Paging Mobile Users Efficiently and Optimally

Amotz Bar-Noy
Computer & Information Science Department
Brooklyn College
Email:amotz@sci.brooklyn.cuny.edu

Yi Feng
Department of Computer Science
The Graduate Center, CUNY
Email: yfeng@gc.cuny.edu

Mordecai J. Golin
Department of Computer Science
Hong Kong U.S.T
Email: golin@cs.ust.hk

[1] ***Abstract*— A mobile user is roaming in a zone composed of N cells in a cellular network system. When a call to the mobile user arrives, the system pages the mobile user in these cells since it never reports its location unless it leaves the zone. The N cells are associated with a probability vector $(p_1, \ldots, p_N)$ where p_i is the probability that the mobile user resides in the ith cell and all the probabilities are independent. A delay constraint paging strategy must find the mobile user within D $(1 \leq D \leq N)$ paging rounds; in each round a subset of the N cells is paged. The goal is to minimize the expected number of paged cells until the mobile user is found. Solutions based on dynamic programming that yield optimal strategies are known. The running time of the known implementations is $\Theta(N^2 D)$. Our first contribution is to improve the running time to $\Theta(ND)$ by proving that the dynamic programming recursive formulation satisfies the Monge property, permitting us to use various dynamic programming speedup techniques. A $\Theta(N)$ heuristic solution is also known. Our second contribution is a heuristic whose running time is $\Theta(N \log D)$. Our heuristic outperforms the known heuristic while running faster for $D << N$. We compare the non-optimal heuristics with the optimal solution demonstrating the tradeoff between optimality and running time efficiency of various solutions.**

I. 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 test searching strategies that are based on some known statistics on the location of the mobile users. We demonstrate a tradeoff between how fast a mobile user is found (*efficiency*) and how much of the wireless links the search strategy consumes (*optimality*).

Consider cellular systems with many cells and many mobile users (*mobiles*) that are roaming 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, future 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., [10], [20], [21]). 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 $C_1, \ldots, C_N$, and that it is possible to page any subset of these N cells in a unit of time (*paging round*) and find out if the mobile is located in one of the cells paged. The trivial solution would page all the N cells in the first and only round (*a blanket search*); this clearly uses the highest possible amount of wireless bandwidth while requiring the lowest possible time. The other extreme is to page the cells sequentially in N rounds terminating once the user is found (*a sequential search*). Without any a-priory 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-priory 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 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 [9], [13], [16], [11] 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

[1]The work of the first and second author was supported by the NSF program award CNS-0626606; the work of the third author was supported partially by Hong Kong RGC CERG grant HKUST6312/04E.

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 $\Theta(N^2D)$. Since the probability vector may change frequently, more efficient solutions are desired. This is the subject of this paper.

Given the parameters N and D, a paging strategy can be viewed as a schedule that partitions the N cells into D disjoint sets. For example, if there are N cells, N even, and the probability of finding the mobile in a cell is $1/N$, then the best 2-round strategy is the one that pages half of the cells in the first round and pages the other half in the second round only if the mobile was not found among the cells that were paged in the first round. The expected number of cells paged by this strategy is $(3/4)N$. This is evidently a gain but at the cost of sometimes conducting two paging rounds. To demonstrate the tradeoff between the number of rounds and the expected number of cells paged, consider the following example. Suppose it is known that a mobile resides either in cell A or in cell B with an equal probability of being in each. There are two possible strategies. In the first strategy, both cells are paged at the same time and, therefore, the mobile is found after one round. However, two cells are always paged. In the second strategy, first cell A is paged and then cell B is paged only if the mobile was not found in cell A. Both the expected number of rounds and the expected number of cells paged in the second strategy is equal to 1.5.

1) Prior art and related work: Modeling uncertainty of locations of mobiles as a probability distribution vector is studied in e.g. [15], which discusses a framework for measuring uncertainty. The paper [12] provides a simple strategy for two paging rounds. The papers [9], [13], [16], [11] describe the optimal solution for any given $N \geq 1$ cells and $1 \leq D \leq N$ rounds. These papers show how to find a D-round paging strategy that locates a mobile with minimum expected number of cells paged using dynamic programming. The papers [16], [11] also study how to minimize the expected number of paged cells given the expected (as opposed to worst-case) delay using relaxation to a continuous model ([16]) or with a weakly polynomial dynamic programming solution [11]. The papers [18], [19] present sub-optimal solutions but are computationally more efficient than the dynamic programming algorithms. 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 [17], [9], [8]. The problem of paging more than one mobile user for a conference call is studied in [5]. The combined cost of reporting and paging is studied by many papers. (See the survey [2] 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. [14], [3].

2) Contributions: The running time of the known algorithms that are based on the dynamic programming solution is $\Theta(N^2D)$ ([9], [13], [16], [11]). This complexity could be too slow for dynamic systems in which the probability vector changes frequently and for large systems where N is very large. Our goal is therefore to design optimal strategies with a faster running time and near optimal strategies with even faster running time.
Our first contribution is to introduce two new algorithms for designing optimal strategies; the running time of these two new algorithms is $\Theta(ND)$, saving a factor of N over the $\Theta(N^2D)$ running time of the previously known optimal algorithms. Our approach is to show that the dynamic programming recursive formulation satisfies the Monge property [6] and the SpeedUp property [7]. Each property enables us to apply a dynamic programming speedup technique. We also describe the original $\Theta(N^2D)$ implementation and another simple algorithm whose running time is $\Theta(N \log ND)$. The four optimal algorithms demonstrate a tradeoff 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 $\Theta(ND)$. The first heuristic is an efficient generalization of the simple solution for the case $D = 2$ (the dynamic programming is another generalization). This heuristic outperforms a known heuristic whose running time is $\Theta(N)$ while running faster for $D << N$. The second heuristic is not faster than one of our optimal solution; nevertheless it is much simpler to describe and implement. Moreover, this heuristic provides the optimal solution for all the instances tested by our simulations. We conjecture that it is actually an optimal solution.

