

# SIGACT News Online Algorithms Column 10

Marek Chrobak  
Department of Computer Science  
University of California, Riverside

**From the editor:** In this quarter’s column, written jointly with Claire Kenyon-Mathieu, we discuss the doubling method used to design online and offline approximation algorithms.

As usual, I would like to invite new contributions to this column: surveys, technical articles, reviews, opinions, conference reports – anything related to online algorithms and competitive analysis. If you are considering becoming a guest writer, please do not hesitate to contact me.

---

## Competitiveness via Doubling

**Marek Chrobak**  
Department of Computer Science  
University of California, Riverside

**Claire Kenyon-Mathieu**  
Computer Science Department  
Brown University

### 1 Introduction

We discuss what we refer to, tentatively, as the “doubling” method for designing online and offline approximation algorithms. The rough idea is to use geometrically increasing estimates on the optimal solution to produce fragments of the algorithm’s solution. The term “doubling” is a little misleading, for often factors other than 2 are used, and suggestions for a better name will be appreciated.

It is not easy to formulate a precise, yet also general and elegant description of this method. In this expository paper, we illustrate this method by discussing several applications where doubling is used, and these examples should be sufficient to elucidate the underlying idea. (This is not meant to be a comprehensive survey; we suspect that there are many more examples of doubling-based approximation algorithms in the literature.)

The *online bidding* problem [15] described in the next section is an attempt to formalize the doubling method. Online bidding captures many but not all applications of the doubling method, excluding, for example, the cow-path problem (especially the randomized case) and its extensions. We hope that this paper will clarify the relationship between similar solutions to apparently quite different problems, and will help to make doubling a standard tool for designing approximation

algorithms.

## 2 Online Bidding

In *online bidding* [15], faced with some unknown integer threshold  $u$ , a player submits a sequence  $(d_\ell)$  of *bids* until one is greater than or equal to  $u$ , paying the sum of its bids. Any strategy is defined by its sequence of bids, and its competitive ratio is

$$\max_{u,k} \left\{ \frac{d_0 + d_1 + \cdots + d_k}{u} : d_{k-1} < u \leq d_k \right\}. \quad (1)$$

What is the best sequence  $(d_\ell)$ ? A natural approach is to double the bid each time, that is  $d_\ell = 2^\ell$  for  $\ell \geq 0$ . The worst case in (1) is when  $u = d_{k-1}$ , and routine calculations show that this strategy's competitive ratio is 4, which also turns out to be optimal in the deterministic setting.

Express the ratio (1) as the product of  $(\sum_{\ell \leq k} d_\ell)/d_k$  by  $d_k/u$ . We can do better with randomization, by choosing the bid sequence  $(d_\ell)$  at random so that  $d_k/u$  is, in expectation, less than the worst case  $d_k/d_{k-1}$ . Let  $d_\ell = \alpha e^\ell$ , where  $\alpha = e^X$  with  $X$  uniformly distributed in  $[0, 1)$ . The first term of the product is  $(\sum_{\ell \leq k} d_\ell)/d_k \sim \sum_{i \geq 0} e^{-i} = 1/(1 - e^{-1})$ . For the second term, observe that  $d_k/u$  is distributed as  $e^X$ , with  $X$  uniform in  $[0, 1)$ , and so its expectation is  $\int_0^1 e^X dX = e - 1$ . Hence the algorithm is  $e$ -competitive, and this is optimal [15].

In some applications, the unknown threshold and the bids are constrained to be in a fixed universe  $U$ : in this paper, expressions (4) and (12) are such examples. It is simple to adapt the algorithms to such cases: to define  $d_\ell$ , choose the largest element of  $U$  which is less than or equal to  $2^\ell$  in the deterministic case and less than or equal to  $\alpha e^\ell$  in the randomized case.

## 3 Cow-Path Problem

The cow-path problem is a classical problem in online algorithms and search theory, studied since the 60's [7, 6], and discussed earlier in this column in the December'03 and June'04 issues. (See also [19] for some recent results and more references.) A cow faces a fence, infinite in both directions, attempting to find a hole in order to get to the green pasture on the other side. The cow's strategy specifies the path traveled in search of the hole and the goal is to minimize the distance traveled.

