ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Techniques for Design and Analysis of Algorithms for Problems in Metric Spaces

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νάκου Βασίλειου

Επιβλέπων: Δημήτριος Φωτάκης
Λέκτορας Ε.Μ.Π

Αθήνα, Νοεμβριος 2012

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Techniques for Design and Analysis of Algorithms for Problems in Metric Spaces

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νάκος Βασίλειος

Επιβλέπων: Δημήτριος Φωτάκης
Λέκτορας Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την -5/11/2012-.

	ΠΑΓΟΥΡΤΖΗΣ ΑΡΗΣ	
ΖΑΧΟΣ ΕΥΣΤΑΘΙΟΣ	ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ	ΦΩΤΑΚΗΣ ΔΗΜΗΤΡΙΟΣ
ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π	Ε.Μ.Π	ΛΕΚΤΟΡΑΣ Ε.Μ.Π

Αθήνα, Οκτώβριος 2012.

....

Ευστάθιος Ζάχος
(Καθηγήτης Ε.Μ.Π)

....

Άρης Παγουρτζής
(Επίκουρος Καθηγητής Ε.Μ.Π)

....

Δημήτριος Φωτάκης
(Λέκτορας Ε.Μ.Π)

Abstract

Abstract

We study problems where we have access to the input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start. We focus to metric spaces problems, where the triangle inequality holds. We take a glimpse at the k-server problem and present some classical, as well as some new results for it. The k-server problem is the most fundamental problem in the field of online algorithms, and has been studied for decades. The previous year, the work of Bansal et al. was breakthrough as it proposed the first (randomized) polylogarithmic algorithm in decades. Moreover, we also present the design and analysis of efficient fractional algorithms for online problems as mentioned before, based on two techniques: the one is puerly combinatorial ,whereas the second one is based on a primal-dual approach and a solution of a system of differential equations to establish the competitive ratio. Last but not least, we study incremental sum-radii clustering and an extension of mobile facility location to an online setting, and present some results about the structures of the problems, algorithms for the general case and more competitive algorithms for some specialised,but not trivial, cases.

Contents

Chapter 1

Introduction

In computer science, an online algorithm is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start. In contrast, an offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand. (For example, selection sort requires that the entire list be given before it can sort it, while insertion sort doesn't.)
Because it does not know the whole input, an online algorithm is forced to make decisions that may later turn out not to be optimal, and the study of online algorithms has focused on the quality of decision-making that is possible in this setting. Competitive analysis formalizes this idea by comparing the relative performance of an online and offline algorithm for the same problem instance. For other points of view on online inputs to algorithms, see streaming algorithm (focusing on the amount of memory needed to accurately represent past inputs), dynamic algorithm (focusing on the time complexity of maintaining solutions to problems with online inputs) and online machine learning.
Many problems may be studied from an online perspective. Some of these are the job scheduling problem,the k-server problem,the metrical task systems problem,the list update problem,the paging problem or the facility location problem. A metrical task system -first introduced and studied by Borodin,Linial and Saks- consists of a metric space and a transition table, in order to represent the set of possible configurations and the distance between each two of them. Metrical task systems are a generalisation of many problems, such as paging,list accessing and k-server.Some of the most influential papers are [50],[51],[52].
The paging problem is one of the most fundamental problems in competitive analysis. It was studied in [44],[45],[46],[47],[48].
Another relevant problem is the page migration problem, where only one copy of each page

exists in the network. A great deal of research has been done on page migration problems in [21],[22],[23],[24],[25],[26] . Also,in [27] the authors assume a potential usage pattern of the databases. If we focus on work with no such assumptions, this work started by the work of Black and Sleator in [28], and were continued in [29],[30],[31],[32],[33],[34],[35],[36].

1.1 Study of Online Algorithms and Competitive Analysis

Competitive analysis is a method invented for analyzing online algorithms, in which the performance of an online algorithm (which must satisfy an unpredictable sequence of requests, completing each request without being able to see the future) is compared to the performance of an optimal offline algorithm that can view the sequence of requests in advance. An algorithm is competitive if its competitive ratio—the ratio between its performance and the offline algorithm's performance—is bounded. Unlike traditional worst-case analysis, where the performance of an algorithm is measured only for ”hard” inputs, competitive analysis requires that an algorithm perform well both on hard and easy inputs, where ”hard” and ”easy” are defined by the performance of the optimal offline algorithm.
For many algorithms, performance is dependent not only on the size of the inputs, but also on their values. One such example is the quicksort algorithm, which sorts an array of elements. Such data-dependent algorithms are analysed for average-case and worst-case data. Competitive analysis is a way of doing worst case analysis for on-line and randomized algorithms, which are typically data dependent.

In competitive analysis, one imagines an ”adversary” that deliberately chooses difficult data, to maximize the ratio of the cost of the algorithm being studied and some optimal algorithm. Adversaries range in power from the oblivious adversary, which has no knowledge of the random choices made by the algorithm pitted against it, to the adaptive adversary that has full knowledge of how an algorithm works and its internal state at any point during its execution. Note that this distinction is only meaningful for randomized algorithms. For a deterministic algorithm, either adversary can simply compute what state that algorithm must have at any time in the future, and choose difficult data accordingly.

1.2 Adversary Model

In computer science, an online algorithm measures its competitiveness against different adversary models. For deterministic algorithms, the adversary is the same, the adaptive offline adversary. For randomized online algorithms competitiveness can depend upon the adversary model used.
The three common adversaries are the oblivious adversary, the adaptive online adversary, and the adaptive offline adversary.

- The oblivious adversary is sometimes referred to as the weak adversary. This adversary knows the algorithm's code, but does not get to know the randomized results of the algorithm.
- The adaptive online adversary is sometimes called the medium adversary. This adversary must make its own decision before it is allowed to know the decision of the algorithm.
- The adaptive offline adversary is sometimes called the strong adversary. This adversary knows everything, even the random number generator. This adversary is so strong that randomization does not help against him.

1.3 Preliminaries

A metric space is an ordered pair (M, d) where M is a set and d is a metric on M,i.e. a function $M \times M \rightarrow \mathbb{R}$ such that $\forall x, y, z \in M$ holds:

- $d(x, y) \geq 0$
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$
- $d(x, z) \leq d(x, y) + d(y, z)$

The function d is also called distance function or simply distance. Often, d is omitted and one just writes M for a metric space if it is clear from the context what metric is used.
As mentioned before,The idea of competitiveness is to compare the output generated by an online algorithm to the output produced by an online algorithm. An offline algorithm is an

omniscient algorithm that knows the entire input data and can compute an optimal output. The better an online algorithm approximates the optimal solution, the more competitive this algorithm is. More formally,we can describe an online problem as follows :
An online algorithm is presented with a request sequence $\sigma = \sigma(1), \sigma(2), .., \sigma(m)$. The requests $\sigma(t)$, $1 \leq t \leq m$ must be served in the order they appear. More specifically, when serving request $\sigma(t)$,algorithm A does not know any request $\sigma(t')$ with $t' > t$. Serving requests incurs a cost and the goal is to minimize the total cost paid on the entire request sequence. Given a request sequence σ,let $C_A(\sigma)$ denote the cost incurred by A and let $C_{OPT}(\sigma)$ denote the cost incurred by an optimal offline algorithm OPT.The algorithm is called c-competitive if there exists a constant a such that
$C_A(\sigma) \leq c * C_{OPT}(\sigma) + a$.

We note a useful theorem,that will be useful later on:

A tree metric is closely related to graph decomposition. The randomized rounding procedure of Calinescu, Karloff, and Rabani [37] for the 0-extension problem de composes a graph into pieces with bounded diameter, cut- ting each edge with probability proportional to its length and a ratio between the numbers of nodes at certain distances. Fakcharoenphol, Rao, and Talwar [38] used the CKR rounding procedure to decompose the graph recursively and obtained the following theorem.

Metric Embeddings,Theorem 1: Given an n-point metric (V, d), there exists a randomized algorithm, which runs in time $O(n^2)$, that samples a tree metric from the distribution D over tree metrics that $O(logn)$-probabilistically approximates (V, d). The tree is also a 2-HST.

Metric Embeddings,Theorem 2: Given an n-point metric (V, d), there exists a polynomial-time deterministic algorithm that finds a dis- tribution D over $O(nlogn)$ tree metrics that $O(logn)$- probabilistically approximates (V, d).

Metric approximation by random trees has applications in on-line and distributed computation, since randomiza- tion works well against oblivious adversaries, and trees are easy to work with and maintain. Alon et al. [41] first used tree embedding to give a competitive algorithm for the k- server problem. Bartal [40] noted a few problems in his pa- per: metrical task system, distributed paging, distributed k-server problem, distributed queuing, and mobile user. After the paper by Bartal in 1996, numerous applica- tions in approximation algorithms have been found. Many approximation algorithms work for problems on tree met- rics or HST metrics. By approximating general metrics with these metrics, one can turn them into algorithms for general metrics, while, usually, losing only a factor of O(log n) in the approximation factors. Sample prob- lems are metric labeling, buy-at-bulk network design, and group Steiner trees. Recent applications include an ap- proximation algorithm to the Unique Games [42], infor- mation network design [43], and oblivious network design [39].

1.4 Examples of Online Algorithms

In this section we will focus on some well-known and elementary problems that appear in online algorithms.

The ski-rental problem: The ski-rental problem was introduced in [ ]. Another relevant problems is the file replication [ ] and the network leasing problem[ ] In this problem, a skier travels to his favourite mountain and wants to skii every day. However, he does not how many days he will stay there(let us forget the reason about that) and so he cannot decide if it is better to buy or rent skis. Denote by B the cost for buying skiis and r the cost of renting them per day, and assume that the rent of the skis holds for one day. If the number of days was known in advance, call it n, then the minimum cost would be $min\{B, r * n\}$, because he either would buy all a ski on the first day, either he would rent skis for all the days. But when the number of days in unknown,as mentioned before, how must the skier act in order to pay as less as he can? The goal is to minimize the cost of the worst-case scenario, a notion which is captured by the competitive ratio, as mentioned before. We will assume that at the morning of each day the skier learns if he departs from the mountain or stays there. What is now a "good" strategy for him to achieve a small competitive ratio? Consider an index(day) i_0 such that $r \cdot i_0 \leq B < r(i_0 + 1)$. If the skier wakes up and it is his $(i + 1)$-th day at the mountain and he will not depart that day, then he buys skiis, otherwise if that day is before the $(i_0 + 1) - th$ day, he rents the skis. We will prove that the algorithm achieves a competitive ratio of 2. Clearly, observe that if $n \leq i_0$ then the algorithm pays what the optimal pays, specifically $i_0 * r$. Otherwise the optimal pays B and the algorithm pays $i_0 * r + B \leq 2B$, 2 times what the optimal pays. Hence, the algorithm is 2-competitive. One can achieve a $(1 - \frac{1}{e})$-competitive randomized algorithm for that problem. Details are in a next section.

We view another one simple problem that must be treated in an online fashion.

The paging problem: In the paging problem,mentioned before, we have to maintain a two-level memory system consisting of a small fast memory and a large slow memory. The memory is partitioned into pages of equal size. The system receives a sequence of requests,

where each request specifies a page in the memory system. A request can be served immediately if the referenced page is available in fast memory. If the requested page is not in fast memory, a page fault occurs. The missing page is then loaded from slow memory into fast memory so that the request can be served. At the same time a page is evicted from fast memory to make room for the missing one. A paging algorithm decides which page to evict on a fault. This decision must usually be made online, i.e. without knowledge of any future requests. The cost to be minimized is the number of page faults.
Suppose that the cache can hold up to k pages and let the algorithm LRU be the one that at each time step evicts-if needed- the page that has been requested least recently. It suffices to prove that every k misses of the LRU algorithm, the optimal algorithm must also miss at least one page. This establishes the result. Observe that if LRU misses the succesive pages $p_1, p_2, .., p_k$ then $p_i \neq p_j, \forall 1\ leqi, j \leq k, i \neq j$, because at time $j > i$ if $p_j = p_i$,that page would not be absent from the cache, as there is another page that was least recently used, by the LRU rule. So, it easy to see that the optimal must incur at least one miss during these pages, if we take into account the page that was in the cache and evicted for the sake of p_1.

Chapter 2

The k-server Problem

2.1 Problem Definition and Conjectures

The k-server problem in a metric space(or weighted graph) corresponds to the problem of moving around in an efficient manner k servers to service requests that appear at the points of the metric space. The k servers start at a fixed set of k points of the metric, called initial configuration. At each time step, a request appears at some point of the metric and one at least server has to move to that point to serve that request. The goal is to minimize the total distance travelled by the servers, when the requests arrive online,one after the other. We assume that the next request appears only after the current one has been serviced.
We study the online k-server on metric spaces as they were defined in previous sections. Some interesting metric spaces include the uniform metric,the line and trees.
We define a configuration to be a k-tuple of points of the metric ,which corresponds to where the k-servers lie. If C_1, C_2 are two configurations, we extend the distance function $d()$ such that $d(C_1, C_2)$ is the minimum weight perfect matching between the points of C_1, C_2. The distance between two configurations corresponds to the minimum distance that must be travelled by the servers in order to move from configuration C_1 to configuration C_2. It is easy to observe that $d(C_1, C_2) = d(C_1 - C_2, C_2 - C_1)$ due to the triangle inequality.
So,the k-server problem is defined by an initial configuration C_0 and a sequence of requests $r = (r_1, .., r_n)$ of points of M. A feasible solution is a sequence of configurations $C_0, C_1, .., C_n$ such that $r_i \in C_i, \forall i \in \{1, .., n\}$. The cost of the solution is the total distance travelled by the servers,namely

$$\sum_{i=0}^{n-1} d(C_i, C_{i+1})$$

.
Rather unexpectedly, it is very likely that the competitive ratio is independent of the metric space (provided that it has more than k distinct points). This is supported by all results on the problem in the last two decades and it is exactly what the k-server conjecture states:

The k-server Conjecture. For every metric space with more than k distinct points, the competitive ratio of the k-server problem is exactly k.

The k-server conjecture is open,however it is proved to be true for some special metric spaces and for $k = 2$. An almost tight result holds : there exists a $(2k - 1)$-competitive algorithm on any metric.
As far as randomization is allowed,the following conjecture holds:
Randomized k-server Conjecture: For every metric space, there is a randomized online algorithm for the k-server problem with competitive ratio $O(logk)$.

2.2 Brief History

The problem was first defined by Manasse, McGeogh, and Sleator in [1]. A few years before, Tarjan and Sleator begun the study of online algorithms using the idea of competitive analysis, studing problems for the list update problem and the paging problem. A great step forward to study these questions in a more general setting was made by Borodin, Linial, and Saks who introduced the online problem of metrical task systems in [2], a generalization of the *k*-server problem, and were able to determine exactly the competitive ratio: $2n - 1$ for a metrical task system on n states. This was the time when the 3 researchers mentioned above introduced the *k*-server problem as a special case of metrical task system, but also as a generalization of the paging problem. Manasse, McGeogh, and Sleator proved that the competitive ratio of any online deterministic algorithm cannot be smaller than k on metrics with at least $k + 1$ points and they also invented an algorithm with competitive ratio of 2 for the special case of $k = 2$. They also generalized this result by proving that the competitive ratio is exactly k for all metric spaces with $k + 1$ points.
There are 2 special cases of the k-server problem that have been studied and are of particular interest: the 3-server problem and the k-server problem on a cycle .Any progress on these problems has the potential to open new paths to attack the general k-server conjecture. For both of these special cases, we know nothing better than the 2k – 1 bound.
The Harmonic Algorithm is a very natural memoryless algorithm but it is not the best one

for metrics of $k+1$ points. For this important special case of metric spaces, it was shown by Coppersmith, Doyle, Raghavan, and Snir that there is a memoryless randomized algorithm for $k+1$ points which has competitive ratio k, by exploiting the connection of the theory of electrical networks with random walks. This is the best possible competitive ratio for memoryless algorithms.

2.3 Deterministic Algorithms

The simplest algorithm that one could invent is the following greedy algorithm: Whenever a demand arrives,move the closest server to that point(breaking ties arbitrarily) . One can easily observe that is algorithm is not competitive. Consider the 2-server point on the real line, where the first server lies on 0, the second on 2 and the demand sequence is $1-\varepsilon, 0, 1-\varepsilon, 0, 1-\varepsilon, 0....$ After n demands have arrived the greedy algorithm would pay $n*(1-\varepsilon)$, whereas the optimal algorithm would pay $1+\varepsilon$,as it would move the server at 2 to the point $1-\varepsilon$.
In this section we propose a lower bound for the k-server problem and two additional algorithms of great significance: the Double Coverage Algorithm and the Work Function algorithm.
The lower bound states that there no deterministic algorithm in a metric space with at least $k+1$ points can achieve a competitive ratio better than k. The Double Coverage algorithm is one an optimal k-server algorithm on the line, proposed by Chrobak,Karloff,Payne and Vishwanathan. The work-function algorithm is the best deterministic algorithm we know till now and was analysed by Koutsoupias and Papadimitriou. Its competitive ratio is $2k-1$ and the conjecture is that it is k. These results are presented in the next three subsections.

2.3.1 A Lower Bound on the Competitive Ratio of any Deterministic Algorithm

The following lower bound is due to Manasse, McGeogh, and Sleator.

