Paging Mobile Users in Cellular Networks: Optimality versus Complexity and Simplicity

Amotz Bar-Noy[a], Panagiotis Cheilaris[b,*], Yi Feng[a], Mordecai J. Golin[c]

[a]*Department of Computer Science, City University of New York, 365 Fifth Avenue, New York, NY 10016, United States*
[b]*Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland*
[c]*Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong*

Abstract

A mobile user is roaming in a zone composed of many cells in a cellular network system. When a call arrives, the system pages the user in these cells since the user never reports its location unless it leaves the zone. Each cell is associated with a positive value which is the probability that the user resides in this cell. A delay constraint requires the user to be found within a predetermined number of paging rounds where in each round a subset of the cells is paged. The goal is to design a paging strategy that minimizes the expected number of paged cells until the user is found. Optimal solutions based on dynamic programming are known. The running time of former implementations is quadratic in the number of cells and linear in the number of rounds. We introduce two implementations whose running times are also linear in the number of cells, by proving that the dynamic programming formulation satisfies properties (like the Monge property) that enable us to use various dynamic programming speed-up techniques. We also propose a new heuristic of almost linear complexity that outperforms a known linear complexity heuristic while running faster when the number of rounds is far less than the number of cells. Our comprehensive simulations compare the non-optimal heuristics with the optimal solutions, demonstrating the trade-off between optimality and running time efficiency as well as implementation simplicity.

Keywords: design and analysis of algorithms, partitioning and scheduling

1. Introduction

Cellular phone systems make it possible to talk with people even if they are not residing in predetermined locations. These systems require efficient methods to find specific mobile users. In this paper, we design, analyze, and evaluate searching strategies that are based on some known statistics about the location of the mobile users. We demonstrate a trade-off between the running time of the algorithm that computes the search strategy (*time complexity*) and the amount of bandwidth the search strategy consumes (*optimality*). We also demonstrate a secondary trade-off between the *simplicity* and the optimality of the algorithm that computes the search strategy.

Consider cellular systems with many cells and many mobile users (*mobiles*) that roam among the cells. If a mobile reports its new location whenever it crosses boundaries of cells, then the cellular system would know its exact location at any time and finding (*paging*) a mobile becomes a trivial task. However, cellular networks are expected to have more cells (mini-cells or micro-cells) and mobiles are expected to move very fast. As a result, a mobile might cross boundaries of cells very frequently, making it infeasible for the mobile to report its new location each time it enters a new cell. This is mostly due to the scarcity of up-link wireless communication and the short life of hand-held device batteries. Indeed, many existing *location management* schemes (reporting and paging strategies) allow mobiles to report less often. A common location management framework partitions the cells into location areas (*zones*) each composed of many cells where a mobile reports its new location only when it crosses boundaries between zones (e.g.,

*Corresponding author
Email addresses: amotz@sci.brooklyn.cuny.edu (Amotz Bar-Noy), panagiotis.cheilaris@usi.ch (Panagiotis Cheilaris), daniel.feng@gmail.com (Yi Feng), golin@cs.hkust.hk (Mordecai J. Golin)

[1–4]). When a call to a mobile arrives, the system may need to locate the cell where the mobile currently resides by paging some or all the cells in a particular zone. Although the choice of a location management scheme to minimize the overall use of wireless bandwidth depends on many parameters, such a paging step is common to most of the schemes.

Suppose that the mobile user is roaming in a zone composed of the N cells $C_1, \ldots, C_N$. An algorithm can page every cell in any subset of these N cells in a unit of time (*paging round*), by paying a cost for every cell paged. If the mobile user is located in any cell of the subset, the algorithm is successful in locating the mobile user. The trivial solution would page all the N cells in the first and only round (*a blanket search*); this clearly is the fastest strategy however it uses the highest possible amount of wireless bandwidth. The other extreme is to page the cells sequentially in N rounds terminating once the user is found (*a sequential search*). Without any a priori knowledge on the whereabouts of the mobile this strategy would page $(N + 1)/2$ cells on average if the cells are paged in a random order. In many cases, some a priori knowledge about the whereabouts of the user is known. This knowledge can be modeled with N probability values, one value associated with each of the N cells: with probability p_i the mobile resides in cell C_i and all the probabilities are independent. This a priori knowledge could either be supplied by the mobile user itself, be extracted from history logs maintained by the system, or be based on recent reports and calls involving this mobile user. It is not hard to see that the best sequential search (N rounds) would page the cells in a non-increasing order of their associated probabilities and that the expected number of paged cells is $\sum_{i=1}^{N} ip_i$. The ultimate goal is to minimize both the number of paging rounds and the expected number of cells paged until the mobile is found. These are the two main criteria in evaluating the efficiency of a specific paging strategy. The first corresponds to the delay incurred until the mobile is found and the second corresponds to the amount of wireless bandwidth used. The first criterion is important to the mobile users and the second criterion is crucial to the system.

The papers [5–8] describe how to trade bandwidth for time. They show how to find a paging strategy that uses at most D rounds ($1 \leq D \leq N$) and that minimizes the expected number of cells paged until the mobile is located. The running time of their algorithms, that are based on a dynamic programming formulation, is $\Theta(DN^2)$, i.e., on the order of DN^2, ignoring multiplicative constants. In papers [5–7] the strategy is computed starting from the last round, whereas in paper [8] starting from the last round. Since the probability vector may change frequently, more efficient solutions are desired. This is the subject of this paper. Another objective of this paper is to find algorithms that on one hand compute almost optimal search strategies and on the other hand have faster running times and simpler implementations.

Although our main focus in this work is locating mobile users, the scope of this paper is very general. Let *mobile data* be an abstraction of any entity in a network whose exact location is not known to the system at the time when a specific query is looking for this data. Instead, the system knows that the mobile data may be found in one out of N locations. Furthermore, the system has a profile for the data which is represented as a vector of probabilities: with probability p_i the data is in location L_i and all the probabilities are independent. Paging mobile users in cellular networks is one application to this general setting but there are more applications. Consider a wireless sensor network that accumulates some information (e.g., weather or traffic). Mobile data may be any information that can be found in this sensor network. In order to save battery energy, the sensors do not push the information but only reply to queries. As a result, the system needs to pull the data by probing the sensors. The above framework models the pull task where the objectives are to minimize the time it takes to get the data and to minimize the expected number of sensors that are probed. The above two applications are for wireless networks, but one could think of similar applications in any kind of network, for example, searching for some data in the Internet or in a peer-to-peer network.

1.1. Prior art and related work

Modeling uncertainty of locations of mobiles as a probability distribution vector is studied in, e.g., [9], which discusses a framework for measuring uncertainty. The paper [10] provides a simple strategy for two paging rounds. The papers [5–8] find a D-round paging strategy with minimum expected number of cells paged using dynamic programming. The papers [5, 8] also study how to minimize the expected number of paged cells given the average (as opposed to worst-case) delay constraint using relaxation to a continuous model [5] or with a weakly polynomial dynamic programming solution [8]. The papers [11, 12] present sub-optimal algorithms which are computationally more efficient than the dynamic programming ones. The papers [13, 14] address the case in which the exact values of the probabilities are not known. In the model studied in [13], the probabilities are learned online and in the model studied in [14], only estimate values of the probabilities are accessible to the algorithm, due to privacy concerns. The

effect of queuing on paging delay when paging requests arrive to the system according to some random process and each request is to page a single mobile user is studied by [6, 15, 16]. The problem of paging more than one mobile user for a conference call is studied in [17–20]. The combined cost of reporting and paging is studied by many papers. (See the survey [21] on location management for mobile users.) The main issue in this line of research is to see how mobiles can reduce the overall wireless cost by reporting their new location according to some rules. Algorithms for efficient maintenance, in the backbone network, of data structures for locating mobiles have been studied in many papers, see, e.g., [22, 23].