Any reasonable strategy will have the cow gradually extend the explored interval of the fence, alternatively to the left and right. We think of the fence as the real line, and assume the start position is 0 and that the hole's position is integer. Such a strategy is then defined by an infinite sequence  $(d_\ell)$  of turning points: go right for distance  $d_0$ , then back to the origin, left for distance  $d_1$ , back to the origin, right for distance  $d_2$ , and so on, until the hole is found, say at distance  $u$  from the origin. The total travel distance is  $2d_0 + 2d_1 + \cdots + 2d_{k-1} + u$ , where  $d_{k-2} < u \leq d_k$ , so the competitive ratio is

$$\max_{u,k} \left\{ \frac{2(d_0 + d_1 + \cdots + d_{k-1}) + u}{u} : d_{k-2} < u \leq d_k \right\}. \quad (2)$$

What is the best sequence  $(d_\ell)$ ? We try doubling again, setting  $d_\ell = 2^\ell$  for  $\ell \geq 0$ . This gives competitive ratio at most 9. (Indeed, (2) is 1 plus twice the ratio (1), since in both cases to determine the ratio we choose the minimum  $u$ .) In fact, it turns out that no better ratio can be achieved by a deterministic strategy [22, 4, 36].

We can reduce this ratio with randomization. Note that for any algorithm, its “mirror image” algorithm that reverses left and right moves has the same competitive ratio. Also, picking one of those two at random can only improve the ratio. So, without loss of generality, we can assume that the algorithm starts by going first left or right with probability  $\frac{1}{2}$  each. If we restrict attention to algorithms such that the sequence  $(d_\ell)$  is monotone, then, for  $d_{k-2} < u \leq d_{k-1}$ , with probability  $\frac{1}{2}$  the hole is on the side of the path of length  $d_{k-1}$ , and so the competitive ratio is

$$\max_{u,k} \left\{ \frac{\mathbb{E}[2(d_0 + \dots + d_{k-2} + \frac{1}{2}d_{k-1})] + u}{u} : d_{k-2} < u \leq d_{k-1} \right\}. \quad (3)$$

Let  $\gamma \approx 3.591$  be the solution of the equation  $\gamma \ln \gamma = \gamma + 1$ . The algorithm chooses first a number  $\alpha = \gamma^X$ , where  $X \in [0, 1)$  is chosen uniformly at random. The  $\ell$ 'th turning point is  $d_\ell = \alpha\gamma^\ell$ . With this strategy the competitive ratio is  $\gamma + 1 \approx 4.591$ , and this is optimal [22, 30].

Again, it is easy to adapt the algorithm to the case where it is constrained to choose its bids from among a fixed universe  $U$ : to define  $d_\ell$ , choose the largest element of  $U$  which is less than or equal to  $2^\ell$  in the deterministic case and less than or equal to  $\alpha\gamma^\ell$  in the randomized case.

## 4 Minimum Latency Tours

Blum *et al* [9], and later Goemans and Kleinberg [23, 24], study the *minimum latency problem* (MLP). Given a path  $v_1v_2\dots$  in a metric space, the *latency* of  $v_i$  on this path is the length of the prefix path  $v_1v_2\dots v_i$ . The objective is to find a path that starts at a specified point  $v_1$ , visits all vertices in a metric space (e.g., a traveling salesman tour), and minimizes the total latency (sum of the individual point latencies). Needless to mention, MLP is NP-hard.

Consider another variation of traveling salesman called  $k$ -TSP. Given an integer  $k$  and a point  $v_1$ , we seek a minimum  $k$ -tour, that is a tour  $T_k$  starting and ending at  $v_1$  that visits at least  $k$  different points and has minimum length  $d_k$ . If the metric space is a weighted tree (the distances are path lengths in this tree),  $k$ -tours can be computed in polynomial time with dynamic programming [9]. This is an interesting case since the minimum latency problem is NP-hard even for weighted trees [37].

The minimum latency algorithm from [9] chooses a certain sequence  $j_0, j_1, \dots, j_m = n$ , and outputs the concatenation  $Q = T_{j_0}T_{j_1}\dots T_{j_m}$ . Some vertices may be visited many times (in fact, some trees may be traversed repeatedly) but that's okay – for the purpose of computing the latency we only count the first visit. The latency of the  $k$ th visited vertex is at most  $d_{j_0} + d_{j_1} + \dots + d_{j_i}$  where  $k \leq j_i$ .

On the other hand,  $k$ -tours also give us an estimate on the optimum latency. Indeed, suppose  $S = v_1v_2v_3\dots$  is some optimal solution of MLP. Traversing the first  $k$  vertices on  $S$  forth and back, we

get a  $k$ -tour  $v_1v_2\dots v_kv_{k-1}\dots v_1$  which, by the optimality of  $T_k$ , has length at least  $d_k$ . So the latency of  $v_k$  in  $S$  is at least  $\frac{1}{2}d_k$ , and if we denote by  $U$  the set  $\{d_k, 1 \leq k \leq n\}$ , the approximation ratio is bounded by

$$\max_{u \in U, k} \left\{ \frac{d_{j_0} + d_{j_1} + \dots + d_{j_i}}{\frac{1}{2}u} : d_{j_{i-1}} < u \leq d_{j_i} \right\}. \quad (4)$$

We observe that the ratio (4) is exactly twice that of the online bidding (1), implying a deterministic 8-approximation [9] on weighed trees (and a randomized  $2e$ -approximation.)

