ISSN 0280–5316
ISRN LUTFD2/TFRT--5636--SE

Sequential Paging for Moving Mobile Users

Jonas Levin

Department of Automatic Control
Lund Institute of Technology
February 2000

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
February 2000

Document Number
ISRN LUTFD2/TFRT--5636--SE

Author(s)
Jonas Levin

Supervisor
Björn Wittenmark
Vikram Krishnamurthy

Sponsoring organization

Title and subtitle
Sequential Paging for Moving Mobile Users.

Abstract
In this thesis, some methods for efficient paging for mobile systems have been investigated. Compared to the conventional method, the methods proposed here increase the mobile stations discovery rate while decreasing the signaling load between the mobile switching centers and the mobile stations. As the cell sizes shrinks, the probability that the mobile stations moves between the cells during the paging process gets higher. These methods takes this probability into consideration and are based on the case when the movement between the cells are described as a Markov model.
One of the algorithms is called the pqup-algorithm. This method works well under both heavy traffic and light traffic. The main method is the POMDP-algorithm. A POMDP is a generalisation of a Markov decision process that allows for incomplete information regarding the state of the system. The POMDP-algorithm does not quite work yet and the method is going to be investigated further. The results so far is presented in this thesis. The methods are fully compatible with current cellular networks and requires small amount of computational power in the mobile switching centers.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
43

Recipient's notes

Security classification

Contents

1 Introduction

In early mobile radio systems the design objective was to achieve a large coverage area by using a single, high powered, transmitter with an antenna mounted on a tall tower. Although this approach did achieve a very good coverage it also meant that it was impossible to reuse the same frequencies throughout the system. This was because any attempt to reuse frequencies would result in interference. Another important issue is the unnecessary high transmitted power. It was a waste of energy for the system and it also resulted in shorter lifelength for the battery of the mobile stations (MS's).

The cellular concept was a major breakthrough in solving the problem of spectral congestion and user capacity. This concept is a system level idea which calls for replacing a single, high power, transmitter (large cell) with many, low power, transmitters (small cells). Each base station (BS) is allocated only a portion of the total number of channels available to the entire system. Neighboring BS's are assigned different groups of channels so that the interference between the BS's (and between the MS's) is minimized.

A more efficient use of limited wireless resources requires much smaller cells (microcells and picocells, described in [1]). Tracking the MS's will become a challenging task as the cell sizes shrink and the number of cells increases. With the expansion of both the number of MS's in service and the number of services available to these users, the radio spectrum increases. In order to make more bandwidth available for voice and data traffic we want, of course, to reduce the signaling load between the MS's and the BS's. This thesis describes some methods to reduce the signaling load associated with paging and location area updating by deploying cellular networks with more intelligent mobile tracking and location management techniques.

Chapter 2 gives an introduction of how a cellular system keeps tracks of the mobile users and how it handles pagings to the mobile stations. In chapter 3 the algorithms for more efficient paging, that are investigated in this thesis, are described. Chapter 4 contains some simulations and results from these, when the algorithms are used for the paging process. The conclusions of this thesis are in chapter 5.

The algorithm called the *pqup*-algorithm is an algorithm that was developed during this thesis and it's a development from the so called *pq*-algorithm that R. Rezaiifar and A.M. Makowski presents in [6]. A description of the *pq*-algorithm is also included in this thesis. The *pqup*-algorithm improved the results significantly, compared to the conventional method. Another al-

gorithm is called the $POMDP$-algorithm. $POMDP's$ has been developed by A.R. Cassandra who has done a lot of research in this area. Running a $POMDP$ gives the optimal solution to the given model. So far there is some problems when running the $POMDP$ on the paging systems. These are described in this thesis.

I would like to take this opportunity to say thanks to my supervisor in Melbourne, Dr. Vikram Krishnamurty, for helping with the research that I've been doing for this thesis. I would also like to say thanks to my supervisor in Lund, professor Björn Wittenmark, for helping me with the contacts with the "University of Melbourne" and for helping me writing this thesis.

2 Description of the system

2.1 Tracking a mobile user

A cellular system has to keep track of the whereabouts of the MS's within the cellular system, in order to be able to deliver incoming calls. In practice, the MS may inform the system of its whereabouts frequently e.g. on power up, just before shut down, or when it crosses special boundaries known as location areas (LA's). An LA is simply a group of cells. Each cell has one base station (BS) and the size of one cell is given by the coverage area of its BS. The size of an LA or the number of cells in it varies.