1.2. Contributions

Our first contribution is to introduce two new algorithms for designing optimal strategies; the running time of these two new algorithms is $\Theta(DN)$, saving a factor of N over the $\Theta(DN^2)$ running time of the previously known optimal algorithms [5–8]. Our approach is to show that the dynamic programming recursive formulation satisfies the Monge property [24] and the SpeedUp property [25]. Each property enables us to apply a dynamic programming speedup technique. We also describe the original $\Theta(DN^2)$ implementation and another simple algorithm whose running time is $\Theta(DN \log N)$. The four optimal algorithms demonstrate a trade-off between simplicity and running time.

Our second contribution is the design of two near-optimal heuristic solutions. The running time of the first is $\Theta(N \log D)$ and the running time of the second is $\Theta(DN)$. The first heuristic, called Divide-and-Conquer, is an efficient generalization of the simple solution for the case $D = 2$ (the dynamic programming is another generalization). Our first heuristic outperforms in terms of the cost of the computed solution a known heuristic, called Boundaries [11, 12], whose running time is $\Theta(N)$, while running faster for small values of D. The running time of the second heuristic, called FirstLocalMin, is $\Theta(DN)$ which is of the same order of our optimal solutions. Nevertheless, it is much simpler to describe and implement. Moreover, this heuristic finds the optimal solution for most instances tested by our simulations and seems well-tailored for real data. However, we show that it might perform badly for some specially crafted inputs, disproving a conjecture in the conference version [26] of this work.

We support our results with a comprehensive simulation study. We implemented all the optimal and heuristic solutions that are described in the paper. Our main objective was to demonstrate the trade-off between optimality and running time efficiency. We tested the solutions on various distributions for the probability vector: Zipf, Gaussian, Uniform Random, and Step. For each of them, we run the algorithms on different values for the two main parameters N and D and the parameters that are related to the distribution. In general, the results coincide with the worst-case running time analysis. Nevertheless, there are some minor differences, mainly because the worst-case analysis ignores constants and due to some required preprocessing.

1.3. Paper organization

Section 2 provides some preliminaries. Section 3 describes the recursive formulation of the optimal dynamic programming solution and presents and analyzes the running time of four algorithms that are based on the recursive formulation. Section 4 describes and analyzes the running time of several heuristics. Section 5 presents the simulation work that compares the running time and performance of the various optimal and heuristic solutions on a "real" machine. Section 6 contains conclusions and some open problems.

2. Preliminaries

The system is looking for a mobile that is roaming in a zone composed of the N cells $C_1, \ldots, C_N$. For $1 \leq i \leq N$, let p_i be the probability that the mobile is in cell C_i at the time of search where all the probabilities are independent. It follows that $\sum_{i=1}^{N} p_i \leq 1$ and with probability $q = 1 - \sum_{i=1}^{N} p_i$ the mobile is *off* the system. For simplicity, we assume that the mobile is *on* and therefore $q = 0$. The algorithms and the analysis can be adapted to the more general case in a straightforward manner.

Paging the mobile is conducted in rounds. In each round a subset of the cells is paged until the subset that contains the actual location of the mobile is paged. The goal is to minimize the expected number of paged cells under the constraint that the paging must be complete in at most D rounds for a given parameter $D \in \{1, \ldots, N\}$. A *paging strategy* $\mathcal{S}_D$ is an ordered D-partition $\langle S_1, \ldots, S_D \rangle$, such that in the d-th round, the cells in set S_d are paged.

The *paging cost* of a strategy $\mathcal{S}$ for the probability vector $(p_1, \ldots, p_N)$ is the expected number of paged cells. Let n_i be the size of the subset S_i and let $p(S) = \sum_{C_i \in S} p_i$ for any subset S of the N cells. Then, the paging cost is

$$cost(\mathcal{S}) = \sum_{i=1}^{D} \Big(\sum_{j=1}^{i} n_j \Big) p(S_i) = \sum_{i=1}^{D} \Big(\sum_{j=i}^{D} p(S_j) \Big) n_i,$$

because with probability $p(S_i)$ the strategy pages $\sum_{j=1}^{i} n_j$ cells, or alternatively, the strategy pages the n_i cells of S_i, when the user is in any of the cells of the sets $S_i, S_{i+1}, \ldots, S_D$. An *optimal paging strategy* O_D is a paging strategy for which $cost(O_D)$ is the minimum among all possible paging strategies. Denote by OPT(D) the cost of optimal paging strategy O_D.

A *paging algorithm* is an algorithm that outputs a paging strategy $\mathcal{S}_D$ for any probability vector $(p_1, \ldots, p_N)$ and delay constraint D. An *optimal paging algorithm* is a paging algorithm that outputs optimal paging strategies on all possible inputs.

The following two observations are used by most of the optimal and non-optimal solutions that appear in the literature and by all the algorithms that appear in this paper. The first observation is that an optimal strategy should use all the rounds.

Observation 1. *Given a delay constraint $D \leq N$, it is always better to page at least one cell in each round.*

Proof. Assume that a paging strategy uses D' rounds for some $D' < D$. Then there must be a round whose associated set of cells contains more than one cell. Splitting this round into two rounds would decrease the paging cost. □

The second observation relates the location probabilities and the order of cells in an optimal paging strategy.

Observation 2. *An optimal paging strategy $O_D = \langle O_1, \ldots, O_D \rangle$ must follow the non-increasing order of probabilities, i.e., for $1 \leq d_i < d_j \leq D$, if $C_i \in O_{d_i}$ and $C_j \in O_{d_j}$, then $p_i \geq p_j$.*

Proof. One can verify that by the definition of the paging cost, if $p_i < p_j$ then the cost is lower for a paging strategy which is identical to O_D except that $C_i \in O_{d_j}$ and $C_j \in O_{d_i}$. □

As a result of the previous observation, without loss of generality we assume that $p_1 \geq p_2 \geq \cdots \geq p_N$.

When $D = 1$, all the cells must be paged in the first and only round and the cost of this optimal strategy is $cost(O_1) = \text{OPT}(1) = N$. When $D = N$, by Observation 2, the optimal paging strategy pages cell C_i in round i for $1 \leq i \leq N$ and the cost is $cost(O_N) = \text{OPT}(N) = \sum_{i=1}^{N} i p_i$. When $D = 2$, an optimal paging strategy pages a subset of the cells in the first round, and if the mobile is not found, the rest of the cells must be paged in the second round. By Observation 2, there exists a pivot n, $1 \leq n < N$, such that cells $\{C_1, \ldots, C_n\}$ are paged in the first round. Define $Q_n = \sum_{i=1}^{n} p_i$ to be the probability of the user to be in cells $C_1, \ldots, C_n$. It follows that $cost(O_2) = \text{OPT}(2) = \min_{1 \leq n < N} \{Q_n n + (1 - Q_n)N\}$.

3. Optimal algorithms

In this section, we provide four algorithms to compute the value of OPT(D) and the corresponding search strategy O_D for a given probability vector $(p_1, \ldots, p_N)$ and $D \in \{1, \ldots, N\}$. The first algorithm is a straightforward implementation of the dynamic programming recursion. This is the known $\Theta(DN^2)$ algorithm that appeared in the literature. The second algorithm reduces the running time to $\Theta(DN \log N)$ with another simple implementation of the dynamic programming. The third and the fourth algorithms further reduce the running time to $\Theta(DN)$; both algorithms are relatively complicated and are based on non-trivial properties of the dynamic programming recursion.

