The Competitiveness of On-Line Assignments *

Yossi Azar †
Computer Science Department
Tel-Aviv University
Tel-Aviv 69978, Israel

Joseph (Seffi) Naor ‡
Computer Science Department
Technion
Haifa 32000, Israel

Raphael Rom
Sun Microsystems
Mountain View
CA 94043-1100

Abstract

Consider the on-line problem where a number of servers are ready to provide service to a set of customers. Each customer's job can be handled by any of a *subset* of the servers. Customers arrive one-by-one and the problem is to assign each customer to an appropriate server in a manner that will balance the load on the servers. This problem can be modeled in a natural way by a bipartite graph where the vertices of one side (customers) appear one at a time and the vertices of the other side (servers) are known in advance. We derive tight bounds on the competitive ratio in both deterministic and randomized cases. Let n denote the number of servers. In the deterministic case we provide an on-line algorithm that achieves a competitive ratio of $k = \lceil \log_2 n \rceil$ (up to an additive 1) and prove that this is the best competitive ratio that can be achieved by any deterministic on-line algorithm. In a similar way we prove that the competitive ratio for the randomized case is $k' = \ln(n)$ (up to an additive 1). We conclude that for this problem, randomized algorithms differ from deterministic ones by precisely a constant factor.

---

*A preliminary version of this paper appeared in Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, Orlando, Florida, (1992), pp. 203-210.

†A portion of this work was done while the author was in the department of Computer Science, Stanford University, CA, and supported by a Weizmann fellowship and contract ONR N00014-88-K-0166.

‡Part of this work was done while the author was a post-doctoral fellow at the Computer Science Department, Stanford University, Stanford, CA, and supported by contract ONR N00014-88-K-0166.

1 Introduction

Consider a wireless communication network, similar to that of the cellular phone system, where customers arrive (i.e., turn on their phone) one-by-one in an arbitrary order. Upon arrival, each customer discloses the extent of service it requires and must be assigned a base-station, of those within range, to service it. Improper station assignment may cause overloading of some stations (or equivalently, entail penalty of hand-off). Thus, it is desirable to spread the load as evenly as possible.

The above is actually an example of the following general problem. Consider a number of servers ready to provide service to a set of customers. Each customer's job can be handled by any of a *subset* of the servers. Customers arrive one by one, each with a different amount of required service, and the problem is to assign each customer to an appropriate server in a manner that will even out the load balance on the servers. This is a typical on-line problem in which an algorithm is sought that must make decisions based upon current state without knowing future events.

The question of evaluating the performance of an on-line algorithm was addressed by Sleator and Tarjan [13] who argued that the traditional approach of measuring the worst case behavior does not seem appropriate for many on-line algorithms. Therefore, they suggested a different measure, the *competitive ratio*. The performance of an on-line algorithm is compared with the performance of an optimal off-line algorithm that knows the sequence of events in advance. The maximum ratio between their respective performances, taken over all sequences, is called the competitive ratio.

Graham [6] considered the above assignment/scheduling problem for the case where each job can be handled by all the servers. He proved that the greedy algorithm achieves a competitive ratio of $2 - 1/n$. On-line scheduling problems were considered in [12] and their references. On-line matching problems in graphs were considered by Karp, Vazirani and Vazirani [9], and Kalyanasundaran and Pruhs [8]. Other on-line graph problems, in particular, on-line vertex coloring, are discussed in [10, 14, 7]. The above are examples of the extensive work that was done in recent years for finding the competitive ratio for different problems which also include paging [5], servers in a metric space [11], and managing a linked list [13]. However, there are only few problems for which the competitive ratio is known in precise.

To further improve the competitiveness of on-line algorithms, extensive work was devoted to exploiting and understanding the power of randomization in such environments [4]. The idea is that randomized on-line algorithms might exhibit a better competitive ratio than deterministic ones, since they manage to "outsmart" the adversary sometimes. The common adversary for randomized algorithms is the oblivious one: one who must construct the request sequence in advance based on the description of the algorithm, but without knowing the random choices made by the algorithm. It should be noted that an exponential gap between the competitive ratio of deterministic on-line algorithms to that of randomized ones might exist. In the paging problem, if k denotes the number of pages in the cache, then k is the competitive ratio of the best deterministic algorithm; in contrast, it is known [5] that randomized algorithms can achieve a competitive ratio of $\log k$.