K-server lower bound: In any metric space with at least $k+1$ points,no online algorithm for the k-server problem can have competitive ratio less than k.

Proof: Let A be any online algorithm. In order to establish the lower bound, we need to indicate a request sequence for which A pays k times what the optimal algorithm pays. Clearly,such a request sequence must be dependent of A and the choices it makes. However,in order to efficiently bound the optimal solution from above, we consider a set of k distinct offline algorithms such that the cost of the online algorithm is equal to the total cost of all k offline algorithms. This would proven immediatetly the desired result,because that would mean that there exists an offline algorithm with at most $\frac{1}{k}$ of the cost of the online algorithm. Initially, let S be a set of $k+1$ points that include the original configuration. All requests are going to be in this set S. What we are going to use is that at any time, we know precisely what an optimal adversary does: at any point in time there is only one point of the metric space that is not covered by the online algorithm and the optimal adverser should bring the next point there. There are exactly $k+1$ configurations of the set S. We consider k offline algorithms $A_1, A_2, .., A_k$ such that the configurations of algorithm A and all A_i are different at each time step.Firstly,before any request arrives,in order for the above invariant to hold,we move the servers of the offline algorithm from the initial configuration to all other configurations,incurring a fixed cost. Suppose now that the online algorithms moves a server from some point a to the current request r_t, which means that the current configuration of the online algorithm is identical to a configuration of one of the offline algorithms, say A_j for some j. Now, A_j must change its configuration in order to hold the invariant true. So, it now moves a server from r_t to a so that its new configuration matches the previous configuration of A. The other algorithms do not change their configuration. It is easy to observe that the invariant is maintained. The cost of A is $d(a, r_t)$ which is equal to the total cost of the offline algorithm A_j. Summing over all time steps we get that the cost of the online algorithm equals the total cost of the offline algorithms minus some fixed cost, the cost incurred at the very first stage of the algorithm,before any request sequence arrived yet. By the discussion before, we get that there exists an algorithm which incurs $\frac{1}{k}$ times the cost of the online algorithm, fact that establishes the lower bound.

2.3.2 Determistic Algorithms for Tree Metrics

Why did the obvious greedy algorithm proposed before failed to be competitive? Because it did not take into account that whenever there exists a region where too many demands arrive, it must start moving the servers there. The Double Coverage algorithm tries to imitate that behaviour in tree metrics.

An important special case of the k-server problem is the case of the 1-dimensional Eu-

clidean space, in other words the line. An elegant algorithm for the line is the Double Coverage Algorithm presented in [3], which is an invariant of the greedy approach which moves the closest server each time to the request. More specifically, when the request is between two servers, the algorithm moves both servers towards the request equal to the distance of the request to the closest server. If the request is outside the interval of the k servers,only the closest server moves to it.
Altough Double Coverage is not a lazy algorithm, it actually can be turned to a greedy algorithm by remembering moves and postponing them until it is necessary to do them, just like any lazy algorithm does. However, here it has some intuitive meaning not to act in a lazy way.
The competitive ratio is established by a potential function argument. Denote $COST_t$, OPT_t the cost of the online and the optimal algorithm respectively at time t. Let C_t be the configuration at time t of the online algorithm,while C'_t the configuration of the optimal offline algorithm. Suppose there exists a function Φ such that $COST_t - \varrho OPT_t \leq \Phi(C_{t-1}, C't-1) - \Phi(Ct, C't)$. Then by adding over all time steps we get that

$$\sum_{t=1}^{n} COST_t - \varrho \sum_{t=1}^{n} OPT_t \leq \Phi(C_0, C'_0)$$

,which implies that the competitive ratio is at most ϱ.
Now, how the potential function Φ should be chosen? Although in general it can be very tricky to devise a good potential function, here it is somehow intuitive. We can view Φ as the sum of two terms. The first one is $kd(C_t, C't)$, and the reasoning behind this choice is that when the adversary moves an offline server some distance x the distance $d(C_t, C't)$ increases by at most x and the online algorithm will pay back k times more this cost. The second term (which is clearly less intuitive) equals the sum of all pairwise distances between the servers of the online configuration. More formally, the potential function is defined as follows:

$$\Phi(C_t, C'_t) = kd(C_t, C'_t) + \sum_{a_1, a_2 \in C_t} d(a_1, a_2)$$

The proof that the Double Coverage Algorithm is k-competitive is based on the observation that when an offline server is already at the request, the double move of the online algorithm guarantees that the first term of the potential function does not increase, because one of the two moving servers moves towards the request point; although the other online server may move away from its matching offline server an equal distance, their total contribution does not increase the matching. As far as the other termis concerned, we note that it decreases by the total distance travelled by the two servers. More specifically, if r_t is the current request

and s_i, s_j the servers which the online algorithm moves at that time with $d(s_i, r_t) \leq d(s_j, r_t)$, the potential function decreases by $2d(s_i, r_t)$.
Let C_t, C'_t be the online an offline configurations after serving request r_t. The above observation implies
$\Phi(C_t, C'_t) \leq \Phi(C_{t-1}, C'_t) - 2d(s_i, r_t) = \Phi(C_{t-1}, C't) - d(C_{t-1}, C_t)$. It is easy to observe that the same inequality holds also when the current request is outside the interval of the k servers and hence only one server moves toward the request.
Till now, we have proven the above inequality when an offline server is at the request r_t. But what about the movement and the cost incurred by the offline algorithm? It holds that
$\Phi(C_t, C'_t) \leq \Phi(C_{t-1}, C'_t) + kd(C_t, C'_t)$. Combining this inequality with the previous one we get
$d(C_{t-1}, C_t) - kd(C'_{t-1}, C'_t) \leq \Phi(C_t, C'_{t-1}) - \Phi(C_t, C'_t)$, which by the discussion above implies that the Double Coverage algorithm is k-competitive.

In tree metrics the Double Coverage algorithm can easily be extended: First of all, we say a server z is free with respece to request r_t if there is no other server in the path from r_t to z. So,to service request r_t we move all free servers towards r_t at equal speed. Observe that there might exist sometime when a server who was free is not anymore. Then we stop moving that server and we leave it where it is. From this point, until the request is serverd, a server that is not free cannot become free.

2.3.3 The Work-Function Algorithm

To design a more efficient algorithm, one must not decide the next move based only on the current situtation. So,it must take into account all the request points from the past, each time it determines which server to move. In other words, if we view each placement of the servers as a configuration, a highly competitive algorithm on any metric space should try to move as close as it can to the optimal configuration till then,while not moving that much; this means that it must also take account the distance from the current configuration.
The following algorithm's competitive ratio was analysed in [4] by Koutsoupias,Papadimitriou and that was a breakthrough at that time.Define $w_\sigma(C)$ be the optimal cost for serving request sequence σ and ending in configuration C. Our online algorithm acts as follows: Assume now that after serving request sequence σ is at configurations C, and then demand r arrives. Then the algorithm moves a server to the point that minimizes $w_{\sigma r}(C - \{c\} \cup \{r\}) + d(c, r)$. The above algorithm is $(2k - 1)$-competitive,as proved by Koutsoupias and Papadimitriou. We omit the proof here, but we state some interesting properties and facts about WFA(work-

function algorithm):

Define C_f to be the configuration that the online algorithm stops. We can also assume that it is the same as the one that the optimal offline algorithm ends to. In order to to derive the $(2k-1)$ competitive ratio for general metric spaces, one can define a function μ_σ such that

- $\mu \leq OPT, \forall x \in C_f$
- $\mu_0(x) \geq -\text{constant}, \forall x \in C_0$
- $\mu'(x) \geq \mu(x), \forall x$ that belong to the metric space.
- $\mu(x) - \mu(y) \leq k \cdot d(x, y)$, for all points x, y of the metric space.
- $\mu'(r) - \mu(r) = max_X\{w'(X) - w(X)\}$(the extened cost), where r is the current request.

Suppose that you have such a function μ and define a potential function Φ such that

$$\Phi = \sum_{u \in U} \mu(u)$$

For the potential function one can immediatetly derive the following:

- $\Phi_f \leq k \cdot OPT$, from the first bullet one gets and that $|U| = k$
- $\Phi_0 \geq -$ constant, because the algorithm starts at the initial configuaration and by the second bullet we can bound the initial value of the potential function from below.
- Whenever no server is moved clearly nothing changes to the potential function because of that server. When a server moves from u to v it holds that $\Delta\Phi \geq -k \cdot d(u, v)$, because of the fourth bullet above.
- When a request
- Whenever a request arrives, $\Delta\Phi \geq max_X\{w'(X) - w(X)\}$, because the optimal algorithm will have a server at r and this changes only because of the change in μ; the property holds as a consequence of the third and the fifth bullet above.

Summing up all the changes in Φ one gets the desired result, by the typical use of potential function in the design and analysis of online algorithms. One needs to construct such a function μ. This is not trivial and we will not present the proof here. However,we note some nice properties about the work-function algorithm:

- $\forall C, w_{\sigma r} \geq w_{\sigma}$
- If $r \in C$, then $w_{\sigma r} = w_{\sigma}$
- $w_{\sigma r} = min_{c \in C}\{w(C - \{c\} \cup \{r\}) + d(c, r)\}$
- $w(C) \leq w(C') + d(C, C')$
- For all configurations X, Y and for all $x \in X$, there exists a y such that $w(X - \{x\} \cup \{y\}) + w(Y - \{y\} \cup \{x\}) \leq w(X) + w(Y)$ (Quasiconvexity Lemma)

2.3.4 Open Problems

The above results may help shed light to the nature of the *k*-server problem,however many practical and theoretical questions still remain. Some of those:

- Does the deterministic *k*-server conjecture holds?
- Can the results of the work-function algorithm be extended in other simple metrics(trees or cycles) in order to prove that the algorithm is *k*-competitive?
- Can a counterexample be found? In particular,does there exist a metric where the work-function algorithm is not *k*-competitive?
- Implementing the work-function algorithm is of course inefficient, because of the combinatorial explosion of the states that one might need to hold. Do there exist simpler algorithms that achieve some a competitive ratio that is a linear function(or even polynomial) of *k* that are easy to implement ?
- Can the proof of the 2-competiviness of the work-function for the 2-server instance be simplified?

2.4 Randomized Algorithms

As mentioned before, the randomized *k*-server conjecture states that there exists a logarithmic in *k* competitive algorithm for the problem in arbitrary metric spaces.The best known lower bound is $\Omega(\frac{logk}{loglogk})$ due to Bartal.One of the first online randomized algorithms is the Harmonic Algorithm, proposed by Raghavan and Snir in [5],with a competitive ratio of $O(2^k logk)$. The algorithm simply states that whenever an uncovered request arrives ,serve the request to some server with a probability that is inversely proportional to the distance of these two. More formally, server s_i serves the request with probability

$$\frac{\frac{1}{d(s_i,r_t)}}{\sum_j \frac{1}{d(s_j,r_t)}}$$

. The Harmonic algorithm is $O(logk)$-competitive on a uniform metric. Some special results have been achieved for other special metrics also. Bartal,Chrobak,Larmore give a $(2-\varepsilon)$-competitive algorithm on the line.
A general result of Ben-David et al. gives a non-constructive proof that the existence of any randomized on-line algorithm, which uses randomization of the form used by the Harmonic algorithm, implies the existence of a deterministic on-line algorithm, at the cost of squaring the competitive ratio. Fiat et al., McGeoch and Sleator , and Karlin et al. propose the scenario of randomized algorithms that use randomization to hide the on-line configuration from the adversary.

2.4.1 Fractional Analysis

As in many online algorithms, the design of efficient randomized algorithms may require using techniques from the continuous world, where the problems seem to be easier (remember that Linear Programming is in P, whereas Integer Programming is NP-complete). For the case of the k-server problem, one can think the servers as fractional entities, which means that in the new (fractional) setting we can break a server to many points of different perhaps weight (which they should however add up to 1) and serving a request would imply that we have moved one unit of servers there in total. In fact, this can be viewed as a probability distribution of configurations, each of which is a set of locations with at most k elements (the number of distinct positions of the metric space that are occupied by servers each time). In fact, any randomized algorithm can be viewed as a probability distribution over the configurations, assigning a probablity to each configuration, which corresponds to the probability that the algorithm is at that configuration at the current time.

For a randomized algorithm ALG, let $P^t_{ALG}(C)$ the probability that at time t, the algorithm has placed the servers at configuration C. We will show that the randomized algortithm corresponds to a fractional one, called ALG'. Let

$$W^t_{ALG'}(x) = \sum_{C,x\in C} P^t_{ALG}(C)$$

be the total weight at location x at time t for the algorithm ALG'. It is easy to see that $W^t_{ALG'}(r_t)$ for all t, otherwise the randomized algorithm would be infeasible. Also, in order to move between two configurations, let ALG' move servers according to the minimum cost shift between those two distributions. The above intuitive algorithm's correctness holds by the above inequality : $cost_{ALG'}(\varrho) \leq E[cost_{ALG}(\varrho)]$,where ϱ is a demand sequence. For the proof observe that if ALG has assigned at time t probabilities $p_1, p_2, ..., p_m$ to configurations $C_1, C_2, .., C_m$, then we can simulate this shift with paying less than the expected cost of ALG. For all i and $x \in C$ move ps_i fraction from x to y_i, where the y_i is the point to which the server at x moves while satisfying the shift from C to C_i. The equalities for the weight functions W hold and that these moves pay the same cost for ALG and ALG'. Since ALG' moves servers according to the minimum cost shift as mentioned before,it may atain a lower value, and so the inequality holds.

The above reduction shows that any randomized algorithm gives birth to a fractional algorithm with the same or lower cost. Does the opposite hold? Does some variant of the opposite hold? It would be nice if any fractional algorithm would give birth to a randomized algorithm with a cost within $O(1)$ of the fractional cost. This is true on some metric spaces such as the line and the circle and when $k = 2$ or $k = n - 1$.

2.5 A Polylogarithmic Competitive Algorithm for the k-server Problem

2.5.1 Introduction

Recently after work done in [6] by Cote et al., new ideas emerged in the randomized setting of the k-server problem. So, Nikhil Bansal, Niv Buchbinder, Aleksander Madry abd Joseph Naor managed to devise and present in [7] a polylogarithmic competitive randomized algorithm(depending on the both the algorithms of n and k) for the k-server problem. Clearly, the above algorithm depends also on the number of demands and improves the result of Koutsoupias and Papadimitriou(the work-function algorithm) only when n is subexponential in k. All the results that follow are presnted in the foremen

Theorem 1. *There is a randomized algorithm for the k-server problem that achieves a competitive ratio of $O(log^2klog^3nlogn)$ on any metric space on n points.*

It is well known that since any metric space can be embedded into a probability distribution over HSTs with relatively low distortion, it suffices to solve the problem on an HST, which seems easier. The algorithm proposed is inspired by the Cote's approach , which includes the definition of a new problem on uniform metrics, called the allocation problem. More specifically, they managed to show that an online randomized algorithm for the allocation problem can lead to a randomized algorithm for the k-server problem on an HST, provided that some certain properties hold. For each node u, they run an instance of the allocation problem on a weighted star which has u as a root and its children as leaves. Combining information from the answers obtained by the algorithm, they determine inductively, over all levels and all time steps, the number of servers at each leaf of the HST, taking care of the feasibility of the solution. However, the properties mentioned above hold on a restricted number of metric spaces and moreover the competitive ratio is polynomial in *logk, logn, log*Δ(Δ denotes the diameter of the metric space). Building on ideas of that paper Buchbinder et al. manage to give the first polylogarithmic algorithm for the k-server problem, which depends only on *logk, logn* and not on the diameter of the underlying metric space. It is interesting that they do not design a randomized algorithm that has the properties that are missing from the previous approach, but they use different techniques. Firstly, the embed the metric space to an HST and then they transform this tree to a weighted $\sigma - HST$, an HST in which the lengths of the edge across a path from the root to the leaf,decrease exponentially but they might be non-uniform. Using these low-dstortion embeddings and transformations they give an online fractional algorithm on this HST based on a suitable fractional algorithm for the allocation problem, which they also design. Last, they propose a low-cost randomized rounding procedure with rounds the fractional algorithm in an online fashion,incurring only a cost of $O(1)$ from the fractional solution. This concludes the design of their algorithm.

