})$. Step 3, therefore, takes only $O(k)$ time. Step 4(a) requires us to update the weight at each vertex which can be done in $O(i+k)$ time. Step 4(b) requires augmenting and updating the residual and alternating graphs each of which can be performed in $O((i+k)^2)$. Therefore, the time taken to process each request is dominated by $O(i^2 + k^2) = O(n^2)$.

Next, assuming the invariants hold, we will show that the algorithm picks the same server as the Work Function Algorithm.

Correctness: The following lemma establishes a link between the net-cost of an augmenting path and the sum of the slacks along its edges.

▶ **Lemma 4.** *Suppose σ_i and the weights $y(\cdot)$ form a feasible solution. For any augmenting path P in the alternating graph that starts at r_{i+1} and ends at an anchor node a, its net-cost is*

$$\Phi(P) = y(r_{i+1}) - y(a) + \sum_{(u,v) \in P} s(u,v). \tag{4}$$

Proof. Every vertex $v' \in P$ with the exception of the first vertex r_{i+1} and the last vertex a will have an incoming edge (u', v') and an outgoing edge (v', w') in P. The weight of v', $y(v')$ is added with respect to (v', w') and subtracted with respect to the edge (u', v') and therefore, the net-contribution of v' to Equation (4) is zero. The first vertex r_{i+1} participates in the first edge of P and contributes $+y(r_{i+1})$ to Equation (4). The last vertex a participates only in the last edge and contributes $-y(a)$ to Equation (4). ◀

From Invariant (I1), Equation 4 and since $y(r_{i+1}) = 0$, the minimum net-cost path P_j from r_{i+1} to some anchor node a_j in the alternating graph $\mathcal{G}^0_{i+1}$ is also the augmenting path that minimizes the sum of slacks along its edges. From invariant (I1) all slacks are non-negative and so, P_j will be the augmenting path returned by the execution of Dijkstra's algorithm in Step 1 of the algorithm. Furthermore, $\ell_{a_j} = \sum_{(u,v)\in P_j} s(u,v)$. Therefore, from Equation 4, we conclude that $\Phi_j = \Phi(P_j) = \ell_{a_j} - y(a_j)$. Therefore, Step 2 of the algorithm will correctly compute the minimum net-cost augmenting path to every anchor node $a_j \in \{a_1, \dots, a_k\}$.

Step 3 of the algorithm selects the server located at the anchor node $a_m = \mathrm{argmin}_{a_j\in\{a_1,\dots,a_k\}}(d(a_j, r_{i+1}) + \Phi_j)$. Let $X_j = \mathcal{C}^{i+1}_j = \mathcal{C}^i \setminus \{a_j\} \cup \{r_{i+1}\}$. By Invariant (I2), $(d(a_j, r_{i+1}) + \Phi_j) = d(a_j, r_{i+1}) + w(\sigma^*_{i+1}(X_j)) - w(\sigma^*_i(\mathcal{C}^i))$ and, $\mathrm{argmin}_{a_j\in\{a_1,\dots,a_k\}}(d(a_j, r_{i+1}) + \Phi_j)$ is

$$\begin{aligned}&= \operatorname*{argmin}_{j\in\{1,\dots k\}} (d(a_j, r_{i+1}) + w(\sigma^*_{i+1}(X_j)) - w(\sigma^*_i(\mathcal{C}^i)))\\ &= \operatorname*{argmin}_{j\in\{1,\dots,k\}} (d(a_j, r_{i+1}) + w(\sigma^*_{i+1}(X_j)).\end{aligned}$$

The last equality follows from the fact that $w(\sigma^*_i(\mathcal{C}^i))$ is the same for every choice of j. Thus, we choose the same server as required by the work function algorithm.

4 Proof of Invariants

4.1 Proof of Invariant (I1)

Recall the definition of feasibility. Any valid solution σ_i with the weight function $y(\cdot)$ associated with each vertex of its alternating graph is *feasible* if for every edge (a, b) directed from a to b in its alternation graph satisfies the following equation.

$$y(a) - y(b) \le c(a, b). \quad (5)$$

Furthermore, any edge satisfying the above inequality is *feasible.*

We prove a slightly stronger version of Invariant (I1)

(I1): The offline solution σ along with the weights $y(\cdot)$ maintained by the algorithm is a valid and feasible solution. Furthermore, for any forward edge (u, v) in the residual graph of σ,

$$y(v) \ge d(u, v). \quad (6)$$

We can prove this invariant by induction. At time $t = 0$, we have the servers in the initial configuration. The vertex set of the initial residual graph G_0 only contains the vertices for the initial configuration and the anchor nodes. The edge set of the initial residual graph G_0 contains only forward edges directed from each server in the initial configuration to its anchor node. Since there are no backward edges in G_0, the alternating graph $\mathcal{G}_0$ does not have edges. Therefore, σ_0 and the vertex weights $y(\cdot)$ are trivially feasible.

Let the solution σ_i after serving i requests be a valid and feasible solution and let $y(\cdot)$ at the end of processing request r_i satisfy (6). Given this, we will now prove that the solution σ_{i+1} is valid and feasible and the updated vertex weight satisfies (6). Note that each new request arrives with a default weight 0. Given a valid feasible solution σ_i along with the vertex weight function $y(\cdot)$, the algorithm first constructs the extended graph G^0_{i+1}. Lemma 6 shows that the corresponding alternating graph $\mathcal{G}^0_{i+1}$ with weight function $y(\cdot)$ continues to be feasible. Steps 1, 2 and 3 do not modify σ_i or the vertex weights. Therefore, σ_i continues to remain valid and feasible till the end of Step 3.

In Step 4(a) of the algorithm, the weights on the vertices of $\mathcal{G}^0_{i+1}$ are updated. Let $y(\cdot)$ be the weights prior to executing step 4(a) and let $y'(\cdot)$ be the weights after executing step 4(a). Lemma 8 proves that $\mathcal{G}^0_{i+1}$ along with the updated vertex weights $y'(\cdot)$ remain feasible. In Step 4(b), the algorithm augments the solution σ_i along the augmenting trail T (as determined in Step 3) leading to σ_{i+1}. Lemma 10 shows that the solution σ_{i+1} remains valid and feasible. Finally, in Lemma 11, we argue that the vertex weights at the end of Step 4(b) satisfies (6).