In this paper we explore the problem of on-line weighted assignment in bipartite graphs for both deterministic and randomized environments and derive exact competitive ratios for either case. Let m and n denote the number of customers and servers respectively. In the deterministic case we do so by providing an on-line algorithm that achieves a competitive ratio of $\lceil \log_2(n) \rceil + 1$ and by proving that the best competitive ratio that can be achieved by any deterministic on-line algorithm is $\lceil \log(n+1) \rceil$. In addition, for randomized algorithms, we prove that the exact competitive ratio in the unweighted case is $\ln n$ (up to an additive 1). We have thus determined the precise performance (including constant factors) in both the deterministic and the randomized cases, and conclude that for the assignment problem, randomized algorithms differ from deterministic ones by precisely a constant factor. It should be noted that the proof of the upper bound in the randomized case holds only for an *oblivious* adversary. For an *adaptive* adversary, we show that randomization does not improve the competitive ratio. (See [4] for a discussion on the different types of adversaries in the randomized case).

A related set of problems is that of the split-weight assignments in which the weight of a vertex can be split and assigned to several of its neighbors. We show that the competitive ratio for this case is $\Theta(\min\{m, \log n\})$.

As a result of our work some related problems were considered. We refer to the general load balancing problem as the case where assigning a customer to a possible server increases the load on that server by an arbitrary value which depends both on the customer and the server. This problem was considered in [1] who presented an $O(\log n)$ competitive algorithm. Our $\log n$ lower bound implies that their algorithm is optimal (up to a constant factor). Another interesting model for the load-balancing assignment problems is the model that customers are not necessarily permanent and hence may arrive and depart at arbitrary times. It is shown in [2, 3] that there is an algorithm with competitive ratio $\Theta(\sqrt{n})$ and that no algorithm can do better.

2 The Problem Statement

Let $G = (U, V, E)$ be a bipartite graph having $|U| = m$ vertices on one side ("the customers") and $|V| = n$ vertices on the other side ("the servers") connected by the set of edges E. We define the *one-sided assignment* (or assignment, in short) as the matching of each and every vertex $u \in U$ with a vertex $v \in V$. In general, there may not be a perfect matching in G, in which case some of the vertices in V will be assigned more than one vertex from U. We reserve the term *matching* to the case in which no server is assigned to more than one customer. Every vertex $u_i \in U$ has a weight w_i associated with it. The sum of the weights of the vertices assigned to a vertex $v \in V$ is referred to as the *load* of vertex v. The goal is to come up with an assignment that minimizes the maximum load on any vertex of V. Clearly, if a perfect matching exists, the maximum load equals the maximum weight. We will assume without loss of generality that $n \leq m$, since all our bounds can be stated as depending on the minimum between $|U|$ and $|V|$ (except for Section 5).

It is not hard to prove that computing an optimal solution in the off-line case is NP-complete for arbitrary weights. (This is done by reduction to the knapsack problem). However, if the weights are all equal, then an optimal solution can be computed in polynomial time by reduction to maximum flow.

In the on-line version, the vertices of U appear either one-by-one, or in groups, in some arbitrary order. The appearance of a vertex includes the disclosure of its identity, its weight, and all the edges incident to it. The on-line algorithm must assign a vertex from V to each vertex of U upon its appearance. We refer to an algorithm's *step* as the appearance of a vertex and its assignment to one of its neighbors (we shall often refer to the beginning of the step—before any appearance, and the end of the step—after the assignment of that step is done). In computing the performance of on-line algorithms, we are interested in the competitive ratio, i.e., the maximum ratio over all possible inputs between the performance of the on-line algorithm and the off-line one. In the deterministic case, the adversary constructs the graph and assigns the weights in advance, but since it knows the behavior of the algorithm for any sequence of requests, it can construct the worst possible sequence.

In the randomized environment we first assume an oblivious adversary, i.e., an adversary that is familiar with the details of the algorithm, but cannot predict the outcome of the coins flipped during the actual run. As in the deterministic case, the adversary has to decide upon the graph and weights in advance but this time without knowing the actual behavior of the algorithm. In Section 4.3 we also consider the case of an adaptive adversary.

3 The Deterministic Upper and Lower Bounds

3.1 The Upper Bound

Let the vertices $u \in U$ of the bipartite graph $G(U, V, E)$ appear one at a time. We assign the appearing vertices of U according to the following algorithm.

Algorithm AW: Upon arrival of a vertex $u \in U$ assign it to a neighbor with the current minimum load (ties are broken arbitrarily).

Theorem 3.1 *Algorithm AW achieves a competitive ratio of* $\lceil \log n \rceil + 1$

Proof: Define $W = \sum_{k=1}^{m} w_k$ and denote by ℓ the maximum load of a vertex in V in an optimal solution, denoted by OPT. Clearly, $\ell \geq W/n$.