Goemans and Kleinberg [23, 24] improve on [9] by traversing the partial tours more efficiently and by using randomization. As before, we consider the concatenated tour  $Q = T_{j_0}T_{j_1}\dots T_{j_m}$ , but now, in  $Q$  we traverse each  $T_j$  either clockwise or counter-clockwise, choosing the direction that produces a smaller contribution to the total latency. Note that  $Q$  is at least as good as the tour  $R$  which chooses the clockwise or counter-clockwise direction uniformly at random. In  $R$ , the average latency of the  $k$ th vertex visited is at most  $d_{j_0} + \dots + d_{j_{i-1}} + \frac{1}{2}d_{j_i}$ , so the competitive ratio is bounded by

$$\max_{u \in U, i} \left\{ \frac{d_{j_0} + \dots + d_{j_{i-1}} + \frac{1}{2}d_{j_i}}{\frac{1}{2}u} : d_{j_{i-1}} < u \leq d_{j_i} \right\}. \quad (5)$$

Applying the doubling method to choose the  $j_i$ s yields a deterministic 6-approximation algorithm. Observe that ratio (5) is just one less than the cow-path ratio (3), so the ratio is  $\approx 3.591$  [23, 24].

We don't actually need randomization to get this ratio, for it is not hard to show that there are at most  $n$  sequences that can result from different choices of  $\alpha$ , so we can simply try all of them. Indeed, as  $\alpha$  ranges from 1 to  $\gamma$ , the sequence changes only when some  $d_k$  exactly equals  $\alpha$  times some power of  $\gamma$ ; but for each fixed  $d_k$ , this happens exactly once, hence there are at most  $n$  changes and therefore at most  $n$  sequences. With some care, this optimal  $\alpha$  can be, in fact, found in time  $O(n \log n)$ . (Goemans and Kleinberg [23, 24] gave a different approach to eliminate randomization.)

Summarizing, the method we outlined above yields a polynomial-time  $\approx 3.591$ -approximation algorithm – if we can compute optimal  $k$ -tours in polynomial time. Thus it applies directly to weighted trees. For general metric spaces, this approach allows us to convert any polynomial-time  $c$ -approximation algorithm for  $k$ -TSP into a polynomial-time  $8c$ -approximation algorithm for MLP. Although no approximation algorithms for  $k$ -TSP were known at the time when [9] was published, Blum *et al* were able to refine this idea to obtain a 72-approximation algorithm for MLP. The ratio has been gradually reduced over time by using better approximations for  $k$ -spanning trees, culminating in the 2003 work of Chaudhuri *et al* [13], who achieved the approximation ratio of  $\approx 3.591$  – same as for weighted trees.

## 5 Non-Clairvoyant Scheduling

Now we turn to online scheduling. Motwani, Phillips, and Torng [34, 35] study preemptive scheduling of jobs in the *non-clairvoyant* setting where a processing time only becomes known upon completion of the job; the objective function is to minimize the sum of completion times. We will focus on the

case of 1-processor scheduling with all jobs released simultaneously at time 0. We will also assume that each job's processing time is at least 1.

The RoundRobin algorithm works in rounds, and in each round it assigns a unit of time to each pending job. As shown in [34, 35], RoundRobin is 2-competitive and (for unlimited  $n$ ) no better ratio is possible even with randomization. This might seem to completely solve the problem — but not quite. In RoundRobin, a job of length  $u$  can be preempted  $\Theta(u)$  times, which is certainly an undesirable feature when a preemption involves a costly context switch. Can we achieve constant-competitiveness with fewer preemptions?

The geometric algorithm (let's call it GeomRR) proposed in [34, 35] works in phases numbered  $0, 1, 2, \dots$ , where, in phase  $\ell$ , each unfinished job is allocated time  $d_\ell$ . Clearly, a job of length  $u$  is preempted now only  $k$  times, where  $d_1 + d_2 + \dots + d_{k-1} < u \leq d_1 + \dots + d_k$ .

