The Minimum-Backlog Problem

Michael A. Bender, Sándor P. Fekete, Alexander Kröller, Vincenzo Liberatore, Joseph S. B. Mitchell, Valentin Polishchuk and Jukka Suomela

Abstract. We introduce and study the minimum-backlog problem (MBP). The MBP arises in sensor networks and is related to the classic k-server problem. It can be understood as a 2-person game played on a graph $G = (V, E)$. The "player" moves along the edges of the graph; the opponent is the "adversary." The game proceeds in timesteps. In each timestep the adversary pours a total of one unit of water into "cups" that are located on the vertices of the graph, arbitrarily distributing the water among the cups. The player then moves from her current vertex to an adjacent vertex and empties the cup at that vertex. The player's objective is to minimize the maximum amount of water (the *backlog*) in any cup at any time.

We show that the *competitive ratio* of any algorithm for the MBP has a lower bound of $\Omega(\Delta)$, where Δ is the diameter of the graph. Thus, we focus on determining a strategy for the player that guarantees a uniform upper bound on the backlog. In general graphs, the deamortization analysis of Dietz and Sleator gives a bound of $O(\Delta \ln |V|)$.

Our main result is that in *geometric* settings (e.g., sensor fields), one can obtain substantially better bounds on the maximum backlog. In particular, for a 2-dimensional n-by-n grid, we achieve a backlog of $O(n\sqrt{\ln \ln n})$, improving the $O(n \ln n)$ upper bound for general graphs, and coming close to the naive $\Omega(n)$ lower bound. Then, in a model of continuous motion of the player and continuous pouring by the adversary, for cups placed at m points in the plane we show that the backlog can be bounded by $O(D\sqrt{\ln \ln m})$, where D is the diameter of the point set. Our methods apply also to higher (fixed) dimensions.

We study also the variant of the MBP in which the adversary has a location within the graph and must act locally (filling cups) with respect to his position, just as the player acts locally (emptying cups) with respect to her position. We prove that deciding the value of this game is PSPACE-hard.

Mathematics Subject Classification (2000). Primary 68W40; Secondary 68Q25.

Keywords. Online algorithms; Competitive analysis; Computational geometry; Games on graphs.

1. Introduction

Consider a sensor network that performs motion-tracking for a set of monitored objects or agents that move within the sensor field. Each sensor acquires data about nearby objects. The total rate of data accumulation within the network remains approximately constant, assuming a relatively fixed set of objects/agents being monitored; however, the distribution of the data rate over the field is nonuniform and unpredictable. If the system is used for a field study where the data is not analyzed until the end of the experiment, it may be much more energy-efficient to store the bulk of the data locally on a memory card and let someone or something gather the data by physically visiting the sensor device (or a neighborhood of the device) [5, 11, 12, 16, 19]. Each sensor uses the wireless channel to report to the data gatherer the amount of data in its local buffer. The objective of the data gatherer is to visit the sensors in an effective order, so that no sensor's storage device is overfull. (An analogous problem arises in scheduling battery recharging/replacement in a field of sensors whose power consumption varies unpredictably with time and location.)

We are motivated to study the **minimum-backlog problem** (**MBP**), in which there is an agent moving around a domain, servicing a set of locations, "emptying" a buffer at each in an effort to make sure that no buffer gets too full. The problem is related to the k-server problem [4, 9, 10, 14, 15, 18] (with $k = 1$), in which requests are popping up at points in a metric space, and the k servers need to minimize the distance traveled to satisfy the requests. In the MBP, the objective is not to minimize the traveled distance, but to keep the backlog at each request at a low level.

Problem Formulation. We define the MBP formally as follows. We have an (unweighted) directed graph $G = (V, E)$ with an (initially empty) cup on each of the $N = |V|$ vertices. There is a **player**, who moves from vertex to vertex along the edges of the graph emptying cups, and there is an **adversary** (not located anywhere in particular), who refills the cups with water. (We talk about filling with water because of historical precedent—see the discussion below about [6].)

The following game is played for an indefinite number of timesteps. In each timestep or round, the two opponents do the following:

- The adversary pours a total of one unit of water into the cups. At each timestep, the adversary is free to distribute the unit of water in any way he likes. The adversary may base his decision on the current location of the player and current water levels in all of the cups.
- The player moves along an edge and empties the cup in its new location. The player can see the amount of water that the adversary has poured into each cup, and the player can use this information to make the decisions. The problem is online, i.e., the player does not know how the water is distributed in the future. Thus, the player's decision of which edge to traverse may be

based only on the amount of the water that has been poured into all cups so far, but not on the future distribution of water.

The *objective* of the player is to minimize the maximum amount of water in any cup at any time. The *performance* of an algorithm is the worst-case maximum amount of water in any cup at any time; the lower the performance, the better the algorithm is. Note that we are not comparing the performance of our algorithm with that of some other. Hence, the performance is not a *competitive* guarantee of any kind; rather, it gives an *absolute* bound on the objective function. In fact, as we show, the competitive ratio of any algorithm for the problem may be as bad as $\Omega(\Delta)$, where Δ is the diameter of the graph.

Cup Emptying in Computer Science. This style of problem, with a player emptying one cup at a time and an adversary distributing water among cups, is a classic problem, which has been independently discovered and rediscovered many times. However, in previous formulations, there is no issue of locality and no graph G (or, equivalently, it is assumed that $G = K_N$, the complete graph). That is, the player can empty any single cup in any timestep.

The earliest reference to cup emptying of which we are aware is the work of Dietz and Sleator [6], who used it as a technique for deamortizing data structures. Dietz and Sleator proved that if there are N cups, and the player always empties the fullest one, then no cup ever contains more than $\ln N$ water, and that this bound is optimal. The bound leads to an optimal, worst-case-constant algorithm for the order-maintenance problem. This deamortization technique has been used many times since.