To facilitate the proof, we adopt the following view of the assignment process. Let M denote the assignment generated by Algorithm AW. We partition M into successive layers so that in layer i there is a subassignment M_i such that the total load on a vertex in each subassignment is maximal but does not exceed ℓ. More formally, for every vertex $v \in V$, let $u_{j_1}, \ldots, u_{j_k}$ be the vertices assigned to it in M, sorted in the order that Algorithm AW assigned them. These vertices are partitioned into the layers as follows. Assignment M_1 contains all the vertices $u_{j_1}, \ldots, u_{j_p}$ where either, p is the smallest index such that $\sum_{i=1}^{p} w_{j_i} \geq \ell$, or $p = k$. The weight of vertex u_{j_p} is adjusted to be

$$w'_{j_p} = \sum_{i=1}^{p} w_{j_i} - \ell.$$

Assignments $M_2, M_3, \ldots$ are defined recursively on the sequence $u_{j_p}, \ldots, u_{j_k}$ along with their adjusted weights (nodes with a 0 adjusted weight are excluded). Thus every layer contains the vertices V and a subset of U as described above along with their adjusted weights. Notice that any vertex from U may belong to at most two adjacent layers and when this happens, its weight is partitioned between the two layers.

Let W_i denote the sum of the weights of the vertices in M_i, and let R_i denote the residual weight of layer i, i.e., the sum of the weights of the vertices that have not completed their assignment in layers j, for $j \leq i$ (in other words, this is the sum of the adjusted weights in layers greater than i). By definition, $R_0 = W$. Clearly, $W_i = R_{i-1} - R_i$. We prove the theorem by first showing that R_i decreases exponentially with i which we do via the next two lemmas.

The following notation and definitions are required.

- W_{ij} - The load of vertex v_j in layer i. Note that $\sum_i W_{ij}$ is the total load on v_j and $W_i = \sum_{j=1}^{n} W_{ij}$.
- z_{ik} - The (partial) weight of vertex $u_k \in U$ assigned in layer i. Note that for every k, there are at most two (adjacent) layers in which $z_{ik} > 0$.
- O_j - The set of vertices in OPT adjacent to vertex v_j.
- O_{ij} - The subset of O_j which has not finished its assignment in layer i, that is, the subset of O_j that appears in layers greater than i.
- R_{ij} - The sum of the (adjusted) weights in layer i of the vertices in O_{ij}. Note that since R_{ij} is a sum of weights of a subset of O_j, it clearly follows that $R_{ij} \leq \ell$.

Lemma 3.1 *For every i and for each vertex $v_j \in V$, $W_{ij} \geq R_{ij}$.*

If $O_{ij} = \phi$, the Lemma follows immediately, since $R_{ij} = 0$. The Lemma also follows immediately if $W_{ij} = \ell$ since $R_{ij} \leq \ell$. Therefore, the only remaining case is $W_{ij} < \ell$ and a nonempty O_{ij}.

$W_{ij} < \ell$ means that in the i^{th} layer, v_j is not completely loaded, implying that in all subsequent layers the load on v_j is 0. Since the set O_{ij} is nonempty, we are lead to the conclusion that all the vertices in O_{ij} were assigned in M to vertices different from v_j. Let v_r be such a vertex, i.e., $u_k \in O_{ij}$ is assigned in M to v_r. Since AW is a greedy algorithm, the load of v_r could not have been greater than the load of v_j at the time of the assignment. The next two observations are crucial:

- $W_{ir} = \ell$; otherwise, vertex u_k would have completed its assignment in layer i contradicting its belonging to O_{ij}.
- $W_{i-1,j} = \ell$; $W_{i-1,j} < \ell$ implies that $W_{ij} = 0$, which in turn implies that the load on v_r is greater than that on v_i (since $W_{ir} = \ell$), which is a contradiction.

Hence, for every k such that $u_k \in O_{ij}$

$$(i-1) \cdot \ell + W_{ij} \geq (i-1) \cdot \ell + (\ell - z_{ik})$$

where the above LHS denotes the total load on vertex v_j up to and including layer i, and the RHS is the load of v_r when u_k was assigned to it. This implies that,

$$\ell - z_{ik} \leq W_{ij}$$

Noting that $\sum_{k;u_k \in O_{ij}} w_k \leq \ell$ (since these are OPT assignments), we get from the above and by the definition of z_{ik},

$$R_{ij} = \sum_{k;u_k \in O_{ij}} (w_k - z_{ik}) = \sum w_k - \sum z_{ik} \leq \ell - \sum z_{ik} \leq W_{ij}$$

which completes the lemma. $\square$

Lemma 3.2 *For all i, $R_i \leq \frac{1}{2} \cdot R_{i-1}$.*

We observe that for fixed i and $1 \leq j \leq m$, the sets O_{ij} are mutually disjoint. Hence, $\sum_{j=1}^{m} R_{ij} = R_i$. Thus, with the aid of Lemma 3.1

$$W_i = \sum_{j=1}^{n} W_{ij} \geq \sum_{j=1}^{n} R_{ij} = R_i \ .$$

The latter inequality implies that $R_i = R_{i-1} - W_i \leq R_{i-1} - R_i$. Hence,

$$R_i \leq \frac{1}{2} \cdot R_{i-1}$$

□

We are now ready to complete the proof of the Theorem. Choose $b = \lceil \log n \rceil$. Clearly, by applying Lemma 3.2 b times we obtain that,

$$R_b \leq \frac{1}{n} \cdot R_0 = \frac{W}{n} \leq \ell$$

Hence, the load on a vertex in V can be at most $\ell \cdot b + R_b. \leq \ell \cdot (b+1)$. Since OPT achieves a load of ℓ, a competitive ratio of $\lceil \log n \rceil + 1$ is established. □