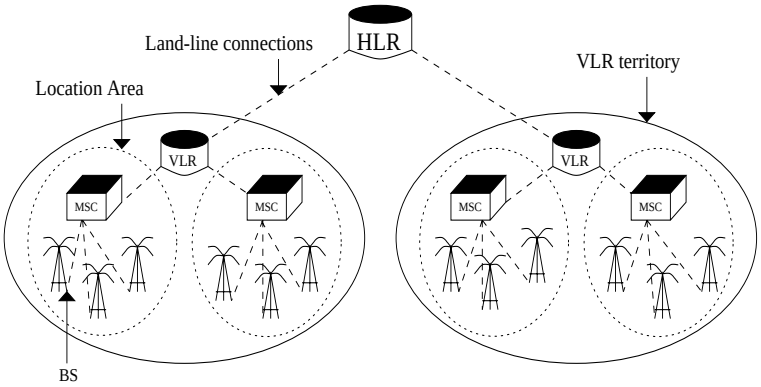

Figure 2.1. Structure of the mobility management system.

Each LA has a Mobile Switching Center (MSC) to where it belongs and each BS periodically broadcasts a unique ID number that specifies the LA/MSC to which it belongs. The MS stores this ID in its memory and when it moves to a new BS it compares the ID broadcasted by the new BS and compares it with the one stored in its memory. If they are different the MS has moved to a new location area and it has to tell the system about its new location. The MS handles this by sending a registration message to the new BS which in turn forwards the message to its MSC (see fig. 2.1).

The information about the MS's location is stored in two types of databases: Home Location Registers (HLR's) and Visitor Location Registers (VLR's). A VLR database is usually smaller than an HLR and isolates the small scale movements of the MS's. As long as an MS travels among the LA's covered by the same VLR, no query is necessary from the HLR because the profile of the MS and other related information already exist in the cache. If an MS leaves its current LA and enters another one handled by the same VLR the MSC will inform the VLR anyway. By doing this the VLR can - given the ID number of an MS - specify the LA where the MS resides.

2.2 Paging

In current cellular networks - when paging an MS - the final step in the call setup procedure starts with the MSC sending a page request (PR) to *all* of the BS's, in the LA where the MS resides, in order to find the MS. However, by monitoring the traffic pattern inside each cell in an LA we can obtain information about the system such as probability mass function and movement between cells. With this information available it thus seems to be a waste of energy to page all the cells within the LA in order to find the MS. For instance, a sequential search plan that starts with the cells with a higher probability of discovery should be more efficient.

The PR's for different MS's arrives to the MSC in order to be processed in a first-in first-out (FIFO) matter. Let N be the number of cells, i.e. the number of BS's, handled by the MSC and let L be the number of paging channels to each BS. This means that at most L MS's can be paged simultaneously in each cell by the MSC. The time epochs at which a PR can be sent to a BS is refered to as a Paging Cycle (PC).

In a first approximation, the paging process is assumed to be perfect in that if a PR is sent to the BS where the MS resides, and there are available paging channels, it is always paged successfully and discovery takes place within that same PC.

Paging with the conventional method goes as follows. First take L PR's from the head of the main queue (if there are L PR's waiting) in the MSC and send these to each the BS's operating under the MSC. With this method exactly L MS's will be discovered per PC and exactly N pagings will be done per MS before discovery. This is a reliable method but it is a waste of system resources. Why is demonstrated in the following example.

Example 2.1
Let say we have a small LA that contains 3 cells and that we have 2 paging channels per BS, i.e. $N = 3$ and $L = 2$. Say we have 4 PR's that are waiting at the head of the main queue and denote these A, B, C and D, respectively, and say that MS A and B are in cell 1, C is in cell 2 and D is in cell 3 (see fig 2.2).

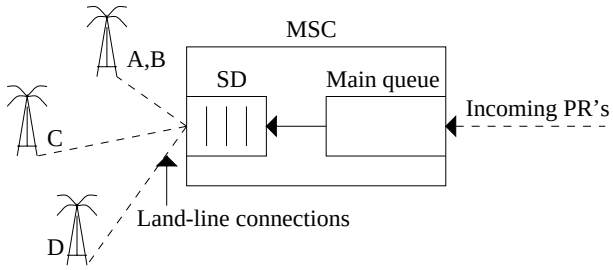

Figure 2.2. Structure of an MSC with a smart distributor.