Let  $u_1 \leq u_2 \leq \dots \leq u_n$  denote the processing times. Given a schedule, for every pair of jobs  $\{i, j\}$ , let  $P_{ij}$  denote the mutual delays, that is, the times at which one of the two jobs is being processed while the other one is not yet finished. We can write the sum of completion times as  $\sum_i u_i + \sum_{\{i, j\}} P_{ij}$ . For a given pair  $\{i, j\}$ , assume  $u_i \leq u_j$ . Clearly, even the optimal schedule must have  $P_{ij} \geq u_i$ . In the GeomRR's schedule, let  $k$  be such that  $d_1 + d_2 + \dots + d_{k-1} < u_i \leq d_1 + \dots + d_k$ . Then job  $j$  delays job  $i$  by at most  $d_1 + \dots + d_k$ , and job  $i$  delays job  $j$  by at most  $u_i$ . Thus, using the notation  $d'_k = d_1 + d_2 + \dots + d_k$ , we have that the competitive ratio of GeomRR is bounded by

$$\max_{u, k} \left\{ \frac{d'_k + u}{u} : d'_{k-1} < u \leq d'_k \right\}. \quad (6)$$

For any  $\beta > 1$ , the sequence  $d'_\ell = \beta^\ell$  gives competitive ratio  $1 + \beta$ . Here, again, we can do better with randomization. First notice that with random ordering of the jobs within the rounds, with probability  $\frac{1}{2}$  job  $j$  comes after job  $i$  and therefore only delays  $i$  by  $d'_{k-1}$ . Hence the randomized competitive ratio is bounded by

$$\max_{u, k} \left\{ \frac{\mathbb{E}[\frac{1}{2}d'_{k-1} + \frac{1}{2}d'_k + u]}{u} : d'_{k-1} < u \leq d'_k \right\}. \quad (7)$$

Choose a number  $\alpha = \beta^X$ , where  $X \in [0, 1)$  is chosen uniformly at random, and let  $d'_k = \alpha\beta^k$ . Since  $\int_0^1 \beta^X dX = (\beta - 1)/\ln \beta$ , this algorithm has competitive ratio  $1 + (\beta + 1)(\beta - 1)/(2\beta \ln \beta)$ . Both algorithms, deterministic and randomized, provide a smooth tradeoff between the competitiveness and the number of preemptions: with  $\beta \rightarrow 1$ , the competitive ratio approaches 2, while the number of preemptions for jobs of length  $u$  increases as  $O(\log_\beta u)$ .

## 6 Scheduling with Min-Sum Criteria

We continue with online job scheduling, although the setting now is quite different from that in Section 5. Specifically, jobs have release dates and arrive online: their existence is revealed only when they are released. At that time, their processing time  $p_j$  and weight  $w_j$  is also given. The objective is to minimize the weighted sum of completion times,  $\sum_{j=1}^n w_j C_j$ . The technique we will

describe applies to a variety of settings: for example, there can be one or many processors, identical or not, and preemption may or may not be allowed.

The algorithm proposed by Hall *et al* [28, 27] executes jobs during phases as follows. Fix an increasing sequence  $(d_\ell)$  such that  $d_\ell - d_{\ell-1} \geq d_{\ell-1}$ . Phase  $\ell$  starts at time  $d_{\ell-1}$  and ends at time  $d_\ell$ ; the algorithm (somehow) selects, from among the jobs pending at time  $d_{\ell-1}$ , a maximum-weight subset that can be processed between  $d_{\ell-1}$  and  $d_\ell$ , and executes them in arbitrary order.

For the analysis, let  $W_\ell$  denote the total weight of the jobs scheduled by this algorithm by time  $d_\ell$ . Letting  $W$  denote the total weight, we have:

$$\sum_{j=1}^n w_j C_j \leq \sum_{\ell} (W_\ell - W_{\ell-1}) d_\ell = \sum_{\ell} (d_\ell - d_{\ell-1}) (W - W_{\ell-1}).$$

Since  $d_{\ell-1} - d_{\ell-2} \geq d_{\ell-2}$ ,  $W_{\ell-1}$  is at least as large as the weight  $W_{\ell-2}^*$  of jobs scheduled in the optimal schedule by time  $d_{\ell-2}$ . Thus:

$$\sum_{j=1}^n w_j C_j \leq \sum_{\ell} (d_\ell - d_{\ell-1}) (W - W_{\ell-2}^*) = \sum_{\ell} (W_\ell^* - W_{\ell-1}^*) d_{\ell+1} = \sum_{\ell} \sum_{j \in I_\ell} w_j d_{\ell+1},$$

where  $I_\ell$  denotes the set of jobs completed in the optimal schedule in time  $(d_{\ell-1}, d_\ell]$ . On the other hand, the optimal schedule has value  $\sum_{\ell} \sum_{j \in I_\ell} w_j C_j^*$ . Thus the competitive ratio can be bounded by:

$$\max_{(c_j), \ell_j} \left\{ \frac{\sum_j w_j d_{\ell_j+1}}{\sum_j w_j c_j} : d_{\ell_j-1} < c_j \leq d_{\ell_j} \right\} \leq \max_{u, \ell} \left\{ \frac{d_{\ell+1}}{u} : d_{\ell-1} < u \leq d_\ell \right\}. \quad (8)$$

The doubling method yields a deterministic 4-approximation [28] (with  $d_\ell = 2^\ell$ ) and a randomized  $\approx 2.891$ -approximation [11] (with  $d_\ell = \alpha 2^\ell$  where  $\alpha = 2^X$  and  $X$  chosen uniformly at random from  $[0, 1)$ ).