2.5.2 The allocation problem

The allocation problem: Suppose that a metric on d points is defined by a weighted star where the edge connecting each leaf i to the root equals w_i. At each time step t, the number of available servers $k(t) \leq k$ is defined. We refer to the vector $(k(1), k(2), ..,)$ as the quota pattern. A request arrives at a point i and it is specified by a $(k+1)$-dimensional vector $h^t = (h^t(0), h^t(1), .., h^t(k)$, where $h^t(l)$ equals the cost paid by the algorithm if it places exactly l servers there at time t(equivalently,if it serves request i using l servers). It is natural to demand that the cost vectors satisfy the monotonicity property $h^t(l) \geq h^t(l+1)$, which means that serving the request with more servers cannot be worse; a rather natural assumption. So,each demand incurs a hit cost according to the previous vector. Moreover,

at each time step the movement of the servers incurs an additional movement cost: moving a server across an edge of weight w incurs a cost of w. The goal is to minimize the total hit cost plus the total movement cost.
As mentioned before, the goal is to use a fractional algorithm for the allocation problem as a building block in order to solve the k-server problem on HST. Clearly, passing from a fractional algorithm to a randomized one, involves the rounding of the fractional solution. We claim that the integrality gap,meaning the gap between the optimal fractional and the optimal integral solution, of the allocation problem can be as high as $\Theta(k)$. This might at first glance seem restrictive for our algorithm, since the during the rounding procedure a factor of k might emerge, but the approach proposed overcomed this difficulty balancing the "need" between the cost of the hit and the move cost.
CLaim: The integrality gap of the allocation problem is $\Omega(k)$.
Proof: Consider a uniform metric over two points(namely an edge) and an instance where $k(t) = k$ for all t. At each odd time step the cost vector $h = (1, 1, 1, .., 0)$ arrives at location 1,whereas the cost vector $h' = (1, 0, .., 0)$ arrives at location 2(the leaf). Now,observe that any integral algorithm must incur a cost of $O(1)$ during two consecutive steps. This holds because if someone wants to avoid the hit cost, he would pay at least 1 for moving the servers from location 1 to location 2. In any case, the cost for a time horizon T must be at least $O(T)$. For the fractional algorithm ,one can write a linear programming formulation of the problem and find an assignment of the variables that gives rise to a solution with $O(\frac{T}{k})$ cost. Let $x^t_{1,j}, x^t_{2,j}$ be the indicator variables that equals 1 if there exist exatcly j servers at location 1,2 at time t. The constraint for the servers allowed then can be written as

$$\sum_j j(x^t_{1,j} + x^t_{2,j}) \leq k(t) = k, \forall t$$

. The total cost of the algorithm then be written as

$$\sum_{t,j} (h(j)x^t_{1,j} + h'(j)x^t_{2,j}) + MV$$

, where MV denotes the movement cost, which expression we will not write in an explicit form, as the assignment of the variables is static,in a sense that the servers never move. Define $x^t_{1,0} = \frac{1}{k}, x^t_{1,k} = 1 - \frac{1}{k}, x^t_{2,1} = 1$. Observe that $MV = 0$ as mentioned before, the constraints holds as $\frac{1}{k} \times 0 + (1 - \frac{1}{k}) \times k + 1 \times 1 = (k - 1) + 1 = 1$, and the algorithm incurs a cost of $\frac{1}{k}$ at each odd time step. Hence the integrality gap of the allocation problem is $O(k)$.

For the allocation problem we say an algorithm is (θ, γ)-competitive if it incurs a hit cost of $\theta(Optcost + \Delta g(k)$ and a move cost of $\gamma(Optcost + \Delta g(k))$,where $Optcost$ is the optimal cost, Δ is the diameter of the metric space and $g(k) = \sum_t |k(t) - k(t-1)|$.

2.5.3 Approaching k-server on HSTs

As mentioned in a previous section, we can view the randomized algorithm as a probability distribution over the configurations at each time step of the algorithm. In the k-server problem, the fractional view corresponds to specifying the probability p_i of having a server at leaf i at the current time step.

We move with the following theorems:

Theorem 2. *For any $\varepsilon > 0$,there exists a fractional $(1+\varepsilon, O(log(\frac{k}{\varepsilon}))$-competitive algorithm for the allocation problem on a weighted star metric.*

Theorem 3. *Let T be a weighted σ-HST of depth l. If,for any $0 \leq \varepsilon 1$,there exists a $(1+\varepsilon, log(\frac{k}{\varepsilon}))$-competitive algorithm for the fractional allocation problem on a weighted star,then there is a $O(l \times log(kl)))$ -competitive algorithm for the fractional k-server problem on T, as long as $\sigma = \Omega(l \times log(kl))$.*

Theorem 4. *Let T be a σ-HST with $\sigma > 5$. Then any online fractional k-server algorithm on T can be converted into a randomized k-server algorithm on T with an $O(1)$ factor loss in the competitive ratio.*

The combination of the above theorems gives a solution to HSTs, and not weighted HSTs. So,we use the following additional theorem:

Theorem 5. *Let T be a σ-HST with n leaves, but possibly arbitrary length. Then T can be transformed into a weighted σ-HST T', such that T' has logarithmic depth($O(logn)$), the leaves of T and T' are the same, and any leaf to leaf distance is distorted by a factor of at most $\frac{2\sigma}{\sigma-1}$.*

The main result is that there is a polylogarithmic competitive algorithm for the k-server. At first, we use the tecnhique of Faclak to embed the metric space into a distribution μ over σ-HSTs with stretch $\sigma = \Theta(logn(log(klogn)))$. The height of the HST is $log_\sigma \Delta$ and the distortion of any distance is multiplied by an asymptotic factor of $O(\sigma log_\sigma n)$. We pick an HST

T at random from the distribution created and transform T to T'. Because T' has a height of $O(logn)$ then $\sigma = \Theta(l \times log(kl))$ and the fractional algorithm gives a $O(lognlog(klogn))$-competitive fractiona algorithm on T. By rounding the solution in an online manner, we get a $O(logn(log(klogn))$-competitive randomized k-server algorithm on T' and hence on T. Now, we will use the fact that the embedding of the metric space to T does increase every distance, but the expected value of these distances does not increase much . Let OPT_M/ALG_M, OPT_T/ALG_T be the optimal cost(the algorithm's cost) on M and T respectively. Observe that $ALG_M \leq ALG_T$ due to the embedding properties, and as ALG_T is $O(lognlog(klogn))$-competitive,it follows that $ALG_M \leq O(lognlog(klogn))c_T$, where c_T denote the cost of our algorithm's solution on T. By taking expected values,

$$E[ALG_M] = O(lognlog(klogn))E[c_T] = O(\sigma log_\sigma n \cdot O(logn \cdot log(klogn))OPT*_M$$

. Hence,the competitive ratio of the algorithm is

$$O(\sigma \frac{logn}{log\sigma})O(logn \cdot log(klogn)) = O(log^2 k \cdot log^3 n \cdot loglogn)$$

.

2.5.4 Fractional Algorithm for the *k*-server Problem on HSTs

For a node p,integer j and time t let $Optcost(p, j \cdot \vec{1}, t)$ be the optimum cost for serving the k-server instance $\{\varrho(1), \varrho(2), .., \varrho(t)\} \cap T(p)$, with the additional constraint that exactly j servers are available.

In our algorithm, each internal node of T, let it be p, will run a number of instances of the allocation problem which are different with respect to their quota patterns(the number of servers they have available), but have the same hit-cost vectors. Each node p also will hold a convex combination of these instances. At every time step, the fractional solutions to the different instance for each node determine how the servers are distributed to each of its children.

We have to specify at each time step which are the allocation problems that run for each node p, and how do they evolve. Denote the convex combination over allocation instances on node p at time t by Λ_p^t, which is specified by the collection

$$\Lambda_p^t = \{(\lambda_{p,s}^t, \kappa_{p,s}^t, H_{p,s}^t)\}, \forall t, p \sum_p \lambda_{p,s}^t = 1$$

Here:

- $\kappa^t_{p,s}$ is the quota pattern of node p until time t
- $\lambda^t_{p,s}$ is the fraction at time t given to the instance with quota pattern $\kappa^t_{p,s}$
- $H^t_p = \{h^1_p, h^2_p, .., h^t_p\}$ is the sequence of hit cost vectors that have appeared until time t.

The description of the algorithm evolves the evolution(or better, definition) of Λ^t_p for each node p at time t and show how the fractional number of servers at the leaves of T are computed.

Hit Costs: Let p be any internal node with children p_1, p_d. For the allocation problem running at p,at time t we give the hist cost vector

$$h_{p_i}(j) = Optcost(p, j \cdot \vec{1}, t) - Optcost(p, j \cdot \vec{1}, t-1)$$

As Cote et al. prove,for the cost vectors $h_{p_i}(t)$ the monotonicity property holds: $h_{p_i}(1) \geq .. \geq h_{p_i}(t)$. We move with two observations:

- $h^t_p(i, 0) = \infty$, because any 0-server solution is infeasible for any instance with at least one request.
- $h^t_p(i', j) = 0$ for all $i' \neq i$ and for all j. This holds because the request is not in the sub-tree of $p_{i'}$ for $i' \neq i$.

Quota patterns: The quoata patterns are determined recursively in a top down manner over the tree T(and inductively over time t) by solutions of the fractional algorithm of the allocation problem that runs on each node. The procedure below specifies how the quota patterns update $\kappa^t_{p,s}$ and the convex combination $\lambda^t_{p,s}$.

- At the root r of the tree T there is a single allocation instance running with a quota of k at all times, which means that Λ^t_r consists of a single allocation instance,hit costs as described above and $\kappa = k \cdot \vec{1}$.
- For any internal node $p \in T$ and time $t = 0$, Λ^0_p consists of a single allocation instance. The quota pattern $\kappa_{p,s}$ for this single instance s,until $t = 0$, is the number of servers present initially at the leaves of subtree $T(p)$ and no hit cost thus far.

Establishing the inductive step: Consider time t. We describe the procedure to obtain Λ_p^t from Λ_p^{t-1} in a top down manner on the HST as follows. Λ_r^t has already been determined for all t. Assume that Λ_p^t has already been determined. Then,for the children $p_1, ..., p_d$ of p,we determine $\Lambda_{p_i}^t$: Consider the allocation instances that are executed at node p. Let $\{x_{i,j,s}^t\}_{i,j,s}$ be the fractional solutions generated at time t. The following consistency property between the quota for servers available at p_i and the fractional solution from the allocation problem at node p, is maintained:

$$\sum_{s\in\Lambda_{p_i}^t|\kappa_s^t(t)=j} \lambda_s^t = \sum_{s\in\Lambda_p^t} \lambda_s^t x_{i,j,s} = x_{i,j}^t$$

, and for each child p_i

$$\sum_{s\in\Lambda_{p_i}^t} \lambda_s^t = 1$$

Suppose now that $x_{i,j}^{t-1}$ changes to $x_{i,j}^t$ due to the allocation instaces $s \in \Lambda_p^t$ at time t. One needs to update $\Lambda_{p_{i-1}}^t$ to $\Lambda_{p_i}^t$,in order to satisfy the previous two equalities, and of course this update can be done in an efficient way. The cost paid by the convex combination of the allocation instances running at some node p equals

$$\sum_s \sum_i w(p,i)\lambda_s^t \sum_{j=1}^k |\sum_{l<j}(x_{i,l,s}^t - x_{i,l,s}^{t-1}| \geq \sum_i w(p,i) \sum_{j=1}^k |\sum_s \lambda_s^t \sum_{l<j}(x_{i,l,s}^t - x_{i,l,s}^{t-1})|$$

Observe that the change from $x_{i,j}^t$ to $x_{i,j}^{t-1}$ can be decomposed into a sequence of elementary moves, in which $\pm\delta_{i,j}$ units of mass are removed from $x_{i,j}$ and are moved to $x_{i,j\pm1}$, such that the total movement cost remains the same. This leads to the conclusion that one can assume that $x_{i,j}^t$ and $x_{i,j}^{t-1}$ differ only an elementary move. Consider now such an elementary move where $x_{i,j}^t = x_{i,j}^{t-1} - \delta$, $x_{i,j-1}^t = x_{i,j}^t + \delta$ and in order to implement this move,let δ be an arbitrary measure of allocation problems $\in \Lambda_{p_i}^{t-1}$ with $\kappa_s(t-1) = j$ and just set $\kappa_s(t) = j$; in other words move them an additional server. For all other κ_s, $\kappa_s(t) = \kappa_s(t-1)$. Observe that hte consistency equation

$$\sum_{s\in\Lambda_{p_i}^t|\kappa_s^t(t)=j} \lambda_s^t = \sum_{s\in\Lambda_p^t} \lambda_s^t x_{i,j,s} = x_{i,j}^t$$

and the fact the λ_s^t add to 1 for each child p_i, meaning

$$\sum_{s\in\Lambda_{p_i}^t} \lambda_s^t = 1$$

hold.

But how does the fractional k-server solution is obtained? As mentioned before, hte k-server solution is established by the fractional number of servers at each leaf q. Consider a leaf q and its parent p. Then we define

$$z(q, t) = \sum_{s \in \Lambda_p^t} \lambda_s^t \sum_j j \cdot x_{q,j,s}^t$$

to be the number of servers at q at time t, where $x_{q,j,s}$ equals the probability of having exactly j servers at q at time t,when the fractional allocation algorithm is applied to the allocation instance $s \in \Lambda_p^t$.

Feasibility: One needs to prove that whenever at some time t there exists a server a q then $z_{q,t} \geq 1$. We will prove this statement, provided the total cost incurred by the allocation problems is finite. Suppose now that leaf q is requested at time t and q is the i-th child of its parent p. Then the hit cost entry $h^t(i, 0)$ for every allocation instance running at p is ∞, which leads to the conclusion that if the total cost of the allocation problem is finite, for each $s \in \Lambda_p^t$, it must hold that $x_{q,0,s}^t = 0$. Since $\sum_j x_{q,j,s}^t = 1$ and $\sum_{s \in \Lambda_p^t} \lambda_s = 1$, it follows that

$$z(q, t) = \sum_{s \in \Lambda_p^t} \lambda_s^t \sum_j j \cdot x_{q,j,s}^t \geq 1$$

2.5.5 Rounding the solution online

First of all, we have to prove Theorem 8, about the transformation of T to T'. We restate Theorem 8:

Theorem 6. *: Let T be a σ-HST with n leaves, but possibly arbitrary length. Then T can be transformed into a weighted σ-HST T', such that T' has logarithmic depth($O(logn)$), the leaves of T and T' are the same, and any leaf to leaf distance is distorted by a factor of at most $\frac{2\sigma}{\sigma-1}$.*

Proof. : We call an HST with a root r balanced balanced if for each node u none of the subtrees rooted at the children of u has more than half of the nodes the tree rooted at u has. Observe that if a balanced tree has n nodes then its height should be $O(logn)$, for if you take any path from the root to a leaf and try to walk across it beginning at r, at each time you "discard" half of the nodes of the subtree on which root you were(in other words,at least half of the nodes are in the other subtrees). Hence, after $O(logn)$ levels, you must have reached the leaf, because by the beforehand geometric decrease, you will have reached a tree with

exactly one node.
We will now describe a procedure that makes a tree balanced. Do a postorder traversal (a botoom-up approach would also work) and assume for a nude u that all its subtrees are balanced, after the routine has returned from them. Then if there exists in the transformed instance a tree with more than the half nodes than the subtree rooted at u. If yes, then unite u with the root of that subtree into one new node, by ”forgetting” (the authors use the word ”contract”) the edge between them.
It is obvious that the tree is indeed balanced ,for we have guaranteed that the property described in the first paragraph indeed holds. But what about the change of the distances? Clearly, since some edges may contract, there is a possibility that the distance between some two nodes is infinite times smaller than what it was. But this is no problem, since we are really interested in the distances between any two leaves, for the embedding procedure guarrantes that all points of the metric space are mapped into the leaves of the tree. Pick two arbitrary leaves u and v let l be their lowest common anchestor. Define l_u be the child of l that is next in the path from l to u. Same for v. Observe that the procedure described above cannot contract both (l, l_u) and (l, l_v) because that would mean that the respective subtrees would have more than half of the nodes of the subtree rooted at u each,a contradiction. Hence, the worst-case scenario is that all edges except l_u(or l_v,but that’s symmetric) are contracted. So the distance from u to v can be distorted only by a factor of

$$\frac{2\sum_{i=0}^{l}\sigma^i}{\sigma^l} \leq 2\sum_{i=0}^{l}\frac{1}{\sigma^i} \leq \frac{2\sigma}{\sigma-1}$$

, and the proof is complete. □

We are now ready to describe a randomized rounding procedure that rounds the fractional solution in an online manner to a feasible solution,while losing only $O(1)$ from the competitive ratio, as long as $\sigma > 5$. The rounding uses ideas from 11, which emerged during the development of a randomized algorithm for the finely competitive paging problem; there the underlying metric space is the uniform metric, so it is necessary to extend them to HSTs, and so does the following theorems.
Let $1, .., n$ be the leaves of the σ-HST T. At each times t,the fractional algorithm defines the numbers x_i^t,which correspond to the probability that there exists a server at time t in leaf i. Clearly, $\forall t \sum_i x_i^t = k$.Observe that using the obvious and naive rounding procedure(create the next configuration of the servers by choosing if there exists a server at i with probability x_i^{t+1} and move the servers as imposed by the minimum weight matching between this and the previous configuration) might lead to an infeasibilty: there is a probability that the algorithm places more than k servers on the leaves of the tree. So, one needs to follow more a more

sophisticated approach, being careful in maintaing feasibility. In order to do this(observe that the main problem at this time seems to be possible infeasibility), we view (as mentioned in previous section) the randomized algorithm as a probability distribution on the configurations. The state S^t at some time t defines a probability distribution $\mu_{S^t}()$ over the configurations(k-tuples of leaves), where $\mu_{S^t}(C)$ is the probability mass of configuration C in state S.

We say that a state S is consistent with a fractional state $\bar{x}$(i.e. values of x_i such that $\sum_i x_i = k$), when

$$\forall i \in \{1, .., n\}, \sum_{C: i \in C} \mu_S(C) = x_i$$

. Observe now that in order to devise an efficient randomized rounding procedure, we need to maintain a sequence of states $S_0, S_1, S_2, ..$ such that each state S_t is consistent with the fractional state $\bar{x}^t$ induced by the fractional algorithm and the total movement cost between the configurations is within $O(1)$ of the movement cost of the fractional algorithm.