▶ **Lemma 5.** *Suppose P is the augmenting path from r_{i+1} to the anchor node a_m chosen in step 3 of the algorithm, then the slack $s(u,v)$ on every edge (u,v) of P after the vertex weight update in step 4(a) is zero.*

Proof. Let $y(\cdot)$ be the vertex weight prior to Step 4(a) and $y'(\cdot)$ be the vertex weight after Step 4(a). By our choice in Step 3, P is the shortest path computed by Dijkstra's algorithm from r_{i+1} to a_m in G'. Since ℓ_{a_m} is the cost of P in G', every vertex $v \in P$ has a shortest path cost at most ℓ_{a_m}, i.e., $\ell_v \le \ell_{a_m}$. Furthermore, by the optimal sub-structure property of shortest paths, for any edge $(u,v) \in P$, $\ell_v - \ell_u = s(u,v) = c(u,v) + y(v) - y(u)$ or $\ell_v - \ell_u = c(u,v) + y(v) - y(u)$.

$$\begin{aligned} c(u,v) + (y(v) - \ell_v) - (y(u) - \ell_u) &= 0, \\ c(u,v) + (y(v) - \ell_v + \ell_{a_m}) - (y(u) - \ell_u + \ell_{a_m}) &= 0, \\ c(u,v) + y'(v) - y'(u) &= 0. \end{aligned}$$

The second to last equality is obtained by simply adding and subtracting ℓ_{a_m} to the LHS. The last equality follows from the fact that $\ell_v \le \ell_{a_m}$ and $\ell_u \le \ell_{a_m}$ and the update of the vertex weights defined in Step 4(a). ◀

▶ **Lemma 6.** *Given a valid feasible solution σ_i, all the edges of the alternating graph $\mathcal{G}^0_{i+1}$ are feasible.*

Proof. From the inductive hypothesis, σ_i is a feasible solution i.e. the edges of $\mathcal{G}_i$ satisfy (5), and, the weights $y(\cdot)$ satisfies (6). The extended graph G^0_{i+1} is created by the addition of r_{i+1} to the vertex set of G_i in the vertex set. Furthermore, for every $j \le i$ a backward edge is added from r_{i+1} to r_j and an edge $(r_{i+1}, f(r_j))$ is added to $\mathcal{G}^0_{i+1}$. We show that for every such edge, $(r_{i+1}, f(r_j)) \in \mathcal{G}^0_{i+1}$, the feasibility condition (5) holds. The cost of the edge $(r_{i+1}, f(r_j))$ is

$$c(r_{i+1}, f(r_j)) = d(r_{i+1}, r_j)) - d(r_j, f(r_j)) \tag{7}$$
$$\ge -d(r_j, f(r_j)). \tag{8}$$

Note that the vertex weight of r_{i+1}, $y(r_{i+1})$ is set to 0. By the inductive hypothesis, $-y(f(r_j)) \le -d(r_j, f(r_j))$. Adding $y(r_{i+1})$ to the LHS and 0 to the RHS, we get $y(r_{i+1}) - y(f(r_j)) \le -d(r_j, f(r_j))$ or $y(r_{i+1}) - y(f(r_j)) \le c(r_{i+1}, f(r_j))$. The last inequality follows from 8. This implies that the edge (r_{i+1}, v) satisfies (5). ◀

Feasibility after Step 4(b): Step 4(b) in the algorithm augments the solution along an augmenting trail T. In doing so, the alternating graph $\mathcal{G}^0_{i+1}$ is updated to $\mathcal{G}_{i+1}$. The edges $\mathcal{E}_{i+1}$ of $\mathcal{G}_{i+1}$ may include several new edges that were not in $\mathcal{G}^0_{i+1}$. Furthermore, there may be edges whose costs $c(\cdot,\cdot)$ change due to augmentation. We classify all such edges in the alternating graph $\mathcal{G}_{i+1}$ as *affected* edges. The following two lemmas establishes important properties of the affected edges.

▶ **Lemma 7.** *Let P be the augmenting path from r_{i+1} to an anchor node a_m in the alternating graph $\mathcal{G}^0_{i+1}$ that is computed in Step 3 of our algorithm. Given any affected edge (u,v) in $\mathcal{G}_{i+1}$, let $\langle(u,x),(x,v)\rangle$ be its projection. Then, $v \neq a_m$ and (v,x) is a backward edge on the augmenting trail $\textsc{Proj}(P)$.*

Proof. Let P be the augmenting path from r_{i+1} to an anchor node a_m in the alternating graph $\mathcal{G}^0_{i+1}$ that is computed in Step 3 of our algorithm. And let T be the augmenting trail in G^0_{i+1} such that $T = \textsc{Proj}(P)$.

First we will prove that given any affected edge (u,v) in $\mathcal{G}_{i+1}$, $v \neq a_m$. After augmentation along T, we add the forward edge (r_{i+1}, a_m). Since all backward edges are directed from a later request to an earlier request and since r_{i+1} is the latest request in the residual graph G_{i+1}, there are no in-coming backward edges to r_{i+1}. Therefore, by its description, there will not be any incoming edges to a_m in the alternating graph $\mathcal{G}_{i+1}$ and so $v \neq a_m$.

Since (u,v) is an affected edge, at least one of the edges in its projection $\langle(u,x),(x,v)\rangle$ is newly introduced by the augment operation. We claim that the forward edge (x,v) was added by the augment operation in Step 4(b) of the algorithm. Suppose, for the sake of contradiction, (x,v) was not added in Step 4(b), i.e., (x,v) is a forward edge in G^0_{i+1}. Therefore, (u,x) must be the backward edge that was newly added by the augment operation. We can conclude that the augmenting trail T contains the forward edge (x,u), implying (x,u) is a forward edge in G^0_{i+1}. Note that (x,u) and (x,v) are both forward edges in G^0_{i+1} which contradicts the fact that σ_i is a valid solution. Therefore, we conclude that the forward edge (x,v) was introduced by the augment operation in Step 4(b), i.e., (v,x) is a backward edge in the augmenting trail T. ◀

▶ **Lemma 8.** *The edges of the alternating graph $\mathcal{G}^0_{i+1}$ remains feasible after the vertex weight update in step 4(a).*