3.2 The Lower Bound

Theorem 3.2 *The competitive ratio of any on-line bipartite assignment algorithm is at least* $k = \lceil \log_2(n+1) \rceil$.

Proof: For simplicity and without loss of generality we assume that m is a power of 2, i.e., $m = 2^{k-1}$. We also assume that all vertices have an equal weight of unity and that $m = n$. A bipartite graph $G = (U, V, E)$ having the following properties is constructed on-line by the adversary based on the decisions made by the on-line algorithm:

1. The vertices in U will be given to the on-line algorithm in steps. In Step i, $1 \leq i \leq k$, the set S_i appears; for $1 \leq i \leq k-1$, $|S_i| = m/2^i$ and $|S_k| = 1$. Hence, $U = \cup_{i=1}^{k} S_i$.
2. For $1 \leq i \leq k$, all the vertices in S_i have the same neighborhood set, denoted by Y_i
3. $Y_{i+1} \subset Y_i$ and $|Y_i| = m/2^{i-1}$.
4. At the beginning of Step i, before the vertices in S_i are assigned, the average load of the vertices in Y_i is at least $i-1$.

If a graph G with the above properties can be constructed, then at the end of Step $k-1$, since $m = n$, there is a vertex in Y_k whose load is at least $k-1$. Since $|Y_k| = 1$ and $|S_k| = 1$, at the end of Step k, the load of that vertex will be increased to k. Such a graph G does have a perfect matching since all the vertices in S_i $(1 \leq i \leq k)$ can be perfectly matched to all vertices in $Y_i - Y_{i+1}$ (define $Y_{k+1} = \phi$). Hence, the competitive ratio of k is maintained. Note that having the input arrive in sets is acceptable since an algorithm that receives its input in sets, as described above, can do at least as well as an algorithm that receives the input one vertex at a time.

To construct G, we need to describe how the set Y_i is chosen so that the above properties are maintained. The set Y_1 is defined to be V. To obtain Y_{i+1}, choose, at the end of Step i, the $n/2^i$ vertices in Y_i having the highest load. Properties 1 and 2 can be maintained independently of Y_i, and property 3 clearly holds. We prove property 4 by induction on the steps of the algorithm. It obviously holds for $i = 1$. Assume inductively that it holds for i, i.e., at the beginning of step i the average load on a vertex in Y_i is at least $i-1$. Hence, the sum of the loads on the vertices in Y_i is at least $(i-1)|Y_i|$. In Step i, all the vertices in S_i are assigned to vertices from Y_i, so that at the end of Step i, the sum of the loads of the vertices of Y_i is at least

$$(i-1)|Y_i| + |S_i| = (i - \frac{1}{2})|Y_i| \; .$$

Recall that Y_{i+1} is defined to be the set of $|Y_i|/2$ vertices of Y_i having the largest load. If the largest load of a vertex of Y_i that was not chosen for Y_{i+1} is at least i, then the load of each vertex in Y_{i+1} will also be at least i and obviously their average as well. Otherwise, the sum of the loads of the vertices in $Y_i - Y_{i+1}$ is at most

$$|Y_i - Y_{i+1}|(i-1) = |Y_{i+1}|(i-1) \; .$$

Thus, the sum of the loads of the vertices in Y_{i+1} is at least

$$(i - \frac{1}{2})|Y_i| - (i-1)|Y_{i+1}| = i|Y_{i+1}|$$

and hence, the average load on vertices in Y_{i+1} is at least i. □

4 The Randomized Upper And Lower Bounds

4.1 The Upper Bound

We restrict ourselves in this section to the case where all vertices have a weight of unity. Our derivation of the upper bound is based on the *ranking algorithm* in [9] for on-line matching in bipartite graphs which proceeds as follows. Choose a random permutation on the vertices of V which induces an unambiguous priority order on the vertices in V. Upon arrival, match vertex $u \in U$ with the eligible unmatched vertex in V of highest priority. If all eligible vertices, i.e., all neighbors of u, are matched, then u is left unmatched. The following is proven in [9]:

Theorem 4.1 (KVV) *Let μ denote the cardinality of the maximum matching in the bipartite graph G with $2n$ vertices. The ranking algorithm finds a matching of size at least $\alpha \cdot \mu$, where*

$$\alpha \geq 1 - \left(1 - \frac{1}{n}\right)^n > (1 - \frac{1}{e})$$

It is interesting to note that [9] show that an on-line algorithm that matches a vertex to a neighbor chosen in random performs poorly.