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 tradeoff between optimality and running time efficiency. We tested the solutions on various distributions for the probability vector: Zipf, Gaussian, Uniform, 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 differences, mainly because the worst-case analysis ignores constants and due to some required preprocessing.

3) Paper organization: Section II provides some preliminaries. Section III 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 IV describes and analyzes the running time of several near-optimal heuristics. Section V presents the simulation work that compares the running time and performance of the various optimal and heuristic solutions on a "real" machine. Section VI discusses a more general scope and some open problems.

II. Preliminaries

Assume that the system is looking for a mobile that is roaming in a zone composed of the N cells $C_1, \ldots, C_N$. For

$1 \le i \le N$, 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 $\sum_{i=1}^{N} p_i \le 1$ 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. Without loss of generality assume that $p_1 \ge p_2 \ge \cdots \ge p_N$.

Definition 1: Let $p(S) = \sum_{i \in S} p_i$ for $S \subseteq \{1, \ldots, N\}$ and let $Q_k = p(S)$ for $S = \{1, \ldots, k\}$.

Paging the mobile is conducted in rounds. In each round a subset of the cells is paged until a subset that contains the actual location of the mobile is found. 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 $1 \le D \le N$. In other words, the goal is to find an ordered partition of the cells into D disjoint subsets that minimizes the expected number of paged cells.

Definition 2: Let $cost(\mathcal{S})$ be the expected number of paged cells given an ordered partition $\mathcal{S}$ of the cells, and a location probability vector $(p_1, \ldots, p_N)$. Denote by $OPT(D)$ the cost of the optimal partition for D paging rounds.

It is not hard to see that given the bound $D \le N$, it is always better to page at least one cell in each round. The following observation relates the location probabilities and the order of paging the cells.

Observation 1: The optimal paging must follow the non-decreasing order of probabilities: for any rounds i and j, $1 \le i < j \le D$, all the cells that are paged in round j are associated with smaller or equal probabilities than all the probabilities associated with cells that are paged in round i.

For $D = 1$, all the cells must be paged in the first and only round and the cost of this strategy is $OPT(1) = N$. For $D = N$, the paging strategy is implied by Observation 1. The optimal solution OPT pages cell i in round i for $1 \le i \le N$. In this case, the cost is $cost(OPT) = OPT(N) = \sum_{i=1}^{N} i p_i$. For $D = 2$, the goal is to find a subset of the cells to be paged in the first round. If the mobile is not found, then the rest of the cells must be paged in the second round. By Observation 1, the goal is to find a strategy T_k that pages the cells $\{1, \ldots, k\}$ in the first round for some $1 \le k \le N$. It follows that $cost(T_k) = Q_k k + (1 - Q_k)N$ since with probability $Q_k = \sum_{i=1}^{k} p_i$ the user is in cells $C_1, \ldots, C_k$. In general, for any $1 \le D \le N$, let $\mathcal{S} = \{S_1, \ldots, S_D\}$ be an ordered partition of $1, \ldots, N$, and let n_i be the size of the subset S_i. Then

$$cost(\mathcal{S}) \;\; = \;\; \sum_{i=1}^{D} \left(\sum_{j=1}^{i} n_j \right) p(S_i) \; . \tag{1}$$

Equation (1) is true since with probability $p(S_i)$ the search strategy that is based on this partition paged $\sum_{j=1}^{i} n_j$ cells.

III. Optimal algorithms

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

A. The dynamic programming

We now define the recursive solution formulation to find $OPT(D)$. We ignore the computation of the optimal partition of the cells into D sets 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 table and then working backwards from the optimal solution. This adds a negligible overhead in the running time complexity. We also assume that the vector of probabilities is given in a non-increasing order ($p_1 \ge p_2 \ge \cdots \ge p_N$). Using radix-sort for example, the sorting can be done in $\Theta(N)$ time assuming that the set of all possible values for probabilities is not too large.

Let $1 \le n \le N$ and $1 \le d \le D$ be given. 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: $OPT(D) = h(N, D)$. The initial values for $h(n, d)$ are the two extreme cases where there is only one round or the number of rounds is the same as the number of cells. In these cases,

$$\begin{array}{ll} \forall_{1 \le n \le N} & h(n,1) = n \sum_{i=1}^{n} p_i \\ \forall_{1 \le d \le D} & h(d,d) = \sum_{i=1}^{d} i p_i \; . \end{array} \tag{2}$$

There are two approaches to formulate the optimal recursion for $h(n, d)$. In the first ([9], [13], [16]), 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 ([11]), 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. The proof of the following lemma can be found in the appendix of [11].