Proof. The alternating graph $\mathcal{G}^0_{i+1}$ before the execution of Step 4(a) is feasible. Let $y(\cdot)$ (resp. $y'(\cdot)$) denote the vertex weights before (resp. after) the weight update of step 4(a). For each edge $(u,v) \in \mathcal{G}^0_{i+1}$, let $s(u,v)$ (resp. $s'(u,v)$) represent the slack on (u,v) before (resp. after) weight update in step 4(a). Let (u,v) be any directed edge in $\mathcal{G}^0_{i+1}$. Note that every edge in $\mathcal{G}^0_{i+1}$ along with the weights $y(\cdot)$ satisfies (3) and so the slack $s(u,v) \geq 0$. Let a_m be the anchor node chosen by algorithm in Step 3 and for any vertex $w \in \mathcal{G}^0_{i+1}$, recollect that ℓ_w is the shortest path cost returned by Dijkstra's algorithm in Step 1 of the algorithm. For the edge (u,v), there are four possibilities after step 4(a):

Case (i) $\ell_u \geq \ell_{a_m}$ and $\ell_v \geq \ell_{a_m}$. In this case, Step 4(a) will not update the vertex weights for u and v. So, $y'(u) = y(u)$, $y'(v) = y(v)$, and $s'(u,v) = s(u,v)$. Therefore, (u,v) remains feasible with respect to $y'(\cdot)$.

Case (ii) $\ell_v < \ell_{a_m}$ and $\ell_u \geq \ell_{a_m}$. In this case, Step 4(a) will update the vertex weights to $y'(v) = y(v) + (\ell_{a_m} - \ell_v) > y(v)$ and $y'(u) = y(u)$. Furthermore, from feasibility of (u,v) with respect to $y(\cdot)$, we have $s(u,v) \geq 0$. Therefore, $s'(u,v) = c(u,v) - y'(u) + y'(v) \geq c(u,v) - y(u) - y(v) \geq 0$ implying that (u,v) remains feasible with respect to the updated vertex weight $y'(\cdot)$.

Case (iii) $\ell_u < \ell_{a_m}$ and $\ell_v \geq \ell_{a_m}$. In this case, Step 4(a) updates the vertex weight to $y'(u) = y(u) + (\ell_{a_m} - \ell_u)$ and $y'(v) = y(v)$. From the property of shortest path distances, the shortest path distance from r_{i+1} to v is bounded by the shortest path distance from r_{i+1} to u and the slack of the edge from (u,v), i.e., $\ell_v - \ell_u \leq s(u,v)$. Using the definition of slack we get,

$$\begin{aligned}\ell_v - \ell_u &\leq c(u,v) - y(u) + y(v),\\ (\ell_{a_m} - \ell_u) - (\ell_{a_m} - \ell_v) &\leq c(u,v) - y(u) + y(v),\\ [(\ell_{a_m} - \ell_u) + y(u) -\\ ((\ell_{a_m} - \ell_v) + y(v))] &\leq c(u,v),\\ y'(u) - ((\ell_{a_m} - \ell_v) + y'(v)) &\leq c(u,v)\end{aligned}$$

The last inequality follows from the fact that $y'(u) = (\ell_{a_m} - \ell_u) + y(u)$. Furthermore, since $\ell_{a_m} < \ell_v$, we get $y'(u) - y'(v) \leq c(u,v)$, i.e., the edge (u,v) remains feasible.

Case (iv) $\ell_u < \ell_{a_m}$ and $\ell_v < \ell_{a_m}$. In this case, Step 4(a) sets $y'(u) = y(u) + (\ell_{a_m} - \ell_u)$ and $y'(v) = y(v) + (\ell_{a_m} - \ell_v)$. Again, the shortest path from r_{i+1} to v is of cost bounded by the shortest path from r_{i+1} to u and the slack on the edge (u,v). Therefore, $\ell_v \leq \ell_u + s(u,v)$ i.e. $\ell_v - \ell_u \leq s(u,v)$. Using the definition of slack we get

$$\begin{aligned}\ell_v - \ell_u \leq &\quad c(u,v) - y(u) + y(v),\\ (\ell_{a_m} - \ell_u) - (\ell_{a_m} - \ell_v) \leq &\quad c(u,v) - y(u) + y(v),\\ [(\ell_{a_m} - \ell_u) + y(u) - &\\ ((\ell_{a_m} - \ell_v) + y(v))] \leq &\quad c(u,v),\\ y'(u) - y'(v) \leq &\quad c(u,v).\end{aligned}$$

implying (u,v) is feasible with respect to $y'(\cdot)$. ◀

▶ **Lemma 9.** *Let P be the augmenting path computed in Step 3 that goes from request r_{i+1} to an anchor node a_m in the alternating graph $\mathcal{G}^0_{i+1}$. For any vertex v on the path P where $v \neq a_m$, let $n(v)$ denote the vertex that succeeds v on P. Given any affected edge (u,v) in $\mathcal{G}_{i+1}$, let $\langle(u,x),(x,v)\rangle$ be its projection. Then,*

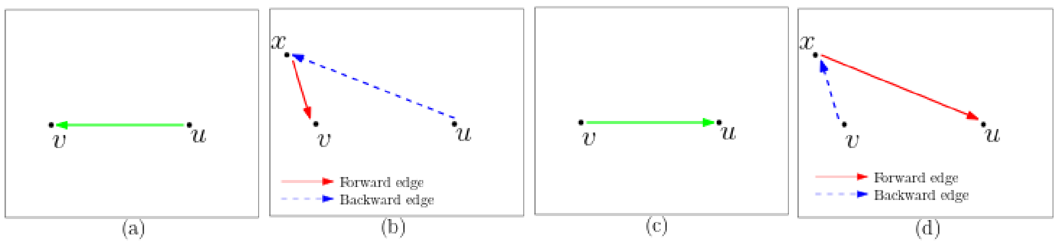
(i) Either the edge (v,u) is on the augmenting path P with $\textsc{Proj}(v,u) = \langle(v,x),(x,u)\rangle$, or
(ii) There is an edge $(u,n(v))$ in $\mathcal{G}^0_{i+1}$ with its projection $\textsc{Proj}(u,n(v)) = \langle(u,x),(x,n(v))\rangle$.

Proof. From Lemma 7, we know that (v,x) is a backward edge in the augmenting trail T. There are two possibilities for the edge (u,x): (a) (u,x) was also added as a backward edge by the augment process, or (b) (u,x) was also an edge in the extended graph G^0_{i+1}.