Coupling the algorithm of [9] with our deterministic algorithm yields the following randomized algorithm:

Algorithm AR: For every $1 \leq i \leq n$, choose a random permutation π_i of the vertices in V. Consider step l in the algorithm in which vertex $u \in U$ arrives, and denote by $j \geq 0$ the minimum load among u's neighbors upon its arrival. Vertex u is then assigned to the neighbor of load j having the highest priority according to π_{j+1}.

Theorem 4.2 *Let $G = (U, V, E)$ be a bipartite graph that contains a matching. The expected competitive ratio of Algorithm AR is at most $k = 1 + \ln(n)$ where $n = |U| = |V|$.*

Proof: We have to show that the expected load on each girl [1] is at most $k = 1 + \ln(n)$. Algorithm AR can be stated in the following equivalent way: let us construct n copies of the original graph, $G_1, \ldots, G_n$, and let the permutation π_i induce priorities on the vertices of G_i. When vertex $u \in U$ appears, let i be the smallest index such that in G_i, u has an unmatched neighbor. Then, u is matched with its highest priority unmatched neighbor in G_i, denoted by v and correspondingly, u is matched to v in the original graph G.

The following properties of the sequence of graphs $\{G_i\}$ are immediate at each step of the algorithm:

- If vertex $v \in V$ is matched in the graph G_i, then it is also matched in the graphs G_j, for $j \leq i$.
- The load on each girl $v \in V$ in G is equal to i, where i is the highest index such that v is matched in G_i .

We have to prove that the expected number of copies used by the algorithm in the sequence $\{G_i\}$ is at most k.

Algorithm AR has an interesting recursive property of which we make use later. Consider some graph G which was subjected to the execution of Algorithm AR, with permutations π_i, $i \geq 1$. Let M_1 be the set of vertices matched in G_1 during this execution. Consider now a new graph G' obtained from G by removing the vertices in M_1 and all their incident edges, and suppose Algorithm AR is executed on G' with permutations $\pi'_i = \pi_{i+1}$, $i \geq 1$, and with the same order of arrival of the vertices as in G. The matching that Algorithm AR produced in ${G'}_1, {G'}_2, \ldots$ is identical to the matching in $G_2, G_3, \ldots$ of the original execution. This property results from the fact that the permutation π_i participates only in the assignments of G_i and from the memoryless operation of Algorithm AR.

To generalize this, consider, as before, the sequence of graphs $\{G_i\}$ after Algorithm AR finished its execution. At that time, let M_i be the set of boys that are matched in G_i, let N_i denote the set of boys that are *not* matched at a lower indexed graph ($N_i = U - \cup_{j=1}^{i-1} M_j$), and let F_i be the graph that contains the set of boys N_i along with all their incident edges and neighbors. Notice that N_i, F_i and M_i all depend on $\pi_1 \ldots \pi_{i-1}$. Clearly, the graph F_i

[1] In the following, as commonly done, we refer to the set U as "the boys" and the set V as "the girls".

has a maximum matching of size $|N_i|$ and in addition, the matching of the M_i boys in G_i is a maximal matching in the graph F_i. Moreover, irrespective of the permutations $\pi_1 \ldots \pi_{i-1}$ themselves, this matching was chosen according to the ranking algorithm by using the random permutation π_i which is independent of the set N_{i-1} since the permutations are mutually independent. (the probability space contains all possible choices of the random permutations). Thus, by Theorem 4.1

$$E[|N_i|] \leq \frac{|N_{i-1}|}{e} \ .$$

Let $T(n)$ be a random variable denoting the maximum load in graph G (where $|U| = |V| = n$). To conclude the proof, we have to show that the expected value of $T(n)$ is at most k'. It is here that we make use of the recursive feature mentioned above. Let n be the number of boys in a graph and let n' be the number of boys left after the removal of those that were matched by Algorithm AR in G_1. Since Algorithm AR involves randomness, for a given n, n' is a random variable. The following probabilistic recurrence is defined on $T(n)$:

$$\begin{array}{rc} & T(n) \leq 1 + T(n') \\ \textit{such that} & E[n'] \leq \frac{n}{e} \\ \textit{and} & T(1) = 1, \ T(0) = 0. \end{array}$$

The first inequality stems from the fact that the graph with n' vertices is obtained from that of n vertices by eliminating the latter's G_1, meaning that the load difference cannot exceed unity. The second inequality results directly from Theorem 4.1.