Lemma 2:

$$\begin{aligned} h(n,d) \;\; &= \;\; \min_{d-1 \le j \le n-1} \{ h(j, d-1) + n \sum_{i=j+1}^{n} p_i \} \\ &= \;\; \min_{d-1 \le j \le n-1} \{ a(n,d,j) \} \end{aligned} \tag{3}$$

where by definition

$$a(n,d,j) = h(j, d-1) + n \sum_{i=j+1}^{n} p_i \tag{4}$$

is the cost of paging the first j cells in $(d-1)$ rounds optimally and paging the rest of the $n - j$ cells in the last round.

Our objective is to implement the recursion defined in Lemma 2 in a dynamic programming fashion. All of our implementations are based on the following schematic algorithm:

Dynamic Programming Schema:

- Step-1:Compute $h(n,1) = n\sum_{i=1}^{n} p_i$ for $n = 1, \dots, N$.
- Step-d, $2 \le d \le D$: compute $h(n,d)$ for $n = 1, \dots, N$, using Equation (3) given that the values of $h(j, d-1)$ has already been computed for all $d-1 \le j \le n-1$.

Step-d can be reduced to the Row-Minima problem for matrices. The input to the Row-Minima problem is an $N \times N$ matrix A and the objective is to find the N minimum values of all the N rows. Alternatively, the objective is to find the indices in the N rows for which the minimum is obtained. We note that in this formulation, A is only given *implicitly* and not explicitly. That is, we are provided with a function that, given the indices n and j, permits us to calculate $A[n,j]$ in constant time.

Definition 3: Fix d for some $2 \le d \le D$, let A be an $N \times N$ matrix such that $A[n,j] = a(n,d,j)$ for $d \le n \le N$ and $d-1 \le j \le n-1$. Otherwise $A[n,j] = \sum_{i=1}^{n} p_i$ (maximum possible value).

First observe that it is possible to compute $A[n,j]$ in constant time. To achieve this, we first assume that we have, in $\Theta(N)$ time, precomputed all of the prefix sums $Q_k = \sum_{i=1}^{k} p_i$. Then by Equation (4)

$$\begin{aligned} a(n,d,j) &= h(j,d-1) + n\sum_{i=j+1}^{n} p_i \\ &= h(j,d-1) + n(Q_n - Q_j) \end{aligned}$$

can be computed in constant time. Note that since d is fixed, we have already computed $h(j, d-1)$ before we compute $a(n,d,j)$. This observation reduces any implementation of Step-d of the Dynamic Programming Schema to the Row-Minima problem for the matrix A (Definition 3).

Definition 4: For $1 \le n \le N$, let $j(n)$ be the index of the minimum value in row n of the matrix A as defined in Definition 3. In case of ties, $j(n)$ will be the largest index that minimizes $A[n, j(n)]$.

We analyze the running time of any of our implementations for the dynamic programming as follows. We first analyze the running time of the solution to the Row-Minima problem. Then we multiply this running time by a factor of D to get the running time of implementing Step-d of the Dynamic Programming Schema for all values of d. Clearly, Step 1 of the Dynamic Programming Schema can be implemented in $\Theta(N)$ time. We note that implementing this schema requires running time of $\Omega(ND)$ because one must compute $\Omega(ND)$ values of the function $h(n,d)$. The above discussion is summarized in the following proposition.

Proposition 3: Assume that algorithm $\mathcal{X}$ solves the Row-Minima problem for the matrix A in time $\Theta(T(N))$. Then the overall running time of $\mathcal{X}$ is $\Theta(T(N)D)$.

B. Algorithms

We now describe four algorithms that are based on the Dynamic Programming Schema. For each of them, we first show how to find the vector $(j(d), \dots, j(N))$ and thus solve the Row-Minima problem for the matrix A (or a shifted matrix B of A in one of the algorithms). Then we state the overall time complexity that is implied by Proposition 3.

1) The $\Theta(N^2 D)$ straightforward implementation: Algorithm-N^{2}D, Row-Minima(A): Fix $d-1 \le n \le N$. Define $m = \min_{j=d-1}^{n-1} \{A[n,j]\}$ and let $j(n)$ be the largest index such that $A[n, j(n)] = m$.

Finding the minimum in each row can be done in $\Theta(N)$ time and therefore solving the Row-Minima problem on A can be done in $\Theta(N^2)$ time. The following proposition is implied by Proposition 3.

Proposition 4: The running time of Algorithm-N^{2}D is $\Theta(N^2 D)$.

2) The $\Theta(N \log ND)$ implementation: The next lemma implies that the sequence of indices $j(1), \dots, j(N)$ is non-decreasing.

Lemma 5: $j(n_1) \le j(n_2)$ for $1 \le n_1 < n_2 \le N$.