Adler et al. [1] used cup emptying as a technique for analyzing scheduling algorithms. In particular, they showed how to use cup emptying to produce "fair" job schedules. Chrobak et al. [3] introduced a generalization of cup emptying and applied it to multiprocessor scheduling with conflicts between tasks. Subsequent work on the problem includes [2, 13, 17]. In the problem, the player can empty more than one cup at a time, but there is a conflict graph between cups. If two cups conflict, only one of them can be emptied at a time. Here we do have a graph (different from our graph G though), but there is still no issue of locality.

To the best of our knowledge, in all previous work on cup emptying there seems to be no issue of locality among the cups; i.e., there is no underlying graph G (or, $G = K_N$), with cups placed at its vertices, and with the constraint that the player is limited to taking steps along edges of G. In contrast, the MBP in this paper is cup emptying on graphs, particularly those with geometric structure, such as the regular n-by-n grid.

It is important to note that although the motivation for studying cup emptying in metric spaces came from online vehicle routing, the locality shows up in non-geometric contexts as well. For instance, in a job scheduling problem, there may be a (set-up) cost associated with switching from executing one job to executing another. This, in particular, makes the Traveling Salesman Problem, which

originated as a geometric problem, applicable also to scheduling tasks. Hence, although we state our problem and results in purely geometric terms, they are also relevant in some more general scheduling applications.

Competitive Analysis is Doomed. It would be natural to try to give a competitive algorithm for MBP. Unfortunately, an online algorithm with a good competitive ratio is not possible. Indeed, consider MBP in the graph G that is a path on N vertices. Suppose that the adversary picks a random permutation of the vertices 2 through $N-1$, and for the first $N-2$ timesteps pours a unit of water per step into the cups according to the permutation. Then, the adversary picks one of the endpoints of the path and pours the water there forever. The best offline strategy would be to rush to the endpoint and stay there—this yields a maximum backlog of 1. On the other hand, without prior knowledge of the "drenched" endpoint, any algorithm will put the player far from the endpoint (since the adversary may choose the endpoint that is farthest from the current player position), thus, making the performance of the algorithm $\Omega(N)$.

Given that the competitive ratio of an online algorithm for the problem may be very high, we concentrate on providing uniform upper bounds on the performance; i.e., on giving universal bounds on the amount of water in any cup at any time.

Intuition for MBP. Before giving our results, we present simple observations on the MBP. The performance of any algorithm has a naive lower bound of $\Omega(\Delta)$, where Δ, as above, is the diameter of the graph. To get another lower bound, the adversary may pick a set of K cups such that the distance between any two cups in the set is at least d, for some number d. The adversary will then pour the water evenly into never-emptied cups from the set. After dK steps, one of the cups will have $\Omega(\frac{d}{K} + \frac{d}{K-1} + \cdots + d) = \Omega(d \ln K)$ units of water.

Combining the above lower bounds, we see that the adversary can ensure $\Omega(\max(\Delta, \ln |V|))$ of water in some cup at some time. In particular, this gives an $\Omega(n)$ lower bound on the performance of any algorithm in an n-by-n grid.

As for the upper bounds, which is the main focus of the paper, a naive algorithm has performance of $O(C)$, where C is the length of the shortest closed path visiting all vertices in V. A direct application of Dietz and Sleator [6] gives an improved upper bound of $O(\Delta \ln |V|)$. For an n-by-n grid this is $O(n \ln n)$. This is the upper bound that we set out to beat in this paper.

Results. Our main result is an algorithm for the MBP in an n-by-n grid whose performance is $O(n\sqrt{\ln \ln n})$. This algorithm is based on a simpler algorithm, whose performance is $O(n\sqrt{\ln n})$. We show that the algorithms for the grid are applicable to the MBP defined on an arbitrary planar point set. In more details, our results are as follows:

- We introduce the $(k\tau, k)$-game, a generalization of the cup-emptying game. We further generalize and introduce the $(k\tau, k, T)$-game, which includes a graph G in the game (Section 3).

- We present an algorithm for the MBP having a performance of $O(n\sqrt{\ln n})$ on an n-by-n grid. We derive this algorithm from our results for the $(k\tau, k, T)$-game applied to the n-by-n grid (Section 3).
- We present our main algorithm, having a performance of $O(n\sqrt{\ln \ln n})$ on an n-by-n grid. We obtain this bound by giving separate bounds for the water of different "ages" and by showing that a better upper bound is possible in a game that lasts for only a few rounds (Section 4).
- We show that our algorithms work for a general geometric domain, achieving a performance of $O(D\sqrt{\ln \ln m})$ on a set of m cups in the plane, where D is the maximum distance between the cups (i.e., D is the diameter of the set). The extension is based on utilizing an approximating grid, but then observing that the performance does not depend on the grid size (Section 5). Our methods apply also to higher dimensional grids and point sets; the generalization is deferred to the full paper.
- We prove that deciding the value of the game is PSPACE-hard (Section 6) in the case when the adversary is also constrained to local actions (pourings) and must move on the graph G, just like the player.

2. Preliminaries

We give basic observations about cup emptying. As mentioned earlier, previous works studied cup emptying with no notion of locality — effectively, when the graph $G = K_N$ is complete. We explain that in such a setting, emptying the fullest cup is a good strategy. In contrast, once more general graphs enter the game, emptying the fullest cup is no longer close to optimal, as we show.

Cup Lemma. Intuitively, always emptying the fullest cup is optimal when there is no graph G (or equivalently, G is complete), and a simple exchange argument validates this intuition.

Lemma 2.1. *The player's strategy of emptying the fullest cup is optimal when there is no graph G.*

Dietz and Sleator [6] analyze the performance of the strategy.

Lemma 2.2 (Cup Lemma [6, Theorem 5]**).** *In the cup emptying game with N cups, if the player always empties the fullest cup, then no cup ever contains more than $\ln N$ units of water.*

The strategy of emptying the fullest cup is far less successful in geometric settings and general graphs. On an n-by-n grid, the empty-the-fullest strategy exactly matches the upper bound given by a direct application of Lemma 2.2 but does no better. (We still seem to have an unsuccessful strategy even if we "weight" the cups based on the water level and distance to the player, and empty the cup with highest weight.) In particular, we show the following performance (the proof is deferred to the full version).