Under the conventional algorithm, in the first PC, PR A and B are being sent to all three cells. In the second PC PR C, and D are being sent to all the cells.
Now let's have a look at how an "ideal" algorithm would have worked. In such a system we would need some kind of distributor that distributes the PR's among the cells. This distributor will hereafter be refered to as a smart distributor (SD). The SD buffers the PR's from the head of the main queue and distributes these among the BS's. In this case it would buffer all the 4 PR's from the main queue, in the first PC, and send PR A and B to cell 1, PR C to cell 2 and PR D to cell 3.
Under the conventional algorithm each MS are paged 3 times (i.e. one time in each cell) as compared to only once in the ideal one and 2 MS's are being found per PC as compared to 4 in the ideal one. Of course, an ideal algorithm would have to know the exact location of each of the MS's so the comparison is therefore not really fair. However, the discussion already points to the possibility of improving the conventional paging algorithm.

3 Algorithms

As the cell sizes shrinks (e.g. *microcells* and *picocells*) the probability that the MS is moving between the cells during the paging process, gets higher. A good paging algorithm should take these movement probabilities into consideration. This chapter describes some algorithms that can be used for paging.

3.1 Model definition

An MS i placed randomly in one of N cells at time step 1 and then, at each time step, it moves between the N cells according to a Markov process.

Let $X = \{1, 2, ..., N\}$ denote a finite state set and let x_k be the state of the MS at time step k, i.e. which cell it remains in at time step k. Let $A = \{1, 2, ..., N\}$ denote a finite action set where action i corresponds to that cell i is being paged. The decision maker that performs these actions will be refered to as the *agent*. Further let $p_k(i)$ denote the probability of the MS being in cell i at time step k. By p_k we mean the N-vector with element i equal to $p_k(i)$ for $1 \leq i \leq N$. Let P denote the transition probability matrix i.e. an NxN matrix with element ij equal to

$$p_{ij} = Pr(x_{k+1} = j | x_k = i). \tag{3.1}$$

Let $q(i)$ denote the probability of being blocked in cell i i.e. the probability that all the paging channels are occupied in cell i when sending a PR to cell i. By q we mean the N-vector with element i equal to $q(i)$ for $1 \leq i \leq N$. Finally, let $\Theta = \{found, missed, blocked\}$ denote a finite observation set and let θ_k denote the observation at time step k. The observations are recieved according to the known probabilities

$$r^a_{jl} = Pr(\theta_k = l | x_k = j, a_k = a). \tag{3.2}$$

The observation *found* corresponds to that the agent is not blocked and the MS is in the cell paged. The observation *missed* corresponds to that the agent is not blocked but the MS is *not* in the cell paged. Finally the observation *blocked* simply means that the agent is blocked in this PC. Consider all vectors to be column vectors.

3.2 The POMDP-algorithm

A partially observable Markov decision process (POMDP) is a generalization of a Markov decision process (MDP) that allows for incomplete information regarding the state of the system. The agent must, at each time step, choose

an action (i.e. in this case which cell to page), based only on the incomplete information at hand. In general, when using POMDP, one of the possible actions might be to expend resources to gather additional information about the system. Here, we have N actions were each action results in one of three observations. Running a POMDP for a specific model gives us the optimal solution for that model.

The process is initiated with a known probability distribution, p_1, among the cells. Let $H_k = \{p_1, a_1, \theta_1, a_2, \theta_2, ..., a_{k-1}, \theta_{k-1}\}$ denote this initial distribution appended with the history of actions and messages recieved up to time k. At the beginning of time step k, H_k contains all the information that the agent can use to assess the state of the system. If, based on this information, the agent chooses action a_k, the following sequence of events is initiated:

1. A real-valued reward $g(x_k, a_k)$ is recieved for action a_k if the state of the system is x_k.

2. The system transits to a new state x_{k+1}, according to the transition probability matrix P.

3. An observation θ_k is received according to the known probabilities $r^{a_k}_{x_k\theta_k}$.

4. Time increments by one, $H_{k+1} = H_k \cup \{a_k, \theta_k\}$, the agent chooses action a_{k+1} and the process repeats.

The system evolves, according to the sequence outlined above, through $T \leq \infty$ time epochs, where T is called the *horizon*. If $T < \infty$ an additional salvage value $\alpha(i)$ is received at the beginning of time epoch $T + 1$ if $x_{T+1} = i$. Let α denote the N-vector with components $\alpha(i), 1 \leq i \leq N$. A search *policy* is an algorithm which specifies the sequence of cells to search. The agent seeks a *policy* that maximizes the expected net present value of the time stream of rewards occured during the process:

$$E\{\sum_{k=1}^{T} \beta^{k-1} g(x_k, a_k) + \beta^T \alpha(x_{T+1})\}, \tag{3.3}$$

where $0 \leq \beta \leq 1$ is a *discount factor*. If $T = \infty$ we require $\beta < 1$. When β is large, future rewards play a larger role in the decision process. As β tends to zero, the agent seeks to maximize the reward for only the next time step with no regard for future consequences. Simulations have shown that a β of about 0.25! is usually good for our paging models.