In order to establish the above, one needs to strengthen the conditions of the induction he uses. More specifically, at each time step t, S^t is not only consistent with $\bar{x}^t$, but also each configuration C does not deviate much from the fractional state x^t. We use the following notation:

- $x_p = \sum_{i \in T(p)} x_i$, for a node p of T; this corresponds to the total fractional amount of servers allocated at the on the leaves of $T(p)$.
- $n_p(C) = |C \cap T(p)|$, is the number of servers in configuration C on leaves of $T(p)$.
- C is balanced with respect to $\bar{x}$ iff $n_p(C) \in \{\lfloor x_p \rfloor, \lceil x_p \rceil\}.\forall p$.
- State S is balanced with respect to $\bar{x}$ if every configuration with non-zero probability mass ($\mu_S(C) > 0$) is balanced with respect to $\bar{x}$.
- $W(p)$ is the length of the edge connecting node p with its parent.
- The balance gap $G(S, x)$ of S with respect to a fractional state $\bar{x}$ is

  $$G(S, \bar{x}) = \sum_p W(p) \sum_{C \in S} \mu_S(C) min(|n_p(C) - \lfloor x_p \rfloor|, |\lceil x_p \rceil - n_p(C)|)$$

  . In a sense, this corresponds to how unbalanced the state S (with respect to $\bar{x}$) is. The term $min(|n_p(C) - \lfloor x_p \rfloor|, |\lceil x_p \rceil - n_p(C)|)$ corresponds to the minimum fractional value that needs to be added or substracted from node p in oder to become balanced.

Before proceeding, we note the following lemma, which appears to be useful in proving the main theorem that follows below.

Theorem 7. *: Let $\bar{x}$ be a fractional state and let S be a k-server state on the leaves of a σ-HST T,with $\sigma > 5$, which is consistent with respect to $\bar{x}$, but not necessarily balanced with respect to it. Then S can be converted to another state S' which is both consistent and balanced with respect to $\bar{x}$, while incurring a cost of $G(S, \bar{x})$.*

Proof. : If no node is imbalanced, then $G(S, x) = 0$ and we have nothing to prove, as $S' = S$. Consider now the case that $G(S, x) > 0$. The high level idea is that if some subtrees in a configuration have more (or less) nodes than what is allowed, there exists a configuration and a set of other subtrees with less(respectively more) nodes than what is allowed, so we will adjust the probability mass of these configurations, by moving probability mass from the missing leaf to the abundant one. So, for all unbalanced subtrees we will have $\mu_S(C) = 0$, meaning that we ignore them in our distribution. We move with the technical details: Pick a node p that is imbalanced and which is at the highest level of T. Observe that this cannot be the root r since $x_r = k$ at each time step, so it has some parent. Let C be a configuration such that $\mu_S(C) > 0$ and $n_p(C) \notin \{\lfloor x_p \rfloor, \lceil x_p \rceil\}$ and assume that $n_p(C) < \lfloor x_p \rfloor$(the other case can be treated similarly). As S is consistent with $\bar{x}$, $\sum_{C''} \mu_S(C'')n_p(C'') = x_p$ and so there must exist a configuration C' with $\mu_S(C')$ such that $n_p(C') \geq \lfloor x_p \rfloor + 1$ (due to the fact that the $\mu_S(C'')$ add up to a value smaller or equal to 1), which leads to $n_p(C') - n_p(C) \geq 2$.
Let P be the parent of p and observe that,since as mentioned before p is not the root and p was the imbalanced node at the highest level, P must be balanced,which means that $|n_P(C') - n_P(C)| \leq 2$. However, since $n_p(C') - n_p(C) \geq 2$, there must exist some other child of P-call it p'- for which $n_{p'}(C') < n_{p'}(C)$. This holds by explotiing the fact

$$\sum_i n_{p_i}(C'') = n_P(C''), \forall C''$$

, where the summation runs over all children of P; just substract the two equations, group terms by nodes and claim the argument.
This means that there exists a leaf i in $T(p)$ which is contained in C' but not in C. In a similar fashion, there exists a leaf i' in $T(p')$ which is not contained in C'. Let $\delta = min(\mu_S(C), \mu_S(C')) > 0$ and do the following modifications:

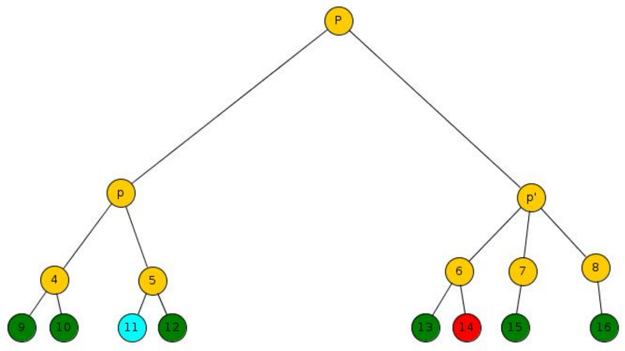
- $\mu_S(C) \leftarrow \mu_S(C) - \delta$
- $\mu_S(C') \leftarrow \mu_S(C') - \delta$

- $\mu_S(C \cup \{i\} - \{i'\}) \leftarrow \mu_S(C \cup \{i\} - \{i'\}) + \delta$
- $\mu_S(C' \cup \{i'\} - \{i\}) \leftarrow \mu_S(C' \cup \{i'\} - \{i\}) + \delta$

One can observe that the only nodes that are affected by these modifications -and hence can become unbalanced- belong to the paths $i - p$ and $i' - p'$. By doing the calculations on the imbalance gap one can observe that it always decreases. The imbalance gap decreases at least by

$$W(p)\delta - 4\delta w(p)(1 + \frac{1}{\sigma} + \frac{1}{\sigma^2} + ..) = W(p)\frac{\sigma - 5}{\sigma - 1} = \Omega(W(p)\delta)$$

. Observe also that the movement cost is at most $4\delta w(P)\frac{\sigma}{\sigma-1}$, because both i and i' lie at $T(P)$. So the movement cost for the modification incurred is within a factor of $O(1)$ from the imbalance gap reduction. Applying this procedure while the imbalance gaps becomes zero(this procedure must end because we proved that it always reduces by this modification), we get a proof of the lemma.



∎

we now prove the next theorem:

Let T be an σ-HST with n leaves, $\sigma > 5$, and let $\overline{x}^0, \overline{x}^1, ..$ be a sequence of states of a fractional k-server algorithm. There is an online procedure that maintains a sequence of randomized k-server states $S^0, S^1..$ with the following properties:

- At any time t,the state S^t is consistent with the fractional state $\overline{x}^t$.
- If the fractional state changes from $\overline{x}^{t-1}$ to $\overline{x}^t$ at time t,incurring a movement cost of c_t,then the state S^{t-1} can be modifies to a state S^t while incurring a cost of $O(c_t)$.

Proof. Let $\overline{x}$ be a fractional state that changes to some other fractional state $\overline{x}'$. Also let S be a state that is both consistent and balanced with respect to $\overline{x}^0$. As S^0 is also consistent with $\overline{x}^0$, it is obvious that it suffices to prove the existence of a k-server state S' such that S'

is balanced and consistent with respece to $\bar{x}'$ and that the cost to move between state S and S' is within a constant factor of the cost of changing from $\bar{x}$ to $\bar{x}'$.
We will consider only the case where $\bar{x}'$ is obtained by $\bar{x}$ by applying only an elementary move, which means that x_i is increased by δ and $x_{i'}$ is decreased by δ, for some leaves i, i'. Also δ can be infinitely small. It is easy to see that this case proves the general case, because we can decompose each change from one state to another to a sequence of finite elementary moves of the above form.
So,consider the two leaves i, i' mentioned before and let p be their lowest common anchestor. The fractional cost of changing $\bar{x}$ to $\bar{x}'$ equals at least $2\delta\frac{W(p)}{\sigma}$.
The transformation of S to S' is the following: We chose a mass of δ arbitrary configurations that do not contain i already and add the leaf i to these configurations. In the same fashion, we remove i' from a mass of δ arbitrary configurations that do contain i', and define S' the new state. The existence of such configurations follows from the fact that S is consistent with $\bar{x}$. Also, since δ can be as small as we want, we can assume that i is added to the mass of a particular configuration C and i' is removed from a mass of another particular configuration C'.
Now, we must prove that S' is consistent with $\bar{x}'$. Observe that the configurations now in S' may not be legal anymore, for they do not have exactly k leaves, and moreover the balance property does not necessarily hold.
Observe that since C contains i and it satisfied the balance property with respece to $\bar{x}$,it must hold now that $n_p(C) \geq \lfloor x_p \rfloor + 1$. Similaryl,for C',it must hold that $n_p(C') \leq \lfloor x_p \rfloor < n_p(C)$. So, there must exist a leaf j in $T(p)$ such that j is contained in C,but not in C'. In the new state(the modified S' state) it holds that $C \leftarrow C - \{j\}$ and $C \leftarrow C \cup \{j\}$.This modification makes all the configurations in the new state legal and keeps the new state consistent with $\bar{x}'$. The total movement cost is at most $4\delta\frac{W(p)}{\sigma-1} = O(\delta\frac{W(p)}{\sigma})$, for $\sigma > 5$, which is within a constant factor of the cost of changing $\bar{x}$ to $\bar{x}'$.
Now, we have to guarantee that S' satisfies the balance property with respect to $\bar{x}'$. We will consider two different occasions:

- For all nodes $\lfloor x_q \rfloor = \lfloor x'_q \rfloor, \lceil x_q \rceil = \lceil x'_q \rceil$: Observe each configuration other than C, C' is balanced with respect to $\bar{x}$. Observe also that $x_q \neq x_{q'}$ only for nodes that belong to the path from i and the child of p and to the path from i' and the child of p. Also, $n_q(C), n_q(C')$ can change only for nodes q belonging to the forementioned paths. However,C,C' were balanced with respect to $\bar{x}$. Observe then that the imbalance gap,as defined before, $G(S', \bar{x}')$ equals

$$3 \cdot 2\delta\frac{W(p)}{\sigma}(1 + \frac{1}{\sigma} + \frac{1}{\sigma^2} + ..) = O(\delta\frac{W(p)}{\sigma})$$

Remeber now the previous theorem and make use of it to obtain a state that is consistent and balanced with respect to $\overline{x}'$ with a cost of $O(\delta\frac{W(p)}{\sigma})$, and finish this part of the proof.

- $\exists q : \lfloor x_q \rfloor \neq \lfloor x'_q \rfloor$ or $\lceil x_q \rceil \neq \lceil x'_q \rceil$. Consider $Q = \{q, \lfloor x_q \rfloor \neq \lfloor x'_q \rfloor \vee \lceil x_q \rceil \neq \lceil x'_q \rceil\ \}$. For each $q \in Q$ either x_q or x'_q is an integer. For the first case observe that $n_q(C'') = x_q = \lfloor x_q \rfloor = \lceil x_q \rceil$ which establishes the balance property. For the second case, since $|x_q - x'_q| \leq \delta$ and $\forall C'' \in S$, such that $n_q(C'') = \lfloor x_q \rfloor, \lfloor x_q \rfloor$ or $n_q(C'') = \lfloor x_q \rfloor > \lceil x'_q \rceil$ it holds that the probability mass of C'' can be at most δ.

After these modifications, the total probability mass of configurations in S' that are not balanced with respect to $\overline{x}'$ is at most 3δ. We calculate the imbalance gap and proceed as before. This leads to the end of our randomized rounding procedure.

□

Chapter 3

Design of efficient fractional algorithms for online problems

3.1 Purely combinatorial methods

3.1.1 Parking Permit

The parking problem was proposed in [8] by Meyerson and is the simplest infrastructure leasing problem, a setting introduced by Antony and Gupta at [10]. In the parking permit problem, we are given K different types of permits which we can purchase. Permit k has cost C_k dollars and duration D_k days. We are given a schedule on which certain days are marked as driving days, and asked to select a set of permits such that the cost is minimized and every driving day is covered. Our goal is to minimize the competitive ratio $\alpha(K)$ of the cost paid by our algorithm to cover all driving days versus the cost paid by the optimum (offline) algorithm which sees the schedule in advance.
Before proceeding, we assume that we can purchase lease k only at times t multiple of l_k, meaning $t \cong 0 mod l_k$. Thus, we may only lose a factor of 2 from the optimal solution, as if the optimal solution would purchase a permit of type k at some moment not a multiple of l_k, we can purchase instead two permits of type k at the greatest multiple of l_k that not exceed t and at the lowest multiple of l_k that exceeds t.

We present a simple fractional algorithm for the parking permit problem. At the start of the algorithm, we set all permits to fraction zero. At any time t we need to drive and the total fraction of permits covering that time is less than one, we use increase x_k^t in the following fashion:

- $\forall 1 \leq i \leq K, x_i^t \leftarrow x_i^t(1+\frac{1}{C_i})$
- $\forall 1 \leq i \leq K, x_i^t \leftarrow x_i^t + \frac{1}{KC_i}$.

We will now prove that this simple fractional algorithm has a cost of $O(logK)$ times the optimal integral solution. First of all observe that during each operation the total cost of the algorithm cannot increase by more than 2 during an operation: Before the operation

$$\sum_{i=1}^{K} F_i(t) < 1$$

and after that

$$\sum_{i=1}^{K}(F_i(t)(1+\frac{1}{C_i})+\frac{1}{KC_i}) \leq \sum_{i=1}^{K} 2F_i(t) + \sum_{i=1}^{K}\frac{1}{C_i} \leq \sum_{i=1}^{K} 2F_i(t)+1$$

,and so the change in the sum of the $F_i(t)$ increases by $\sum_{i=1}^{K} F_i(t) + 1 < 2$.
Now consider an optimal solution and consider one by one the lease the algorithm purchases. Suppose that it purchases the permit (i, k). We prove that after at most $O(logK)$ operations we will have covered that permit , meaning that the fractional value assigned to it is greater or equal to 1. The first C_i operations increase the fractional value for this permit by at least $\frac{1}{K}$ of the permit. After this happens, each operation multiplies the value by a factor of $1+\frac{1}{C_i}$. Let T be the number of the next operations. It holds that $\frac{1}{K}(1+\frac{1}{C_i})^T \leq 1$. This leads to $T \leq logK\frac{1}{log(C_i+1)-logC_i} = \xi_k logK$,for some $\xi_k \in [C_i, C_{i+1}]$ by the mean value theorem. So,after roughly $T = O(C_i logK)$ operations, we have purchased the whole permit, and after that no more operation will occur while permit is active. So, since each operation costs $O(1)$ we conclude that for each permit in the optimal solution the algorithm pays $O(logK)$ what the optimal pays. Hence, the competitive ratio is $O(logK)$ with respect to the integral solution.

3.1.2 Sum-radii-Clustering

Clustering problems are typical in computer science and especially approximation algorithms. These type of problems are studied in [11] and also in [12],[13], where they are treated from a Facility-Location point of view(they follow a Facility-Location relaxation).

In the offline version of Sum-Radii Clustering, the input consists of a metric space (M, d), a cluster opening cost f, and a set $D = u1, ..., un$ of demand points in M, which appear one by one to the algorithm. Also n is not necesarilly known in advance. The goal is to find a collection of clusters $C(p_1, r_1), ..., C(p_k, r_k)$ that cover all demand points in D and minimize the total cost, which equals

$$\sum_{i=1}^{k}(f + r_i)$$

.
We may assume that we can open clusters at the demands points by losing only a factor of 2 from the optimal solution. This is clear since each optimal cluster includes a demand point inside it. Opening a cluster at that point with radius two times bigger than the optimal, we cover the optimal cluster and the claim is proved. We also assume,for simplicity, that we can open clusters with radius of the form $2^k f, \forall k$, losing another factor of 2 from the competitive ratio. That holds because if the optimal algorithm opens the cluster $C(u, r)$ we may open instead the cluster $C(u, 2^k f)$ where $2^{k-1} f < r + f \leq 2^k f$, so in the worst case we only pay double what the optimal pays for that cluster.

We now show a deterministic $O(loglogn)$-competitive fractional algorithm(with respect to an integer solution) for the problem on general metric spaces, where the triangle inequality does not necessarily hold. The algorithm is a generalisation of the algorithm proposed for the Parking Permit problem in the previous section. Let x_{ik} be the extend to which we have opened the cluster centerd at i with radius r_k. Clearly, for the feasibility of a solution one needs to have that for each demand u_j,

$$\sum_{(i,k):(d(i,u_j)\leq r_k} x_{ik} \geq 1$$

. The cost of the solution is

$$\sum_{(i,k)} x_{ik} r_k$$

. In the fractional setting x_{ik} can take also fractional values.
We assume that n is a power of 2 and known in advanced and we also consider the demand locations only as potential cluster centers. So $x_{uk} = 0$ demand a point in u arrives. With these assumptions only lose $O(1)$ from the competitive ratio of our algorithm.
The fractional algorithm acts as follows: Whenever a demand u_j arrives,if $\sum_{(i,k):d(i,u_j)\leq r_k} x_{ik} \geq 1$, then do nothing. Else,as before,while the previous sum is lower than 1, do the following operation:

- $\forall i \in \{1, .., K+1\}, x_{jk} \leftarrow x_{jk} + \frac{1}{K+1}$
- $\forall i \in \{1, .., K+1\}, u_i \in C(u_j, r_k), x_{ik} \leftarrow x_{ik}(1 + \frac{1}{c_k})$

The analysis is very similar to the analysis of the fractional algorithm of the previous section. We first prove that each operation operation increases the cost by at most 1. Indeed, the increase in the cost from the second part of the operation equals

$$\sum_{(i,k):d(i,u_j)\leq r_k} (x_{ik}(1 + \frac{1}{c_k}) - x_{ik}) = \sum_{(i,k):d(i,u_j)\leq r_k} x_{ik} < 1$$

. The increase in the cost from the first part of the operation equals

$$\sum_{k=1}^{K+1} c_k * \frac{1}{c_k(K+1)} = \sum_{k=1}^{K+1} \frac{1}{K+1} = 1$$

.