Lemma 2.3. *In the cup emptying game on an n-by-n grid, if the player always empties the fullest cup, then there will be times when the fullest cup contains $\Theta(n \ln n)$ units of water.*

Despite the limitations of the Cup Lemma on graphs, it will serve as the starting point for analyzing all of our algorithms.

The Cup Game is Fully Determined by the Pouring Sequence. Suppose that we know the pouring sequence up to some time t. Then, by Lemma 2.1, we also know which cup an optimal player empties in each timestep until t (up to a renumbering of cups with an equal level of water).

We formalize this observation with the following notation. Let $M_{i,t}$ be the amount of water that the adversary pours into cup i in timestep t. We call M the **pouring matrix**. Let $X_{i,t}$ be the *level* of water in cup i at timestep t when the player always empties the fullest cup. We call X the **level matrix**.

Observation 1. *The first t columns of the level matrix, and hence — the cup emptied at t, are fully determined by the first t columns of the pouring matrix (up to renumbering the cups with an equal level of water).*

3. First Algorithm for Grids: $O(n\sqrt{\ln n})$ Water Level

In order to obtain an algorithm with a performance of $O(n\sqrt{\ln n})$, we introduce and analyze generalizations of the cup game. We then apply our analysis to the MBP in an n-by-n grid.

The $(k\tau, k)$ Cup Game. We now define the $(k\tau, k)$-game. (There is no graph G yet, or $G = K_N$.) Our strategy for this game is used as a subroutine in all subsequent algorithms. For $\tau, k \in \mathbb{N}$, the **$(k\tau, k)$-game** has a player (whom we call the **$(k\tau, k)$-player**), who sits still for $k\tau$ timesteps and then empties k cups. For instance, in the $(\tau, 1)$-game, the $(\tau, 1)$-player empties 1 cup every τ timesteps.

Our algorithm for the $(k\tau, k)$-game proceeds as follows. Run an imaginary $(\tau, 1)$-game with the same pouring matrix as in the $(k\tau, k)$-game. Divide time into **epochs** of length $k\tau$. For each epoch, determine which k cups would be emptied by the imaginary $(\tau, 1)$-player. Empty these k cups in the $(k\tau, k)$-game at the end of the epoch (Fig. 1). (In fact, we might empty fewer than k cups because the imaginary player may empty the same cup multiple times during the epoch.)

Observe that the $(\tau, 1)$-game can be easily analyzed by generalizing the Cup Lemma.

Corollary 3.1. *In the $(\tau, 1)$-game with N cups, there exists a strategy (in particular, empty the fullest) ensuring that at any time, each cup contains at most $\tau \ln N$ units of water.*

Proof. The original cup game is, in fact, a $(1, 1)$-game. The $(\tau, 1)$-game may be viewed as a $(1, 1)$-game, in which the amount of water poured at any timestep is scaled up by τ. □

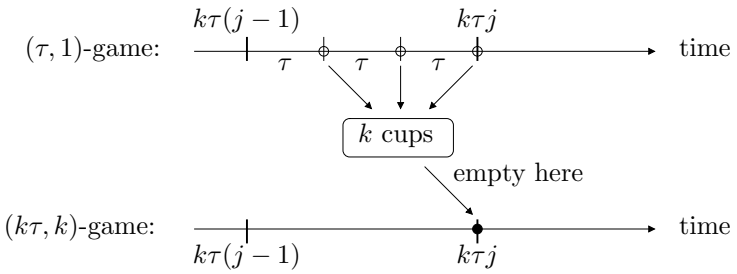

FIGURE 1. Using the $(\tau, 1)$-game to deduce the k cups to empty in the $(k\tau, k)$-game.

We now analyze the performance of our algorithm for the $(k\tau, k)$-game.

Lemma 3.2. *There exists a strategy for the $(k\tau, k)$-game on N cups such that at all times no cup contains more than $\tau \ln N + k\tau$ units of water.*

Proof. Let M be the pouring matrix in the $(k\tau, k)$-game. For any $j = 1, 2, \ldots,$ by the end of the jth epoch, i.e., by time $k\tau j$, the $(k\tau, k)$-player knows the first $k\tau j$ columns of M. Thus, by Observation 1, the $(k\tau, k)$-player knows which k cups are emptied by the imaginary $(\tau, 1)$-player in the jth epoch. By our strategy, the $(k\tau, k)$-player empties these k cups at her jth move (see Fig. 1). Thus,

Observation 2. *At the end of any epoch j, i.e., at time $k\tau j$, any cup in the $(k\tau, k)$-game is at most as full as it is in the $(\tau, 1)$-game.*

By Observation 2 and Corollary 3.1, at time $k\tau j$ (the *end* of the jth epoch) the amount of water in any cup is not greater than $\tau \ln N$. The proof of the lemma concludes with the following observation about water accumulated between ends of epochs.

Observation 3. *Between the times $k\tau(j-1)+1$ and $k\tau j$ at most $k\tau$ units of water can be added to any cup.*

□

We call the imaginary $(\tau, 1)$-game, played to deduce the cups to empty in the $(k\tau, k)$-game, **attached** to the $(k\tau, k)$-game.

The $(k\tau, k, T)$-Game on Graph G. As a further generalization, we now introduce the graph $G = (V, E)$ into the cup game. Define $T(k)$ to be the smallest integer such that for any set of $k+1$ vertices, there exists a closed path of length at most $T(k)$ that visits all $k+1$ vertices in the set. In other words, starting at any vertex we can empty k cups and return back to our starting point in $T(k)$ steps.