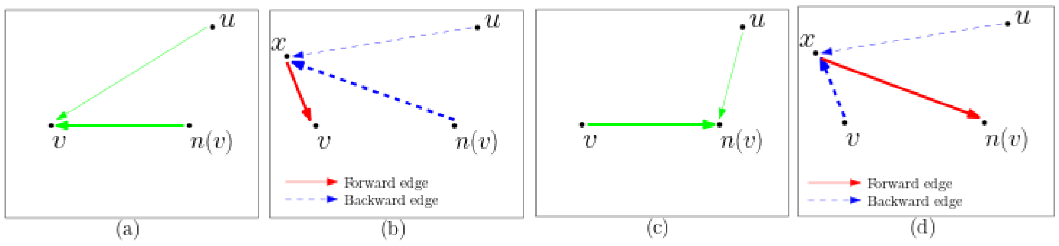
For case (a), the edge (x,u) was a forward edge prior to augmentation. Since both the backward edge (v,x) and the forward edge (x,u) are in the augmenting trail T, we will have an edge (v,u) in the augmenting path P, implying (i). An instance of this case is shown in Figure 2I where directed edge (u,v) is the affected edge.

In case (b), (u,x) is also a backward edge in G^0_{i+1} (Figure 2II(b)). Since (v,x) is a backward edge in T, the projection of the edge $(v,n(v))$ will contain the backward edge (v,x) followed by the forward edge $(x,n(v))$. Since (u,x) is a backward edge and $(x,n(v))$ is a forward edge in G^0_{i+1}, we will have an edge $(u,n(v))$ in the alternating graph $\mathcal{G}^0_{i+1}$. An example of this case is demonstrated in Figure 2II where the directed edge (u,v) in Figure 2II(a) is the affected edge. Figure 2II(b) shows the projection of (u,v). Figure 2II(c) shows the scenario before augmentation where $(v,u) \in \mathcal{G}^0_{i+1}$ and Figure 2II(d) shows (v,u)'s projection in G^0_{i+1}. ◀

▶ **Lemma 10.** *σ_{i+1} is a valid and feasible solution after the Augment operation in Step 4(b).*



(I) Demonstrating Case(a): (u, v) is an affected edge with projection $\langle (u, x), (x, v)\rangle$ such that (x, u) was a forward edge prior to augmentation.
(a) Affected edge $(u, v) \in \mathcal{G}_{i+1}$. (b) Projection of the affected edge (u, v), $\text{Proj}(u, v) = \langle (u, x), (x, v)\rangle$. (c) Alternating graph edge $(v, u) \in \mathcal{G}^0_{i+1}$ before augmentation. (d) Projection of the edge (v, u), $\text{Proj}(v, u) = \langle (v, x), (x, u)\rangle$ before augmentation.



(II) Demonstrating Case(b): (u, v) is an affected edge with projection $\langle (u, x), (x, v)\rangle$ such that (u, x) was a backward edge prior to augmentation as well.
(a) Affected edge $(u, v) \in \mathcal{G}_{i+1}$. (b) Projection of the affected edge $(u, v) \in \mathcal{G}_{i+1}$, $\langle (u, x), (x, v)\rangle$. (c) Alternating graph edges of $\mathcal{G}^0_{i+1}$ before augmentation where $(v, n(v)) \in P$ and $(u, n(v)) \in \mathcal{G}^0_{i+1}$ (d) Projection of the edge $(u, n(v)) \in \mathcal{G}^0_{i+1}$, $\langle (u, x), (x, n(v))\rangle$ before augmentation.

Figure 2

Proof. Let P be the augmenting path from r_{i+1} to the anchor node chosen in Step 3. To prove that σ_{i+1} is a feasible solution after Step 4(b), we need to show that the edges of alternating graph $\mathcal{G}_{i+1}$ that are affected by the augment operation along the path P continue to be feasible and satisfy (5).

Let $c(\cdot, \cdot)$ be the cost function of edges in the alternating graph $\mathcal{G}^0_{i+1}$, i.e., prior to augmentation and let $c'(\cdot, \cdot)$ be the cost function of edges in the alternating graph $\mathcal{G}_{i+1}$, i.e., after augmentation. From Lemma 9, one of the following cases is true.

(i) Edge (v, u) is on the augmenting path P with $\text{Proj}(v, u) = \langle (v, x), (x, u)\rangle$, or
(ii) There is an edge $(u, n(v))$ in $\mathcal{G}^0_{i+1}$ with its projection $\text{Proj}(u, n(v)) = \langle (u, x), (x, n(v))\rangle$.

We will consider both these cases separately.

Case (i): For the edge (v, u), $c(v, u) = d(v, x) - d(x, u)$. After augmentation, $\text{Proj}(u, v) = \langle (u, x), (x, v)\rangle$ and the cost of the affected edge $(u, v) \in \mathcal{G}_{i+1}$ is $c'(u, v) = d(u, x) - d(x, v) = -c(v, u)$. From Lemma 5, after Step 4(a), the slack on every edge of the augmenting path P, including $(v, u) \in P$ is 0 i.e. $s(v, u) = (c(v, u) + y(u) - y(v)) = 0$. The slack on the affected edge $(u, v) \in \mathcal{G}_{i+1}$ can be calculated as,

$$\begin{aligned} s(u, v) &= c'(u, v) - y(u) + y(v) \\ &= -(c(v, u) + y(u) - y(v)) \\ &= 0. \end{aligned}$$

Hence, the affected edge $(u, v) \in \mathcal{G}_{i+1}$ is feasible.

Case (ii): There is an edge $(u, n(v))$ in $\mathcal{G}^0_{i+1}$ and $\text{Proj}(u, n(v)) = \langle (u, x), (x, n(v))\rangle$. From Lemma 5, after Step 4(a), every edge on P including $(v, n(v)) \in P$ has a slack of 0.

Hence, $y(v) - y(n(v)) = c(v, n(v)) = d(v, x) - d(x, n(v))$, or

$$y(n(v)) = y(v) + d(x, n(v)) - d(v, x). \tag{9}$$

Since $(u, n(v))$ is an edge in $\mathcal{G}^0_{i+1}$, it is a feasible edge after the step 4(a) (Lemma 8) and we get $y(u) - y(n(v)) \leq c(u, n(v))$. Substituting $c(u, n(v))$ as $d(u, x) - d(x, n(v))$, we get $y(u) - y(n(v)) \leq d(u, x) - d(x, n(v))$. From equation (9), we can rewrite this inequality as

$$\begin{aligned} y(u) - y(v) - d(x, n(v)) + d(v, x) &\leq d(u, x) - d(x, n(v)), \\ y(u) - y(v) &\leq d(u, x) - d(x, v), \\ y(u) - y(v) &\leq c'(u, v), \end{aligned}$$

implying that the affected edge (u, v) is a feasible edge.