Let $R^a(\theta)$ denote the NxN diagonal matrix with element (j, j) equal to $r^a_{j\theta}$ and let e denote an N-columnvector with a 1 in each component. Define

$$\sigma(\theta|p,a) = p'PR^a(\theta)e \tag{3.4}$$

$$T(\theta,p,a) = \frac{p'PR^a(\theta)}{\sigma(\theta|p,a)}. \tag{3.5}$$

Then σ is the probability of obtaining observation θ given distribution p and action a and T is, by Baye's rule, the posterior probability vector p_{k+1} given the prior probability vector p_k. Given this, then for $T < \infty$, the problem of finding a policy that maximizes (3.3) can theoretically be solved (Lovejoy [2]) via the dynamic programming recursion

$$\begin{cases} V(p_{T+1}) = p'_{T+1}\alpha \\ V(p_k) = \max_{a\in A}(p'_k g^a + \beta \sum_{\theta\in\Theta} \sigma(\theta|p_k,a)V(T(\theta,p_k,a))) & ,k = 1,...,T, \end{cases} \tag{3.6}$$

where g^a denotes the N-vector with components $g(i,a), 1 \leq i \leq N$. Any *policy* that for each time epoch k maps p_k into a maximizing argument in eq. (3.6) will be an optimal policy. $V(p_k)$ has the interpretation of the optimal value function for the analogous problem with *horizon* $T-k+1$.

3.2.1 Rewards

As mentioned above, the agent seeks a *policy* that maximizes the stream of rewards. There are several ways of choosing the reward but not many will give the optimal *policy*. We can give the agent a reward every time it finds the MS, or every time it gets close to the MS. We might also introduce penalties by simply put a minus sign in front of the rewards, e.g. give the agent a penalty every time it gets observation *missed*.

In [3], Eagle gives an account of optimal search in a 2 observation case. The two observations in Eagles paper is *found* and *not found*. If the object is in the searched box i the agent miss it with probability $q(i)$ and gets observation *not found*. Eagles reward function is

$$V(p_k,i) = \max_{a\in A_i}(p_k(a)*(1-q(a))+(1-(1-q(a))*p_k(a))*V(T_a(p_k),a)), \tag{3.7}$$

where T_a is p updated for an unsuccesful search (i.e. observation *not found*) in cell a, according to Bayes's rule (3.5) and $A_i \subseteq A$. Comparing the first part of eq.(3.7) with the first part of eq. (3.6) gives that the rewards g^a should be chosen to

$$g^a = (0, 0, ..., 0, 1 - q(a), 0, ..., 0, 0)' \tag{3.8}$$

i.e. $1-q(a)$ in the $a : th$ position and 0 elsewhere. If we look at the definition of g^a (eq. (3.6)) this has the interpretation that each time we search the cell where the MS resides the agent gets a reward. This reward is equal to the probability of detecting the MS, given that the MS is in the searched cell. That makes sense because cells with lower $q(a)$, i.e. blocking probability, should be searched more often. This is for the 2 observation case (*found* and *not found*), but g^a is independent of the observations. It only depends on the current state and the current action. This means we should be able to use the same g^a in the 3 observation case.

Choosing the reward as above is actually the same as giving the agent a reward of 1 each time it finds the MS. The proof of this is available in appendix A.

3.2.2 Solving POMDP's

When trying to solve eq. (3.6), the fundamental problem is that at each iteration of the dynamic programming recursion, V_k must be evaluated for each possible p_k-vector. This is an infinite set. All feasible numerical methods involve reducing the number of distributions p_k considered, for each k, to a finite number.

Tony Cassandra has done a lot of research in POMDP's and in [4] he describes the POMDP approach to find optimal or near optimal control strategies for partially observable stochastic environments, given a complete model of the environment. On his webpage there are POMPD code available to solve POMDP's. These programs are all written in C and the webpage is: *http://www.cs.brown.edu/research/ai/pomdp/index.html*. Methods that are available with these programs includes, among others, the "linear support algorithm" (described by Lovejoy in [2]), the "witness algorithm" and "incremental pruning" (described in [5]).

How to specify the POMDP-files is described in Appendix B.

3.3 The *pq*-algorithm

In [6] Rezaiifar and Makowski suggests an algorithm that they call the *pq*-algorithm. They show that this algorithm is optimal in the 2 observation case (i.e. *found* and *not found*) and when the MS doesn't move between the cells during the paging process, and they use this algorithm for the 3 observation

case when the MS doesn't move. The case when the MS doesn't move will be refered to as the stationary case.

The *pq*-algorithm is defined according to a slightly different model. The algorithm is then modified to fit to our model.