The **$(k\tau, k, T)$-game** on G is defined provided $k\tau \geq T(k)$, as follows. As before, divide time into epochs of length $k\tau$. During any epoch, the **$(k\tau, k, T)$-player** spends the first $T(k)$ timesteps moving around the graph to empty k cups. Then the player sits still for $k\tau - T(k)$ steps until the end of the epoch. Meanwhile, the adversary pours one unit of water into the cups during each timestep.

Our algorithm for the $(k\tau, k, T)$-game on G proceeds as follows. Run an imaginary $(k\tau, k)$-game with the same pouring matrix as in the $(k\tau, k, T)$-game. For each epoch, determine which k cups would be emptied by the imaginary $(k\tau, k)$-player. Empty these k cups in the $(k\tau, k, T)$-game at the beginning of the *next* epoch. Since by definition $T(k) \leq k\tau$, there is enough time for the player to empty the k cups.

Theorem 3.3. *There exists a strategy for the $(k\tau, k, T)$-player such that at all times no cup contains more than $\tau \ln |V| + k\tau + T(k)$ units of water.*

Proof. Similarly to Observation 2, we have:

Observation 4. *In each epoch j, after $T(k)$ timesteps from the beginning of the epoch, i.e., at time $k\tau j + T(k)$, any cup in the $(k\tau, k, T)$-game contains at most the same amount of water as in the $(k\tau, k)$-game at the beginning of the epoch, i.e., at time $k\tau j$, plus the amount of water that has arrived during the $T(k)$ timesteps.*

By Corollary 3.1, at the beginning of the epoch the amount of water in any cup is not greater than $\tau \ln |V|$. The proof of the lemma concludes with the following observation.

Observation 5. *Between the times $k\tau(j-1)+1$ and $k\tau(j-1)+T(k)$ at most $T(k)$ units of water may be added to any cup, and between the times $k\tau(j-1)+T(k)+1$ and $k\tau j + T(k)$ at most $k\tau$ units of water may be added to any cup.* □

MBP on an n-by-n Grid. We can finally turn our attention to the MBP when G is an n-by-n grid. For a grid, $T(k)$ is as follows.

Lemma 3.4 ([8]). *In an n-by-n grid, $T(k) \leq 5n\sqrt{k}$.*

We now apply Theorem 3.3.

Theorem 3.5. *There exists a strategy for MBP on an n-by-n grid such that the amount of water in any cup is $O(n\sqrt{\ln n})$.*

Proof. Choose $k = \Theta(\ln n)$ and $\tau = \Theta(n/\sqrt{\ln n})$. Then, by Lemma 3.4, $T(k) = O(n\sqrt{\ln n})$, so Theorem 3.3 implies an upper bound of $\tau \ln n + k\tau T(k) = O(n\sqrt{\ln n})$ on the amount of water ever at a vertex. □

4. Refined Algorithm for Grids: $O(n\sqrt{\ln \ln n})$ Water Level

We improve the performance of the algorithm from the previous section by applying several observations. First, we observe that the analysis of Dietz and Sleator [6] can be extended to show that if the $(1, 1)$-game is played for $r \leq N$ timesteps, the empty-the-fullest strategy guarantees $O(\ln r)$ units of water in any cup at any time. Second, we note that if in a $(k\tau, k, T)$-game only "old" water is counted (i.e., water from two or more epochs previously), an additive term can be removed from the upper bound in Theorem 3.3. Finally, we split the time $T(k)$, used by the

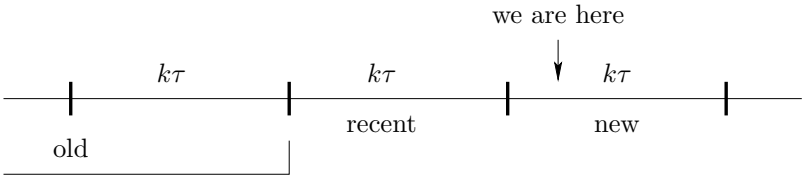

FIGURE 2. Water classes in $(k\tau, k, T)$-game.

$(k\tau, k, T)$-player to empty k cups, into several subintervals. We use the time between the subintervals to clear "new" water. Altogether this gives an $O(n\sqrt{\ln \ln n})$ upper bound on the amount of water in the MBP on an n-by-n grid.

"Aging" of Water. The upper bound on the amount of water in the $(k\tau, k)$-game (Lemma 3.2) consists of two terms. The first term (Observation 2) bounds the amount of water that arrived *before* the current epoch. The second term (Observation 3) bounds the amount of water that arrived *during* the current epoch. We capture this fact in the following definitions and lemma (Fig. 2).

Definition 4.1. The water that arrived during the current epoch in a $(k\tau, k)$-game is called **new**. The water that arrived in previous epochs is called **old**.

Similarly, in the $(k\tau, k, T)$-game, we may define:

Definition 4.2. The water that arrived during the current epoch in a $(k\tau, k, T)$-game is **new**. The water that arrived in the previous epoch is **recent**. The water that arrived earlier is **old**.

From the above definitions and Observations 2–5 we have:

Lemma 4.3. *Both in the $(k\tau, k)$-game and in the $(k\tau, k, T)$-game the amount of* old *water in any cup at any time is at most $\tau \ln N$.*

Playing for Less Than N Rounds in the $(\tau, 1)$-Game. The Cup Lemma gives an upper bound of $\ln N$ on the water level in a $(1, 1)$-game played for an *indefinite* number of rounds. Following the original proof of the Lemma, we show that if a game is played for a small number r of rounds, the upper bound may be improved to $\ln r$. We use the following improved result to bound the level of new water in our refined algorithm.