Next, we show that σ_{i+1} is a valid solution after Step 4(b). To show that σ_{i+1} remains a valid solution, we need to show that the forward edges of G_{i+1} satisfies (Q1) and (Q2). By construction, every backward edge is directed from a later request to an earlier one. During augmentation, for every backward edge on the augmenting trail, we reverse its direction and label it as a forward edge. Therefore, every newly introduced forward edge is from an earlier request to a later one implying (Q1).

Next, we will show (Q2)

The first vertex of P is r_{i+1}. Let a be the anchor node at the end of P. Consider any edge (u, v) of the simple directed path P. Note that its projection is $\langle (u, p(v)), (p(v), v) \rangle$ where $(u, p(v))$ is a backward edge and $(p(v), v)$ is a forward edge. Due to augmentation, this projection is modified as follows: $(v, p(v))$ becomes a backward edge and $(p(v), u)$ is now a forward edge. We abuse notation and refer to the modifications made by augmentation along the projection of (u, v) as augmentation along the edge (u, v).

Augmentation along (u, v) modifies the outgoing forward edge from $p(v)$. It also removes the incoming forward edge to v and adds an incoming forward edge to u. Therefore, augmentation along (u, v) does not change the number of forward edges coming in and going out of $p(v)$, i.e., $p(v)$ continues to satisfy (Q2). However, it increases the number of forward edges coming into u by 1 and reduces the number of incoming forward edges incident on v by 1.

Every vertex v' along P, except for the first and the last vertex, i.e., $v' \notin \{r_{i+1}, a\}$, will be the tail for some edge $(u', v') \in P$ and the head for some edge $(v', w') \in P$. Augmentation along (u', v') will reduce the incoming forward edge on v' by one. On the other hand, augmenting along (v', w') will increase the incoming forward edge on v' by one. As a result the incoming and outgoing forward edges incident on v' remain unchanged. Therefore, for every vertex except r_{i+1} and a, we can conclude that (Q2) holds.

The first vertex r_{i+1} is the tail of the first edge in P, augmentation will result in a new incoming forward edge in G_{i+1}. Finally, the last vertex $a \in P$ is the head of the last edge. Therefore, augmentation causes removal of the only incoming forward edge to a. Instead, we add a forward edge from r_{i+1} to a. This guarantees that r_{i+1} contains exactly one incoming forward edge and one outgoing forward edge satisfying (Q2). Furthermore, the anchor node a will have exactly one incoming forward edge satisfying (Q2).

◀

Finally, we will show that the updated vertex weights satisfy (6).

▶ **Lemma 11.** *Consider any solution σ_i maintained by the algorithm with residual graph G_i, for any forward edge (u, v) in the residual graph,*

$$y(v) \geq d(u, v). \tag{10}$$

4.2 Proof of Invariant (I2)

Before we present the proof for (I2), we would like to remind the reader of the following discussion.

From Invariant (I1), Equation (4) and since $y(r_{i+1}) = 0$, the minimum net-cost path P_j from r_{i+1} to some anchor node a_j in the alternating graph G^0_{i+1} is also the augmenting path that minimizes the sum of slacks along its edges. From invariant (I1) all slacks are non-negative and so, P_j will be the augmenting path returned by the execution of Dijkstra's algorithm in Step 1 of the algorithm. Furthermore, $\ell_{a_j} = \sum_{(u,v)\in P_j} s(u,v)$. Therefore, from Equation (4), we conclude that $\Phi_j = \Phi(P_j) = \ell_{a_j} - y(a_j)$. Therefore, Step 2 of the algorithm will correctly compute the minimum net-cost augmenting path to every anchor node $a_j \in \{a_1,\dots,a_k\}$.

Invariant (I2): Let $\mathcal{C}^i$ be the final configuration of σ_i. Then, $\sigma_i = \sigma^*_i(\mathcal{C}^i)$. Furthermore, for every anchor node $a_j \in \{a_1,\dots,a_k\}$, let $\mathcal{C}^{i+1}_j = \mathcal{C}^i \setminus \{a_j\} \cup \{r_{i+1}\}$. Then $\Phi_j = w(\sigma^*_{i+1}(\mathcal{C}^{i+1}_j)) - w(\sigma^*_i(\mathcal{C}^i))$.

Proof: Prior to processing any request, all the servers are in their initial configuration and σ_0 with zero cost is indeed the optimal solution. Assume that, after processing i requests, $\sigma_i = \sigma^*_i(\mathcal{C}^i)$. We will use this to show that $\Phi_j = w(\sigma^*_{i+1}(\mathcal{C}^{i+1}_j)) - w(\sigma^*_i(\mathcal{C}^i))$ and $\sigma_{i+1} = \sigma^*_{i+1}(\mathcal{C}^{i+1})$.

Consider $\sigma^j_{i+1} = \sigma^*_{i+1}(\mathcal{C}^{i+1}_j)$ to be the smallest cost solution that serves $i+1$ requests and ends in $\mathcal{C}^{i+1}_j$. If there are many minimum-cost solutions, we set σ^j_{i+1} to be the one that has the fewest edges in the symmetric difference with $\sigma_i = \sigma^*_i(\mathcal{C}^i)$. Consider the symmetric difference of the edges of σ_i and σ^j_{i+1}. Since their final configurations $\mathcal{C}^i$ and $\mathcal{C}^{i+1}_j$ differ in only the locations of a_j and r_{i+1}, the symmetric difference will include exactly one augmenting trail from r_{i+1} to a_j and possibly a set $\mathbb{C}$ of alternating cycles.

First, we show that $\mathbb{C}$ is an empty set. For the sake of contradiction, assume $\mathbb{C}$ is not empty. Let C be an alternating cycle with respect to G^0_{i+1} (extended graph for solution σ_i) in the symmetric difference. The net-cost of C cannot be zero, since otherwise applying the flip operation on the cycle C in $G_{\sigma'}$ will lead to another valid solution σ'' whose cost is identical to that of σ^j_{i+1} and the final configuration is $\mathcal{C}^{i+1}_j$. However, the flip operation will reduce the size of the symmetric difference and so, σ^j_{i+1} has more edges that σ'' in the symmetric difference with σ_i. This contradicts our assumption that σ^j_{i+1} is the minimum-cost valid solution that ends in configuration $\mathcal{C}^{i+1}_j$ and has the smallest symmetric difference with σ_i.