Proof: Let $X = A(n,i) - A(n,j)$ and $Y = A(n+1,i) - A(n+1,j)$ for $d-1 \le i < j \le n-1$. By definition,

$$\begin{aligned} X &= a(n,d,i) - a(n,d,j) \\ &= ((h(i,d-1) - h(j,d-1)) + n\sum_{k=i+1}^{j} p_k \\ Y &= a(n+1,d,i) - a(n+1,d,j) \\ &= (h(i,d-1) - h(j,d-1)) + (n+1)\sum_{k=i+1}^{j} p_k \end{aligned}$$

Therefore,

$$Y - X = \sum_{k=i+1}^{j} p_k \ge 0 \ \Rightarrow \ Y \ge X \ .$$

Since $h(n_1,d) = A(n_1, j(n_1))$ it follows that $A(n,i) > A(n, j(n_1))$ for all $d-1 \le i < j(n_1)$. By the above inequality, this implies that $A(n+1,i) > A(n+1, j(n_1))$ for all $d-1 \le i < j(n_1)$. Consequently, $h(n_2,d) = A(n_2, j(n_2))$ for some $j(n_2) \ge j(n_1)$. ■

Algorithm-N $\log$ ND, Row-Minima(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 5, each stage can be computed in $\Theta(N)$ time because each one of the 2^{i-1} values that is 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 Algorithm-N $\log$ ND for the Row-Minima problem is $\Theta(N \log N)$. The following proposition is implied by Proposition 3.

Proposition 6: The running time of Algorithm-N $\log$ ND is $\Theta(N \log ND)$.

3) The first $\Theta(ND)$ algorithm: We prove a stronger property for the matrix A, called the Monge property, that in particular implies Lemma 5.

Definition 5: An $N \times N$ matrix M satisfies the Monge property [6] if for any $1 \leq n, j < N$:

$$M[n,j] + M[n+1,j+1] \leq M[n+1,j] + M[n,j+1] \ .$$

Lemma 7: The matrix A defined in Definition 3 satisfies the Monge property.

Proof: By definition, proving that $A[n,j] + A[n+1,j+1] \leq A[n+1,j] + A[n,j+1]$ is equivalent to proving that

$$a(n,d,j) + a(n+1,d,j+1) \leq a(n+1,d,j) + a(n,d,j+1) \ .$$

Let

$$X = a(n,d,j) - a(n,d,j+1)$$

and

$$Y = a(n+1,d,j) - a(n+1,d,j+1) \ .$$

We need to show that $X \leq Y$. By definition,

$$\begin{aligned} X &= np_{j+1} + h(j,d-1) - h(j+1,d-1) \\ Y &= (n+1)p_{j+1} + h(j,d-1) - h(j+1,d-1) \ . \end{aligned}$$

Therefore, $Y - X = p_{j+1} \geq 0$. ■

Paper [1] introduces an algorithm that solves the Row-Minima problem for Monge matrices in $\Theta(N)$ time. Paper [6] gives an alternative description of this algorithm (called SMAWK after the authors of [1]) along with many later applications.

Theorem 8: If an $N \times N$ matrix satisfies the Monge property, then all of its row minima can be found in $\Theta(N)$ time.

Algorithm-SMAWK, Row-Minima(A): Apply algorithm SMAWK from [1] to solve the Row-Minima problem for the matrix A.

The following proposition is implied by Theorem 8 and Proposition 3.

Proposition 9: The running time of Algorithm SMAWK is $\Theta(ND)$.

4) The second $\Theta(ND)$ algorithm: While the SMAWK technique does imply a $\Theta(ND)$ algorithm for solving the problem, it can be a bit tricky to be implemented properly. We now describe another property of a shifted version of A that yields a simpler (but still complicated) $\Theta(ND)$ algorithm to solve the Row-Minima problem. The shifted matrix is defined as follows:

Definition 6: Fix d for some $2 \leq d \leq D$, let B be the $N \times N$ matrix such that $B[n,j] = a(n+(d-1), d, j+(d-2))$ for $d \leq n \leq N$ and $d-1 \leq j \leq n-1$. Otherwise $B[n,j] = \infty$.

Definition 7: An $N \times N$ matrix M satisfies the SpeedUp property if

$$M[n+1,j] - M[n,j] = C(n) + D(j)$$

for $1 \leq j \leq N$ and $1 \leq n < N$ where $C(n)$ is a function of n and $D(j)$ is a monotonically decreasing function of j.

Note that this property can be shown to imply the Monge property.

Lemma 10: The matrix B defined in Definition 6 satisfies the SpeedUp property.

Proof:

$$\begin{aligned} & B[n+1,j] - B[n,j] \\ &= a(n+d,d,j+d-2) - a(n+d-1,d,j+d-2) \\ &= (n+d)\left(\sum_{i=j+d-1}^{n+d} p_i\right) - (n+d-1)\left(\sum_{i=j+d-1}^{n+d-1} p_i\right) \\ &= (n+d)p_{n+d} + \sum_{i=1}^{n+d-1} p_i - \sum_{i=1}^{j+d-2} p_i \\ &= C(n) + D(j) \end{aligned}$$

for $C(n) = (n+d)p_{n+d} + \sum_{i=1}^{n+d-1} p_i$ and $D(j) = -\sum_{i=1}^{j+d-2} p_i$. It follows that $D(j)$ is a monotonic decreasing function. Note, that we assume that d is fixed and therefore C is a function of n and D is a function of j. ■

We can show that for matrices that satisfy the SpeedUp property there exists an algorithm, called FGZ, that solves the Row-Minima problem in $\Theta(N)$ time. This algorithm is a modification of the one given in [7] for solving the dynamic program describing placing K medians on a line.

Theorem 11: If an $N \times N$ matrix satisfies the SpeedUp property, then all of its row minima can be found in $\Theta(N)$ time.

Algorithm-SpeedUp, Row-Minima(A): Apply algorithm FGZ to solve the Row-Minima problem for the matrix B.

The following proposition is implied by Theorem 11 and Proposition 3.