Lemma 4.4. *Let the $(\tau, 1)$-game on N cups be played for $r \le N$ timesteps. Let the player follow the empty-the-fullest strategy. Then the amount of water in any cup is $O(\tau \ln r)$.*

Proof. We follow the proof of Lemma 2.2 due to Dietz and Sleator [6]. For $j = 1 \ldots r$, $i = 1 \ldots N$, let $X_{(i)j}$ be the order statistics of X^j — the jth column of X; $X_{(i)j}$ is the ith highest water level at time j. Let S_j be the sum of the $r - j + 1$

largest elements in X^j:

$$S_j = \sum_{i=1}^{r-j+1} X_{(i)j} \,.$$

Then, after emptying the fullest cup and adding at most 1 unit of water to the $r-j$ cups, fullest at time $j+1$,

$$S_{j+1} \leq \sum_{i=2}^{r-j+1} X_{(i)j} + 1 = 1 + S_j - X_{(1)j} \leq 1 + S_j - \frac{S_j}{r-j+1}$$

which can be rewritten as

$$S_{j+1} \leq (r-j)\left(\frac{S_j}{r-j+1} + \frac{1}{r-j}\right)$$

leading to

$$S_{j+1} \leq (r-j)\left(\frac{S_1}{r} + \frac{1}{r-1} + \cdots + \frac{1}{r-j}\right) .$$

Substituting $j = r-1$ and noting that $S_1 = 0$,

$$S_r \leq \frac{1}{r-1} + \cdots + 1 \,.$$

□

Similarly to Corollary 3.1,

Corollary 4.5. *Let $(\tau, 1)$-game on $|V|$ cups be played for $r \leq |V|$ rounds. Let the player follow the* Empty-the-Fullest *strategy. Then the amount of water in any cup is $O(\tau \ln r)$.*

Playing for Less Than $|V|/k$ Epochs in the $(k\tau, k, T)$-Game. For $r \in \mathbb{N}$, $r \leq |V|/k$, let the **$(k\tau, k, T)^r$-game** be the $(k\tau, k, T)$-game played for r epochs.

Lemma 4.6. *There exists a strategy for the player in a $(k\tau, k, T)^r$-game such that the amount of* old *water in any cup at any time is at most $\tau \ln kr$.*

Proof. We expand upon the analysis from Theorem 3.3. Since we are only counting old water, only the first term remains in the upper bound in the theorem. The amount of old water in the $(k\tau, k, T)^r$-game is at most the amount of old water in the imaginary $(k\tau, k)$-game played for the same number, r, of epochs (observe the difference in the definition of old water for $(k\tau, k)$- and $(k\tau, k, T)$-games, Definitions 4.1 and 4.2). Finally, if the $(k\tau, k)$-game is played for only r epochs, the amount of water in the attached imaginary $(\tau, 1)$-game is at most $\tau \ln kr$ (by Lemma 4.4). □

Observe that the bound in the above lemma is independent of $|V|$.

Corollary 4.7. *There exists a strategy for the player in a $(k\tau, k, T)^r$-game such that the amount of water in any cup at any time is at most $\tau \ln kr + 2k\tau$.*

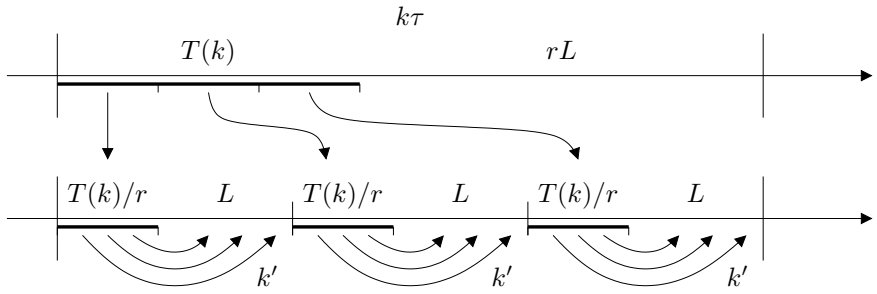

FIGURE 3. Split $T(k)$. Play $(k\tau, k, T)$-game emptying k cups during the intervals of total length $T(k)$. Within each epoch play $(T/r + L, k', T)^r$-game emptying k' cups during the intervals of length L.

Proof. The amount of old water is at most $\tau \ln kr$ by Lemma 4.6. The total amount of recent water in any cup is $k\tau$ and the total amount of new water is at most $k\tau$. □

Splitting $T(k)$. In the $(k\tau, k, T)$-game the player was given a *contiguous* interval of time, of length $T(k)$, to empty k cups. There is nothing wrong in *splitting* the interval into smaller subintervals and spreading the subintervals along the epoch. Instead of emptying the k cups in "one shot" the player would empty some cups during the first subinterval, have a rest, then empty some other cups during the second one, have a rest, etc.

In particular, we may pick $r \in \mathbb{N}$, split $T(k)$ into r *equal-length* subintervals, and spread the subintervals evenly along the epoch. During the subintervals, k cups may be emptied ensuring at most $\tau \ln |V|$ of *old* water in any cup at any time (Fig. 3).

Each of the r *gaps* between the subintervals will have length $L = (k\tau - T(k))/r$. For $k' \in \mathbb{N}$, such that $L \geq T(k')$, during a gap the player may empty any k' cups. This means that a $(T/r + L, k', T)$-game (equivalently, $(k\tau/r, k', T)$-game) may be defined. We will play the game with the *new* water.

To summarize (refer to Fig. 3), we divide the time into epochs of length $k\tau = T(k) + rL$. We play the $(k\tau, k, T)$-game (with the old water) emptying k cups during the total-length-$T(k)$ periods. We also divide each epoch into r periods of length $k\tau/r = T(k)/r + L$. We play the $(k\tau/r, k', T)$-game (with the water that is classified as new in the above $(k\tau, k, T)$-game) emptying k' cups during the length-L segments. The $(k\tau/r, k', T)$-game (which, equivalently, is a $(k'\frac{k\tau}{rk'}, k', T)$-game) "resets" every r periods, as a different epoch from the $(k\tau, k, T)$-game is entered. Thus, with the new water a $(k'\frac{k\tau}{rk'}, k', T)^r$-game is played.