Similarly, the net-cost $\Phi(C)$ cannot be negative, since otherwise applying the flip operation along the cycle C on G^0_{i+1} will lead to another valid solution that ends in $\mathcal{C}^i$ and has a smaller cost than σ_i . This contradicts the fact that $\sigma_i = \sigma^*_i(\mathcal{C}^i)$ is a minimum-cost solution.

If the net-cost $\Phi(C)$ with respect to G^0_{i+1} is positive, i.e., $\Phi(C) > 0$, then let C' be the alternating cycle corresponding to C in $G_{\sigma^j_{i+1}}$ (the residual graph with respect to σ^j_{i+1}). From Corollary 13 presented in Section 4.3, it follows that the net-cost $\Phi(C') = -\Phi(C) < 0$. Again, applying the flip operation along the cycle C' in $G_{\sigma^j_{i+1}}$ will lead to a valid solution whose cost is smaller than σ^j_{i+1} contradicting the fact that σ^j_{i+1} is the smallest cost solution.

From the above discussion, it follows that the symmetric difference of σ_i and σ^j_{i+1} is an augmenting trail T'. We claim that T' is in fact the minimum net-cost augmenting trail that starts at r_{i+1} and ends at a_j. For the sake of contradiction, suppose T' is not the minimum net-cost augmenting trail and $\Phi(T') > \Phi_j$. Let T be some minimum net-cost augmenting trail in G^0_{i+1} that starts at r_{i+1} and ends at an anchor node a_j. Note that $\Phi_j = \Phi(T)$. Let $\overline{\sigma}^j_{i+1}$ be the valid solution obtained by augmenting σ_i along T in the extended graph G^0_{i+1}.

The final configuration of $\overline{\sigma}^j_{i+1}$ is $\mathcal{C}^j_{i+1}$. Then, by its definition,

$$\begin{aligned} w(\sigma^j_{i+1}) - w(\sigma_i) &> w(\overline{\sigma}^j_{i+1}) - w(\sigma_i), \\ w(\sigma^j_{i+1}) &> w(\overline{\sigma}^j_{i+1}), \end{aligned}$$

contradicting the fact that σ^j_{i+1} is a minimum cost solution to serve $i+1$ requests and end in configuration $\mathcal{C}^j_{i+1}$. Thus the net-cost of the augmenting trail T' in the symmetric difference of σ_i and σ^j_{i+1} is Φ_j. From the definition of net-cost, $\Phi_j = w(\sigma^j_{i+1}) - w(\sigma_i) = w(\sigma^*_{i+1}(\mathcal{C}^{i+1}_j)) - w(\sigma^*_i(\mathcal{C}^i))$.

Our algorithm chooses the minimum net-cost path from r_{i+1} to a_m and augments the valid solution along this path. As a result, the cost of the solution σ_{i+1} increases precisely by $w(\sigma^*_{i+1}(\mathcal{C}^{i+1}_m)) - w(\sigma^*_i(\mathcal{C}^i))$ and the new valid solution will end in configuration $\mathcal{C}^{i+1}_m = \mathcal{C}^{i+1}$ and have a cost equal to $w(\sigma^*_{i+1}(\mathcal{C}^{i+1}))$ proving invariant (I2).

4.3 Symmetric Difference of Valid Solutions

Next, we introduce properties of the symmetric difference of two valid solutions. These properties are used in the proof of invariant (I2).

Let σ and σ' be two valid solutions where σ serves the first i requests and σ' serves the first $i+1$ requests. Let G^0_σ be the extended graph with respect to σ and $G_{\sigma'}$ be the residual graph with respect to σ'. We show that the edges in the symmetric difference of σ and σ' can be decomposed into a edge-disjoint set of alternating trails, augmenting trail and alternating cycles.

Let $\mathcal{C}$ and $\mathcal{C}'$ be the final configurations of σ and σ'. Let X denote the edges in the symmetric difference of σ and σ'. For any edge in X, there is a corresponding directed edge in G^0_σ. We assign the same direction for edge in X. Therefore, by construction, the edges of $X \cap \sigma$ will be forward edges and the edges of $X \cap \sigma'$ will be backward edges in G^0_σ. Figures 3(a), (b) and (c) highlight the edges of σ, σ' and $(X \cap G^0_\sigma)$ respectively. For each vertex $v \in V_i$, suppose v is not an anchor node (resp. if v is not a vertex in the initial configuration), let $f(v)$ (resp. $p(v)$) denote the vertex that appears after (resp. before) v in σ. Similarly, for all $v \in V_{i+1}$, where v is not a vertex from the initial configuration (resp. not an anchor node), let $p'(v)$ (resp. $f'(v)$) denote the vertex that appears before (resp. after) v in $G_{\sigma'}$. Let $\{a_1, \ldots, a_k\}$ denote the anchor nodes of G^0_σ and $\{a'_1, a'_2, \ldots, a'_k\}$ denote the anchor nodes of $G_{\sigma'}$. Let $v'_j = p'(a'_j)$ in $G_{\sigma'}$ and $v_j = p(a_j)$ in G^0_σ. Let $Y = \{v_1, \ldots, v_k\}$ and let $Y' = \{v'_1, \ldots, v'_k\}$. We use these notations throughout this section.

Next, consider any edge $(u, v) \in X$. Suppose (u, v) is a forward edge and $v \notin \{a_1, \ldots, a_k\}$. Since (u, v) is in the symmetric difference and since v is not an anchor node, there is a different in-coming forward edge to v, namely (u', v) in $G_{\sigma'}$ with $u \neq u'$. The backward edge (v, u') will be in X and we denote the edge (v, u') as $\text{next}(u, v)$ in G^0_σ. For example, in Figure 3(c), the backward edge (r_9, r_8) is $\text{next}(r_6, r_9)$. For the forward edge $(u, v) \in X$, there is a different out-going forward edge (u, v') in $G_{\sigma'}$. This edge appears as the backward edge (v', u) in X provided $v' \notin \{a'_1, a'_2, \ldots, a'_k\}$. Therefore, we define the backward edge (v', u) to be the $\text{prev}(u, v)$ in X. For example, in Figure 3(c), $(r_3, r_1) = \text{prev}(r_1, r_7)$.