The MS is placed randomly in one of N cells at time step 1 where it remains throghout the paging process. The distribution among the cells is $p_1(i)$ for $i = 1, ..., N$ and the observations are *found* and *not found*. Let $q(i)$ for $i = 1, ..., N$ be the probability that we get observation *not found* on a single PR to cell i, given that the MS is in cell i. This q-vector defined here is identical to the q-vector defined in chapter 3.1, because the probability that we miss the MS, given that we page the cell where the MS resides, is simply the probability that there are no available paging channels.

The probability of failing to detect the object on the first $j-1$ looks in box i and of succeeding on the j:th, given that the object is in cell i, is

$$(1 - q(i)) * q^{j-1}(i), \quad i = 1, ..., N, \quad j = 1, 2, ... \tag{3.9}$$

Let $r(i, k)$ denote the number of PR's out of the k first PR's that are sent to cell i. Then the *pq*-algorithm is: choose the $(k+1) : st$ PR according to

$$a_{k+1} = \arg\max_{a \in A}(p_1(a) * (1 - q(a)) * q(a)^{r(a,k)}), \quad k = 0, 1, 2, 3, ... \tag{3.10}$$

This is for the 2 observation case. When applying this to the 3 observation case, Rezaiifaar and Makowski suggest the following in [6]:

- If the agent recieves observation *found*, then simply stop searching.
- If the agent recieves observation *blocked* in cell i in the $k : th$ PC, then increment $r(i, k)$ by one.
- If the agent recieves observation *missed* in cell i in the $k : th$ PC, then set $r(i, k)$ to infinity.

The last point might need some explanation. Setting $r(i, k)$ to infinity has the interpretation that the cell has been paged infinite times. Thus it is very unlikely for the MS to be in that cell. This is for the non-moving case.

Seperate observation *not found* to the observations *blocked* and *missed* actually invalids eq. (3.9) because the probability of being blocked in cell i on one paging attemp now, given that we have been blocked there an arbitrary

number of times before is still $q(i)$. This is because we now can observe if we get blocked or not in each cell, compared to before when we only knew the probability that we were blocked. The *pq*-algorithm still works well, because of the way it spreads out the PR's among the cells (which leads to more paging channels available among the cells) and still keeps the number of PR's per MS low. However, it's not optimal anymore when we seperate the observations.

3.4 The *pqup*-algorithm

This is an algorithm that was developed during my research and it's based on the *pq*-algorithm. In order to be able to apply the *pq*-algorithm to the moving case, we have to somehow keep track of the mobile user. One way of doing this is to update the p-vector after each time step. We can do this by using Baye's rule (eq. (3.5)). However, this equation calculates the p-vector in the *filter* case. Filter means that the order of events in one time step is as follows: action→move→observation. For example, if we get observation *missed* in cell i, then $p(i) = 0$ according to eq. (3.5). We can't use this p in the *pq*-algorithm, because the MS moves before next action is taken. The order we really want for this algorithm, in one time step, would be: action→observation→move. This is called the *predictor* case. This can be obtained by Baye's rule by simply change the order between P and $R^a(\theta)$. Thus,

$$\sigma_p(\theta|p,a) = p'R^a(\theta)Pe \tag{3.11}$$

$$T_p(\theta,p,a) = \frac{p'R^a(\theta)P}{\sigma(\theta|p,a)}, \tag{3.12}$$

will give us the p-vector that we want.

Now, we can't set $r(i,k)$ to infinity, when we get observation *missed* as in the non-moving case, because then we will never page that cell again. We must go back to the original definition of r, i.e. $r(i,k)$ denotes the number of PR's out of the k first PR's that are sent to cell i. The updated p-vector will take care of the different observations.

By using eq. (3.11) and eq. (3.12), we can calculate p_{k+1}, given p_k, a_k and θ_k. Since we stop searching when we get observation *found*, we only have to calculate p_{k+1} for the observations *missed* and *blocked*. The observation matrix $R^a(\theta)$ (defined on page 7) for observation *missed* and action i is

$$R^{a_k=i}(missed) = \begin{pmatrix} 1-q(i) & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1-q(i) & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 1-q(i) & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 1-q(i) \end{pmatrix} \tag{3.13}$$

i.e. 0 on the i:th position and $1-q(i)$ elsewhere on the diagonal. Inserting this in eq. (3.11) yields

$$\begin{aligned} \sigma_p(missed|p_k, a_k = i) &= p' R^{a_k=i}(missed) P \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= p' R^{a_k=i}(missed) \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = p' \begin{pmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ 1 \\ \vdots \\ 1 \end{pmatrix} (1-q(i)) \\ &= (p_k(1) + \cdots + p_k(i-1) + 0 + p_k(i+1) + \cdots + p_k(N))(1-q(i)) \\ &= (1-p_k(i))(1-q(i)) \end{aligned} \tag{3.14}$$