To bound the number of such operations, observe that after the first c_{k+1} operations caused by demands in $C(p, r_k)$, $\sum_{(i,k):d(i,u_j)\leq r_{k+1}} x_{i(k+1)}$ becomes at least $\frac{1}{K+1}$ due to the first step of these operations. For each subsequent operation caused by a demand in $C(p, r_k)$, all fractions $x_{j(k+1)}$ with $u_j \in C(p, r_k)$ increase by a factor of $1 + \frac{1}{c_{k+1}}$. Therefore increases by a factor of $1 + \frac{1}{c_{k+1}}$. After $O(c_{k+1}logK)$ such increases , $\sum_{(i,k):d(i,u_j)\leq r_{k+1}} x_{i(k+1)}$ becomes at least 1 and so there will be no additional operations for the demands arriving inside that cluster.

So, the total fractional cost of the algorithm for the demands in an optimal cluster of cost c_k is $O(c_{k+1}logK)$. Exploting the fact that $c_{k+1} \leq 2c_k$ (by rounding all radius to the highest power of 2 that does not exceed this radius) and $K = O(logn)$ we get the desired result.

3.1.3 Generalized Connectivity

The Generalized Connectivity problem is treated in [14], where the authors give a logarithmic competitive algorithm for the problem. The requirement function f is a set of demands of the form D = (S, T), where S and T are subsets of vertices in the graph such that $S \cap T = \emptyset$. A feasible solution is a set of edges, such that for each demand D = (S, T) there is a path from a vertex in S to a vertex in T.

Let OPT_{FRAC} be the value of the optimal fractional solution. We claim that OPT_{FRAC} is known up to a factor of two. We skip the proof here,as our goal is only the design of the fractional algorithm.

We describe an online algorithm with competitive factor $O(logm)$. Initially the algorithm assigns to each edge a fractional weight of $\frac{1}{2|E|^3}$. Whenever a demand (S, T) arrives check if the maximum flow from S to T is at least 1. If it is greater than 1, then do nothing. Else, while the flow between S and T is less than 1, then compute a minimum cut C between S and T and for each edge $e \in C, x_e \leftarrow x_e(1 + \frac{1}{c_e})$. It is easy to notice the feasibility of the fractional solution. We bound the number of weight augmentation steps performed during the run of the algorithm.

Claim: The number of weight augmentation stops is at most $O(OPT_{FRAC}log(|E|))$.

Let's start with the following observation that for each edge $e \in E$ always holds $x_e \leq 1 + \frac{1}{c_e}$, because no edge of weight > 1 can be a part of a minimum cut with total weight less than 1. Let

$$\Phi = \sum_{e \in E} c_e x_e^* log(x_e)$$

, where x_e is the weight edge in the optimal solution. Now observe that :

- The initial value of the potential function is $-6alog(|E|) - 2OPT_{FRAC}$.
- The potential function never exceeds OPT_{FRAC}.
- In each step that x_e increases(augmentation step), Φ increases by at least 1.

The first property holds because

$$\Phi = \sum_{e \in E} c_e x_e^* log(\frac{1}{2|E|^3}) = -\sum_{e \in E} c_e x_e^* (3log2 + 3log|E|) = -6alog(|E|) - 2OPT_{FRAC}$$

.
The second propery holds because

$$\Phi \leq \sum_{e \in E} c_e x_e^* log2 \leq 2OPT_{FRAC}$$

.
For the third property observe that the increase in the potential function is

$$\sum_{e \in E} c_e x_e^* log(1 + \frac{1}{c_e}) \geq \sum_{e \in E} x_e^* \geq 1$$

,because the optimal solution guarrantes that the flow between S and T is greater than 1 and also $xlog(1 + \frac{1}{x}) \geq 1$, which can be proved by studying the monotonicity of $f(x) = xlog(x + \frac{1}{x})$ in $[1, \infty]$.
In order to prove the theorem,it suffices to prove that

$$sum_{e \in E} c_e x_e \geq 6OPT_{FRAC} log(|E|) + 4OPT_{FRAC} + 1 = O(OPT_{FRAC} log(|E|)$$

. At each augmentation step observe that -if we denote by C the current minimum (S, T) cut then

$$\sum_{e \in C} \frac{x_e}{c_e} c_e = \sum_{e \in C} x_e < 1$$

. At the first step

$$\sum_{e \in E} x_e c_e \leq |E| \frac{1}{2|E|^3} 2|E|^2 = 1$$

and since the number of augmentations is at most $O(OPT_{FRAC} log(|E|))$ the cost of the fractional solution is $O(OPT_{FRAC} log(|E|)$, which proves the desired competitive ratio.

3.1.4 Generalized Cuts

The generalized cuts problem is almost identical to the generalized connectivity. The next approach is used in the same paper as the forementioned problem. We present a $O(log(|E|)$-competitive algorithm for the generalized cuts problem. As previously, the algorithm assigns each edge a length of $\frac{1}{2|E|^3}$. The main body of the algorithm is almost identical with the algorithm presented in the previous section. Again,whenever a demand (S, T) arrives, if the length of the shortest path from S to T is already at least 1 , then do nothing. Else, while the length of the shortest path from S to T is less than 1 perform a length augmentation:

- Compute the shortest path P from S to T.
- $\forall e \in P, x_e \leftarrow x_e(1 + \frac{1}{c_e})$.

Claim: The number of weight augmentation stops is at most $O(OPT_{FRAC}log(|E|)$. For each edge $x_e \leq 1 + \frac{1}{c_e}$, for an analogous reason as before. Define the potential function

$$\Phi = \sum_{e \in E} x_e^* c_e log(x_e)$$

. Then it holds that:

- The initial value of the potential function is $-6alog(|E|) - 2OPT_{FRAC}$.
- The potential function never exceeds OPT_{FRAC}.
- In each step that x_e increases(augmentation step), Φ increases by at least 1.

The first property holds because

$$\Phi = \sum_{e \in E} c_e x_e^* log(\frac{1}{2|E|^3}) = -\sum_{e \in E} c_e x_e^* (3log2 + 3log|E|) = -6alog(|E|) - 2OPT_{FRAC}$$

.

The second propery holds because

$$\Phi \leq \sum_{e \in E} c_e x_e^* log2 \leq 2OPT_{FRAC}$$

.

For the third property observe that the increase in the potential function is

$$\sum_{e \in E} c_e x_e^* log(1 + \frac{1}{c_e}) \geq \sum_{e \in E} x_e^* \leq 1$$

,because the optimal solution guarrantes that the flow between S and T is greater than 1 and also $xlog(1+\frac{1}{x}) \geq 1$, which can be proved by the inequality $logx+1 \leq x$.
In order to prove the theorem,it suffices to prove that

$$sum_{e\in E}c_e x_e \geq 6OPT_{FRAC}log(|E|)+4OPT_{FRAC}+1 = O(OPT_{FRAC}log(|E|)$$

. At each augmentation step observe that -if we denote by C the current minimum (S,T) cut then

$$\sum_{e\in C}\frac{x_e}{c_e}c_e = \sum_{e\in C}x_e < 1$$

. At the first step

$$\sum_{e\in E}x_e c_e \leq |E|\frac{1}{2|E|^3}2|E|^2 = 1$$

and since the number of augmentations is at most $O(OPT_{FRAC}log(|E|))$ the cost of the fractional solution is $O(OPT_{FRAC}log(|E|)$, which proves the desired competitive ratio.

3.2 A primal-dual approach

3.2.1 Introduction

The primal-dual tecnhique is a very strong technique that has proved to be very helpful for many NP-hard problems, in the area of the design and analysis of approximation algorithms, although it first started as a tecnhique for designing exact algorithms to problems such as matching. In the area of approximation algorithms, the tecnhique was firstly introduced by Williamson and Goemans in [15].
When a facing a problem with a linear programming formulation, we can also write its dual problem, which general form is presented below. The solution of the dual problem provides a lower bound to the solution of the primal problem.[1] However in general the optimal values of the primal and dual problems need not be equal. Their difference is called the duality gap. For convex optimization problems, the duality gap is zero under a constraint qualification condition. Thus, a solution to the dual problem provides a bound on the value of the solution to the primal problem; when the problem is convex and satisfies a constraint qualification, then the value of an optimal solution of the primal problem is given by the dual problem.

In this section, we present competitive fractional algorithms for online problems, by exploiting the relation between the primal and the dual problems that occur from the linear programming formulation of each of these problems. These techniques have been established

Primal program	**Dual program**
min $\sum_{i=1}^{n} c_i x_i$	max $\sum_{j=1}^{m} b_j y_j$
s.t.:	s.t.:
$\sum_{i=1}^{n} a_{ij} x_i \geq b_j, \forall j : 1 \leq j \leq m$	$\sum_{j=1}^{m} a_{ij} y_j \leq c_i, \forall i, 1 \leq i \leq n$
$x_i \geq 0, \forall i : 1 \leq i \leq n$	$y_j \geq 0, \forall j, 1 \leq j \leq m$

Ski-rental primal program	**Ski-rental dual program**
$B \cdot x + \sum_j z_j$	$\sum_j y_j$ s.t.:
s.t.:	$\sum y_j \leq B$
$x + z_k \geq 1$	$\forall j, 0 \leq y_j \leq 1$
$x \geq 0, \forall j, z_j \geq 0$	

and extended mainly by Niv Buchbinder, and they were a breakthrough as they indicate an easier way to obtain fractional algorithms that are highly competitive. So, they appear useful in designing efficient randomized algorithms, followed by the essential randomized rounding tecnhiques. The main idea of these algorithms is that whenever a new constraint arrives, if this constraint is not satisfied then the algorithm raises the variables of the primal and dual variables in a certain way, such that some properties hold. This is non-trivial and at first sight someone cannot understand why the variables are updated in such a way. However, this occurs from a solution of a system of differential equations. Examples are presented below and which system is this is indicated in the last subsection. These examples are taken from [21], by the work of Niv Buchbinder and Joseph(Seffi) Naor, who established the method.

3.2.2 The Ski-Rental problem

The ski-rental problem is reffered again in chapter 1. We begin by indicating a fractional competitive algorithm for the problem via a primal-dual approach. The online algorithm follows:

- Set $x \leftarrow 0$.
- Each new day, if $x < 1$

- $z_j \leftarrow 1 - x$
- $x \leftarrow x(1+\frac{1}{B}) + \frac{1}{cB}$,where c a constant to be determined later.
- $y_j \leftarrow 1$

We now break the analysis into two parts:

- The primal and dual solutions are feasible.
- In each day, the ratio between the change in the optimal and the dual objective function is bounded by $1+\frac{1}{c}$.

The above would imply that the algorithm is $(1+\frac{1}{c})$-competitive, by the weak duality theorem.
In order to prove the feasibilty of the primal observe that at each day at the beggining of which $x < 1$ it holds that at the end of it that $x' + z_j \geq x + (1-x) = 1$,hence the constraints of the primal program hold true. Here, x' equals the new value of x, namely $x' = x(1+\frac{1}{B}) + \frac{1}{cB}$. For the feasibility of the dual observe that since x only increases and each time this happens one y_j becomes 1, we need to prove that the update of x does not happen more than B times. At the k-th time x is updated, we have that

$$x = \frac{1}{cB}\sum_{i=0}^{k-1}(1+\frac{1}{B})^i$$

Setting $k = B$ and exploting the geometric series we get that the expression equals $x = \frac{(1+\frac{1}{B})^B - 1}{c}$. Thus setting $c = (1+\frac{1}{B})^B - 1$, we get the feasibility of the dual solution.
We proceed with the second part of the analysis. If $x \geq 1$ then there is nothing to prove. Else , the change in the primal objective function equals $B[x(1+\frac{1}{B}) + \frac{1}{cB} - x] + 1 - x = \frac{1}{c} + 1$ and the change in the dual cost is exactly 1,hence the algorithm is indeed $(1+\frac{1}{c})$ competitive, and in particular $\frac{e}{e-1}$-competitive, because $c \leq lim_{B\to+\infty}[(1+\frac{1}{B})^B - 1] = e - 1$

3.2.3 Weighted Caching Problem

In the weighted caching problem there is a size of size k, and pages $\{1, .., n\}$, associated with weights $w_1, w_2, .., w_n$. The weights denote the cost of fetching the pages into the cache. Pages are requested online and the goal is to minimize the total weight of the pages we fetch into the cache. The following fractional $O(logk)$-algorithm is presented in .

The variables $y_{p,t}$ denote the fraction of page p missing from the cache at time t. Let p_t denote the page p missing from the cache at time t. Let p_t denote the page requested at time

Weighted Caching primal program

min $\sum_{p=1}^{n} \sum_{t=1}^{T} w_p z_{p,t} + \sum_{t=1}^{T} \infty y_{p,t}$

s.t.:

$\forall t, S \subset [n], |S| > k : \sum_{p \in S} y_{p,t} \geq |S| - k$

$\forall t, p z_p, t \geq y_{p,t-1} - y_{p,t}$

$\forall t, p z_{p,t}, y_{p,t} \geq 0$

Weighted Caching Dual Program

max $\sum_{t=1}^{T} \sum_{S} (|S| - k) a_{s,t}$

$\forall t, p \neq p_t : \sum_{S: p \in S} a_{S,t} - b_{p,t+1} + b_{p,t} \leq 0$

$\forall t, p : b_{p,t} \leq w_p$

$\forall t, p, |S| > k : a_{t,S}, b_{p,t} \geq 0$

t. The primal LP constraints states that at any time t,for any set S of pages with $|S| > k$,then the total number of pages outside the cache greater or equal to $|S| - k$. The variables $Z_{p,t}$ denote the fraction of page p that is fetched at time t. The first term in the objective function is the fetching cost and the second term enforces the requirement that page $p_t$must be in the cache at time t. The dual of the LP consists of variables for each set S and time t, and variables for each page p and time t.

The algorithm follows:

For each page p and time t,we maintain the following relation between primal and dual variables:

$$y_{p,t} = \frac{1}{k}(exp(\frac{b_{p,t+1}}{w_p} ln(1 + k)) - 1)$$

Consider now a request for page p_t at time t. Initially we set $y_{p,t} = y_{p,t-1}$ for all p and hence $b_{p,t+1} = b_{p,t}$, because $y_{p,t}$ if seen as a one-variable function of $b_{p,t+1}$ is one to one. Also, $y_{p_t,t} = b_{p_t,t+1} = 0$. Let $S = \{p : y_{p_t} < 1\}$, namely the set of the pages that at time t have a fraction of them present in the cache. We start increasing $a_{S,t}, b_{p,t+1}$ at the same rate for all pages $p \in S - \{p_t\}$ uniformly. By the relation between dual and primal variables one can observe that $y_{p,t}$ increases, thus meaning that pages start being evicted from the cache(more specifically, a fraction of their). The following procedure continues until the primal constraints are satisfied. If $y_{p,t} = 1$ for some page p during this procedure, we redefine S by forgetting that page: $S \leftarrow S - \{p\}$.

The analysis follows:

- **Feasibility**: We first note the feasibility of the above algorithm. Observe that $b_{p,t+1}$ never exceed w_p for some page p, and also $y_{p,t} = 1$ leads to $b_{p,t+1} \leq w_p$, because the function described above is increasing in $b_{p,t+1}$. The dual constraint is also satisified as $a_{S,t}$ and $b_{p,t+1}$ increase at the same rate for any page $p \in S - \{p_t\}$ and hence the contribution of the variable $a_{S,t}$ to constraint is cancelled by the contribution of $b_{p,t+1}$ (they have opposite signs).

- **Cost Analysis**: Let P, D be the values of the primal and dual solutions procuded by the algorithm. In order to obtain an $O(logn)$-competitive algorith, it suffices to show that the derivative of P is at most $O(logn)$ times the derivative of D. Due to the increase of the variable $a_{S,t}$ one gets at some time when the algorithm decides to increase that dual variable that $\frac{\partial D}{\partial a_{S,t}} = |S| - k$.

Observe now that $\frac{\partial y_{p,t}}{\partial b_{p,t+1}} = \frac{ln(1+k)}{w_p}(y_{p,t} + \frac{1}{k})$. Also , as $b_{p,t+1}, a_{S,t}$ are raised at the same rate then the derivative of the primal cost at that time equals

$$\sum_{p \in S - \{p_t\}} w_p \frac{\partial y_{p,t}}{\partial b_{p,t+1}} = \sum_{p \in \{p_t\}} ln(1+k)(y_{p,t} + \frac{1}{k}) \leq$$

$$\leq ln(1+k)(|S| - k + \frac{|S|-1}{k}) \leq 2ln(1+k)(|S| - k) = 2ln(1+k)\frac{\partial D}{\partial a_{S,t}}$$

,since $\sum_{p \in S} y_{p,t} < (|S| - k)$ and $\frac{x-1}{k} \leq x - k, \forall x \geq k+1$, as it reduces to $k^2 - 1 \leq x(k-1)$ or $k + 1 \leq x$.