Finally, we define $\text{next}(u, v)$ and $\text{prev}(u, v)$ for the case where (u, v) is a backward edge in X. Since (u, v) is in the symmetric difference, the edge (v, u) is an out-going forward edge from v in $G_{\sigma'}$. Since (u, v) is in the symmetric difference, there is a different out-going forward edge from v, namely $(v, v') \in X$ with $v' \neq u$. We denote the forward edge (v, v') as $\text{next}(u, v)$ in G^0_σ. For instance, in Figure 3(c), $(r_1, r_7) = \text{next}(r_3, r_1)$. Similarly, for the backward edge $(u, v) \in X$, if $u \neq r_{i+1}$, there is a different in-coming forward edge to u (v', u)

in G_σ. This edge appears as a forward edge (v', u) in X. Therefore, we define the forward edge (v', u) to be the $\mathrm{prev}(u, v)$ in X. For instance, in Figure 3(c), $(r_3, r_5) = \mathrm{prev}(r_5, i_4)$.

Thus, every edge in X have a unique $\mathrm{next}(\cdot, \cdot)$ edge except those that are directed towards an anchor nodes $\{a_1, \dots, a_k\}$. Edges directed towards the anchor nodes $\{a_1, \dots, a_k\}$ do not have any $\mathrm{next}(\cdot, \cdot)$ edge.

Similarly, every edge in X has a unique $\mathrm{prev}(\cdot, \cdot)$, except the edges going out of $\{v'_1, v'_2, \dots, v'_k\}$. Edges going out of $\{v'_1, v'_2, \dots, v'_k\}$ do not have any previous edge.

Decomposing X into augmenting trail, alternating trails and cycles: Consider any vertex in $v \in Y \cap Y'$. The forward edge (v, a) is in X. However, since $v \in Y'$, the $\mathrm{prev}(v, a)$ does not exist. Similarly, since $a \in \{a_1, \dots, a_k\}$, $\mathrm{next}(v, a)$ does not exist. So, we create a trivial alternating trail with one edge (v, a).

For every vertex $v \in Y' \setminus Y$, let (v, v') be the outgoing edge in X. We initialize T to be (v, v') We construct an alternating trail incrementally by concatenating the last edge (u, v) of T with $\mathrm{next}(u, v)$. This construction stops when we reach some edge (u', a) for which $\mathrm{next}(u', a)$ is not defined. Suppose the vertex $v \neq r_{i+1}$, then this trail starts with a forward edge and ends at an anchor node. On the other hand, suppose $v = r_{i+1}$, this trail is an augmenting trail that starts with a backward edge and ends at an anchor node. Any edge (u, v) of X that did not participate in the alternating and augmenting trails have a well defined $\mathrm{next}(u, v)$. Therefore, we can construct an alternating cycle that contains (u, v) by repeatedly concatenating the last added edge (u', v') with its $\mathrm{next}(u', v')$. The construction stops when $\mathrm{next}(u', v') = (u, v)$ and we get an alternating cycle. Figure 3(c) illustrates an example such decomposition. Vertex r_{10} and r_5 are in $Y \cap Y'$, so $\langle r_{10}, a_1 \rangle$ and $\langle r_5, a_3 \rangle$ are trivial trails. Vertex r_6 and r_3 are in $Y' \setminus Y$ and therefore we have two alternating trails $\langle r_6, r_9, r_8, a_2 \rangle$ and $\langle r_3, r_5, i_4, r_4, i_3, r_3, r_1, r_7, r_4, a_4 \rangle$. The vertex r_{11} is r_{i+1}, therefore we have an augmenting trail $\langle r_{11}, r_9, r_{10}, r_7, a_5 \rangle$. All the remaining edges form an alternating cycle $\langle r_2, r_6, i_1, r_2, i_2, r_8, r_2 \rangle$.

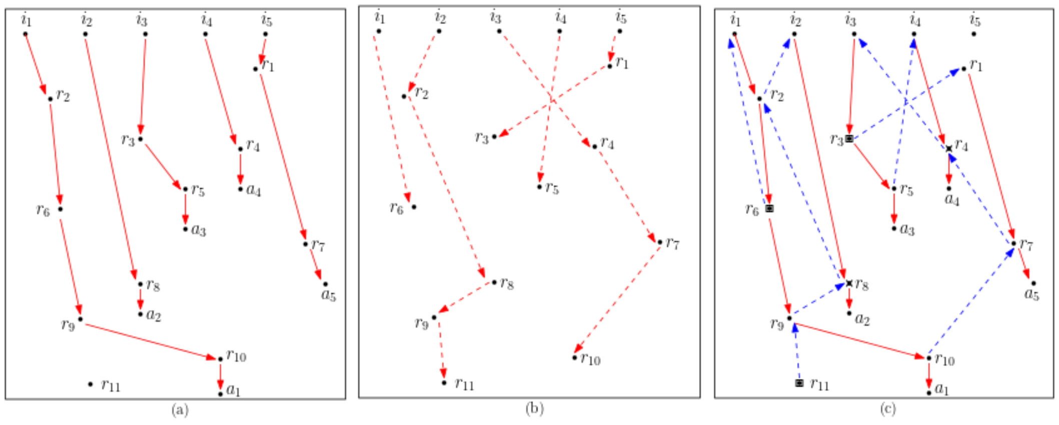


Figure 3 (a) Forward edges of G^0_σ, representing σ, (b) Forward edges of $G_{\sigma'}$, representing σ', (c) Edges in the symmetric difference of σ and σ'. Edges of σ are shown as forward edges in G_σ (red edges), and edges of σ' are shown as the backward edges in G_σ (dashed blue edges). This graph has an augmenting trail $\langle r_{11}, r_9, r_{10}, r_7, a_5 \rangle$, two alternating trails $\langle r_6, r_9, r_8, a_2 \rangle$, $\langle r_3, r_5, i_4, r_4, i_3, r_3, r_1, r_7, r_4, a_4 \rangle$ and one directed alternating cycle $\langle r_2, r_6, i_1, r_2, i_2, r_8, r_2 \rangle$

Note that the construction described above also extends to the residual graph $G_{\sigma'}$ and we obtain the following lemma.