We now prove that $E[T(n)] \leq 1 + \ln(n)$ for all positive n. To that end we define the continuous function $\Lambda(x)$ as follows:

$$\Lambda(x) = \begin{cases} 1 + \ln(x) & x > 1 \\ x & 0 \leq x \leq 1 \end{cases}$$

Note that the function $\Lambda(x)$ is concave as a result of the concavity of the log function and the fact that the straight line has the same derivative as the ln function at the point $x = 1$. In addition, since n' is a random variable, we denote $p_i = Prob[n' = i]$ and $E[n'] = \sum_i ip_i$. Note that always $n' < n$.

Assume inductively that $\Lambda(n)$ is the solution to the above recurrence relation. Then,

$$\begin{aligned} E[T(n)] &\leq 1 + E[T(n')] = 1 + \sum_{j=0}^{n-1} p_j T(j) \\ &\leq 1 + \sum_{j=0}^{n-1} p_j \Lambda(j) \leq 1 + \Lambda(\sum_{j=0}^{n-1} jp_j) \\ &= 1 + \Lambda(E[n']) \leq 1 + \Lambda(\frac{n}{e}) \end{aligned}$$

In the inequalities above, the transition in the second line takes advantage of the concavity of $\Lambda(x)$.

For $n > 2$ we get

$$E[T(n)] \leq 1 + \Lambda(\frac{n}{e}) = 1 + \left(1 + \ln(\frac{n}{e})\right) = 1 + \ln(n) \ .$$

Direct computation shows that $E[T(2)] \leq 1.5 < 1 + \ln(2)$. Hence, $E[T(n)] \leq 1 + \ln(n)$ for all positive n. $\square$

At first sight it seems that identical bounds should also hold for graphs which do not have a matching. Unfortunately, there does not seem to be a straightforward way of showing this. Let β denote the maximum load on a vertex in the off-line case. The obvious approach is to reduce the case of $\beta > 1$ to the case where $\beta = 1$ by replacing each vertex $v \in V$ by β copies of it, each having the same neighborhood set as v. The new graph, $G'(U, V', E')$, has a one-sided assignment. The requirement in Theorem 4.1 is that random permutations of size βn are chosen on V' by the randomized on-line algorithm. But, in effect, Algorithm AR chooses permutations of size n on the vertex set V. We conjecture that the bounds of Theorem 4.2 still hold in the case where $\beta > 1$.

4.2 The Lower Bound

Theorem 4.3 *The competitive ratio of any randomized on-line assignment algorithm is at least $k - 1 = \ln(n)$*

Proof: The theorem is proved by constructing a bipartite graph $G(U, V, E)$ in which the expected value of the competitive ratio is at least k. Again $|U| = |V| = n$. The vertices $u \in U$ of G appear one-by-one to a randomized on-line algorithm and G will be constructed on-line (step-by-step) by the adversary based on the on-line algorithm, but not on the outcome of the coin-flips of the algorithm. Let Y_i denote the set of neighbors of vertex $u_i \in U$ $(1 \leq i \leq n)$ and let H_n be the harmonic series sum, i.e., $H_n = \sum_{i=1}^{n} \frac{1}{i}$. The graph G has the following properties:

1. For $1 \leq i$, $Y_{i+1} \subset Y_i$.
2. $|Y_i| = n + 1 - i$
3. The sum of the expected loads of the vertices in Y_i at the beginning of Step i is at least $(n - i + 1)(H_n - H_{n-i+1})$.

If a graph G with such properties can be constructed, then at the beginning of Step n, the sum of the expected loads on the vertices of Y_n is at least $H_n - H_1 = H_n - 1$. Since $|Y_n| = 1$, the expected load on that vertex at the beginning of Step n is $H_n - 1$ and is increased to H_n during this step. Note that the set $Y_i - Y_{i+1}$ for $1 \leq i \leq n$ ($Y_{n+1} = \phi$) contains a single vertex which can be matched with the vertex v that appears in the i-th step, meaning that G contains a perfect matching. Thus, since $\ln(n) < H_n \leq 1 + \ln(n)$, construction of the graph proves the theorem.

To construct the graph G, we define Y_1 to be V. To obtain Y_{i+1}, at the end of step i the adversary chooses from Y_i the set of $n - i$ vertices with the largest expected load. Being familiar with the algorithm, the adversary can compute the expected load of each vertex by averaging over all possible coin-flips of the algorithm. Properties 1 and 2 are clearly satisfied. We prove by induction on the step number that property 3 holds as well. Property 3 trivially holds for Step $i = 1$ and assume inductively that it holds at the beginning of Step i (before vertex u_i appeared). Thus, at the beginning of step i the sum of the expected loads of the vertices in Y_i is at least $(n - i + 1)(H_n - H_{n-i+1})$. By definition, the neighborhood of vertex u_i is precisely Y_i, which implies that the sum of the expected load on the vertices in Y_i at the end of step i is at least