Proposition 12: The running time of Algorithm SpeedUp is $\Theta(ND)$.

IV. Non-optimal heuristics

In the previous section, we show how to reduce the running time complexity of algorithms for finding *optimal* strategies from $\Theta(N^2D)$ to $\Theta(N \log ND)$ and then to $\Theta(ND)$. In highly dynamic systems the probabilities might change frequently and for large N even $\Theta(ND)$ might be too slow. Moreover, in all of the optimal algorithms, we found out that the actual constant in the running time is not small enough. We therefore propose several non-optimal heuristics whose running time is smaller than $\Theta(ND)$.

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 [19] 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 [19] in both categories: performance and running time. Our last heuristic has a $\Theta(ND)$ running time and is much simpler to describe and implement than the two optimal $\Theta(ND)$ running time algorithms. Moreover, this heuristic provided the optimal solution for all the instances tested by our simulations. We conjecture that it is actually an optimal solution.

A. 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 another paper by the authors [4].

Algorithm-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.

Algorithm-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, Algorithm-Uniform and Algorithm-LargeSuffix complement each other.

Algorithm-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 Algorithms-Uniform and Algorithm-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 another paper ([4]), we show that Algorithm-Doubling has the best worst-case performance against an adversary that may choose maliciously the values for $p_1, \ldots, p_N$.

B. The Boundaries heuristic

The heuristic described in [19] 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 next lemma.

Lemma 13: 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 boundary condition is

$$\ell_{i+1}(n_{i+1} - 1) \le \sum_{j \in S_i} p_j$$

and the backward boundary condition is

$$s_i(n_{i-1} + 1) \ge \sum_{j \in S_i} p_j \ .$$

Algorithm-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 14: The running time of Algorithm-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 [19] was the best for most of the instances tested by the simulation.

C. The $\Theta(N \log D)$ 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.

Algorithm-$N \log D$: 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 $\log_2 D$.
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 $\Theta(N)$. To make sure that there are $\log_2(D)$ stages, if one of the sets in a partition gets n cells and d rounds where $n < d$, then the algorithm extends this set to have d cells and allocates one round for each cell.

Proposition 15: The running time of Algorithm-$N \log D$ is $\Theta(N \log D)$.

D. The $\Theta(ND)$ heuristic

We now define another property for matrices called the OneMinimum property.

Definition 8: An $N \times N$ matrix M satisfies the OneMinimum property if the global minimum $j(n)$ is the only local minimum in the nth row. That is, $M[n,j]$ is a monotonically non-increasing function of j for $1 \leq j \leq j(n)$ and a monotonically non-decreasing function of j for $j(n) \leq j \leq N$.

We cannot prove that matrix A (Definition 3) satisfies the OneMinimum property. We based the following conjecture on the fact that for all the instances tested by our simulation, the algorithm that relies on the existence of the OneMinimum property always provided the optimal solution.

Conjecture 16: The matrix A defined in Definition 3 satisfies the OneMinimum property.

We note that a stronger property for which $A[n,j]$ is a concave function as a function of j is wrong. It fails on the following parameters: $N=3$, $D=1$, $p_1=\frac{4}{9}$, $p_2=\frac{4}{9}$, and $p_3=\frac{1}{9}$.

Algorithm-OneMinimum, Row-Minima(A): Fix $d-1 \leq n \leq N$. Informally, starting from $j(n-1)$ ($j(0)=1$), find $j(n)$ as the first local minimum in the nth row. That is $A[n,j(n)] \leq A[n,j(n)-1]$ and $A[n,j(n)] < A[n,j(n)+1]$. Lemma 5 allows the search for $j(n)$ to start from $j(n-1)$ and Conjecture 16 would justify stopping after detecting the first local minimum. The search for all values of $j(n)$ takes $\Theta(N)$ time since the minimum search ranges overlap only at their extremes. The following proposition is implied by Proposition 3 and Conjecture 16.

Proposition 17: The running time of Algorithm OneMinimum is $\Theta(ND)$.

V. Simulation

We coded all the algorithms presented in this paper with Matlab 7 Release 14 with SP3. We run the codes on a workstation with the following specifications: AMD Opteron 165 CPU, Dual Core 2GHz×2, 1MB Cache×2, 2GB 400MHz RAM. The operating system of this machine is Windows Server 2003 Standard Edition with SP1. We tested the correctness of the optimal algorithms, the performance of the heuristics, and the time complexity of all the algorithms by running them on different input scales (N and D) as well as different distributions with various parameters. We considered the following distributions for the probability vector ($p_1 \geq p_2 \geq \cdots \geq p_N$): Uniform, Zipf, Gaussian, and Step.