An easy improvement: the algorithm can schedule the jobs in phase  $\ell$  (by taking the better of one ordering and its reverse ordering) so that their average weighted completion time is  $(d_{\ell-1} + d_\ell)/2$  instead of the crude upper bound  $d_\ell$ ; this variant leads to the bound

$$\max_{u, \ell} \left\{ \frac{\frac{1}{2}d_\ell + \frac{1}{2}d_{\ell+1}}{u} : d_{\ell-1} < u \leq d_\ell \right\}, \quad (9)$$

yielding, by the doubling method, a deterministic 3-approximation [28] and a randomized  $\approx 2.16$ -approximation.

Unfortunately, because of the “somehow” step, this algorithm, as described, is not polynomial-time. We have effectively reduced our *online* min-sum problem with release times, to an *offline* makespan problem without release dates: given a deadline  $D$  and a set of jobs released at time 0, construct a schedule with makespan at most  $D$  that maximizes the total weight of completed jobs. Now, suppose that there exists a “dual- $\rho$ -approximation” algorithm *DualPack* for that problem – it outputs a schedule with makespan at most  $\rho D$  and with total weight at least as large as that of an optimal schedule with makespan  $D$ . We can then modify the algorithm so that during phase  $\ell$  it

uses DualPack with  $D = d_\ell - d_{\ell-1}$  to schedule the next batch in the interval  $[\rho d_{\ell-1}, \rho d_\ell)$ , and the resulting approximation ratio will simply be multiplied by  $\rho$ .

For example, in the setting of one-processor non-preemptive scheduling, Hall *et al* [28, 27] give a DualPack with ratio  $(1 + \epsilon)$ , which, incorporated into the framework above, immediately yields an online polynomial time deterministic algorithm with competitive ratio  $3 + \epsilon$  (see also [14].) Numerous improvements and extensions of the doubling method in the context of min-sum scheduling method appeared subsequently in the literature, see, for example, [10, 16]. For one processor, a 2-competitive algorithm, that does not use doubling, was recently given in [1], and this ratio is optimal [29].

## 7 List Scheduling on Related Machines

The list scheduling problem is a classical online problem first studied by Graham [26] in 1960's. The input consists of a list of jobs of varying processing times that need to be assigned to  $m$  machines. The jobs are processed one by one. At step  $j$ , the size  $p_j$  of job  $j$  is revealed and we need to assign it to a machine. The objective is to minimize the *makespan*, that is, the maximum machine load (sum of job processing times assigned to this machine). List scheduling differs from the “real-time” online scheduling model discussed in Sections 5 and 6 in that the decision time is unrelated to the job execution time.

We focus here on the version  $Q|pmtn|C_{\max}$  – namely the case of preemptive scheduling on related machines, where each machine can run at a different speed. Eberlendr and Sgall [20] provide two online algorithms for this problem: a 4-competitive deterministic algorithm and an  $e$ -competitive randomized algorithm. Their work is based on earlier ideas from [3, 8, 5]. (See also [21] for some recent improvements.)

The key observation is that knowing the optimal makespan  $C_{\max}^*$  helps. In fact, [20, 21] give a “semi-online” algorithm  $\text{InTime}(T)$  that is 1-competitive (that is, it computes an *optimal* schedule) if it so happens that  $C_{\max}^* = T$ .

In the truly online case we don't know  $C_{\max}^*$  so, instead, we use a sequence  $(d_\ell)$  of geometrically increasing estimates, i.e., such that  $d_{\ell+1} - d_\ell \geq d_\ell$ . In phase  $\ell = 0, 1, 2, \dots$  we assume that the optimal makespan is  $d_\ell$  and we apply  $\text{InTime}(d_\ell)$  to the remaining jobs, attempting to schedule them in the time interval  $[d_\ell, d_{\ell+1})$ . If  $\text{InTime}(d_\ell)$  fails, that is, a job arrives that cannot be squeezed into this interval, we know that  $C_{\max}^* > d_{\ell+1} - d_\ell \geq d_\ell$ , so we increase our estimate to  $d_{\ell+1}$ , and continue.