▶ **Lemma 12.** *The edges of symmetric difference X of two valid solutions σ and σ' in G^0_σ can be decomposed into (a) one augmenting trail A, (b) a set $\mathbb{T}$ of $|Y' \setminus Y| - 1$ non-trivial alternating trails that start with a forward edge, (c) a set $\mathbb{T}'$ of $|Y' \cap Y|$ trivial alternating trails and (d) a set $\mathbb{C}$ of alternating cycles. Similarly, the edges of X in $G_{\sigma'}$ can be decomposed into alternating trails and cycles each of which are obtained by simply applying the flip operation to the trails and cycles in A, $\mathbb{T}$, $\mathbb{T}'$ and $\mathbb{C}$*

▶ **Corollary 13.** *Given the set X of edges in the symmetric difference, consider the decomposition of X into $\{A\} \cup \mathbb{T} \cup \mathbb{C}$ in G^0_σ as described in Lemma 12. Then the edges of X in $G_{\sigma'}$ can be decomposed into alternating trails such that for each alternating trail (resp. cycle) $T \in \{A\} \cup \mathbb{T} \cup \mathbb{C}$, there is an alternating trail T' (resp. cycle) induced by the edges of X in $G_{\sigma'}$ such that T' is obtained by applying the flip operation on T and $\Phi(T) = -\Phi(T')$.*

5 Missing Proofs

Proof of Lemma 11: The initial solution σ_0 is a trivially valid solution and the only edges in the residual graph are forward edges that go from a vertex in the initial configuration to an anchor node. For any such forward edge (u, v), v is an anchor node with $y(v) = 0$. The cost $d(u, v)$ is also 0 since the anchor node v and the vertex u share the same location. Therefore, inequality (10) holds.

Let us assume that the inequality (10) holds after request r_i is processed by our algorithm. To complete the proof, we will show that the inequality will continue to be satisfied after request r_{i+1} is processed. To do so, we will show that the any of the changes made by the algorithm will not violate inequality (10).

At the start of the algorithm, we add r_{i+1} to G_i to create G^0_{i+1}. Since all the edges incident on r_{i+1} in G^0_{i+1} are backward edges, all forward edges will continue to satisfy inequality (10). Steps 1, 2 and 3 do not alter the weights $y(\cdot)$ or the alternating graph. Therefore, the inequality (10) continues to hold for every forward edge during these three steps. In Step 4(a), we modify the vertex weights for every vertex v with $\ell_v < \ell$. Consider any such vertex and let (u, v) be the in-coming forward edge to v. Recollect that $y(v)$ is the weight prior to the execution of Step 4(a) and $y'(v)$ is the weight after Step 4(a). Since inequality (10) is true prior to execution of Step 4(a), we have $y(v) \geq d(u, v)$. In Step 4(a), the weight is updated to $y'(v) \leftarrow y(v) - \ell_v + \ell$ provided $\ell_v < \ell$. Since $\ell_v < \ell$, it follows that $y'(v) \geq y(v) \geq d(u, v)$ and inequality (10) continues to hold.

Next, we show that the inequality (10) continues to hold after Step 4(b). In Step 4(b), we apply the augment operation along an augmenting trail T (computed in Step 3). Recollect that the first vertex of T is r_{i+1} and the last vertex of T is an anchor node a. Note that the weights of every vertex, except the anchor node a remains unchanged. However, for any v along the alternating trail T, its incoming forward edge may change. As a result of augmentation along T, every backward edge (v_2, v_1) in T changes to a forward edge (v_1, v_2). Let P be the augmenting path in the alternating graph $\mathcal{G}^0_{i+1}$ such that trail $T = \textsc{Proj}(P)$. Let (v_2, v_1) be a backward edge in T. Let (v_2, w) be the edge in P such that $\textsc{Proj}(v_2, w)$ contains the backward edge (v_2, v_1), i.e., $\textsc{Proj}(v_2, w) = \langle (v_2, v_1), (v_1, w) \rangle$. By definition of slack,

$$\begin{aligned} s(v_2, w) &= c(v_2, w) - y(v_2) + y(w) \\ &= d(v_2, v_1) - d(v_1, w) - y(v_2) + y(w). \end{aligned}$$

Since $(v_2, w) \in P$, at the end of Step 4(a), the slack $s(v_2, w)$ becomes 0 (Lemma 5). Therefore, we have $(d(v_2, v_1) - d(v_1, w)) - y'(v_2) + y'(w) = 0$ which can be rearranged as

$$y'(v_2) = d(v_2, v_1) - d(v_1, w) + y'(w). \tag{11}$$

Since, (v_1, w) is a forward edge in T, after Step 4(a), we have $y'(w) \geq d(v_1, w)$. Substituting this in 11, we get

$$y'(v_2) \geq d(v_1, v_2). \tag{12}$$

After augmentation, the backward edge $(v_2, v_1) \in \textsc{Proj}(v_2, w)$ changes to a forward edge $(v_1, v_2) \in G_{i+1}$. Hence, inequality (11) implies inequality (10) continues to hold. Therefore, inequality (10) continues to hold after the augment operation in Step 4(b).

Finally, step 4(b) also adds a forward edge from r_{i+1} to a_m and modifies the vertex weight of the anchor node a_m to 0. Since the cost of this forward edge $d(r_{i+1}, a_m)$ is 0, inequality (10) holds for the vertex a_m. This concludes the argument that inequality (10) holds after Step 4(b).

References

1 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 3–16. ACM, 2018. doi:10.1145/3188745.3188798.

2 Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discret. Math.*, 4(2):172–181, 1991. doi:10.1137/0404017.

3 Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991. doi:10.1137/0220008.

4 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

5 B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993. URL: https://www.sciencedirect.com/science/article/pii/S0196677483710266, doi:https://doi.org/10.1006/jagm.1993.1026.

6 Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994. doi:10.1016/0304-3975(94)90042-6.

7 Elias Koutsoupias. The k-server problem. *Comput. Sci. Rev.*, 3(2):105–118, 2009. doi:10.1016/j.cosrev.2009.04.002.

8 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

9 Harold W. Kuhn. The hungarian method for the assignment problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 29–47. Springer, 2010. doi:10.1007/978-3-540-68279-0_2.

10 James R. Lee. Fusible hsts and the randomized k-server conjecture. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 438–449. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00049.

11 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

12 Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 505–515. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.53.

13 Sharath Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.18.

14 Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, volume 99 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.SoCG.2018.67.

15 Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. A fast work function algorithm for solving the k-server problem. *Central Eur. J. Oper. Res.*, 21(1):187–205, 2013. doi:10.1007/s10100-011-0222-7.

16 Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. A fast work function algorithm for solving the k-server problem. *Central European Journal of Operations Research*, 21:187–205, 2013.

17 Tomislav Rudec and Robert Manger. A new approach to solve the k-server problem based on network flows and flow cost reduction. *Comput. Oper. Res.*, 40(4):1004–1013, 2013. doi:10.1016/j.cor.2012.11.006.