By the above, we conclude that the above algorithm is $O(logn)$-competitive.

3.2.4 The online Packing-Covering Framework

The online packing-covering problem is a special case of linear programming and a generalisation of many important problems, like the well-known NP-complete Set Cover problem. In the covering problem the goal is to minimize an objective function of the form $\sum_i c_i x_i$, and at the same time all the coefficients of the indicator variables in the constraints are either 0 or 1.

.

We present three algorithms for the covering/packing problem. All three algorithms achieve a competitive ratio of $O(logd)$ and hence their performance is the same. However, their properties vary and some algorithms are more suitable than other algorithms for certain applications.

Whenever a new primal constraint $\sum_{i \in S(j)} x_i \geq 1$ arrives do:
While $\sum_{i \in S(j)} x_i < 1$

Packing-covering primal program	**Packing-Covering dual program**
min $\sum_i c_i x_i$	max $\sum_j y_j$
s.t.:	s.t.:
$\forall 1 \leq j \leq m, \sum_{i \in S(j)} x_i \geq 1$	$\forall 1 \leq i \leq n, \sum_{j \mid i \in S(j)} y_j \leq c_i$
$\forall 1 \leq i \leq n, x_i \leq 0$	$\forall 1 \leq j \leq m, y_j \leq 0$

- $\forall e \in S(j) : x_i \leftarrow x_i(1 + \frac{1}{c_i}) + \frac{1}{|S(j)|c_i}$
- $y_j \leftarrow y_j + 1$

We may assume that $c_i \geq 1$. Let $d = max_j |S(j)|$ be the maximum size of a covering constraint. As before we move with the following:

- The algorithm produces a feasible solution
- In each iteration $\Delta P \leq 2\Delta D$
- Each packing constraint is violated by $O(logd)$.

The proof of the first bullet is obvious, as it is the condition for the while loop to terminate. For the second bulltet ,observe that $\Delta D = 1$ for an execution of the while loop, and also:

$$\Delta P = \sum_{i \in S(j)} c_i(x_i(1 + \frac{1}{c_i}) + \frac{1}{|S(j)|c_i} - x_i) = \sum_{i \in S(j)} (x_i + \frac{1}{|S(j)|}) = 1 + \sum_{i \in S(j)} x_i < 2$$

For the third bullet observe that whenever we increase a variable y_j by one unit,we also increase the variable x_i and one can observe that

$$x_i \geq \frac{1}{d}((1 + \frac{1}{c_i})^{\sum_{j \mid i \in S(j)} y_j} - 1)$$

We prove the above relation by induction. At the start, $x_i = 0$ and all y_j equal zero so the claim follows. Whenever some y_k increases by 1 then the new value of x equals

$$x(1 + \frac{1}{c_i}) + \frac{1}{|S(j)|c_i} \geq x(1 + \frac{1}{c_i}) + \frac{1}{dc_i} \geq \frac{1}{d}((1 + \frac{1}{c_i})^{\sum_{j \mid i \in S(j) - \{k\}} y_j} - 1)(1 + \frac{1}{c_i})^{y_k} + \frac{1}{dc_i} -$$

$$= \frac{1}{d}((1 + \frac{1}{c_i})^{\sum_{j \mid i \in S(j)} y_j} - 1)$$

Now,It is easy to see that the algorithm never updates a variable greater than 1 as it would not need to,for the condition of the while loop would be false. So, the last time a x_i was updated, its value is $x_i(1+\frac{1}{c_i})+\frac{1}{|S(j)|c_i} \leq 1(1+1)+1=3$,as $c_i \geq 1$. So it holds that

$$3 \geq x_i \geq \frac{1}{d}((1+\frac{1}{c_i})^{\sum_{j|i\in S(j)} y_j}-1)$$

and hence

$$\sum_{j|i\in S(j)} y_j \leq O(c_i logd)$$

We move with the second algorithm: Whenever a new primal constriant $\sum_{i\in S(j)} x_i \geq 1$ do:
While $\sum_{i\in S(j)} x_i < 1$:

- Increase variable y_j in a continuous fashion.
- For each variable x_i that appears in the primal constraint increase x_i as below:

$$x_i \leftarrow \frac{1}{d}[exp(\frac{ln(1+d)}{c_i}\sum_{j|i\in S(j)} y_j)-1)$$

Clearly, there are variables in the function presented above that may refer to future,unknown constraints. We ignore these variables, as we do not know them and set them to zero. We follow the typical approach presented above:

- The algorithm produces a primal feasible solution
- In each iteration j: $\frac{\partial P}{\partial y_j} \leq 2ln(1+d)\frac{\partial D}{\partial y_j}$
- Each constraint in the dual program is feasible.

The first claim is obvious by the while-loop condition. Let us now prove thse second claim.Observe that $\frac{\partial D}{\partial y_j}=1$. The derivative of the primal cost is

$$\frac{\partial P}{\partial y_j}=\sum_{i\in S(j)} c_i\frac{\partial x_i}{\partial y_j}=$$

$$=\sum_{i\in S(j)} c_i\frac{ln(1+d)}{c_i}\frac{1}{d}exp(\frac{ln(1+d)}{c_i}\sum_{j|i\in S(j)} y_j)=$$

$$= ln(1+d) \sum_{i \in S(j)} \frac{1}{d}(exp(\frac{ln(1+d)}{c_i} \sum_{j|i \in S(j)} y_j) - 1) + \frac{1}{d}) =$$

$$ln(1+d) \sum_{i \in S(j)} (x_i + \frac{1}{d}) \leq 2ln(1+d)$$

, where the last inequality follows from the infeasibility of the constraint.
To prove the third bullet, we observe that $x_i \leq 1$ since it would not belong to some unsatisfied constraint. So, by the update rule one gets that

$$x_i \leftarrow \frac{1}{d}[exp(\frac{ln(1+d)}{c_i} \sum_{j|i \in S(j)} y_j) - 1) \leq 1$$

, which simplifies to

$$\sum_{j|i \in S(j)} y_j \leq c_i$$

, hence the feasibility olf the dual constraint.

We move with the third algorithm:
Whenever a new primal constraint $sum_{i \in S(j)} x_i \geq 1$ arrives do: While $sum_{i \in S(j)} x_i < 1$ do:

- Increase variable y_j continuously.
- If $x_i = 0$ and $\sum_{j|i \in S(j)} y_j = c_i$ then set $x_i \leftarrow \frac{1}{d}$
- For each variable x_i, $\frac{1}{d} \leq x_i \leq 1$ such that $x_i \in S(j)$ increase x_i according to the rule:

$$x_i \leftarrow \frac{1}{d} exp(\frac{\sum_{j|i \in S(j)} y_j}{c_i} - 1)$$

We move with the above claims:

- The algortihm produces a primal solution.
- Each packing constraint is violated by a factor of at most $O(logd)$.
- $P \leq 2D$.

For the proof of the first claim, observe that the condition of the while-loop guarantees the feasibility of the solution. For any dual constraint $\sum_{j|i \in S(j)} \leq c_i$, as before, the variable

x_i which corresponds to that constraint cannot exceed 1, otherwise it would not belong to an unsatisfied constraint. This means that

$$x_i \leq \frac{1}{d}exp(\frac{\sum_{j|i\in S(j)} y_j}{c_i} - 1) \leq 1$$

After the calculations one get that

$$\sum_{j|i\in S(j)} \leq c_i(1 + lnd) = O(c_ilnd)$$

We now need to prove the third bullet. We can partition the contribution ot the primal cost into $Cost_1$, $Cost_2$, where the first denotes the contribution to the primal cost from the first step of the algorithm(the increase of the variables to $\frac{1}{d}$, and the second one denotes the contribution from the uniform increase of the variables x_i. Let $\overline{x_i} = min(x_i, \frac{1}{d})$. Clearly, $Cost_1 = \sum_i c_i\overline{x_i}$. This means that if $\overline{x_i}$ is positive, then $\sum_{j|i\in S(j)} y_j \geq c_i$. and if $y_j > 0$ then

$$\sum_{i\in S(j)} \overline{x_i} \leq \sum_{i\in S(j)} \frac{1}{d} \leq 1$$

Using the complementary slackness conditions one gets that

$$\sum_i c_i\overline{x_i} \leq \sum_i (\sum_{j|i\in S(j)} y_j)x_i =$$

$$= \sum_j \sum_{i\in S(j)} y_j \leq \sum_j y_j$$

The above inequality is translated into $Cost_1 \leq \sum_j y_j$. We now bound $Cost_2$: Whenever the algorithm updates the primal and dual solutions, observe that it must hold $\frac{\partial D}{\partial y_j} = 1$, and also notice that

$$\frac{\partial P}{\partial y_j} = \sum_{i\in S(j)} c_i \frac{\partial x_i}{\partial y_j} = \sum_{i\in S(j)} c_i \frac{x_i}{c_i} \leq 1$$

, because when updating the covering constraint is infeasible. By the above one gets that $Cost_1 + Cost_2 \leq 2D$.

3.2.5 Indicating the basic idea

Observe for example the second algorithm for the online packing-covering problem. It is clear not obvious why these update rules where chosen, why someone would use this function between primal and dual variables. The idea behind this to bound the derivative of the primal cost(let it be P) as a function of the derivative of the dual cost. So, one needs to prove that

$$\frac{\partial P}{\partial y_j} = \sum_{i \in S(j)} c_i \frac{\partial x_i}{\partial y_j} \leq a \frac{\partial D}{\partial y_j}$$

,where a is the desired competitive ratio. Now,suppose that the derivative of the primal cost equals

$$\sum_{i \in S(j)} c_i \frac{\partial x_o}{\partial y_j} = A \sum_{i \in S(j)} (x_i + \frac{1}{d})$$

,for some A, depending on a(their relation will be found later). Then since

$$\sum_{i \in S(j)} x_i \leq 1, \sum_{i \in S(j)} \frac{1}{d} \leq 1, \frac{\partial D}{\partial y} = 1$$

one gets that

$$A \sum_{i \in S(j)} (x_i + \frac{1}{d}) \leq 2A \frac{\partial D}{\partial y_j}$$

Clearly, $a = 2A$ and in order to satisfy the previous equality one needs to solve the following system of differential equations:

$$\frac{\partial x_i}{\partial y_j} = \frac{A}{c_i}(x_i + \frac{1}{d})$$

By solving the previous system one gets that x_i are:

$$x_i = B \cdot exp(\frac{A}{c_i} \sum_{l | i \in S(j)} y_l - \frac{1}{d})$$

, where B is an arbitrary constant, that is derived from the boundary conditions. The first boundary conditions corresponds to the beginning of the algorithm when $x_i = 0$ and athis happens when $\frac{1}{c_i} \sum_{j | i \in S(j)} = 0$. The second boundary condition corresponds to the tightness of the dual constraint, where $x_i = 1$ and $\frac{1}{c_i} \sum_{j | i \in S(j)} y_j = 1$. By the previous boundary conditions one gets $B = \frac{1}{d}$ and $A = ln(d+1)$. So, the function used in the algorithm for connecting the primal with the dual variables is

$$x_i = \frac{1}{d} \cdot exp(\frac{ln(d+1)}{c_i} \sum_{l | i \in S(j)} y_l - \frac{1}{d})$$

Chapter 4

Incremental Clustering

4.1 Problem Definition

Given a metric d defined on a set P of n points, we define the ball $B(v, r)$ centered at $v \in P$ and having radius $r \geq 0$ to be the set $\{q \in P | d(v, q) \leq r\}$. For $k > 0$, a k-cover for subset Q of P is a set of at most k balls, each centered at a point in P , whose union covers (contains) Q. The cost of a set D of balls, denoted $cost(D)$, is the sum of the radii of those balls. In this chpater, we consider the (metric) minimum cost k-cover problem: Given a metric d on a set P of n points as above, and an integer $k > 0$, compute a minimum cost k-cover for P. We follow the approach of [16],[17] and [18] and study the incremental sum-radii-clustering problem (ISRC) is the next one: Find a permutation of the request sequence such that for each $k \in \{1, .., n\}$ the optimal solution for the k-sum-radii-clustering with the clusters opening at the first k centers in the permutation is within a factor of c from the optimal solution for the k-sum-radii-clustering problem. Then,the approximation ratio of the algorithm is c.

4.2 The case of the line

Consider n points on the line sorted in increasing order,hence $r_1 < r_2 < .. < r_n$. If the optimal algorithm would answer the k-sum-raddii clustering question how will it react? Observe that all interval(clusters) that the optimal algorithm chooses must have their end-points at two points of the request sequence and the center of it would be the midpoint of these. Instead of choosing the k clusters,the optimal algorithm would choose the $k - 1$ ”spaces”,meaning the intervals of the form $[r_i, r_{i+1}]$ that it would not cover with some cluster. Hence it would sort all intervals of the form $[r_i, r_{i+1}]$ for $i \in \{1, .., n-1\}$ in decreasing length and discard the first k of them.

Theorem 8. *There is a 2-approximation algorithm for ISRC on the line.*

Proof. The incremental algorithm works as follows: For $i = 1$ to n the next point in the permutation is the left endpoint of the interval of the optimal solution that is has not been put yet in the permutation. Observe that due to the nature of the optimal solution,the transition from k to $k+1$ means that the optimal algorithm discards one additional interval of the above form and hence there is exactly one interval,call it I, of the optimal solution with k points that is partitioned into two new intervals in the new solution. Clearly,the left interval that I breaks to has its left endpoint in the permutation,while the right one does not have it. Hence we put it next in the permutation and observe that now all intervals in the new solution have their left endpoint in the permutation so far. At each time step, the optimal for all intervals it creates,puts the optimal center at their midpoint,while the proposed algorithm has opened a center at the left endpoint of the interval,hence it pays at most 2 times what the optimal pays for each interval(exactly 2 for intervals of non zero length and exactly the same cost-zero-for single point clusters). Hence in overall it pays 2 times what the optimal pays and the theorem is proved. □

Theorem 9. *There is a 2-approximation algorithm for ISRC on the circle.*

Proof. We reduct this problem to the instance on the line. Let r_i be named in clockwise order and let x_i be the distance of point i in this order from point r_1 .Also,let $\delta = max_{i\in\{1,..,n\}}|x_{i+1} - x_i|$, where the indices are taken modulo n. We say an interval with endpoints r_i, r_{i+1} in the clockwise order a primitive interval. We prove that no optimal algorithm would cover a primitive interval of length δ with a cluster if there was an uncovered primitive interval with length l such that $l < \delta$. Consider all clusters that cover a primitive interval of length δ. Consider any cluster C with endpoints r_i, r_j that covers a primitive interval $r_{i'}, r_{i'+1}$ of length δ. Then observer that r_{j+1} must be covered by a cluster C'. We extend C' from $r_{i'+1}$ to r_{j+1} and restrict C from r_i to $r_{i'}$(obviously the centers of the clusters also move). Observe that the cost is increased by $\frac{|x_{j+1}-x_j|}{2}$ and it is decreased by $\frac{\delta}{2}$, which is non-negative. If it strictly negative we are done. Else we repeat the same procedure with the cluster C' and so on,until the change in the cost is negative(cluster C' must now have a primitive interval of length δ). Clearly, this procedure must end sometime because there is an uncovered primitive interval of length $l < \delta$,proving the desired result. For the algorithm we can safely ”forget” any primitive interval of length δ and thus transform the circle into the line,applying the previous algorithm and yielding the desired approximation ratio. □

4.3 Algorithms on Hierarchically Separated Trees

A Hierarchically Separated Tree(or HST from now on) is a rooted tree for which there is a number σ such that $\sigma * d(p(u), u) = d(u, c(u))$ for any node u (nor a leaf,neither the root) of the tree, where $p(u), c(u)$ denotes the parent and child of u respectively. We focus on HSTs with $\sigma \leq \frac{1}{2}$ and all leaves have equal length from the root. We denote by T_u the tree rooted at node u and by h_u the length of the longest path from u to some leaf of T_u. We also assume that the demands lie at all the leaves of the HST, as we can discard any leaf u which does not have a request at it(and similarly any node u such that no member of the request sequence lies in T_u). We prove the following:

Claim: For the the k-sum-raddi clustering problem on HSTs, if we decide to open a cluster at a node u then there is no point in opening a cluster with radius greater than h_u.