$$(n - i + 1)(H_n - H_{n-i+1}) + 1 = (n - i + 1)(H_n - H_{n-i}) \ .$$

By the above choice of the set Y_{i+1}, the sum of the expected loads on the vertices in Y_i (at the end of Step i) is at least

$$\frac{n-i}{n-i+1} \cdot (n - i + 1)(H_n - H_{n-i}) = (n - i)(H_n - H_{n-i})$$

which completes the proof. □

4.3 An Adaptive Adversary

A different type of adversary for randomized algorithms is the *adaptive adversary* (See [4]). This adversary does not have to construct the graph in advance but rather is allowed to construct it step by step, taking advantage of the coin flips and assignments made by the on line algorithm up to that point. However, the adversary is not familiar with the outcome of future coin flips. The *adaptive on-line* adversary has to serve the request immediately, whereas the *adaptive off-line* adversary serves the request only after the sequence of requests is completed.

It is clear that the adaptive on-line adversary is at least as powerful as the oblivious one and at most as powerful as the adaptive off-line adversary. It is also known (see [4]) that the adaptive off-line adversary has the same power as any adversary for deterministic algorithms. (For deterministic algorithms all adversaries are equivalent).

We now show that randomization does not improve the competitive ratio (denoted by k'') against an adaptive on-line adversary. From the discussion above and from the previous sections we conclude that $\ln n \leq k'' \leq \lceil \log_2(n + 1) \rceil$. We now extend the lower bound for the deterministic algorithm for this case. Recall that in each step in the proof of the lower bound (see Section 3.2), the adversary provides the on-line algorithm with a set of vertices S_i . We will now consider each vertex from this set *separately* and describe the decisions of the off-line algorithm as a function of the decisions made by the on-line algorithm. When the first vertex v of the set S_i appears, its set of neighbors is Y_i, as described in the proof of Theorem 3.2. The on-line algorithm chooses to assign v to some vertex of Y_i, say u, while the adversary chooses to assign v to a *different* vertex of Y_i, say w. The neighborhood set of the next vertex in S_i is updated by deleting the two vertices u and w from Y_i. Since, initially, $|Y_i|$ is twice as large as $|S_i|$, every vertex in S_i has at least two neighbors, and the adversary can match each vertex to a different vertex than the on-line algorithm. It is easy to verify that the adversary's matching is perfect. Moreover, after the on-line algorithm has matched the entire set S_i, exactly half the vertices in Y_i will have load $i-1$ and the other half will have load i. Y_{i+1} will be exactly the latter half of the set Y_i. Thus, at the end, the load on vertex S_k will be exactly k, which completes the proof.

Notice that our proof assumes that the adversary serves each request after the algorithm has served it. If both the adversary and the algorithm are required to serve the request simultaneously, the same lower bound holds as well, up to additive small lower order terms.

5 Split Assignments

In this section we consider a variant of the assignment problem where a vertex from the set U is allowed to split its weight among any of its neighbors. We allow the weights to be split into arbitrary positive real portions. Notice that in this case the optimal off-line solution can be computed in polynomial time by reduction to the maximum flow problem.

Algorithm AS: Upon arrival of vertex $u_i \in U$ with weight w_i: let Y denote its set of neighbors, and let Y_m be those vertices of Y with minimal load before the assignment of w_i takes place. Assign the weight w_i to the set of nodes $Y_a \subseteq Y$ such that $Y_m \subseteq Y_a$ and all vertices of Y_a end up having equal load.

Algorithm AS is a very natural one despite its complex description. Essentially the weight w_i is distributed among the neighbors of the arriving nodes in a manner that evenly raises "the low water mark" of these neighbors.

Theorem 5.1 *The competitive ratio of any on-line algorithm solving the split-assignment problem is* $\Theta(\min\{m, \log n\})$ *where* $|U| = m$ *and* $|V| = n$.

Proof: We first show that m is an upper bound on the performance of this algorithm. Upon arrival of vertex u_i, let v_{j_i} denote the neighbor which has maximum load. After u_i splits its weight among its neighbors, v_{j_i} will still be the neighbor which has maximum load (not necessarily the only one). Hence, its load could have increased by at most $\frac{w_i}{d_i}$, where

d_i denotes the degree of u_i. Implying that at the end, the maximum load on a vertex in V is at most,

$$m \cdot \max_{i \leq m} \left\{ \frac{w_i}{d_i} \right\}$$

Obviously, for the off-line algorithm, there exists a vertex whose load is at least