By Lemma 4.3 the amount of old water is at most $\tau \ln |V|$. By Corollary 4.7 the amount of new water is at most $\frac{k\tau}{rk'} \ln k'r + 2k\tau/r$. Finally, recent water was new

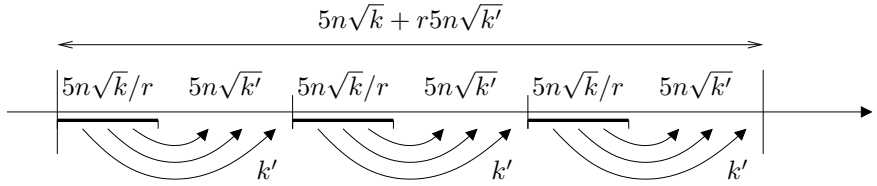

FIGURE 4. $k\tau = 5n\sqrt{k} + r5n\sqrt{k'}$. Play $(k\tau, k, 5n\sqrt{k})$-game with the old water and $(k'\frac{k\tau}{k'r}, k', 5n\sqrt{k'})^r$-game with the new water.

in the previous time segment, and thus, its amount is also at most $\frac{k\tau}{rk'} \ln rk' + 2k\tau/r$. Overall, we have:

Lemma 4.8. *Let k', r be such that $T(k') \le (k\tau - T(k))/r$. Then in a $(k\tau, k, T)$-game the player has a strategy with performance $\tau \ln |V| + (2k\tau \ln rk')/(rk') + 4k\tau/r$.*

MBP on an n-by-n Grid. Suppose that k, τ, r, k' are such that $k\tau = 5n\sqrt{k} + r5n\sqrt{k'}$ (see Fig. 4). By Few's result (see Lemma 3.4), a $(k\tau, k, 5n\sqrt{k})$-game is well defined in an n-by-n grid. Note that since $T(k) \le 5n\sqrt{k}$, we have $(k\tau - T(k))/r \ge 5n\sqrt{k'} \ge T(k')$. Thus, Lemma 4.8 can be applied to conclude that there is a strategy for a player in an n-by-n grid with performance $\tau \ln n^2 + (2k\tau \ln rk')/(rk') + (4k\tau)/r$. Substituting $\tau = (5n\sqrt{k} + r5n\sqrt{k'})/k$, we obtain

Lemma 4.9. *For any k, k', and r, there exists a strategy for a player in an n-by-n grid with performance $10n((\ln n)/\sqrt{k} + (r\sqrt{k'} \ln n)/k + (\sqrt{k} \ln rk')/(rk') + (\ln rk')/\sqrt{k'} + 2\sqrt{k}/r + 2\sqrt{k'})$.*

We are now ready to give our main result:

Theorem 4.10. *There exists a strategy for MBP on an n-by-n grid with a performance of $O(n\sqrt{\ln \ln n})$.*

Proof. Choose $k = (\ln^2 n)/(\ln \ln n)$, $k' = \ln \ln n$, $r = (\ln n)/(\ln \ln n)$ and apply Lemma 4.9. □

5. General Geometric Domains

In this section we consider the MBP in which the cups are installed at a set P of m points in the plane. The game no longer proceeds in discrete time steps. Instead, the player may move continuously in the plane with maximum speed of 1. The adversary pours the water also continuously; the maximum amount of water poured during any length-t time interval is t. The adversary is free to choose how to distribute the water among the cups.

Let D be the diameter of set P; i.e., D is the maximum distance between two points of P. By pouring into the diametrical points of the set, the adversary can ensure a water level of $\Omega(D)$ at one of the points. We show that there is a strategy for the player to ensure $O(D\sqrt{\ln \ln m})$ water in any cup at any time.

Our result is based on a generalization of Theorem 4.10. Specifically, suppose that in a standard (discrete-time) MBP on an n-by-n grid, the adversary may pour water not into all n^2 cups, but only into a subset S of them. Following the analysis of the algorithm of Section 4, we get the following corollary of Theorem 4.10:

Corollary 5.1. *There exists a strategy for MBP on a subset S of grid points with a performance of $O(\Delta\sqrt{\ln\ln|S|})$, where Δ is the diameter of the grid.*

We use the above corollary to get

Theorem 5.2. *Let P be a set of m points in the plane. There exists a strategy for the player, with the performance $O(D\sqrt{\ln\ln m})$, where D is the diameter of P.*

Proof. (Sketch.) By laying down a fine enough grid, we may assure that the points of P are at the nodes of the grid. As the fineness of the grid grows, the game tends to the MBP on a grid. The difference though, is that the adversary may now pour water only into the subset of nodes, in which the cups from P are situated. From Corollary 5.1, the performance of the player is $O(D\sqrt{\ln\ln m})$. The theorem follows from the fact that the performance is independent of the grid size. □

Remark. In the above theorem we only prove the *existence* of the strategy; we do not discuss the *running time* of an implementation of it. In the full version we show that it is enough to lay a grid of polynomial size and snap P onto the grid. The snapping does not increase the length of a tour by much, and we can tolerate the increase by taking a larger constant in Few's result (Lemma 3.4).

6. PSPACE-Hardness for a Localized Adversary

In this section we consider the MBP on (directed) graphs with a localized adversary. More specifically, we consider the following game, which we call the *localized* MBP:

- The game is played on a directed graph; each vertex carries some integer load (= water level).
- At each time step, both player and adversary are located at some vertices, beginning at (distinct) starting positions.
- Both participants take turns in moving from their respective current position along an outgoing edge to an adjacent node. (They are not allowed to stay in place.)
- A move by the player ends with her removing the load from the vertex she reached.
- A move by the adversary ends with him increasing the load on the vertex he reached by one unit.
- The game ends when the player steps onto the vertex currently occupied by the adversary, or when the adversary manages to get the load on some vertex to a pre-specified target value.

- The player wins if she can keep the adversary from reaching the target value; the adversary wins if he can reach the target value.