Using this σ_p in eq. (3.12) gives us the posterior p-vector for observation *missed*.

$$\begin{aligned} p'_{k+1} = T_p(missed, p_k, a_k = i) &= \frac{p'_k R^{a_k=i}(missed) P}{\sigma_p} \\ &= \frac{(p_k(1), \ldots, p_k(i-1), 0, p_k(i+1), \ldots, p_k(N))(1-q(i))P}{(1-p_k(i))(1-q(i))} \\ &= \frac{(p_k(1), \ldots, p_k(i-1), 0, p_k(i+1), \ldots, p_k(N))P}{1-p_k(i)} \end{aligned} \tag{3.15}$$

For observation *blocked* the observation matrix is

$$R^{a_k=i}(\theta_k = blocked) = \begin{pmatrix} q(i) & 0 & \cdots & 0 & 0 \\ 0 & q(i) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & q(i) & 0 \\ 0 & 0 & \cdots & 0 & q(i) \end{pmatrix} \tag{3.16}$$

Inserting this in eq. (3.11) gives us

$$\sigma_p(blocked|p_k, a_k = i) = p' R^{a_k=i}(blocked) \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = p' \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} q_k = q_k, \tag{3.17}$$

and finally inserting this in eq. (3.12) gives us the posterior p-vector for observation *blocked*

$$p'_{k+1} = T_p(blocked, p_k, a_k = i) = \frac{p'_k R^{a_k=i}(blocked) P}{\sigma_p} = \frac{p'_k q_k P}{q_k} = p'_k P. \tag{3.18}$$

3.5 Estimating the q-vector

As mentioned before the probability of being blocked in cell i is the same as the probability that all the paging channels to cell i are occupied when sending a PR to the cell. The SD-buffer has a size of K and under one PC the SD distributes the PR's (maximum K) among the N cells, according to some algorithm. If more than L PR's are being sent to a specific cell, in one PC, then some PR's will be blocked. This means that the true blocking probabilities, lets call these probabilities q_{true}, will depend on the algorithm used, i.e. how it distributes the PR's among the cells. In turn the algorithms used here have the q-vector as an in-parameter. The distribution among the cells depends on the choice of this q-vector. This means that the q_{true}-vector will depend on the choice of q-vector. Of course, we want q to be the same as q_{true}. The question is, how do we choose q to obtain this?

What we can get is an estimate of the q_{true}-vector. Let's call this estimated vector $\hat{q}_{true}$ and this has the form

$$\hat{q}_{true} = q_{true} + \epsilon, \tag{3.19}$$

where ϵ is a random vector representing the estimation error. The $\hat{q}_{true}$-vector is computed as follows. Fix the p-vector and the q-vector and let the

algorithm run for a *paging window* which consists of W PC's. Pick W large enough so that the system reaches steady state. Then, for $i = 1, ..., N$, let $\hat{q}_{true}(i)$ be the fraction of times that a PR has been denied access to a paging channel in cell i within the paging window. It can be shown that this is the maximum likelihood estimation of the q_{true}-vector.

The problem is to pick a vector q such that

$$q(i) - \hat{q}_{true}(i) = 0, \quad i = 1, ..., N. \tag{3.20}$$

The mapping $q \rightarrow \hat{q}_{true}$ is virtually impossible to find. There are two many parameters that affects the mapping, e.g. the number of paging channels, the number of cells, the movement of the MS's, the size of the SD-buffer, the algorithm used, etc. So when solving eq. (3.20), it calls for some kind of numerical approximations.

What we have at our disposal is the input q and the output $q - \hat{q}_{true}$. We don't have the derivative of the function. This limits the numerical methods we can use. The method used here is the Robbins-Monro algorithm [7] which, when trying to solve eq. (3.20), yields

$$q_{n+1} = q_n - b_n * (q_n - \hat{q}_{true,n}), \quad n = 1, 2, ... \tag{3.21}$$

and $\{b_n, n = 1, 2, ...\}$ is a sequence of positive numbers. As part of conditions for convergence it is required that the gain sequence b_n satisfies $\sum_{n=1}^{\infty} b_n = \infty$ and $\sum_{n=1}^{\infty} b_n^2 < \infty$. The usual candidate is $b_n = n^{-1}$. When eq. (3.21) is defined like this it is required that $q(i) - \hat{q}_{true}(i)$ has a positive derivative. This is actually the case, because a higher $q(i)$ means a higher blocking probability in cell i and the algorithm used want's to page cell i less. This leads to that more channels will be available in cell i and $\hat{q}_{true}$ will be smaller. Hence, $q(i) - \hat{q}(i)_{true}$ will be higher and the derivative is positive.