For the analysis, suppose that  $d_{k-1} < C_{\max}^* \leq d_k$ , for some integer  $k \geq 1$ . The algorithm will produce a schedule with makespan at most  $\sum_{\ell=0}^k d_\ell$ , and so the competitive ratio is bounded by

$$\max_{u,k} \left\{ \frac{d_0 + d_1 + \dots + d_k}{u} : d_{k-1} < u \leq d_k \right\}. \quad (10)$$

Notice that this is simply the online bidding ratio (1), so the deterministic competitive ratio is bounded by 4 and the randomized competitive ratio is bounded by  $e$ .

We focused in this section on the preemptive case, but a fair amount of work has been done on the non-preemptive case as well, see [3, 8, 5] and references therein.

## 8 Incremental and Hierarchical Clustering

We now consider the *k-clustering problem*, where we want to partition a given set  $X$  of points in a metric space into  $k$  clusters  $C_1, \dots, C_k$  so as to minimize the maximum cluster *diameter* (the maximum distance between any two points in a cluster).

In the incremental version studied by Charikar *et al* [12], we wish to maintain a  $k$ -clustering of  $X$  while new points are inserted into  $X$  over time. The online algorithm is not allowed to divide existing clusters. When a new point is inserted into  $X$ , it can be either added to an existing cluster or it can become a new, singleton cluster. Additionally, at any time, the algorithm can merge some clusters into one. Besides efficiency and simplicity, these restrictions guarantee that the produced clustering will have a tree-like hierarchical structure useful for certain applications.

The incremental  $k$ -clustering algorithm in [12] uses a sequence  $d_\ell$ ,  $\ell = 0, 1, \dots$ . At each step it has up to  $k$  clusters  $C_1, \dots, C_h$ , each with a designated *center*  $c_i \in C_i$ . The algorithm works in phases, and in phase  $\ell = 0, 1, 2, \dots$ , it will satisfy the invariant that the distance between any two centers is greater than  $d_\ell$ .

Initially, each of the first  $k$  points becomes its own cluster. Without loss of generality, assume that the minimum distance between the first  $k$  points is greater than 1. During the  $\ell$ th phase, whenever a new point  $x$  arrives, we attempt first to add it to an existing cluster. If there is a center  $c_i$  at distance at most  $d_\ell$  from  $x$ , we add  $x$  to  $C_i$ . Otherwise, if there are fewer than  $k$  clusters, we create a new cluster  $\{x\}$ . In the remaining case, we cannot preserve the invariant for this phase anymore and the phase ends. To prepare for the next phase, we create a temporary  $(k + 1)$ st cluster  $\{x\}$ , and to reduce the number of clusters we merge some, as follows: Greedily find a maximal set  $J$  of centers whose pairwise distances are greater than  $d_{\ell+1}$ . For each other center  $c_i \notin J$  there is a center  $c_j \in J$  at distance at most  $d_{\ell+1}$ . We merge  $C_i$  into  $C_j$ , with  $c_j$  becoming the center of the new cluster. When we are done, all inter-center distances are greater than  $d_{\ell+1}$ . So now we are ready to start the new phase  $\ell + 1$ . (Except if no merging occurred, in which case phase  $\ell + 1$  will be empty and we go directly to the next phase, etc.)

To analyze this algorithm, define the *radius* of  $C_i$  to be the maximum distance between  $c_i$  and a point in  $C_i$ . Note that at the beginning of phase  $\ell$  the  $k + 1$  centers are all at distance greater than  $d_{\ell-1}$  from each other, which implies that the diameter of the optimal  $k$ -clustering must be greater than  $d_{\ell-1}$ . The merges at the end of phase  $\ell - 1$  increased the maximum radius by at most  $d_\ell$ , so the maximum radius during phase  $\ell$  is at most  $d_1 + d_2 + \dots + d_\ell$ , and the maximum diameter is at most twice that. The competitive ratio is therefore bounded by

$$\max_{u,k} \left\{ \frac{2(d_0 + d_1 + \dots + d_k)}{u} : d_{k-1} < u \leq d_k \right\}. \quad (11)$$

Again, this is simply twice the online bidding ratio (1), and thus the above method yields a deterministic 8-competitive algorithm and a randomized  $(2e)$ -competitive algorithm. (The same technique works if we measure the cluster size by the radius instead of the diameter.)

Dasgupta and Long [18] have a similar *hierarchical clustering* algorithm. They are initially given all the points. First, they order them according to a farthest point traversal [25]:  $p_2$  is the point

farthest from  $p_1$ , at distance  $x_1$ ,  $p_3$  is the point farthest from  $\{p_1, p_2\}$ , at distance  $x_2$ , etc. This naturally induces a tree structure  $T$  rooted at  $p_1$ . Given an increasing sequence  $(d_\ell)$ , Define a new tree  $T'$  by setting  $p_i$ 's parent to be its closest ancestor  $p_j$  in  $T$  such that  $x_i$  and  $x_j$  are separated some element of the sequence:  $x_j > d_\ell \geq x_i$  for some  $\ell$ . The  $k$ -clustering is the partition into connected components obtained from  $T'$  by removing the edges from  $p_2, \dots, p_{k+1}$  to their parents.