3.1. The dynamic programming scheme

We now define the recursive solution formulation to find OPT(D). We ignore the computation of the optimal partition O_D that yields the optimal cost. We note that in implementing dynamic programming solutions, it is a standard technique to find the actual solution by first filling in the dynamic programming table with costs and then tracing backwards from the optimal solution to obtain the corresponding strategy. This adds a negligible $\Theta(D)$ overhead

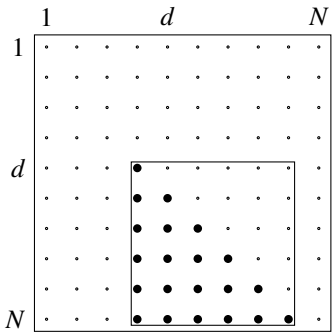


Figure 1: Matrix $A^{(d)}$, matrix B, and the lower triangular part of the latter

in the running time complexity. We also assume that the vector of probabilities is given in a non-increasing order ($p_1 \geq p_2 \geq \cdots \geq p_N$) since all the algorithms use this order. We note that the sorting can be done in $\Theta(N)$ time (using radix-sort for example) if the set of distinct values of probabilities is not too large.

Let $n \in \{1, \ldots, N\}$ and $d \in \{1, \ldots, D\}$. Define $h_n^{(d)}$ to be the optimal cost of finding the mobile user in the first n cells, $C_1, \ldots, C_n$, in d rounds. By definition, $\mathrm{OPT}(D) = h_N^{(D)}$. For the base of the dynamic programming computation ($d = 1$) and for all $n \in \{1, \ldots, N\}$: $h_n^{(1)} = n \cdot \sum_{i=1}^{n} p_i$.

There are two approaches to formulate the optimal recursion for $h_n^{(d)}$. In the first, by [5], [6], the size of the first set of cells to be paged in the first round is fixed and then the rest of the cells are recursively and optimally paged in $d - 1$ rounds. In the second, by [8], the size of the last set of cells to be paged in the last round is fixed and then the rest of the cells are recursively and optimally paged in $d - 1$ rounds. We adopt the second approach.

Definition 3. For fixed $d \in \{2, \ldots, D\}$, let $A = A^{(d)}$ be the $N \times N$ matrix with entries: $a_{n,j}^{(d)} = h_j^{(d-1)} + n \cdot \sum_{i=j+1}^{n} p_i$, for $d \leq n \leq N$ and $d - 1 \leq j \leq n - 1$. In other words, $a_{n,j}^{(d)}$ is the cost of paging the first j cells optimally in $d - 1$ rounds and the next $n - j$ cells in the d-th round. Let $N' = N - d + 1$, then matrix B is an $N' \times N'$ sub-matrix of A, where $b_{n',j'} = a_{n'+d-1,j'+d-2}^{(d)}$, for $1 \leq j' \leq n' \leq N'$.

We remark that the rest of the entries of A are irrelevant. We can set them to ∞ or $a_{n,j}^{(d)} = h_j^{(d-1)}$, when $j \geq n$. B is the submatrix of A that precisely contains relevant entries (see Figure 1). We will use interchangeably A and B; their index variables are related as follows:

$$n' = n - (d - 1) \quad \text{and} \quad j' = j - (d - 2). \tag{1}$$

The proof of the following lemma can be found in the appendix of [8].

Lemma 4. $h_n^{(d)} = \min_{j=d-1}^{n-1} a_{n,j}^{(d)}$.

Our objective is to implement the recursion defined in Lemma 4 in a dynamic programming fashion. All of our implementations are based on the schematic Algorithm 1.

Algorithm 1 Dynamic programming: compute the minimum cost of paging N cells in D rounds; cost = $\mathrm{DP}(N, D)$

```
for n = 1 ... N do {base d = 1}
    h_n^(1) ← n · Σ_{i=1}^{n} p_i
for d = 2 ... D do
    for n = d ... N do
        h_n^(d) ← min_{j=d-1}^{n-1} a_{n,j}^(d)
return h_N^(D)
```

For a fixed d, the innermost loop in Algorithm 1 computes the minimum in each of the N' rows of $A^{(d)}$ with relevant entries. Formally, the input to the Row-Minima problem is an $N' \times N'$ lower triangular matrix B and the

objective is to find the minimum value in each row. Alternatively, the objective is to compute the following sequence $\{j'(n')\}_{n'=1}^{N'}$:

Definition 5. For $1 \le n' \le N'$, let $j'(n')$ be the smallest column index in which the minimum value of row n' appears in matrix B. Alternatively, for $d \le n \le N$, let $j(n)$ be the smallest column index in which the minimum value of row n appears in matrix $A = A^{(d)}$.

In the following theorem, we state the complexity relationship between the Row-Minima problem of matrix B and the optimal paging problem.

Theorem 6. *Let algorithm $\mathcal{X}$ be an algorithm that solves the Row-Minima problem for the matrix B in time $\Theta(T(N'))$ bounded by a polynomial w.r.t. N'. If $T(N') = \Omega(N')$, then the overall running time of algorithm $\mathcal{X}$ on the optimal paging problem is $\Theta(D \cdot T(N))$.*

Proof. In the first round, it takes $\Theta(N)$ time to compute $h_n^{(1)}$ for $n = 1 \dots N$. For rounds $d = 2 \dots D$, it takes $\Theta(T(N-d))$ time to compute the row minima of matrix B. Overall, the running time is $\Theta(N + \sum_{d=2}^{D} T(N-d))$. Since $\sum_{d=2}^{D} T(N-d) \ge (D/2)T(N/2) = \Omega(D \cdot T(N))$ (because $T(N)$ is bounded by a polynomial) and $\sum_{d=2}^{D} T(N-d) \le (D-1)T(N)$, the overall running time of $\mathcal{X}$ on the optimal paging problem is $\Theta(D \cdot T(N))$ □

For efficiency reasons, it will be crucial that our implementations do not compute all relevant entries of A (or B); an entry is computed only when it is needed. Such computation can be done in constant time per entry. To achieve this, we first assume that we have, in $\Theta(N)$ time, precomputed all of the prefix sums $Q_k = \sum_{i=1}^{k} p_i$, for $1 \le k \le N$. Then $a_{n,j}^{(d)} = h_j^{(d-1)} + n(Q_n - Q_j)$, which can be computed in $O(1)$ time (since d is fixed, and we have already computed $h_j^{(d-1)}$ before computing $a_{n,j}^{(d)}$). That is, we have an oracle that, given $h_j^{(d-1)}$ and the indices n, j, permits us to calculate $a_{n,j}^{(d)}$ in constant time.

3.2. Algorithms

We now describe four algorithms, called Seq, Bin, SMAWK, and SpeedUp, that are based on the *Dynamic Programming Scheme*. For each of them, we first show how to find the vector $(j(d), \dots, j(N))$, i.e., solve the *Row-Minima problem* for the matrix A (or B), and analyze its time complexity. Then, for each algorithm the overall time complexity is an immediate corollary of Theorem 6.

1. The $\Theta(DN^2)$ straightforward implementation:

Row-Min-Seq(A): For each n with $d \le n \le N$, sequentially find $m = \min_{j=d-1}^{n-1} a_{n,j}^{(d)}$ and set $j(n)$ to the smallest index such that $a_{n,j}^{(d)} = m$.

Finding the minimum in each row can be done in $\Theta(N)$ time and therefore Row-Min-Seq has time complexity $\Theta(N^2)$.

Corollary 7. *The running time of Seq is $\Theta(DN^2)$.*

2. The $\Theta(DN \log N)$ binary search implementation:

The next lemma implies that the sequence of column indices $j(d), \dots, j(N)$ is non-decreasing.