4 Numerical results

To be able to compare the different algorithms we introduce two performance measures which captures the effiency of the paging. These are

1. $F \equiv$ the expected number of MS's found per PC.
2. $S \equiv$ the expected number of times that an MS is paged before it is found.

An efficient algorithm is one that has a higher F and a lower S than the conventional method.

In the simulations the PR's are generated according to a Poisson process with rate λ (PR's/PC). For the conventional method F and S are quite straightforward to obtain. Each MS is paged exactly N times, since we send one PR to each of the BS's for every MS. Therefore

$$S_{conv} = N.$$

F_{conv} is a bit more difficult to compute. Exactly L MS's can be paged simultaneosly with the conventional method. If we assume heavy traffic so that there is more than L MS's to page, then we will discover exactly L MS's in that PC. Assume that λ is constant long enough for the system to reach steady state. Then if λ is bigger than L (heavy traffic) the expected number of MS's found per PC will be equal to L. However, if λ is smaller than L then the expected number of MS's found per PC will simply be equal to λ, since we can't discover more MS's than is coming in. Hence,

$$\begin{cases} F_{conv} = \lambda, & \lambda \leq L \\ F_{conv} = L, & \lambda \geq L. \end{cases}$$

The performances of the different algorithms will be compared with the conventional method as the increase in percent in F and the decrease in percent in S.

In all the simulations, the size of the SD buffer (i.e. the maximum number of PR's that can be processed per PC) is set to the fixed value K=200. This value could have been tuned adaptively in response to how many cells being used, the number of channels per cells, variations in the the input rate λ, etc. This possibility will not be considered further here.

When the agent get the observation *blocked*, that doesn't count as a paging because we don't actually page the MS. With the observation *blocked* we

don't get any information about the MS location and the PR will remain in the SD. The higher traffic it is, the higher will the blocking probabilities be.

4.1 The stationary case

Intuitively, in the stationary case (i.e. when the MS doesn't move), a good way of paging should be to simply page the cells in decreasing order of the probabilities $p(i), i = 1, ..., N$. This is true as long as the traffic isn't too high. The method only minimizes S. With this method, as the traffic gets higher and the main queue starts to grow, the channels to the more attractive cells will be occupied too much and a method that has a higher F is more attractive. A higher F is exactly what the pq-algorithm gives us. The method that pages the cells in decreasing order of $p(i)$ will be referred to as the p-algorithm.

In the stationary case the performance of the p-algorithm and the pq-algorithm will be compared.

Simulation I. The stationary case
This simulation compares the performances as a function of the incoming paging rate λ. The number of channels L per cell is equal to 9 and the number of cells N is equal to 10. The p_1-vector is given in table 4.1.

cell	1	2	3	4	5	6	7	8	9	10
$p_1(i)$	0.023	0.12	0.05	0.083	0.21	0.14	0.038	0.095	0.18	0.061

Table 4.1. Listing of the location probabilities among the cells.

In figure 4.1 the increase in discovery rate $((F_{seq} - F_{conv})/F_{conv})$ and the decrease in signaling load $((S_{conv} - S_{seq})/S_{conv})$ are plotted as a function of the incoming paging rate.

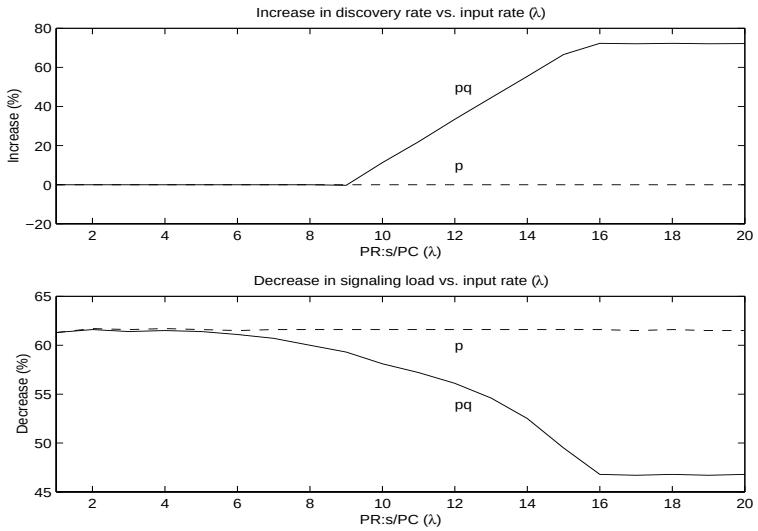

Figure 4.1. Performance as a function of the incoming paging rate.