The Uniform distribution is the most natural distribution to test. Each p_i is a random number between 0 and 1 that is normalized by the sum of all the N original random numbers. Formally, $p_i = \frac{q_i}{\sum_{i=1}^N q_i}$ for $1 \leq i \leq N$ where $q_i = rand(0,1)$. The Zipf distribution is motivated by the observation that frequency of occurrence of some event as a function of the rank when the rank is determined by the above frequency of occurrence, is a power-law function. Formally, $p_i = \frac{i^{-\alpha}}{\sum_{i=1}^N i^{-\alpha}}$ for $1 \leq i \leq N$ and a parameter $0 \leq \alpha$. The Gaussian is a continuous distribution that is another common distribution to model the notion of ranking that is based on some frequency of occurrence. Formally, $p_i = \frac{q_i}{\sum_{i=1}^N q_i}$ for $1 \leq i \leq N$ where

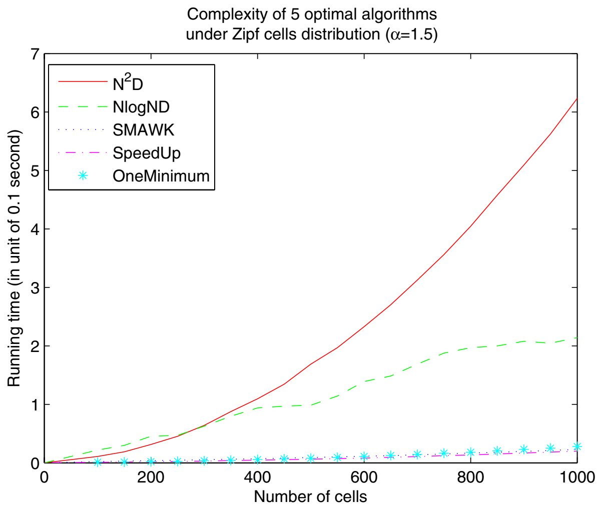

Fig. 1. Complexity of optimal algorithms

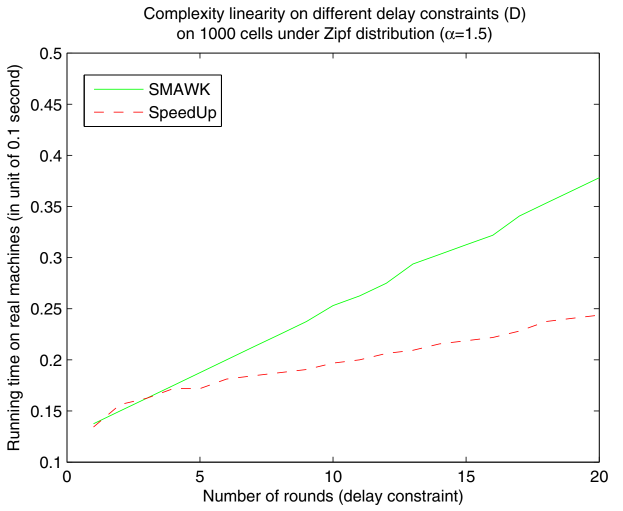

Fig. 2. Linearity of ND algorithms

$q_i = \frac{2}{\sigma\sqrt{2\pi}} \exp -\frac{i^2}{2\sigma^2}$ for a parameter $0 < \sigma$. The Step distribution is a semi-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. This distribution provided us with a counter example to a stronger property for the dynamic programming formulation that would have implied that our simplest $\Theta(ND)$ implementation is optimal. Formally, $p_i = \frac{\alpha^k}{\sum_{k=1}^s \sum_{j=1}^{N/s} \alpha^k}$ for $1 \leq i \leq N$, a step-ratio parameter α, and a step-number parameter s.

The rest of this section reports three sample tests out of the many tests conducted by our simulations. When running time was measured, each test was repeated $100--1000$ times and the average running time is reported. For each test, only a representative figure for some specific parameters is provided.

a) First test: We tested the actual running time of the optimal implementations and the one that is only conjectured to be optimal. Figure 1 illustrates the results for the Zipf