Lemma 8. *For $d \le n_1 < n_2 \le N$, $j(n_1) \le j(n_2)$.*

Proof. Assume for the sake of contradiction that there is some $n > d$ such that $j_1 = j(n) < j(n-1) = j_2$. Then: $a_{n-1,j_1}^{(d)} > a_{n-1,j_2}^{(d)}$ and $a_{n,j_2}^{(d)} \ge a_{n,j_1}^{(d)}$, which imply $a_{n-1,j_1}^{(d)} + a_{n,j_2}^{(d)} > a_{n-1,j_2}^{(d)} + a_{n,j_1}^{(d)}$. Therefore, using Definition 3, $(n-1) \cdot \sum_{i=j_1+1}^{n-1} p_i + n \cdot \sum_{i=j_2+1}^{n} p_i > (n-1) \cdot \sum_{i=j_2+1}^{n-1} p_i + n \cdot \sum_{i=j_1+1}^{n} p_i$ or $\sum_{i=j_1+1}^{j_2} p_i < 0$, which is a contradiction. □

Row-Min-Bin(A): Instead of computing $j(n)$ sequentially, compute it in a binary fashion. Informally (ignoring ceilings and floors), the algorithm has $\log_2 N$ stages. In the first stage, it computes $j(N/2)$; in the second stage, it computes $j(N/4)$ and $j(3N/4)$; and in general, in the ith stage it computes $j(kN/2^i)$ for all odd numbers k between 1 and 2^i.

By Lemma 8, each stage can be computed in $\Theta(N)$ time because each one of the 2^{i-1} values that are computed in stage i is computed over a unique range and the total size of all these ranges is N. Since there are $\log_2 N$ stages, the running time of Row-Min-Bin is $\Theta(N \log N)$.

Corollary 9. *The running time of Bin is $\Theta(DN \log N)$.*

3. The Θ(DN) SMAWK algorithm:

We prove a stronger property for matrix A (or B), called the *Monge* property.

Definition 10. An $N' \times N'$ matrix M satisfies the lower triangular *Monge* property [24] if for $1 < j' < n' \leq N'$:

$$(m_{n'-1,j'-1} + m_{n',j'}) - (m_{n',j'-1} + m_{n'-1,j'}) \leq 0.$$

Lemma 11. *Matrix B satisfies the lower triangular Monge property.*

Proof. Because of (1), it is enough to prove $(a^{(d)}_{n-1,j-1} + a^{(d)}_{n,j}) - (a^{(d)}_{n,j-1} + a^{(d)}_{n-1,j}) \leq 0$ and indeed:

$$\left(h^{(d-1)}_{j-1} + (n-1)\cdot\sum_{i=j}^{n-1} p_i + h^{(d-1)}_{j} + n\cdot\sum_{i=j+1}^{n} p_i\right) - \left(h^{(d-1)}_{j-1} + n\cdot\sum_{i=j}^{n} p_i + h^{(d-1)}_{j} + (n-1)\cdot\sum_{i=j+1}^{n-1} p_i\right) = (n-1)p_j - np_j = -p_j < 0. \quad \square$$

In [27] the authors introduce an algorithm that solves the Row-Minima problem for Monge matrices in $\Theta(N)$ time. The paper [24] gives an alternative description of this algorithm and call it *SMAWK* (after the initials of the authors of [27]).

Theorem 12. *If an $N \times N$ matrix satisfies the Monge property, then all of its row minima can be found in $\Theta(N)$ time.*

Row-Min-SMAWK(A)**:** Apply the SMAWK technique from [27] to solve the Row-Minima problem for the matrix A.

Corollary 13. *The running time of SMAWK is $\Theta(DN)$.*

4. The Θ(DN) SpeedUp algorithm:

While the SMAWK technique does imply a $\Theta(DN)$ algorithm, it is challenging to be implemented properly. We now describe another property of A (or B) that yields a simpler (but still complicated) $\Theta(DN)$ algorithm to solve the Row-Minima problem.

Definition 14. An $N' \times N'$ matrix M satisfies the *SpeedUp* property if $m_{n',j'} - m_{n'-1,j'} = u_{n'} + v_{j'}$ for $1 \leq j' < n' \leq N'$, where $u_{n'}$ is a function of n' and $v_{j'}$ is a monotonically decreasing function of j'.

Lemma 15. *Matrix B satisfies the SpeedUp property.*

Proof. Because of (1), it is enough to show:

$$a^{(d)}_{n,j} - a^{(d)}_{n-1,j} = h^{(d-1)}_{j} + n\cdot\sum_{i=j+1}^{n} p_i - h^{(d-1)}_{j} - (n-1)\cdot\sum_{i=j+1}^{n-1} p_i = \underbrace{(n-1)p_n + \sum_{i=1}^{n} p_i}_{u_n} + \underbrace{\left(-\sum_{i=1}^{j} p_i\right)}_{v_j},$$

where u_n depends only on n and v_j is a decreasing function of j, since for every i, $p_i > 0$. $\square$

We can show that for matrices that satisfy the SpeedUp property there exists an algorithm that solves the Row-Minima problem in $\Theta(N)$ time. This algorithm is an adaptation of the one given in [25] for solving the dynamic program for placing K medians on a line.

Theorem 16. *If an $N \times N$ matrix satisfies the SpeedUp property, then all of its row minima can be found in $\Theta(N)$ time.*

Row-Min-SpeedUp(A)**:** Apply an adaptation of the algorithm in [25] to solve the Row-Minima problem for the matrix B.

Corollary 17. *The running time of SpeedUp is $\Theta(DN)$.*

4. Non-optimal heuristics

In the previous section, we show how to reduce the running time complexity of algorithms for finding *optimal* strategies from $\Theta(DN^2)$ to $\Theta(DN \log N)$ and then to $\Theta(DN)$. In highly dynamic systems the probabilities might change frequently and for large N even $\Theta(DN)$ might be too slow. We therefore propose several non-optimal heuristics whose running time is better than $\Theta(DN)$.

We first present three oblivious solutions that for given values of N and D ignore the values of the probabilities and provide the same partition. These algorithms are very fast to compute and each performs sufficiently well under some conditions. Next, we describe a heuristic that was proposed in [11] whose running time is $\Theta(N)$ and propose a new heuristic whose running time is $\Theta(N \log D)$ with a small constant coefficient. In the next section, we will demonstrate that for $D << N$, our heuristic outperforms the heuristic of [11] in both categories: performance and running time. Our last heuristic has a $\Theta(DN)$ running time and is much simpler to describe and implement than the two optimal $\Theta(DN)$ running time algorithms. Moreover, this heuristic provided the optimal solution for most of the instances in our simulations.

4.1. Oblivious heuristics

The following three heuristics depend only on the value of N and D and ignore the values of $p_1, \ldots, p_N$. All of them can be computed in $\Theta(D)$ time assuming the cells are ordered by a non-decreasing order of their associated probabilities. The first two are folklore and the third is based on a known technique and is discussed in [14].

LargeSuffix: *In rounds* 1 *to* $D-1$ *page only one cell. In the last round page the rest of the* $N-D+1$ *cells.* This heuristic performs very well when the mobile user is likely to be found in $D-1$ cells. It performs poorly when the values of the N probabilities are almost the same.

Uniform: *In the first* $(D-(N \bmod D))$ *rounds page* $\lfloor N/D \rfloor$ *cells; in the last* $(N \bmod D)$ *rounds page* $\lceil N/D \rceil$ *cells.* This heuristic performs very well when the values of the N probabilities are almost the same. It performs poorly when the mobile user is likely to be found in one particular cell. In a way, *Uniform* and *LargeSuffix* complement each other.