$$\max_{i \leq m} \left\{ \frac{w_i}{d_i} \right\} .$$

To prove that the performance of this algorithm is at most $\log n$, we slightly modify the proof of Theorem 3.1. Suppose that the weight w_i of vertex u_i was split in the off-line solution, OPT, into weights $w_{i1}, \ldots, w_{id_i}$. To simplify the proof, we now change the rules for the off-line algorithm: (i) vertex u_i appears as d_i different vertices, each with weight w_{iq}, $1 \leq q \leq d_i$; (ii) the off-line algorithm is not allowed to split the weights anymore. (Clearly, now, splitting cannot improve the performance of the off-line algorithm). However, the on-line algorithm receives the vertices as before and is allowed to split their weight. To prove the competitive ratio, we now follow the proof of Theorem 3.1. The only lemma whose proof needs to be modified is Lemma 3.1, although with splitting the proof becomes much simpler. A modified proof of Lemma 3.1 (using the same notation) is presented next.

Lemma 5.1 *For every i and for each vertex $v_j \in V$, $W_{ij} \geq R_{ij}$.*

If the set S_{ij} is empty, then $R_{ij} = 0$ and the lemma holds. Otherwise, if it is not empty for some i and j, we claim that $W_{ij} = \ell$. The reason is that S_{ij} cannot be nonempty if layer i of vertex v_j is only partially filled; since the on-line algorithm can split the weights, it will first finish filling layer i of vertex v_j before filling layer $i+1$ of any other vertex. Since $R_{ij} \leq \ell$ the lemma holds in this case as well. This completes the proof of the upper bound. □

The proof of the lower bound follows from the graph constructed in the proof of Theorem 3.2 by contracting the vertices of each set S_i into one vertex whose weight is $|S_i|$. More specifically, the cardinality of vertex set U is m and the cardinality of V is 2^{m-1}. (Notice that here m and $\log n$ differ by one unit). For $1 \leq i < m$, the weight of vertex $u_i \in U$ is 2^{m-1-i} and the weight of u_m is 1. The degree of vertex u_i is 2^{m-i}. The adversary chooses the 2^{m-i} vertices of largest load from the neighborhood set of u_{i-1} to be the neighborhood set of u_i. It is easy to verify that after vertex u_i was served by the algorithm, the average load of the vertices in its neighborhood set is at least $i/2$. At the end, therefore, the load of the (single) neighbor of u_m is $m/2$.

It is easy to see that $\log n$ is a lower bound on the competitive ratio in case $\log n < m$. The same proof applies, but now only the first $\log n + 1$ vertices in U have non-zero weights. All the other vertices have zero weight. □

Notice that the proof of the lower bound holds for the randomized case as well, by considering the expected load of the vertices in V instead of their actual load.

Acknowledgement

We would like to thank Samir Khuller for useful discussions.

References

[1] Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., and Waartz, O. On-line machine scheduling with applications to load balancing and virtual circuit routing. In *Proc. 25th ACM Symp. on Theory of Computing* (May 1993), pp. 623–631.

[2] Azar, Y., Broder, A., and Karlin, A. On-line load balancing. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science* (October 1992), pp. 218–225.

[3] Azar, Y., Kalyanasundaram, B., Plotkin, S., Pruhs, K., and Waarts, O. On-line load balancing of temporary tasks. In *Proc. Workshop on Algorithms and Data Structures* (August 1993), pp. 119–130.

[4] Ben-David, S., Borodin, A., Karp, R. M., Tardos, G., and Wigderson, A. On the power of randomization in on-line algorithms. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (Baltimore, Maryland, May 1990), pp. 379–386.

[5] Fiat, A., Karp, R. M., Luby, M., McGeoch, L. A., Sleator, D. D., and Young, N. E. Competitive paging algorithms. *Journal of Algorithms 12* (1991), 685–699.

[6] Graham, R. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal 45* (1966), 1563–1581.

[7] Irani, S. Coloring inductive graphs on-line. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science* (St. Louis, Missouri, October 1990), pp. 470–479.

[8] Kalyanasundaran, B., and Pruhs, K. Online weighted matching. *Journal of Algorithms 14* (1993), 478–488.

[9] Karp, R. M., Vazirani, U. V., and Vazirani, V. V. An optimal algorithm for on-line bipartite mathcing. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (Baltimore, Maryland, May 1990), pp. 352–358.

[10] Lovász, L., Saks, M., and Trotter, W. T. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Math 75* (1989), 319–325.

[11] Manasse, M. S., McGeoch, L. A., and Sleator, D. D. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, May 1988), pp. 322–333.

[12] Shmoys, D., Wein, J., and Williamson, D. Scheduling parallel machines on-line. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science* (1991), pp. 131–140.

[13] Sleator, D. D., and Tarjan, R. E. Amortized efficiency of list update and paging rules. *Communications of the ACM 28*, 2 (1985), 202–208.

[14] Vishwanathan, S. Randomized on-line graph coloring. *Journal of Algorithms 13* (1992), 657–669.