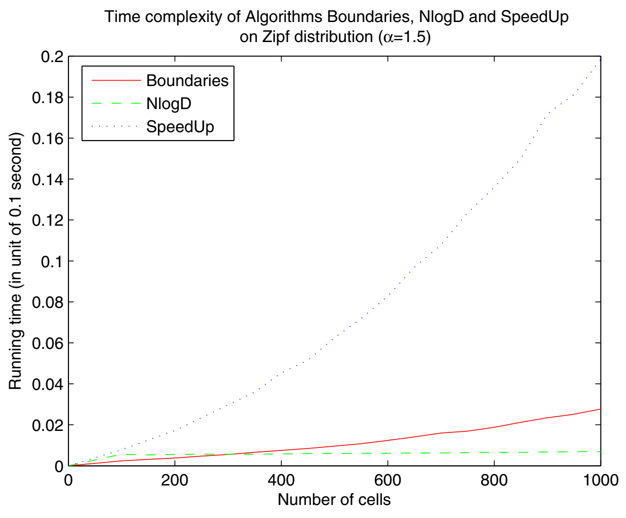

Fig. 3. Complexity of heuristics

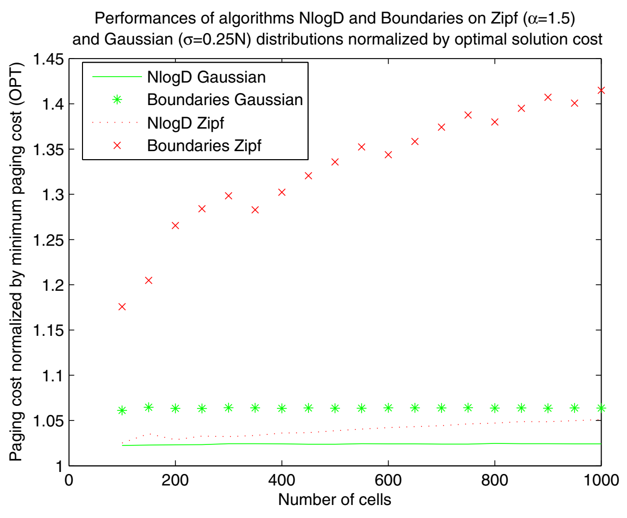

Fig. 4. Optimality of heuristics

distribution with parameter $\alpha = 1.5$ and $D = 10$, where N varies from $N = 100$ to $N = 1000$. Indeed, the plot coincide with the worst-case analysis. We note, that similar behavior was found for all the distributions and for many other parameters. Interestingly, we found out that although Algorithm-OneMinimum is very simple to implement, its actual running time is inferior to the one of Algorithm-SpeedUp. This implies that the complications of the latter save in the running time. Figure 2 compares only the two $\Theta(ND)$ algorithm. It illustrates the results for the Zipf distribution with parameter $\alpha = 1.5$ and $N = 1000$ where D varies from $D = 1$ to $D = 20$. The plot coincides with the worst-case analysis by showing the linearity of both algorithms as a function of D. In all the parameters we checked, Algorithm-SpeedUp outperforms Algorithm SMAWK. This was expected from the exact details of both algorithms. From now on, we adopt Algorithm-SpeedUp as the optimal implementation. We note that there were some irregularities in the linearity property for small D due to the preprocessing.

b) Second test: We compared the running time and performance of the two non-optimal heuristics Algorithm-N $\log$ D and Algorithm-Boundaries with the optimal solution. Figure 3 illustrates the results for the running time for the Zipf distribution with parameter $\alpha = 1.5$ and $D = 10$ where N varies from $N = 100$ to $N = 1000$. Both heuristics, as expected, were faster than the optimal solution. For this set of parameters and for all the instances we checked where $D << N$, we found out that Algorithm-N $\log$ D runs faster than Algorithm-Boundaries. When D is not much smaller than N or when N is small, the $\Theta(N)$ time Algorithm-Boundaries was faster than the $\Theta(N \log D)$ time Algorithm-N $\log$ D. Nevertheless, for small N, the optimal implementation already runs very fast, and when D approaches N already oblivious solutions perform well. We demonstrate the superiority in performance of Algorithm-N $\log$ D to Algorithm-Boundaries in Figure 4. This figure illustrates the performance ratio over the optimal cost for $D = 10$ where N varies from $N = 100$ to $N = 1000$. It shows the results for the Zipf distribution with parameter $\alpha = 1.5$ and for the Gaussian distribution with parameter $\sigma = 0.25N$. The advantage of Algorithm-N $\log$ D is more impressive for the Zipf distribution.

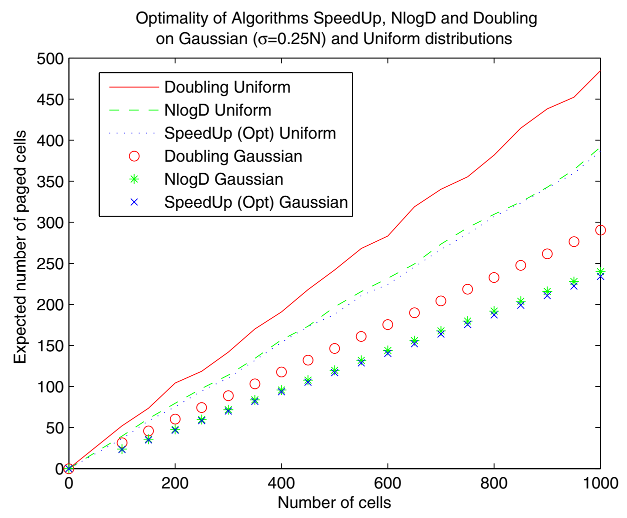

Fig. 5. Optimality of complexity class hierarchy

c) Third test: Figure 4 coupled with Figure 3 depict one of the main contributions of our paper. They show a clear tradeoff between optimality and running time efficiency. Figure 6 coupled with Figure 5 emphasize this tradeoff by adding Algorithm-Doubling to the charts. We selected the best algorithm in each of the following complexity class of algorithms: the *above linear* class of algorithms (Algorithm-SpeedUp), the *almost linear* class of algorithms (Algorithm-N $\log$ D), and the *sub linear* class of algorithms (Algorithm-Doubling). Figure 6 illustrates the results for the running time and Figure 5 illustrates the results for the performance each for the Uniform distribution and for the Gaussian distribution

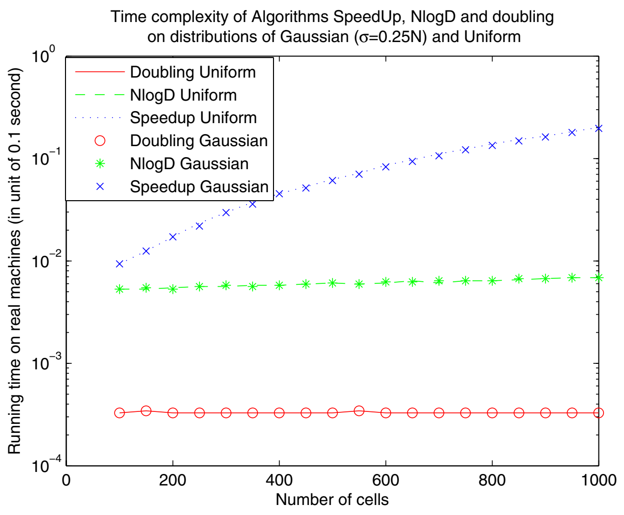

Fig. 6. Complexity of complexity class hierarchy

with parameter $\sigma = 0.25N$. In both figures, $D = 10$ and N varies from $N = 100$ to $N = 1000$. The results speak for themselves. We emphasize again that this tradeoff between time complexity and performance repeated itself for all the instances tested by our simulations.