Doubling: *Find a parameter* α *such that* $\alpha + \alpha^2 + \cdots + \alpha^D = N$ *(*$\alpha \approx N^{1/D}$*). For* $1 \le d \le D$*, in round* d *page either* $s_d = \lfloor \alpha^d \rfloor$ *or* $s_d = \lceil \alpha^d \rceil$ *cells. The floor and ceiling decisions are made such that* $s_1 + \cdots + s_D = N$ *and* $s_1 \le s_2 \le \cdots \le s_D$. This heuristic is a compromise between Uniform and LargeSuffix. The Doubling technique (usually $\alpha = 2$) is a well known and useful technique when some of the parameters are not known in advance. Indeed, in [14], it is shown that Doubling has the best worst-case performance against an adversary that may choose maliciously the values for $p_1, \ldots, p_N$.

4.2. The Boundaries heuristic

The heuristic described in [11] starts with some initial partition of the cells into D sets. Then cells in the boundaries of the sets are moved from one set to an adjacent set if some conditions are not satisfied. These conditions, that are necessary conditions for a paging partition to be optimal, are stated in the following definition.

Definition 18. Let $\{S_1, \ldots, S_D\}$ be a partition of the N cells, let n_i be the size of the subset S_i, let ℓ_i be the largest probability in S_i, and let s_i be the smallest probability in S_i. Then, the *forward* and the *backward* boundary condition are, respectively:

$$\ell_{i+1}(n_{i+1} - 1) \le \sum_{j \in S_i} p_j \quad \text{and} \quad s_i(n_{i-1} + 1) \ge \sum_{j \in S_i} p_j \, .$$

Boundaries: Partition the N cells into D sets of almost the same size ($\lfloor N/D \rfloor$ or $\lceil N/D \rceil$). Let the partition be $\{S_1, \ldots, S_D\}$. In the first stage, from S_1 to S_{D-1}, check the forward boundary condition and move a cell to the next set each time the condition is violated. In the second stage, from S_D to S_2, check the backward boundary condition and move a cell to the previous set each time the condition is violated.

The first stage is a forward scan of almost all the N cells and the second stage is a backward scan of almost all the N cells. Each scan can be implemented in $\Theta(N)$ time and therefore,

Proposition 19. *The running time of Boundaries is* $\Theta(N)$.

We note that the initial partition could be any of the partitions generated by the three oblivious heuristics. Our simulations show that the choice made by [11] was the best for most of the instances tested by the simulation.

4.3. The divide and conquer heuristic

We propose a new non-optimal heuristic whose running time is $\Theta(N \log D)$. The basic idea is to apply $\lceil \log_2(D) \rceil$ times the optimal algorithm for the case $D = 2$ that can be implemented efficiently in $\Theta(N)$ time. For simplicity assume that D is a power of 2.

DQ: In stage 1, find the best partition of the N cells into two sets such that each set gets $D/2$ paging rounds. For $2 \le i \le \log_2 D$, in stage i, partition the N cells into 2^i sets such that each set gets $D/2^i$ paging rounds. The final partition is the one after stage $\log_2 D$. To divide a set of cells into two, we look for a pivot cell C_m in the set, such that cells up to C_m are paged in the previous round(s), cells after C_m are paged in the later round(s), and the expected cost of paging the two sets is minimized. Formally, we apply Algorithm 2 by $\mathrm{DQ}(\langle p_1, \ldots, p_N \rangle, 1, D, 1, N)$.

The method used to find the best partition in each stage for each range is an adaptation of the way an optimal partition is found for the case $D = 2$. This implies a running time which is linear in the number of cells to be partitioned. Since in each stage all the ranges are disjoint, it follows that the running time of each stage is $\Theta(N)$.

Proposition 20. *The running time of DQ is $\Theta(N \log D)$.*

Algorithm 2 Divide and conquer: Recursively divide cells $\{C_{n_1} \ldots C_{n_2}\}$ that contain rounds $\{d_1 \ldots d_2\}$ into two parts. parts = $\mathrm{DQ}(\mathbf{p}, d_1, d_2, n_1, n_2)$

```
if d_1 = d_2 or n_1 = n_2 then
  return [n_1 ... n_2]
else
  d ← ⌊(d_1 + d_2)/2⌋
  m ← argmin_{n_1+d−d_1 ≤ i ≤ n_2+d_2−d} { i · Σ_{n_1 ≤ j ≤ i} p_j + n_2 · Σ_{i < j ≤ n_2} p_j }
  S_left  = DQ(p, d_1, d, n_1, m)
  S_right = DQ(p, d + 1, d_2, m + 1, n_2)
  return S_left, S_right
```

4.4. The First Local Minimum heuristic

Row-Min-FirstLocalMin(A): For each row n from d to N, start at column $j(n-1)$ (by convention $j(d-1) = d-1$), find the first local minimum in row n, and store its column in $j(n)$.

The search for values $j(d), \ldots, j(N)$ takes $\Theta(N')$ time since the minimum search ranges overlap only at their extremes, and therefore immediately from Theorem 6:

Corollary 21. *The running time of FirstLocalMin is $\Theta(DN)$.*

A heuristic solution is usually compared against an optimal solution algorithm by the ratio of the cost computed by the heuristic to the cost of an optimal solution. Although our heuristic computed the optimal solution in all simulations we tried, we prove that for some specially crafted inputs it can compute solutions quite far from optimal, settling an open problem in [26].

Proposition 22. *There are inputs for which FirstLocalMin finds a solution worse than optimal, even for $D = 3$, and the ratio of the two solutions is unbounded.*

Proof. Consider the probability vector with N probabilities $p = (x, x, \overbrace{sy, \ldots, sy}^{t}, \overbrace{y, \ldots, y}^{N-2-t})$, where $x < 1/2$, s, and t are parameters, to be specified later. Since $\sum_{i=1}^{N} p_i = 1$, we have $y = (1 - 2x)/(N - 2 - t + st)$.

We consider the following two strategies for $D = 3$:

$$\mathcal{S}' = \langle \{1\}, \{2\}, \{3, \ldots, N\} \rangle \quad \text{and} \quad \mathcal{S} = \langle \{1, 2\}, \{3, \ldots, t+2\}, \{t+3, \ldots, N\} \rangle,$$

whose respective costs are:

$$C' = 3x + (1 - 2x)N \quad \text{and} \quad C = 4x + syt(t+2) + (N - 2 - t)yN.$$

In order for FirstLocalMin to output strategy $\mathcal{S}'$, it must be the case that: for all $n \in [4, N]$, $a^{(3)}_{n,2} \leq a^{(3)}_{n,j}$ or equivalently, $a^{(3)}_{n,3} - a^{(3)}_{n,2} = x + 3sy - syn \geq 0$, or, since $n \leq N$,

$$x + 3sy - syN \geq 0. \tag{2}$$

We need to choose the parameters s, t, x, so that condition (2) is true and the ratio C'/C goes to infinity as a function of N. We set $s = N^{\sigma}$, $t = N^{\tau}$, and $x = \frac{1}{2} - \frac{1}{2}N^{-\alpha}$, where σ, τ, α are positive reals. With these new parameters, condition (2) is satisfied whenever

$$\alpha + \sigma + \tau > 1 + \alpha \quad \text{and} \quad \alpha + \sigma + \tau > 1 + \sigma. \tag{3}$$

If $1 - \alpha > 0$, the cost of each partition is $C' = \Theta(N^{1-\alpha})$ and $C = \Theta(N^{\tau-\alpha} + N^{2-(\alpha+\sigma+\tau)})$, and by choosing $\alpha = 1 - \varepsilon/2$, $\sigma = 2$, and $\tau = \varepsilon$, where ε is a positive real arbitrarily close to 0, condition (3) is satisfied and the ratio becomes $C'/C = \Theta(N^{1-\varepsilon})$. □

5. Simulation

In complement to our theoretical results, we conduct a comprehensive simulation to show the trade-off between efficiency and complexity and the trade-off between efficiency and implementation simplicity. We report some sample tests out of the many conducted in our simulation. For each test, only representative figures for some specific parameters are provided. More tests can be found in [28].