Again, it is not hard to prove that the diameter of the  $k$ -clustering is at most  $2(d_1 + d_2 + \dots + d_\ell)$ , and that the optimal  $k$ -clustering has diameter greater than  $d_{\ell-1}$ , hence this, via online bidding again, leads to a deterministic  $8$ -competitive algorithm and a randomized  $(2e)$ -competitive algorithm. The algorithms of [12] and of [18] can be formally related in some settings [17].

## 9 Incremental Medians

Given an integer  $k$  and a metric space, we want to find a set  $F$  of  $k$  points called *facilities* for which the sum of distances between each point and its closest facility in  $F$  is minimized. Naturally, this can be thought of as another version of  $k$ -clustering.

In the incremental version studied by Mettu and Plaxton [32, 33], the metric space is given offline, but the number  $k$  is not specified in advance. Instead, authorizations for additional facilities arrive over time, and each time the incremental algorithm can add another facility to those chosen previously. In other words, the task is to compute an incremental sequence  $F_1 \subseteq F_2 \subseteq \dots \subseteq F_n$  of facility sets, where  $|F_k| \leq k$  for all  $k$ . The competitive ratio is the maximum, over all  $k$ , of the cost of  $F_k$  divided by the optimum  $k$ -median cost. (We stress that the term “incremental” is used here in a very different sense than in Section 8.)

This problem has recently been studied by Lin *et al* [31] and by Chrobak *et al* [15], who, independently, improved the first constant ratio of  $\approx 30$  from [32, 33] to  $8$ , in the deterministic case, and to  $2e$  in the randomized case.

Let  $d_k$  denote the optimum  $k$ -median cost and let  $F_k^*$  be the corresponding optimal set. Note that  $(d_k)$  is a monotone non-increasing sequence. Here is an algorithm for oblivious median. Given a subsequence  $d_1 = d_{i_1} \geq d_{i_2} \geq \dots \geq d_{i_p} = d_m$ , the algorithm construct the ordering backwards starting with the set  $F_n$  of all facilities and removing facilities in batches as follows. Given the current set  $F_{i_{j+1}}$  of at most  $i_{j+1}$  facilities, the algorithm defines  $F_{i_j} \subseteq F_{i_{j+1}}$  by taking, for each  $f \in F_{i_j}^*$ , the facility closest to  $f$  in  $F_{i_{j+1}}$ . The output is any ordering that starts with the facilities in  $F_{i_1}$ , continues with the additional facilities in  $F_{i_2}$ , then  $F_{i_3}$ , and so on until  $F_{i_p}$ .

For the analysis, using the triangle inequality, it is not hard to prove that the cost of  $F_{i_j}$  exceeds the cost of  $F_{i_{j+1}}$  by at most  $2d_{i_j}$ . For  $i_j \leq k < i_{j+1}$ , the cost of the distances to the first  $k$  facilities is at most the cost of distances to  $F_{i_j}$ . Hence the approximation ratio is bounded by

$$\max_k \left\{ \frac{2d_{i_j} + 2d_{i_{j+1}} + \dots + 2d_{i_m}}{d_k} : i_j \leq k < i_{j+1} \right\}. \quad (12)$$

Up to reversing the sequence  $(d_k)$  (say, by defining  $d'_k = d_{m-k}$ ), (12) is the same as the ratio (1) for the online bidding, giving a deterministic  $8$ -approximation and a randomized  $(2e)$ -approximation

algorithms.

Of course, to achieve these ratios we need to compute  $d_k$  for each  $k$  – an NP-hard problem. But instead of the optimal medians  $F_k^*$  we can use  $c$ -approximate medians in the above method; this simply increases the approximation ratio by a factor of  $c$ . In particular, since  $k$ -medians can be approximated with ratio  $(3 + \epsilon)$  in polynomial time [2], we obtain polynomial time algorithms with ratio  $24 + \epsilon$  (deterministic) and  $6e + \epsilon$  (randomized).

With a somewhat more sophisticated approach, [31] were able to improve the ratio to 16 in deterministic polynomial time. They also provide a number of extensions to other related clustering-like problems, as well as a generalization of their approach in terms of searching in partially ordered sets. Other versions of incremental  $k$ -medians are considered in [15].