Proof. Suppose that we open a cluster C at node u with radius greater than h_u,say R. Let the leaf l served in C with the highest common ancestor with u, say l. Clearly, l must also belong to C. Opening however a cluster at l with radius h_l covers C and its radius is smaller, by exploiting the geometric decrease of the lengths of the edges and that $\sigma \leq \frac{1}{2}$. □

Theorem 10. *Let an HST T with $\sigma \leq \frac{1}{2}$. Then there exists a 2-approximation algorithm for ISRC on T.*

Proof. Let r be the root of the tree and u_1, u_2 be its children. We prove that the optimal algorithm for the $(k+1)$-cluster instance would choose the cluster C centered at a non-lead node u of the k-cluster instance with the smallest $f_u = h_u - 2 * d(c_1(u), u)$ value and open two clusters at each two children, forgetting the cluster at u. We then say that cluster C is split.We prove the claim by induction on k. For $k = 1$ the statement is trivial. Assume that the algorithm holds for k and we now prove it for $k+1$. Let l'_1, l'_2 be the number of clusters the optimal algorithm for the $(k+1)$-instance opens at subtrees T_{u_1}, T_{u_2} respectively. Let l_1, l_2 be the same for the k-instance. Clearly, $0 < l'_1, l'_2, l_1, l_2 < k+1$. Observe that ,due to the induction hypothesis, the algorithm is valid on subtrees T_{u_1}, T_{u_2} with l'_1, l'_2 clusters respectively. Assume now,without loss of generality, that the center-call it w- of the cluster with the largest f value in the optimal k instance solution, lies in T_{u_2}. We show that each optimal solution for which $l'_2 > l_2 + 1$,can be transformed to an optimal solution with $l_2 + 1$ clusters in T_{u_2} . Running the algorithm for the k instance at r, let T_x be the last cluster that was split at T_{u_1}. If $l'_2 > l_2 + 1$ then the optimal algorithm for the $k+1$ instance does not open T_x, while it opens two clusters at w_1, w_2, the children of w. Observe however that since T_w has not been split in the k instance by our algorithm, either there is another subtree T_y

of T_{u_2} such that $f_y \le f_w$ (T_w might be a subtree of T_y, but the inequality still holds) and the algorithm split T_y after T_x leading to $f_x \le f_y$, either T_x was the last cluster that was split for the k instance. Whatever holds, $f_x \le f_w$ and opening two clusters at the children of x instead of the children of w cannot increase the cost. The transformed instance has one less cluster in T_{u_2}. Applying this procedure we can make the numbers of clusters in T_{u_2} equal to $l_2 + 1$. A similar transformation holds when $l_2' \le l_2$, which means that we can assume that $l_2' = l_2 + 1$. The optimality of the algorithm now holds by induction hypothesis on each subtree.
Our algorithm now proceeds in a divide and conquer approach. For T, solve recursively the problem on T_{u_1} and T_{u_2} and let π_1, π_2 be the permutations that are returned by the algorithm for these two subtrees. The algorithm returns a permutation π of the nodes of T. The first two clusters open at the first node that π_1 indicates and at the first node that π_2 indicates, , meaning that the first two elements in π are the first elements of π_1, π_2. Then the optimal algorithm at each time adds one additional cluster at exactly one of these two subtrees. If it adds it at T_{u_1} then next in the permutation is the next node in π_1,else the next node in π_2. Inductively over the number of available clusters and over all subtrees assume that the algorithm is 2-competitive. The induction basis is trivial. Then at each time step if the optimal has l_1 clusters at T_{u_1} and l_2 clusters at T_{u_2}, so our algorithm does. So it pays the double for each subtree and hence the double in total. $\square$

We extend the above algorithm to arbitraty HSTs:

Theorem 11. *: There is a 2-approximation algorithm for ISRC on an HST where $\sigma \le \frac{1}{\Delta+1}$, where Δ is the maximum number of children one node has. There is also a 2-approximation algorithm for ISRC on an HST where all children,except for the leaves,have the same number of children Δ and $\sigma \le \frac{1}{\Delta}$.*

Proof. Define as before for any node u the function $f_u = d_u(h_u - D_u) - h_u$, which corresponds to the change in the cost if we substitute a cluster rooted at u with clusters rooted at the children of u. Here $D_u = d(u, c_1(u))$(for ease of notation below we drop the notation for u and say D meaning D_u) and d_u equals the number of children of u. The analysis is very similar to the previous one but instead of applying the greedy procedure at each time step, it applies it only when the optimal solution is strictly better.It is also based on the following observations(which are trivial in the case of the binary tree):

- $f_u < 0, \forall u$: It suffices to show that $h_u(d_u - 1) < d_u D$. However $h_u(d_u - 1) < \frac{D}{1-\sigma}(d_u - 1)$. Then $\frac{D}{1-\sigma}(d_u - 1) \le d_u D$ because it reduces to $1 \ge d_u\sigma$, which holds by the definition of σ.

- $f_u \leq f_x$, for each child x of u: The above expression reduces to $d_u(h_u - D) \leq d_x(h_u - D - D\sigma) + D$. Clearly,the worst-case scenario occurs when d_u equals Δ and d_x equals 2. After that observe that the inequality, after bringing all terms at the left-hand side, is a linear function of h_u with non-negative slope. Hence it suffices to prove that the statement holds for the maximum h_u, meaning $h_u = \frac{D}{1-\sigma}$. Because $(\Delta+1)\sigma \leq 1$, one can observe that $\frac{\Delta\sigma D}{1-\sigma} \leq D$, which establishes the inequality.

The above inequalities in the case of an HST with the same number of nodes and $\sigma \leq \frac{1}{\Delta}$ can easily be derived again. $\square$

4.4 General metric spaces

Theorem 12. *Given an a-approximation algorithm for the k-sum-radii-clustering problem,we can consruct a $8a$-approximation deterministic algorithm and a $2ea$-approximation randomized algorithm for the ISRC problem.*

Proof. We follow the approach of Huolong Lin et.al. at [19]. For a solution S define C_S the set of centers of the clusters that they are open. Define $S_1 \preceq S_2$ to be $C_{S_2} \subseteq C_{S_1}$. $ben(S)$ equals n minus the number of clusters S uses. $cost(S)$ is defined in the obvious way. To establish the result, one needs to show that given a state S and a number k there exists a state $S' \subseteq S$ with at most k clusters such that $cost(S') \leq 2cost(S) + cost(opt(k))$. Define also $c_S(x)$ the cluster in which x is included in in S and also $r_s(c_S(x))$ the radius of that cluster. Let's proceed with the algorithm.If $|C_S| \leq k$, there is nothing to prove. Else let S'' be a state which uses k clusters. Set $A = C_S, B = C_{S''}, C_{S'} = \emptyset$. We will say that $x \in C_S$ dominates y if and only if y belongs to the cluster in S centered at x.Now, apply the following procedure until $A = \emptyset$. Choose a $y \in B$ and choose a $x \in A$ such that the cluster of x in S and the cluster of y in S'' have a non-empty intersection. .Then let m be the largest radius(in S'') among the clusters with centers in B such that the cluster of x in S has a non-empty interesection with these clusters. Set $C_{S'} = C_{S'} \cup \{x\}$ and open a cluster at x with radius $2m + r_S(c_S(x))$. Observe that all clusters mentioned above and belonging to S'' are covered from the one opened, due to triangle inequality. Delete all these centers from B. At the end,observe that there is no uncovered point(as all clusters in S'' are completely covered) and we have a feasible solution using at most k clusters. It is easy to observe that the cost of the new clusters is at most $2cost(S) + cost(S'')$, as each cluster's radius in S and S'' is charged at most one time. Since $S \preceq S'$, the theorem is proved. $\square$

Chapter 5

Online Mobile Facility Location

5.1 Problem formulation

In the online mobile facility location, we are given a finite metric space $M(V, d)$ and an initial configuration of facilities C on points of the metric with a certain weight each. We assume that no two facilities initially lie at the same point.Each time a request arrives and we must serve it with one of the facilities. However, the request can move also, so we have 3 choices:move the facility onto the request,move the request onto the facility, or move both of them to an intermediate point. After each requested is served,the facility's weight it is attached to increases by the weight of the demand. The cost for moving a unit of mass D for x equals $D * x$. The objective is to minimize the total movement cost. This is one online variant of the problem that was introduced in [20] by Friggstad and Salavatipour. There they tried the offline variant of the problem and proposed a constant approximation algorithm rounding of a linear relaxation of the problem.

5.2 Problem simplification

Proof. : We prove the two-facility occasion which can easily be generalized. Consider a triangle with vertices a, S, b where ab is of weight 1 and the other two of weight Δ for some Δ sufficiently larger than n. The two facilities of weight 1 are placed at a and b. Consider $n = 2k + 1$ and bring the first k demands to a and the next k to b,each with a weight of Δ. Clearly,an online algorithm that moves during these first $2k$ moves any facility or demand(in order to put all the weight in one of the two facilities) cannot achiece a finite competitive ratio,because the adverser can bring the last demand to a and pay nothing. Now, the adverser brings a demand of weight Δ^2 to S.The online algorithm either will move the demand to one of the facilities paying a cost of Δ^3 or move one facility to it,paying a cost of $\Delta^2 * k$. However,the optimal algorithm would move the facility from a to S at the beginning and serve all demands arriving in a to b paying a total cost of $\Delta + \Delta * k$.Hence the competitive ratio is $O(\Delta)$ which is unbounded. □

However,if we focus on the more realistic ocassion that all facilites and all demands arriving during the algorithm have weights that do not differ much for each other, e.g. there is a constant c such that $w_i \leq cw_j, \forall i, j$, where w_i, w_j are drawn from the set W that contains the weights of all facilities and demands,then we can achieve a much better competitive ratio. So we may concetrate in the case just mention.We also do one more observation to simplify the problem: We can treat all demands and facilities like having equal weight with losing $O(1)$ from the competitive ratio. We can view each demand apart from the facility it is attached to and its contribution to the total cost as its weight times the distance it travelled. Rounding to the greatest weight appeared we see that this contribution is multiplied by at most c. Hence we lose only $O(c) = O(1)$ of the optimal solution.

Here we prove that in metric spaces where the triangle inequality holds, in any optimal solution for a request sequence $\varrho(t)$ all the facilities would firstly move to a configuration and then serve all the demands without further moving the facilities. Consider a request sequence $\varrho(t)$ on a metric space $M(V, d)$ and an optimal solution where there exists some facility,let's say f, such that the above does not hold. Let u be the final position of f and v be the position where it was before it moved to u. Then at v this facility must have served some requests,let them be $r_1, r_2, .., r_l$. The total cost incurred by the movement of these requests is

$$\sum_{i=1}^{l}(d(v, u) + d(r_i, v)).$$

Consider the instance where f does not serve any requests to v, but it moves instantly at v. Then for each one of the requests mentioned above the cost incurred is

$$\sum_{i=1}^{l} d(r_i, u)$$

, whereas for all the other requests the cost does not change. By triangle inequality, the new solution costs at least as much as the previous solution.Repeating this argument for all facilities proves our claim.

5.3 Warm-up:The offline mobile facility location on the line

Let us consider the offline facility location on the line and let $\{r_1, .., r_n\}$ be the demand sequence in increasing order,.eg:$r_1 \leq r_2 \leq .. \leq r_n$. Consider two facilities l and k and let s_l, s_k be their starting positions and t_l, t_l be their final positions. It is easy to observe that if $s_l < s_k$ then $t_k < t_l$ and vice versa. This holds because if there exists an optimal solution that does not satisfy this property,we can change t_k to t_l and t_l to t_k and decrease the cost. Moreover, if S_i be the set of demands attached to facility i in the optimal solution, then i must move to the median of the elements in $S_i \cup \{s_i\}$. This holds because we want to minimize

$$|t_i - s_i| + \sum_{j \in S_i} |r_j - t_i|$$

and a simple exchanging argument shows that we must choose t_i as the median mentioned above. Moreover,if for some i, j with $i < j$ r_i and r_j are served at the same facility,then all intermideate demands ($r_k : i < k < j$) will also be served to the same facility. This can be proved by an exchanging argument. So the indexes of the demands attached to a facility form a sequence of consecutive indices. This lead to the following natural dynamic programming formulation: Denote $dp[i][j]$ be the optimal cost of serving the first i demands with the first j facilities(everything is assumed to be sorted in increasing order). Then :

- dp[0][j] = 0 ,
- dp[i][0] = $+\infty, i > 0$
- $dp[i][j] = min_{0 \leq k \leq i}\{dp[i-k][j-1] + opt(i-k+1, i, j)\}, i > 0, j > 0$

,where $opt(i-k+1, i, j)$ is the optimal cost to serve the demands $\{r_{i-k+1}, .., r_i\}$ to facility j. We do not need to compute this function each time,but we can compute $opt(i-k+1, i, j)$ from $opt(i-k+2, i, j)$ efficiently(notice that the median moves one position left and we can compute the new value in constant time). The total running time is $O(n^2k)$.

5.4 Deterministic algorithms on OMFL

Theorem 13. *No deterministic algorithm can obtain better than $\Omega(n)$ competitive ratio,when $k \geq 2$.*

Proof. Define the following metric on $k+1$ points: There are 3 points a, b, S and $k-2$ points u_i. a, b, S form a triangle with weights $d(a,S) = d(b,S) = n$ and $d(a,b) = 1$. Also , S is connected to all $k-2$ points u_i with an edge of weight n^2. Each facility initially lies on a, b, u_i.Then the adverser,brings the first $\frac{n}{4}$ demands on point a. If the online algorithm decides not to serve one of these demands with a,then it would incur at least 1 cost. So,the adverser bringing all the remaining points on a would force the competitive ratio to be infinity. Hence,all first $\frac{n}{4}$ demands are served to point a.Then the adverser brings the next $\frac{n}{4}$ demands to point b and by the same reasoning,they are served there. At last,he brings the last $\frac{n}{2}$ demands to point S. If the online algorithm moves no facility,it would pay a cost of $\frac{n}{2}n$. If it moves one of the u_i it would pay at least n^2. If it moves one of a and b, it would pay $\frac{n}{4}n$. In any ocassion,the online algorithm incurs a cost of $O(n^2)$. The offline algorithm would move facility from b to S and serve all demands arriving in b to its neighbour a,paying a total cost of $\frac{n}{4}+n$, which is $\Omega(n)$ times better than what the online algorithm pays. $\square$

Next,we prove that the "stay-where-you-are" algorithm,where the facilities don't move,but the requests attach to the nearest one is $O(n)$-competitive.

Theorem 14. *The "stay-where-you-are" algorithm is $O(n)$-competitive.*

Proof. Denote the initial facilites' locations by $f_1, f_2, .., f_{|C|}$. For the j-th facility let F_j be the set of requests that are attached to that facility in the optimal solution and f_j^* the final position where it moves. Then the optimal cost equals

$$OPT = \sum_{j=1}^{|C|}(\sum_{i\in F_j} d(ri, f_i^*) + d(f_j, f_j^*))$$

The algorithm's cost equals

$$\sum_{i=1}^{n} min_j(d(r_i, f_j)) \leq \sum_{j=1}^{|C|}\sum_{i\in F_j} d(r_i, f_j) \leq \sum_{j=1}^{|C|}\sum_{i\in F_j}(d(r_i, f_j^*) + d(f_j, f_j^*)) =$$

$$\sum_{j=1}^{|C|}(\sum_{i\in F_j} d(r_i, f_j^*) + |F_j| d(f_j, f_j^*)) \leq \sum_{j=1}^{|C|} |F_j|(\sum_{i\in F_j} d(r_i, f_j^*) + d(f_j, f_j^*)) \leq n * OPT$$

$\square$

Let’s look at the one-facility occasion where the bound mentioned above does not hold anymore.
For convenience of analysis of Theorem 3(mentioned below), we state the following lemma.
Lemma: In any metric space $M(\{r_1, .., r_{|M|}\}, d())$ and any multiset C containing nodes of the metric space,we can assume that the node that minimizes the sum of distances from all nodes of C is contained in C,losing a factor of 2.

Proof. Let u any node not in C. Then,

$$2(|C|-1)\sum_{i\in C} d(u, r_i) = \sum_{i\in C}\sum_{j\in C-\{i\}} (d(r_i, u)+d(u, r_j)) \geq \sum_{i\in C}\sum_{j\in C} d(r_i, r_j) \geq |C|\sum_{j\in C} d(r^*, r_j)$$

,where r^* is the node of C with the minimum total distance from the others. □

Note: The lemma above is essentialy tight. Consider a metric with n nodes,$n-1$ of which have distance 1 from each other and the other $\frac{1}{2}$ from all the others.
The above lemma will be used in the next theorem when bounding the costs without explicitly mentioning it.