5.1. Setup

Algorithms implemented: We implement all the algorithms in this paper, including the optimal algorithms Seq, Bin, SMAWK, and SpeedUp, and the non-optimal algorithms LargeSuffix, Uniform, Doubling, Boundaries, DQ, and FirstLocalMin.

Benchmark: We measure the running time of all the algorithms and the paging cost (compared with the optimal algorithms) of the non-optimal heuristics. We run these algorithms for 100 to 1 000 000 iterations on each instance and measure the average running time.

Environment: We implement the algorithms in C++ using the g++ compiler with default options and then run programs in a dedicated Intel Core 2 Quad 9550 CPU node of a Linux cluster. All programs are single threaded using only one core of the CPU. We also implement all the algorithms using Matlab on a personal computer and some of the algorithms using C. We conclude that our results are platform independent.

5.2. Data

In this paper, we mostly use data following the Zipf distribution. The probability vector follows the Zipf distribution with parameter α if $p_i = i^{-\alpha}/\sum_{i=1}^{N} i^{-\alpha}$, $(1 \leq i \leq N)$. In [29], the authors point out that human beings are more likely to move at a hop length that follows the *Zipf* distribution. The same Zipf model also applies to Internet traffic and congestion [30]. We have obtained 171 929 appearances of 996 users in 5 625 cells on 31 consecutive days from the boundary of a metro and a suburban region (courtesy China Unicom). These data also confirm that the user probability vector $\mathbf{p}$ follows the Zipf distribution with parameter $\alpha \approx 0.5$ (See Figure 2.).

We also test the algorithms for data that is generated from the Gaussian, Uniform Random, and Step distributions. The *Gaussian* distribution, $p_i = q_i/\sum_{i=1}^{N} q_i$, where $q_i = \frac{2}{\sigma\sqrt{2\pi}} \exp(-i^2/(2\sigma^2))$, $(1 \leq i \leq N)$, is a continuous distribution commonly used to model the notion of ranking based on some frequency of occurrence [11]. The *Uniform Random* distribution is a natural distribution, in which each p_i is a uniform random number between 0 and 1 that is normalized by the sum of all the N original random numbers. The *Step* distribution is a local-uniform distribution in which the cells are partitioned into groups such that all the cells in a group are associated with the same location probability. In particular, for the step distribution with ratio α and count s, $p_i = \alpha^{\lceil i/(N/s) \rceil}/\sum_{k=1}^{s}\sum_{j=1}^{N/s} \alpha^k$, $(1 \leq i \leq N)$.

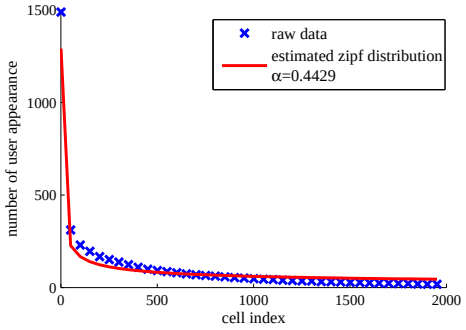


Figure 2: Appearances of the 996 users in the 2000 most frequent cells from real system: x-axis: indices of 2000 cells, y-axis: number of users' appearances in the cells

5.3. Implementation Complexity

For algorithms of time linear in N, there are differences in the multiplicative factor in front of the linear term. For example, SMAWK uses recursion and an upper bound on the number of comparisons is $6DN - 3D(D+1)$ (see [27]), whereas Boundaries is doing about $2N$ comparisons, checking the conditions from C_1 to C_N and then back to C_1.

First Test: We test the running time of all optimal algorithms and non-optimal heuristics using all four types of data. We measure the running time with respect to problem scale (N). Figure 3(a) illustrates results for the Zipf distribution, when N varies from 100 to 1000. As expected, the plot coincides with our complexity analysis. We note that similar behavior occurs for all the distributions and for many other parameters. A direct observation is that all our optimal algorithms and heuristics run much faster than the original $\Theta(DN^2)$ optimal algorithm Seq.

We observe that Bin is efficient when N is in a reasonable range (hundreds to thousands). For this range, $\log_2 N$ is comparable to the constant 6 coefficient in the complexity of SMAWK. Moreover Bin only involves simple algorithmic operations and is faster than SMAWK in this test. Only when $N \geq 8\,000$, SMAWK outperforms Bin in our simulation. We also observe that both FirstLocalMin and SpeedUp are of complexity DN. Since FirstLocalMin has a faster row minima operation, it is slightly faster than SpeedUp. Although FirstLocalMin is not optimal in theory, in all of our simulations it found the optimal strategy. Moreover, it is also implementation-wise simpler. We also note that the running time of SMAWK fluctuates as N grows. This is because during the recursion of SMAWK, the process of eliminating entries differs greatly among different instances of the same length N, affecting the actual running time.

We also test the complexity of the three $\Theta(DN)$ algorithms w.r.t. the delay constraint D. Figure 3(b) illustrates the results for a Zipf distribution where D varies from 2 to 10. The plot coincides with the worst case analysis by showing the linearity of $\Theta(DN)$ algorithms as a function of D. For all parameters we checked, SpeedUp outperforms SMAWK, as was expected from the exact details of both algorithms. From now on, we adopt SpeedUp as the optimal implementation.

Second Test: We test the performance of the two non-optimal heuristics DQ and Boundaries. Both heuristics, as expected, are faster than SpeedUp. This indicates a trade-off of optimality for efficiency. For this set of parameters and for all instances we checked, DQ runs slower than Boundaries. We demonstrate the superiority in performance of DQ to Boundaries in Table 1. The table shows the ratio of computed cost over optimal cost for $D = 10$, where N varies from 100 to 1000 with a step of 100, and for data following the four distributions. We note that the performance of Boundaries heavily relies on the initial partition of cells and the user location distribution because of its limited ability to adjust cells among partitions. Thus, for all tests we run on Boundaries, we try three initial partitions (large-suffix, uniform, doubling) and then pick the best paging strategy. We observe that the doubling initial partition performs better on uneven distributions, while the uniform initial partition performs better on even distributions.

In the table, we observe that DQ remarkably outperforms Boundaries on most distributions, but is slightly worse in the uniform random distribution. This is because for uniform random data, an even initial partition is already very close to optimal. Given an almost optimal initial partition, Boundaries performs very well. In fact, for uniform random data, the oblivious Uniform heuristic already performs close to optimal. DQ performs better on inputs for which no good oblivious initial partition could be provided for Boundaries.

Table 1: Competitive Ratio of Algorithms DQ and Boundaries On Four Types of Input Data: $D = 10$, $N = 100, 200, \ldots, 1000$

Zipf, $\alpha = 0.5$			
Algorithm	Best	Worst	Average
DQ	1.0190	1.0238	1.0222
Boundaries	1.1187	1.1215	1.1209
Gaussian, $\sigma = 0.25N$			
Algorithm	Best	Worst	Average
DQ	1.0257	1.0301	1.0291
Boundaries	1.0252	1.0347	1.0319
Step, $s = 10$, $\alpha = 0.5$			
Algorithm	Best	Worst	Average
DQ	1.0486	1.0583	1.0554
Boundaries	1.2366	1.2366	1.2366
Uniform Random			
Algorithm	Best	Worst	Average
DQ	1.0188	1.0215	1.0201
Boundaries	1.0145	1.0180	1.0159

Third Test: To demonstrate the trade-off between paging cost and delay constraint, we test the actual paging cost against the delay constraint D on all four types of data. Figures 3(c), 3(d), 3(e) and 3(f) show the results on all four different types of input data. We find that paging cost drops remarkably by increasing the delay constraint at small values, but when we increase the delay at larger values the drop in paging cost is not as dramatic.