## References

- [1] E. Anderson and C. Potts. Online scheduling of a single machine machine to minimize total weighted completion time. *Mathematics of Operations Research*, 29:686–697, 2004.
- [2] A. Archer, R. Rajagopalan, and D. Shmoys. Lagrangian relaxation for the  $k$ -median problem: new insights and continuity properties. In *Proc. 11th European Symp. on Algorithms (ESA)*, pages 31–42, 2003.
- [3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *J. ACM*, 44:486–504, 1997.
- [4] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Inform. and Comput.*, 106:234–252, 1993.
- [5] A. Bar-Noy, A. Freund, and J. Naor. New algorithms for related machines with temporary jobs. *J. Sched.*, 3:259–272, 2000.
- [6] A. Beck. On the linear search problem. *Naval Research Logistics Quarterly*, 2:221–228, 1964.
- [7] R. Bellman. An optimal search problem. *SIAM Review*, 5:274, 1963.
- [8] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35:108–121, 2000.
- [9] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proc. 26th Symp. Theory of Computing (STOC)*, pages 163–171. ACM, 1994.
- [10] S. Chakrabarti and S. Muthukrishnan. Resource scheduling for parallel databases and scientific computing. In *Proc. 8th Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 329–335, 1996.
- [11] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In *Proc. 23rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1099 of *Lecture Notes in Comput. Sci.*, pages 646–657. Springer, 1996.
- [12] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proc. 29th Symp. Theory of Computing (STOC)*, pages 626–635. ACM, 1997.
- [13] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proc. 44th Symp. Foundations of Computer Science (FOCS)*, pages 36–45, 2003.

- [14] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. In *Proc. 8th Symp. on Discrete Algorithms (SODA)*, pages 609–618. ACM/SIAM, 1997.
- [15] M. Chrobak, C. Kenyon, J. Noga, and N. Young. Online medians via online bidding. In *Proc. 7th Latin American Theoretical Informatics Symp. (LATIN)*, volume 3887 of *Lecture Notes in Comput. Sci.*, pages 311–322, 2006.
- [16] F. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *J. Algorithms*, 30:323–343, 1999.
- [17] A. Das and C. Kenyon. On hierarchical diameter-clustering and the supplier problems. In *Proc. WAOA'06, 4th Workshop on Approximation and Online Algorithms, September 2006, Zurich, Switzerland, LNCS, Springer*, 2006.
- [18] S. Dasgupta and P. Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555–569, 2005.
- [19] E. D. Demaine, S. P. Fekete, and S. Gal. Online searching with turn cost, 2004.
- [20] T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 199–210. Springer, 2004.
- [21] T. Ebenlendr, J. Sgall, and W. Jawor. Preemptive online scheduling: Optimal algorithms for all speeds. In *Proc. 13th European Symp. on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Comput. Sci.*, pages 327–339. Springer, 2006.
- [22] S. Gal. *Search Games*. Academic Press, 1980.
- [23] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proc. 7th Symp. on Discrete Algorithms (SODA)*, pages 152–158. ACM/SIAM, 1996.
- [24] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82:111–124, 1998.
- [25] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Comput. Sci.*, 38:293–306, 1985.
- [26] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45:1563–1581, 1966.
- [27] L. Hall, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proc. 7th Symp. on Discrete Algorithms (SODA)*, pages 142–151. ACM/SIAM, 1996.
- [28] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22:513–544, 1997.
- [29] J. A. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proc. 5th Conf. Integer Programming and Combinatorial Optimization (IPCO)*, volume 1084 of *Lecture Notes in Comput. Sci.*, pages 404–414. Springer, 1996.
- [30] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proc. 4th Symp. on Discrete Algorithms (SODA)*, pages 441–447. ACM/SIAM, 1993.
- [31] G. Lin, C. Nagarajan, R. Rajamaran, and D. Williamson. A general approach for incremental approximation and hierarchical clustering. In *Proc. 17th Symp. on Discrete Algorithms (SODA)*. ACM/SIAM, 2006.

- [32] R. R. Mettu and C. G. Plaxton. The online median problem. In *Proc. 41st Symp. Foundations of Computer Science (FOCS)*, pages 339–348. IEEE, 2000.
- [33] R. R. Mettu and C. G. Plaxton. The online median problem. *SIAM J. Comput.*, 32:816–832, 2003.
- [34] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *Proc. 4th Symp. on Discrete Algorithms (SODA)*, pages 422–431. ACM/SIAM, 1993.
- [35] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. *Theoret. Comput. Sci.*, 130:17–47, 1994.
- [36] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoret. Comput. Sci.*, 84:127–150, 1991.
- [37] R. Sitters. The minimum latency problem is NP-hard for weighted trees. In *Proc. 9th Integer Programming and Combinatorial Optimization Conference*, 2002.