VI. Discussion and Open problems

We described, analyzed, and tested optimal and heuristic solutions to the problem of paging a mobile user in N cells within D rounds. We improved the complexity of optimal solutions from $\Theta(N^2D)$ to $\Theta(ND)$ and demonstrated the tradeoff between the time complexity and optimality.

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 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). A 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 other applications in wired networks. For example, the task of looking for an information in the Internet or looking for an IP-address of a user.

Open Problems: We were not able to prove the correctness of our conceptually simplest $\Theta(ND)$ implementation of the dynamic programming solution. This algorithm produced the optimal solution for all the instances tested by our simulation. We concentrated on reducing the running time complexity of the implementations. We have ideas how to reduce the space complexity from $\Theta(ND)$ to $\Theta(N)$ for Algorithm-SpeedUp. The tradeoff 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 location probabilities might change their value frequently. The best solution we have is to re-compute the dynamic programming after 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 $\Theta(TND)$.

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber. *Geometric Applications of a Matrix Searching Algorithms.* Algorithmica, 2:195-208 (1987).

[2] I. F. Akyildiz, J. Mcnair, J. S. M. Ho, H. Uzunalioğlu, and W. Wang. *Mobility Management in Next-Generation Wireless Systems.* IEEE Proceedings Journal, 87:1347–1385 (1999).

[3] B. Awerbuch and D. Peleg. *Online Tracking of Mobile Users.* J. of the ACM, 42:1021–1058 (1995).

[4] A. Bar-Noy and J. Klukowska. *Paging Mobile User Under Delay Constraints: Privacy vs. Efficiency.* Submitted Manuscript.

[5] A. Bar-Noy and G. Malewicz, *Establishing Wireless Conference Calls Under Delay Constraints.* Journal of Algorithms, 51:145–169 (2004).

[6] R. E. Burkard, B. Klinz, and R. Rudolf. *Perspectives of Monge Properties in Optimization.* Discrete Applied Mathematics, 70:95–161 (1996).

[7] R. Fleischer, M .J. Golin, and Y. Zhang. *Online Maintenance of K-Medians and K-Covers on a Line.* Proceedings of the 9th Scandinavian Workshop on Algorithm Theory, pp. 102–113 (2004).

[8] R. H. Gau and Z. J. Haas. *Concurrent Search for Mobile Users in Cellular Networks.* ACM/IEEE Transactions on Networking, 12:117–130 (2004).

[9] D. J. Goodman, P. Krishnan, and B. Sugla. *Minimizing Queuing Delays and Number of Messages in Mobile Phone Location.* Mobile Networks and applications, 1:39–48 (1996).

[10] R. Jain, Y. Lin, and S. Mohan. *Location Strategies for Personal Communications Service.* The Mobile Communications Handbook, ed. J. Gibson (CRC Press, 1996).

[11] B. Krishnamachari, R. H. Gau, S. B. Wicker, and Z.J. Haas. *Optimal Sequential Paging in Cellular Wireless Networks.* Wireless Networks 10:121–131 (2004).

[12] G. L. Lyberopoulos, J. G. Markoulidakis, D V. Polymeros, D. F. Tsirkas, and E. D. Sykas. *Intelligent Paging Strategies for Third Generation Mobile Telecommunication Systems.* IEEE Transactions on Vehicular Technology, 44:543-553 (1995).

[13] S. Madhavapeddy, K. Basu, and A. Roberts. *Adaptive Paging Algorithms for Cellular Systems.* Wireless Information Networks: Architecture, Resource Management and Mobile Data 83–101 (1996).

[14] S. J. Mullender and P. B. M. Vitányi. *Distributed Match-Making.* Algorithmica, 3:367–391 (1988).

[15] C. Rose and R. Yates. *Location Uncertainty in Mobile Networks: a Theoretical Framework.* IEEE Communications Magazine, 35 (1997).

[16] C. Rose and R. Yates. *Minimizing the Average Cost for Paging Under Delay Constraints.* Wireless Networks, 1:211–219 (1995).

[17] C. Rose and R. Yates. *Ensemble Polling Strategies for Increased Paging Capacity in Mobile Communication Networks.* Wireless Networks, 3:159–167 (1997).

[18] W. Wang and I. F. Akyildiz. *An Optimal Paging Scheme for Minimizing Signaling Costs Under Delay Bounds.* IEEE Communication Letters, 5:43–45 (2001).

[19] W. Wang, I. F. Akyildiz, and G. L. Stüber. *Effective Paging Schemes with Delay Bounds as QoS Constraints in Wireless Systems.* Wireless Networks, 7:455–466 (2001).

[20] ETSI/TC. *Mobile Application Part (MAP) Specification, version 4.8.0.* Technical Report, Recommendation GSM 09.02 (1994).

[21] EIA/TIA. *Cellular Radio-Telecommunications Intersystem Operations.* EIA/TIA Technical Report IS-41 Revision C (1995).