Fourth Test: Figure 3(g) coupled with Figure 3(h) depict a clear trade-off between optimality and running time efficiency. Let *linear* class of algorithms be of complexity $\Theta(N)$. We select the best algorithm in each of the following complexity classes: the *above linear* class (SpeedUp), the *almost linear* class (DQ), the linear class (Boundaries) and the *sub-linear* class (Doubling). Figure 3(g) illustrates the results for the performance and Figure 3(h) illustrates the results for the running time, on data following the Zipf distribution. We emphasize that this trade-off between time complexity and performance repeated itself for all the instances tested by our simulations.

Fifth Test: Finally, we test the performances and running time of the algorithms on real user data. We extract the user location probability vector from the 30 most frequent cells for each of the 996 users as mentioned in Subsection 5.2 and normalize it. We measure the paging cost divided by the optimal paging cost.

In Table 2, we demonstrate the *average*, *worst* and *best* competitive ratio of four classes of non-optimal algorithms. We observe three facts from the table. First, FirstLocalMin always generates an optimal paging strategy for real user data. Second, DQ outperforms the later two heuristics on average optimality in most of the cases. Only when D is comparable to N, the two heuristics performs close to DQ, since there is less flexibility on partitioning the cells. Third, DQ has a significantly better worse case performance than the two faster heuristics. This implies applying DQ in real system offers great optimality and efficiency.

In Table 3, we present the running time of all algorithms on real user data. We measure the best, worst, and average running time for $D = 4$ rounds on each of the 996 users. We run different algorithms tens to tens of thousands of times for accurate time measurement because of the small scale of input data. We observe that the actually running time coincide with our theoretical analysis with minimal variance. DQ is slightly slow for small D.

6. Discussion and Open problems

In this paper, we described, analyzed, and evaluated optimal and heuristic solutions for the problem of paging a mobile user in N cells within D rounds. We improved the complexity of computing optimal solutions from $\Theta(DN^2)$ to $\Theta(DN)$. We then demonstrated a trade-off between time complexity and simplicity of the algorithms.

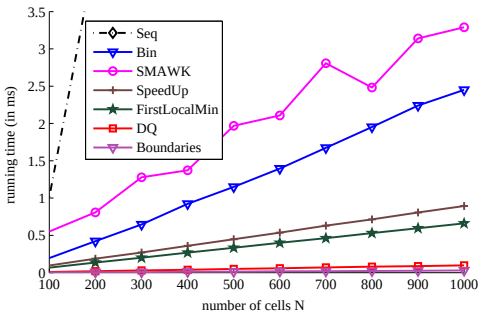
We conclude with some open problems. In this paper, we concentrated on reducing the running time complexity of the implementations. The trade-off between the running time and optimality can be refined and possibly improved. In particular, it is not clear that we have the best $\Theta(N)$ heuristic. Finally, in dynamic systems the values of location probabilities might change frequently. The best solution we have is to re-compute the dynamic programming after

Table 2: Competitive Ratio of Algorithms on Real User Data

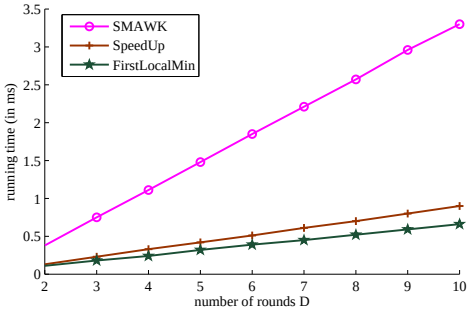
$D = 2$			
Algorithm	Average	Best	Worst
FirstLocalMin	1.0000	1.0000	1.0000
DQ	1.0153	1.0000	2.0000
Boundaries	1.0999	1.0000	10.000
Doubling	1.0999	1.0000	10.000
$D = 3$			
Algorithm	Average	Best	Worst
FirstLocalMin	1.0000	1.0000	1.0000
DQ	1.0356	1.0000	1.3389
Boundaries	1.0868	1.0000	7.0000
Doubling	1.0773	1.0000	5.0000
$D = 5$			
Algorithm	Average	Best	Worst
FirstLocalMin	1.0000	1.0000	1.0000
DQ	1.0252	1.0000	1.2014
Boundaries	1.0462	1.0000	4.0000
Doubling	1.1232	1.0000	1.3333
$D = 10$			
Algorithm	Average	Best	Worst
FirstLocalMin	1.0000	1.0000	1.0000
DQ	1.0192	1.0000	1.3531
Boundaries	1.0163	1.0000	2.0000
Doubling	1.1561	1.0000	1.2381

Table 3: Running time of algorithms on real user data: $D = 4$, $N = 30$, in milliseconds

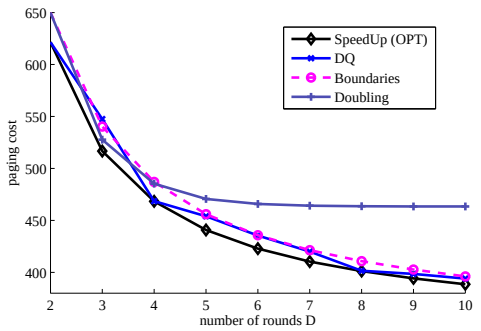
Algorithm	Best	Worst	Average
Seq	0.10409	0.19331	0.18792
Bin	0.06691	0.10409	0.09592
SMAWK	0.07532	0.12597	0.11464
SpeedUp	0.01447	0.01559	0.01511
FirstLocalMin	0.01237	0.01406	0.01368
DQ	0.01781	0.02783	0.02643
Boundaries	0.00391	0.00484	0.00464
Doubling	0.00015	0.00017	0.00016



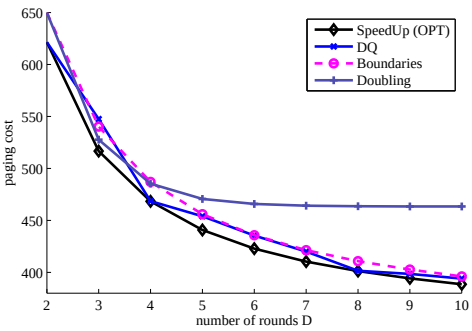
(a) *Running time* vs *number of cells*: $D = 5$, Zipf, $\alpha = 0.5$



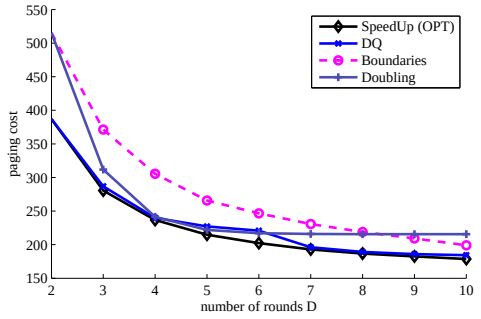
(b) *Running time* vs *delay constraint*: $N = 1000$, Zipf, $\alpha = 0.5$