Theorem 15. *:There is a simple $O(\sqrt{n})$-competitive algorithm for the OMFL with one facility.*

Proof. : The online algorithm acts as follows: For the first $\sqrt{n}$ demands the facility stays at its initial position and serves them there. Then,the algorithm moves to the optimal position for these demands and serves each other demand there. Define OPT be the cost of the optimal algorithm and OPT_0 the optimal position to move the facility if we restrict to the first $\sqrt{n}$ demands. We can assume that OPT_0 is one of the first $\sqrt{n}$ demands,losing a factor of 2. Let also u be the optimal position for the facility to move and define r_0 the facility’s initial position.
The cost of the online algorithm is

$$\sum_{i=1}^{\sqrt{n}} d(r_i, r_0) + (\sqrt{n}+1)d(r_0, OPT_0) + \sum_{i=\sqrt{n}+1}^{n} d(r_i, OPT_0)$$

As in the ”stay-where-you-are” algorithm it easy to see that

$$\sum_{i=1}^{\sqrt{n}} d(r_i, r_0) + (\sqrt{n}+1)d(r_0, OPT_0) \leq OPT + (2\sqrt{n}+1)d(r_0, OPT_0) \leq (2\sqrt{n}+2)OPT$$

To bound the second term observe that :

$$\sum_{i=\sqrt{n}+1}^{n} d(r_i, OPT_0) \leq \sum_{i=\sqrt{n}+1}^{n} (d(r_i, u) + d(u, OPT_0)) \leq$$

$$\sum_{i=\sqrt{n}+1}^{n} d(r_i, u) + (n - \sqrt{n})d(u, OPT_0) \leq$$

$$OPT + (n - \sqrt{n})d(u, OPT_0)$$

In order to establish the competitive ratio it suffices to prove that

$$d(u, OPT_0) \leq \frac{2OPT}{\sqrt{n}}$$

. To do so observe that

$$d(u, OPT_0) \leq \frac{1}{\sqrt{n}} \sum_{i=1}^{\sqrt{n}} (d(u, r_i) + d(r_i, OPT_0)) =$$

$$\frac{1}{\sqrt{n}} (\sum_{i=1}^{\sqrt{n}} (d(u, r_i) + \sum_{i=1}^{\sqrt{n}} d(r_i, OPT_0)) \leq$$

$$\frac{OPT}{\sqrt{n}} + \frac{OPT}{\sqrt{n}} = \frac{2OPT}{\sqrt{n}}$$

$\square$

Consider now the algorithm when the 2^i-th demand arrives moves the facility to the optimal position at that time. Call it COFL.

Theorem 16. *The COFL is $O(logn)$-competitive for the OMFL with one facility.*

Proof. : Let c_l be the optimal position to move the facility when the 2^l-th demand arrives.We say that the l-th phase starts when the $2^l + 1$-th demand arrives and ends when the facility moves from c_l to c_{l+1}. Also define $c_0 = r_0$ to be the facility's initial position.The algorithm starts at phase 0.The cost incurred during phase l equals

$$2^{l+1}d(c_l, c_{l+1}) + \sum_{i=2^l+1}^{2^{l+1}} d(r_i, c_l) \leq 2^{l+1}d(c_l, c_{l+1}) + \sum_{i=2^l+1}^{2^{l+1}} (d(r_i, c_{l+1}) + d(c_l, c_{l+1}))$$

$$\leq (2^l + 2^{l+1})d(c_l, c_{l+1}) + \sum_{i=2^l+1}^{2^{l+1}} d(r_i, c_{l+1})$$

Observe that

$$d(c_l, c_{l+1}) \leq \frac{1}{2^l}(\sum_{i=1}^{2^l} d(r_i, c_l) + \sum_{i=1}^{2^l} d(r_i, c_{l+1}))$$

$$\leq \frac{1}{2^l}(OPT_{2^l} + OPT_{2^{l+1}}) \leq \frac{2}{2^l}OPT$$

Moreover

$$\sum_{i=2^l+1}^{2^{l+1}} d(r_i, c_{l+1}) \leq \sum_{i=0}^{2^{l+1}} d(r_i, c_{l+1}) \leq OPT$$

Combining the above and summing over all $l = O(logn)$ stages,we get the desired result. □

5.5 OMFL in special metric spaces

Because OMFL in a general metric space seems inatractable,we try some special cases to obtain better bounds. The next two theorems imply that COFL in the metric spaces described achieves a constant competitive ratio,however we give another more intuitive analysis,discriminating them from the COFL algorithm.

Theorem 17. *There is an algorithm with constant competitive ratio for the OMFL with one facility in a uniform metric space.*

Algorithm for Theorem 15: Whenever the 2^i-th demand arrives,check if there exists a point of the metric space where more demands have arrived than in the current position of the facility. If so,move there and serve the demand there. In any other occasion,stay where you are and serve the demand there.

Proof. : We can suppose that the optimal algorithm incurs some cost, otherwise also the online will incur zero cost. So, let u the node where the facility moves in the optimal solution and observe that u must be the node where the most demands have arrived to. Let OPT be the optimal cost and observe that either OPT or $OPT-1$ demands have been served in u. If there is no integer j such that after the 2^j-th demand, the facility stays in u forever,the optimal would pay at least $\frac{n}{2}$ and the online algorithm $O(n)$. Else,suppose that $j \geq 2$ otherwise the statement is trivial.The cost of the online algorithm for the demands not arriving in u after the 2^j-th demand is what the optimal also pays for these demands. The algorithm pays a cost of $O(2^j)$

for the first 2^j demands.Observe that the optimal algorithm must pay at least 2^{j-2}, because there are at least 2^{j-2} demands not arriving at u through the first 2^{j-1} demands,otherwise u would be the node with the maximum number of demands at 2^{j-1} and so the online algorithm would have moved the facility to u by $j-1$,contradiction. □

Theorem 18. *There is an algorithm with constant competitive ratio for the OMFL with one facility in a weighted star.*

We assume that the facility starts from the root of the star and the demands arrive only at leaves of the star.These assumptions can be removed with standard tricks.Denote by n_i the number of demands at the i-th leaf and w_i the weight of the edge connecting it with the root. Observe now that if the optimal offline algorithm leaves the facility at the root, it will pay

$$\sum_j w_j r_j$$

, while if it moves it to leaf i it will pay

$$w_i + \sum_{j \neq i} (w_i + w_j) n_j = (n + 1 - 2n_i) w_i + \sum_j w_j n_j$$

So it is better to move to a leaf i if and only if there are more than $\frac{n+1}{2}$ demands that have arrived there, and there can be only one such point.The following algorithm tries to imitate the above behaviour:

Algorithm for Theorem 6: Whenever the 2^i-th demand arrives,check if there exists a leaf with more than half of the demands arrived so far to have arrived there.If yes then move to that leaf,else move to the root.

Proof. : We may assume for ease of analysis that n is a power of 2. At first,suppose that in the optimal solution the facility moves to some leaf ,say m. Then,let 2^j be the last time when the online algorithm moves the facility to m. Observe for the demands arriving after that time, the algorithm pays what the optimal pays. For the previous demands we focus on a leaf i different from m and the weight incurred through that edge. If the facility leaves this leaf at some time t then it pays $2^t w_i$ for movement through this edge at that time. It is easy to see that if 2^{l_i} is the last time where the facility departs from i,the online algorithm pays $O(2^{l_i} w_i)$ for the demands passing through that edge till then and the optimal must pay at least $2^{l_i - 1} w_i$ for the demands till then,because there are at least $2^{l_i - 1}$ demands arriving to that point till then. After this step,the optimal pays the same as the online algorithm for the demands passing through that edge. So, summing over all leaves i different from m we get that the online algorithm pays constant times what the optimal pays for each edge connecting a leaf with the root. If the best is to stay at the root, observe that if 2^l is the last time that the facility is at leaf i the online algorithm pays $O(2^l w_i)$ for cost till then passing through that edge and it must also hold $n_i \geq 2^{l-2}$. For all subsequent demands or demands arriving in leaves that the facility never moves to , the algorithm pays what the optimal does for crossing the edge connecting the respective leaf to the root. □

5.6 Another variant of the online mobile facility location problem

Consider the previous scenario,but from each facility's initial position,we can draw as much facilities as we want(like a hole that gives birth to facilities at zero cost). In particular,imagine that these points are like factories from which we can take a facility at cost 0 and move it where we want to. We note that there is an $O(\frac{logn}{loglogn})$ algorithm for this variant of OMFL, by doing a reduction to non-uniform facility location. Like in the classical OMFL model,it is easy to observe that in any optimal solution each facility created moves to a certain point and serves all demands there. For the reduction,denote by F_{init} the set of facilities' initial positions and associate each point u in the metric space with a value $f(u)$ such that $f(u) = min_{v \in F_{init}} d(u, v)$. Observe that any optimal solution to this variant of online mobile facility location can be transformed to a feasible solution of non-uniform facility location with equal cost. Clearly,moving a facility from a point $u \in F_{init}$ to a point p is like opening a facility in point u with cost $d(p, u)$, but since in no optimal solution there exists a facility that moves from p to v if $d(p, u) > min_{v \in F_{init}} d(u, v) = f(u)$, we can asssume that the opening cost is exactly $f(u)$. In a similar fashion each instance of the non-uniform facility location with the costs described above can be mapped to a solution of this variant of online mobile facility location:When the algorithm opens a facility at v we move a facility to v from $u = argmax_{i \in F_{init}} d(i, v)$.Each demand will now be served to the closest facility and the reduction is complete.

Notice that this solves the even more general problem where each facility must move within a distance D (the above problem is a special case where D equals the diameter of the metric space, or even simpler $D = +\infty$). We can set $f(u) = min_{\{v \in F_{init}: d(u,v) \leq D\}} d(u, v)$ and obtain again the same competitive ratio as before. Clearly,the above reduction still holds even if D is not the same for all facilities in F_{init} but may vary among them. [**?**]

Bibliography

[1] Mark S. Manasse and Lyle A. McGeoch and Daniel Dominic Sleator, *Competitive Algorithms for On-line Problems*. STOC, 1988.

[2] Allan Borodin and Nathan Linial and Michael E. Saks, *An Optimal Online Algorithm for Metrical Task Systems*. STOC, 1987.

[3] Marek Chrobak and Lawrence L. Larmore. *An optimal online algorithm for k servers on trees*. SIAM J. Comput., 20:144–148, 1991.

[4] Elias Koutsoupias and Christos H. Papadimitriou, *On the k-Server Conjecture*, J. ACM, 1995.

[5] Prabhakar Raghavan and Marc Snir, Memory versus randomization in on-line algorithms, *IBM Journal of Research and Development*. 1994.

[6] Aaron Cote and Adam Meyerson and Laura J. Poplawski, *Randomized k-server on hierarchical binary trees*. STOC,2008.

[7] Nikhil Bansal and Niv Buchbinder and Aleksander Madry and Joseph Naor, *A Polylogarithmic-Competitive Algorithm for the k-Server Problem*. FOCS, 2011,

[8] Nikhil Bansal and Niv Buchbinder and Joseph Naor, *A Primal-Dual Randomized Algorithm for Weighted Paging*. FOCS, 2007.

[9] Adam Meyerson, *The Parking Permit Problem*. FOCS, 2005.

[10] Barbara M. Anthony and Anupam Gupta, *Infrastructure Leasing Problems*. IPCO, 2007.

[11] Satu Elisa Schaeffer, *Graph clustering*. Computer Science Review, 2007.

[12] Moses Charikar and Rina Panigrahy, *Clustering to minimize the sum of cluster diameters*. J. Comput. Syst. Sci., 2004

[13] Jnos Csirik and Leah Epstein and Csand Imreh and Asaf Levin, *Online Clustering with Variable Sized Clusters*. MFCS, 2010.

[14] Noga Alon and Baruch Awerbuch and Yossi Azar and Niv Buchbinder and Joseph Naor, *A general approach to online network optimization problems*. ACM Transactions on Algorithms, 2006

[15] Michel X. Goemans and David P. Williamson, *A General Approximation Technique for Constrained Forest Problems*. SIAM J. Comput., 1995.

[16] Marek Chrobak and Claire Kenyon and John Noga and Neal E. Young, *Incremental Medians via Online Bidding*, Algorithmica,2008

[17] Ramgopal R. Mettu and C. Greg Plaxton, *The Online Median Problem*. IEEE, New York 2000.

[18] Mettu, R.R., Greg Plaxton, C., *The online median problem*. SIAM J. Comput. 32, 2003.

[19] Guolong Lin, Chandrashekhar Nagarajan, Rajmohan Rajaraman, David P. Williamson: *A General Approach for Incremental Approximation and Hierarchical Clustering*. SIAM J. Comput. ,2010.

[20] Zachary Friggstad, Mohammad R. Salavatipour: *Minimizing Movement in Mobile Facility Location Problems*. FOCS 2008.

[21] Marshall L. Fisher and Dorit S. Hochbaum, *Database Location in Computer Networks*. J. ACM, 1980.

[22] Jehan-Francois Paris and Walter F. Tichy, *Stork: An Experimental Migrating File System for Computer Networks*. INFOCOM, 1983

[23] *File Migration and Process Migration in a Local Area Network*. In Proc. of INFOCOM86, pages 488-495,1986.

[24] W.Tichy and R.Zuwang. Towards a Distributed File System.In PRoc.of the 1984 USENIX Summer Conf.pages 87-97,1984.

[25] O.L. Sheng. *Models for Dynamic File Migration in Multiprocessors with Distributed computer systems*. Ph.D. dissertation.

[26] C.Sheurich and M.Dubois. *Dynamic Page Migration in Multiprocessors with Distributed Global MEmory*. In IEEE TRansactions on Computers,38(8):1154-1163,1989.

[27] B.Gavish and O.R.L.,Sheng. *Dynamic File Migration in Distributed Computer Systems*. In Communications of the ACM, 33(2):177-189:1990.

[28] D.Sleator and R.E. Tarjan. *Amortized Efficiency of List Update and Paging Rules*. Communications of ACM, 28(2):202-208,1985.

[29] J.Westbook. *Randomized Algorithms for Multiprocess or Page Migration*. In Proc. of DIMACS Workshop on On-Line Algorithms. American Mathematical Society,February, 1991.

[30] Y.Bartal,A.Fiat and Y.Rabani. *Competitive algorithms for distributed data management*. In Proc.of the 24th ACM Symp. on Theory of Compuring, pages 39-50,1992.

[31] M.Chrobak,L.Lamore,N.Reingold and J.Westbrook. *Optimal Multiprocessor Migration Algorithms Using Work Functions*. In PRoc. of the 4th International Symp.On Algorithms and OCmputation. Also Lecture Notes in Computer Science,vol.762,pages 406-415,Hong Kong,1993,Springer-Verlag.

[32] B.Awerbuch,Y.Azar nad Y.Bartal. *On-line Generalized Steiner Problem*. In Proc. of the 7th Ann.ACM-SIAM Symp. on Discrete Algorithms,pages 68-74,January 1996.

[33] B.Awerbuch,Y.Bartal and A.Fiat, *Heat and Dump:Randomized competitive distributed paging*.In Proc.34rd IEEE symp.on Foundations of Computer Science. IEEE,pages 574-583,January 1996.

[34] B.Awerbuch,Y.Bartal and A.Fiat. *Distributed Paging for General Networks*. In Proc.of the 7th Ann.ACM-SIAM Symp.on Discrete Algorithms,pages 574-583,January 1996.

[35] S.Albers and H.Koga. *Page Migration with Limited Local Memory Capacity*. In Proc. of the 4th Workshop on Algorithms and Data Structures, August 1995.

[36] Y.Bartal. *Probabilistic Approximation of Metric Spaces and its Algorithmic Applications*. Proc. of the 37rd Ann.IEEE Symp.on Foundations of COmputer Science,1996.

[37] Calinescu, G., Karloff, H., Rabani, Y.: Approximation algorithms for the 0-extension problem. In: SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 8–16. (2001)

[38] Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approx- imating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. 69, 485–497 (2004)

[39] Gupta, A., Hajiaghayi, M.T., Räcke, H.: Oblivious network de- sign. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 970–979. ACM Press, New York (2006)

[40] Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 161–168. ACM Press, New York (1998).

[41] Alon, N., Karp, R.M., Peleg, D., West, D.: A graph-theoretic game and its application to the k-server problem. SIAM J. Comput. 24, 78–100 (1995)

[42] Gupta, A., Talwar, K.: Approximating unique games. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM sym- posium on Discrete algorithm, New York, NY, USA, pp. 99–106. ACM Press, New York (2006)

[43] Hayrapetyan, A., Swamy, C., Tardos, É.: Network design for information networks. In: SODA '05: Proceedings of the six- teenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Math- ematics, pp. 933–942. (2005)

[44] F IAT, A., K ARP, R. M., L UBY, M., M C G EOCH , L. A., S LEATOR , D. D., AND Y OUNG, N. E. 1991. *Competitive paging algorithms*. J. Algorithms 1, 4, 685–699

[45] I RANI , S. 1997. *Page replacement with multi-size pages and applications to web caching*. In Proceedings of the 29th Annual ACM Symposium on Theory of Computing. 701–710

[46] I RANI , S. 2002. *Randomized weighted caching with two page weights*. Algorithmica 32, 4, 624–640.

[47] Y OUNG, N. 1991. *On-line caching as cache size varies*. In SODA'91: Proceedings of the 23rd Annual ACM- SIAM Symposium on Discrete Adlgorithms. 241–250.

[48] A DAMASZEK , A., C ZUMAJ, A., E NGLERT, M., AND R ACKE , H. 2012. *An $O(logk)$-competitive algorithm for generalized caching*. In SODA. 1681–1689.

[49] B LUM , A., F URST, M., AND T OMKINS, A. 1996. *What to do with your free time: Algorithms for infrequent requests and randomized weighted caching*. Manuscript. (www.cs.cmu.edu)

[50] B ARTAL , Y., B LUM , A., B URCH , C., AND T OMKINS, A. 1997. *A polylog(n)-competitive algorithm for metrical task systems*. In Proceedings of the 29th Annual ACM Symposium on Theory of Computing. 711–719.

[51] Allan Borodin, Nathan Linial, and Michael Saks. *An optimal online algorithm for metrical task systems*. In Proc. 19th Symp. Theory of Computing (STOC), pages 373–382. ACM, 1987.

[52] Allan Borodin, Nathan Linial, and Michael Saks. *An optimal online algorithm for metrical task system*. J. ACM, 39:745–763, 1992.