As it turns out, this version is already quite difficult when it comes to computing the value of the game for just the smallest nontrivial target value of 2:

Theorem 6.1. *The localized MBP is PSPACE-hard, even for a target value of 2.*

Proof. We present a reduction from Quantified 3SAT (Q3SAT), where the Boolean formula F, containing m clauses $c_1, c_2, \ldots, c_m$ and n variables $x_1, x_2, \ldots, x_n$, is in conjunctive normal form with 3 literals per clause; without loss of generality, we assume that n is even. A Q3SAT instance I_F asks for the truthfulness of the expression $\forall x_1 \exists x_2 \forall x_3 \ldots \exists x_n : c_1 \wedge c_2 \wedge \ldots \wedge c_m$. It is helpful to think of this as a game between two players who take turns at setting the variables in ascending order of indices; the first player tries to set the odd variables in a way that will keep F from being true, while the second player (adversary) sets the even variables in a way that aims at F ending up satisfied.

Now we construct an instance of the localized MBP by specifying the digraph $D = (V, A)$ on which it is played; for more details of a somewhat related construction, see [7]. The initial vertices for player and adversary are u_{-1} and u_0, respectively, and the player starts the game. We use the vertex set $V = \{x_i, \overline{x}_i, u_i : 1 \le i \le n\} \cup \{u_{-1}, u_0\} \cup \{c_j, \overline{c}_j, d_j : 1 \le j \le m\}$ and the edge set

$$\begin{aligned} A = {} & \{(x_i, u_i), (\overline{x}_i, u_i) : 1 \le i \le n\} \cup \{(u_i, x_{i+2}), (u_i, \overline{x}_{i+2}) : -1 \le i \le n-2\} \\ & \cup \{(u_n, c_j), (u_{n-1}, \overline{c}_j), (\overline{c}_j, d_j) : 1 \le j \le m\} \cup \{(\overline{c}_j, c_k) : 1 \le j \le m, k \ne j\} \\ & \cup \{(c_j, x_i), (d_j, x_i) : \text{iff } c_j \text{ contains } x_i,\ 1 \le j \le m,\ 1 \le i \le n\} \\ & \cup \{(c_j, \overline{x}_i), (d_j, \overline{x}_i) : \text{iff } c_j \text{ contains } \overline{x}_i,\ 1 \le j \le m,\ 1 \le i \le n\}. \end{aligned}$$

We single out the subset $V_C = \{x_{2i-1}, \overline{x}_{2i-1} : 1 \le i \le n/2\} \subset V$ that start with an initial load of one; all other vertices start with an initial load of zero. Note that $|V| = O(m+n)$, $|V_C| = O(n)$, $|A| = O(m^2+n)$, so the construction is clearly polynomial. The construction is illustrated in Figs. 5 and 6.

Now consider the game on D. For easier reference, we denote by *diamonds* the subgraphs induced by $(u_{i-1}, x_i, \overline{x}_i, u_i)$. The players traverse the diamonds according to their chosen truth assignments in the given instance of Q3SAT, i.e., the adversary traverses x_i if $x_i = 1$, otherwise traversing $\overline{x}_i$; analogously, the player traverses $\overline{x}_i$ if $x_i = 1$, otherwise x_i; obviously, both participants are forced to move this way, implying a corresponding truth assignment. We argue in the following that the adversary wins if and only if all clauses are satisfied.

After arriving at vertex u_{n-1}, the player selects a clause c_j by moving to the corresponding clause selector vertex $\overline{c}_j$. This forces the adversary to move by (u_n, c_j) in order to avoid being caught. As the player has no way of catching the adversary in her next move, the adversary wins if the clause vertex c_j is adjacent to a variable vertex with load one, i.e., the corresponding variable setting satisfies the clause. On the other hand, the player can prevent the adversary from reaching

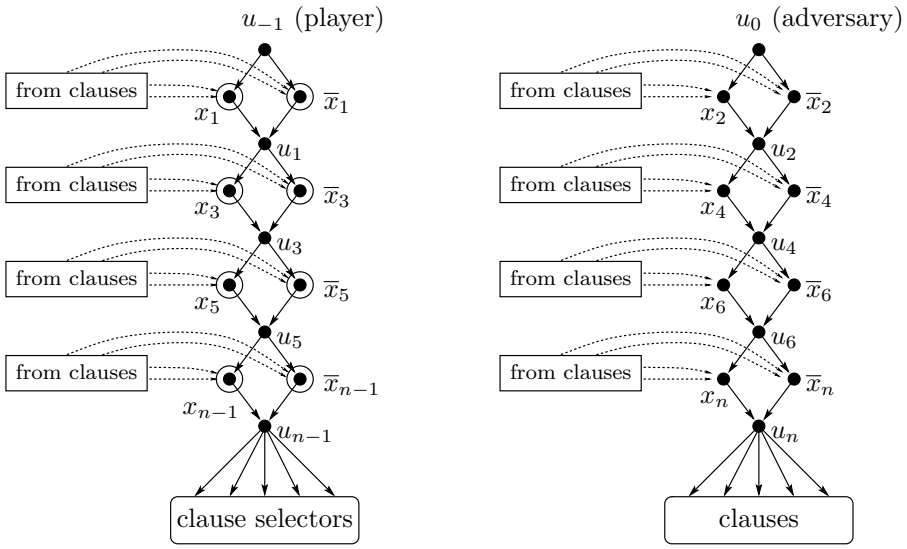

FIGURE 5. The variable gadget: The player chooses a truth setting for the odd variables by running from u_{-1} to u_{n-1}, while the adversary chooses a truth setting for the even variables by running from u_0 to u_n. Note that initially, the odd-numbered vertices carry a load of 1 (indicated by circles), while all other vertices start out with a load of 0.