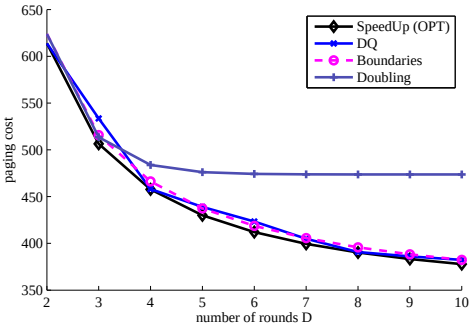
(c) *Cost* vs *delay constraint*: Zipf $\alpha = 0.5$, $N = 1000$



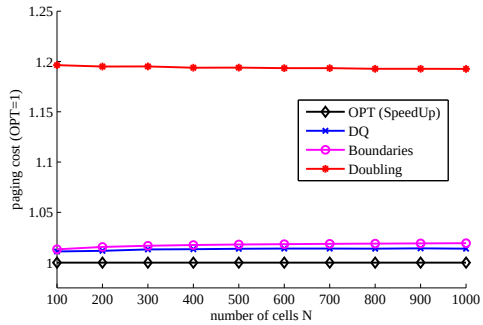
(d) *Cost* vs *delay constraint*: Gaussian $\sigma = 0.25N$, $N = 1000$



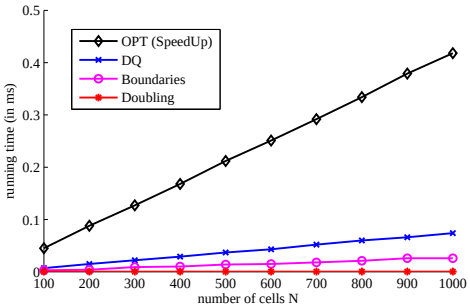
(e) *Cost* vs *delay constraint*: Step $s = 10$, $\alpha = 0.5$, $N = 1000$



(f) *Cost* vs *delay constraint*: Uniform random $N = 1000$



(g) *Optimality* hierarchy: Zipf $\alpha = 0.5$, $D = 10$



(h) *Complexity* hierarchy: Zipf $\alpha = 0.5$, $D = 10$

Figure 3: Simulation results: optimality, complexity and other properties of different algorithms

each change. We are looking for algorithms and data structures that would improve the amortized running time of T executions of the algorithm. That is, we are looking for a total running time which is better than $\Theta(TDN)$.

Acknowledgement

The work of A. Bar-Noy and Y. Feng was partially supported by the NSF program award CNS-0626606; the work of M.J. Golin was partially supported by Hong Kong RGC CERG grant HKUST6312/04E. The authors would also like to thank the Research Computing Group at the Graduate Center of CUNY for providing computational resources.

References

[1] R. Jain, S. Mohan, Two user location strategies for personal communications services, IEEE Wireless Commun. 1 (1) (1994) 42–50.
[2] ANSI/EIA/TIA, Cellular radiotelecommunications intersystem operations, Tech. rep., ANSI/TIA/EIA-41, rev. D (1997).
[3] R.-H. Gau, C.-W. Lin, Location management of correlated mobile users in the UMTS, IEEE Trans. Mob. Comput. 4 (6) (2005) 641–651.
[4] TSG/WG, 3GPP Tech. Spec. 23.012 Location management procedures, version 8.2.0, Section 2 (2009).
[5] C. Rose, R. D. Yates, Minimizing the average cost of paging under delay constraints, Wireless Netw. 1 (2) (1995) 211–219.
[6] D. J. Goodman, P. Krishnan, B. Sugla, Minimizing queuing delays and number of messages in mobile phone location, Mobile Netw. and Appl. 1 (1) (1996) 39–48.
[7] S. Madhavapeddy, K. Basu, A. Roberts, Adaptive paging algorithms for cellular systems, in: Wireless Information Networks, Kluwer Academic Publishers, 1996, pp. 83–101.
[8] B. Krishnamachari, R.-H. Gau, S. B. Wicker, Z. J. Haas, Optimal sequential paging in cellular wireless networks, Wireless Netw. 10 (2) (2004) 121–131.
[9] C. Rose, State-based paging/registration: a greedy technique, IEEE Trans. Veh. Technol. 48 (1) (1999) 166–173.
[10] G. L. Lyberopoulos, J. G. Markoulidakis, D. V. Polymeros, D. F. Tsirkas, E. D. Sykas, Intelligent paging strategies for third generation mobile telecommunication systems, IEEE Trans. Veh. Technol. 44 (3) (1995) 543–553.
[11] W. Wang, I. F. Akyildiz, G. L. Stüber, An optimal paging scheme for minimizing signaling costs under delay bounds, IEEE Commun. Lett. 5 (2) (2001) 43–45.
[12] W. Wang, G. Xue, A cost-minimization algorithm for fast location tracking in mobile wireless networks, Comput. Netw. 50 (15) (2006) 2713–2726.
[13] A. Bar-Noy, Y. Mansour, Competitive on-line paging strategies for mobile users under delay constraints, in: Proc. ACM Symposium on Principles of Distributed Computing (PODC), 2004, pp. 256–265.
[14] A. Bar-Noy, J. Klukowska, Finding mobile data: efficiency vs. location inaccuracy, in: Proc. Annual European Symposium on Algorithms (ESA), 2007, pp. 111–122.
[15] C. Rose, R. D. Yates, Ensemble polling strategies for increased paging capacity in mobile communication networks, Wireless Netw. 3 (2) (1997) 159–167.
[16] R.-H. Gau, Z. J. Haas, Concurrent search of mobile users in cellular networks, IEEE/ACM Trans. Netw. 12 (1) (2004) 117–130.
[17] A. Bar-Noy, G. Malewicz, Establishing wireless conference calls under delay constraints, J. Algorithms 51 (2) (2004) 145–169.
[18] A. Bar-Noy, Z. Naor, Efficient multicast search under delay and bandwidth constraints, Wireless Networks 12 (6) (2006) 747–757.
[19] L. Epstein, A. Levin, The conference call search problem in wireless networks, Theor. Comput. Sci. 359 (1-3) (2006) 418–429.
[20] L. Epstein, A. Levin, A PTAS for delay minimization in establishing wireless conference calls, Discrete Optimization 5 (1) (2008) 88–96.
[21] I. F. Akyildiz, J. Mcnair, J. Ho, H. Uzunalioglu, W. Wang, Mobility management in next-generation wireless systems, in: Proc. IEEE, 1999, pp. 1347–1384.
[22] S. J. Mullender, P. M. B. Vitányi, Distributed match-making, Algorithmica 3 (1988) 367–391.
[23] B. Awerbuch, D. Peleg, Online tracking of mobile users, J. ACM 42 (5) (1995) 1021–1058.
[24] R. E. Burkard, B. Klinz, R. Rudolf, Perspectives of Monge properties in optimization, Discrete Appl. Math. 70 (2) (1996) 95–161.
[25] R. Fleischer, M. J. Golin, Y. Zhang, Online maintenance of k-medians and k-covers on a line, Algorithmica 45 (4) (2006) 549–567.
[26] A. Bar-Noy, Y. Feng, M. J. Golin, Paging mobile users efficiently and optimally, in: Proc. IEEE Conference on Computer Communications (INFOCOM), 2007, pp. 1910–1918.
[27] A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, R. E. Wilber, Geometric applications of a matrix-searching algorithm, Algorithmica 2 (1987) 195–208.
[28] Y. Feng, Paging mobile users in celluar networks with statistical a priori information of their whereabouts, Ph.D. thesis, The Graduate School and University Center, City University of New York (2011).
[29] M. Buchanan, Ecological modelling: the mathematical mirror to animal nature, Nature 453 (2008) 714–716.
[30] L. A. Adamic, B. A. Huberman, Zipf's law and the internet, Glottometrics 3 (2002) 143–150.