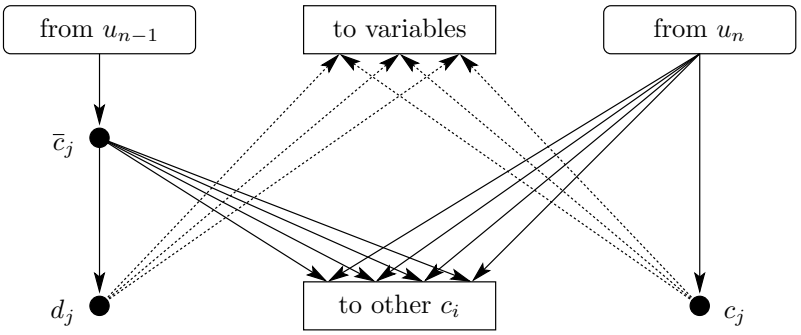

FIGURE 6. A clause gadget: The player picks a clause by moving to a clause selector vertex $\overline{c}_j$, which is connected to all clause nodes c_k for $k \neq j$. This forces the adversary to move to c_j in order to avoid being caught prematurely. Then the player moves to d_j, catching the adversary after he moves to one of the three variable vertices corresponding to the clause c_j. The adversary wins iff that vertex had already carries a load of 1, i.e., if the corresponding variable satisfies the clause.

a load of two if the clause is unsatisfied, by moving to vertex d_j, assuring himself of catching the adversary in her next move.

This shows that the player wins iff there is an unsatisfied clause. □

Acknowledgments. We gratefully acknowledge Gerhard Woeginger for discussions that lead to the formulation of this problem. We thank Estie Arkin, Leonidas Guibas, Patrik Floréen and Petteri Kaski for many helpful discussions.

MAB is supported in part by NSF Grants CCF-0621439/0621425, CCF-0540897/05414009, CCF-0634793/0632838, and CNS-0627645. AK is supported by DFG Grants FE407/9-1 and FE407/9-2. VL is supported in part by NSF Grants CCR-0329910, Department of Commerce TOP 39-60-04003, Department of Energy DE-FC26-06NT42853, and the Wright Center for Sensor Systems Engineering. JSBM is supported in part by the U.S.-Israel Binational Science Foundation (2000160), NASA (NAG2-1620), NSF (CCF-0528209, ACI-0328930, CCF-0431030), and Metron Aviation. JS is supported in part by the Academy of Finland, Grant 116547, and by Helsinki Graduate School in Computer Science and Engineering (Hecse).

References

[1] M. Adler, P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, and M. Paterson. A proportionate fair scheduling rule with good worst-case performance. In *Proceedings of SPAA*, pages 101–108, 2003.

[2] A. Bar-Noy, A. Freund, S. Landa, and J. S. Naor. Competitive on-line switching policies. In *Proceedings of SODA*, pages 525–534, 2002.

[3] M. Chrobak, J. Csirik, C. Imreh, J. Noga, J. Sgall, and G. J. Woeginger. The buffer minimization problem for multiprocessor scheduling with conflicts. In *Proceedings of ICALP*, pages 862–874, 2001.

[4] M. Chrobak and L. L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.

[5] Y. Diao, D. Ganesan, G. Mathur, and P. Shenoy. Rethinking data management for storage-centric sensor networks. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR)*, 2007.

[6] P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *Proceedings of STOC*, pages 365–372, 1987.

[7] S. P. Fekete, R. Fleischer, A. Fraenkel, and M. Schmitt. Traveling salesmen in the presence of competition. *Theoretical Computer Science*, 313:377–392, 2004.

[8] L. Few. The shortest path and the shortest road through n points. *Mathematika*, 2:141–144, 1955.

[9] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. In *Proceedings of FOCS*, pages 454–463, 1990.

[10] A. Floratos and R. Boppana. The on-line k-server problem. Technical Report TR1997-732, NYU, Computer Science Department, 1997.

[11] Y. Gu, D. Bozdağ, R. W. Brewer, and E. Ekici. Data harvesting with mobile elements in wireless sensor networks. *Computer Networks*, 50(17):3449–3465, 2006.

[12] D. Jea, A. Somasundara, and M. Srivastava. Multiple controlled mobile elements (data mules) for data collection in sensor networks. In *Proceedings of the First IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 244–257, 2005.

[13] H. Koga. Balanced scheduling toward loss-free packet queuing and delay fairness. In *Proceedings of ISAAC*, pages 61–73, 2001.

[14] E. Koutsoupias and C. H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[15] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[16] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks (IPSN)*, pages 374–381, 2006.

[17] G. Rote. Pursuit-evasion with imprecise target location. In *Proceedings of SODA*, pages 747–753, 2003.

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

[19] A. A. Somasundara, A. Ramamoorthy, and M. B. Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS)*, pages 296–305, 2004.

Michael A. Bender
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400
USA
e-mail: `bender@cs.sunysb.edu`

Sándor P. Fekete
Department of Computer Science
Braunschweig University of Technology
Pockelsstr. 14
D-38106 Braunschweig
Germany
e-mail: `s.fekete@tu-bs.de`

Alexander Kröller
Department of Computer Science
Braunschweig University of Technology
Pockelsstr. 14
D-38106 Braunschweig
Germany
e-mail: `a.kroeller@tu-bs.de`

Vincenzo Liberatore
Department of Computer Science
Case Western Reserve University
10900 Euclid Avenue
Cleveland, Ohio 44106-7071
USA
e-mail: vl@case.edu

Joseph S. B. Mitchell
Department of Applied Math and Statistics
Stony Brook University
Stony Brook, NY 11794-3600
USA
e-mail: jsbm@ams.sunysb.edu

Valentin Polishchuk
Corresponding author
Helsinki Institute for Information Technology HIIT
Department of Computer Science
University of Helsinki
P. O. Box 68
FI-00014 University of Helsinki
Finland
e-mail: valentin.polsichuk@cs.helsinki.fi

Jukka Suomela
Helsinki Institute for Information Technology HIIT
Department of Computer Science
University of Helsinki
P. O. Box 68
FI-00014 University of Helsinki
Finland
e-mail: jukka.suomela@cs.helsinki.fi