The performance of the p-algorithm is independent of the incoming paging rate. This is because the incoming paging rate change the blocking probabilities q and the p-algorithm is independent of these. The p-algorithm always has the minimum possible S. However, when the traffic gets higher and the MQ starts to grow a higher F should be prioritised. When using the p-algorithm the MQ starts to grow when $\lambda \geq 9$. When $\lambda \leq 9$ both algorithms are able to take care of all the incoming PR's. For the pq, the MQ starts to grow when $\lambda \geq 15.5$.

A good way to page would be to use the p-algorithm when the traffic isn't heavy, i.e. when the MQ doesn't grow, because it minimizes S and it takes care of all the incoming PR's. When the traffic gets higher we should switch to the pq-algorithm, because it's able to take care of more PR's per PC.

4.2 The moving case

Running a POMDP for a given model gives us the optimal search policy for that model. When there is a probability for the MS to move, during the paging process, the best way of paging is not necessarily to page the cell with the highest $p_k(i)$ for $k = 1, 2, ...$, when the traffic is low as in the stationary case. This is illustrated in the following example. The example is from [8].

Example 4.1
Suppose that we have two cells and that $q_1 = q_2 = 0$, i.e. we will never be blocked. Also suppose that the transition probability matrix is

$$P = \begin{pmatrix} 0 & 1 \\ 0.5 & 0.5 \end{pmatrix}$$

Now, if $p_1 = (0.55, 0.45)$, then an immediate search in cell 1 will lead to discovery with probability 0.55 and a search in cell 2 will lead to discovery with probability 0.45. However, an unsuccessful search in cell 2 will lead to certain discovery in the next time step, because it will always move from cell 1 to cell 2, whereas an unsuccesful search in cell 1 leads to complete uncertainty as to where it will be in the next time step. Starting to search in cell 2 results in an expected cost of 1*0.45+2*0.55=1.55, whereas to start search in cell 1 will at best result in an expected cost of 1*0.55+(2*0.5+3*0.5)*0.45=1.675. Hence, starting to search cell 2 is better than to start with cell 1. This is because searching cell 2 gives more information about the location.

POMDP solves problems like the one in the example above. There are however, two problems when using POMDP with the paging system. First of all, the computations becomes very complex and takes a very long time to make. However, once the computations are made the search policy is given as a policy graph (described in appendix C) and can easily be applied to the system. The second problem - and this is the main problem - is that eq. (3.20) doesn't seem to have any zeros for the POMDP. The function is both negative and positive but it contains discontinuities and it doesn't seem like there is any value of q that solves eq. (3.20) for all i. The individual $q_{true}(i)'s$ actually depends on the whole q-vector, so the only case we can plot eq (3.20) is when $N = 2$. This is done for the *pqup* in figure 4.2 and for the POMDP in figure 4.3.

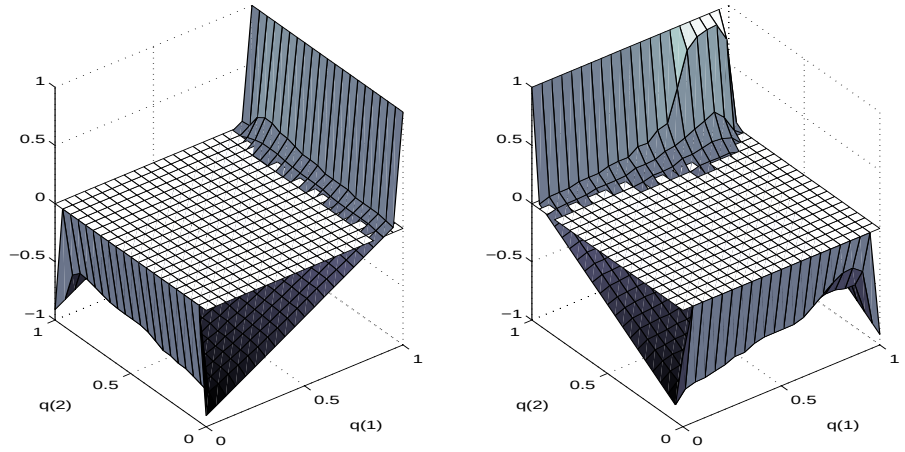

Figure 4.2. In the left plot, $q(1) - \hat{q}_{true}(1)$ is plotted and in the right plot, $q(2) - \hat{q}_{true}(2)$ is plotted, for the *pqup*-algorithm.

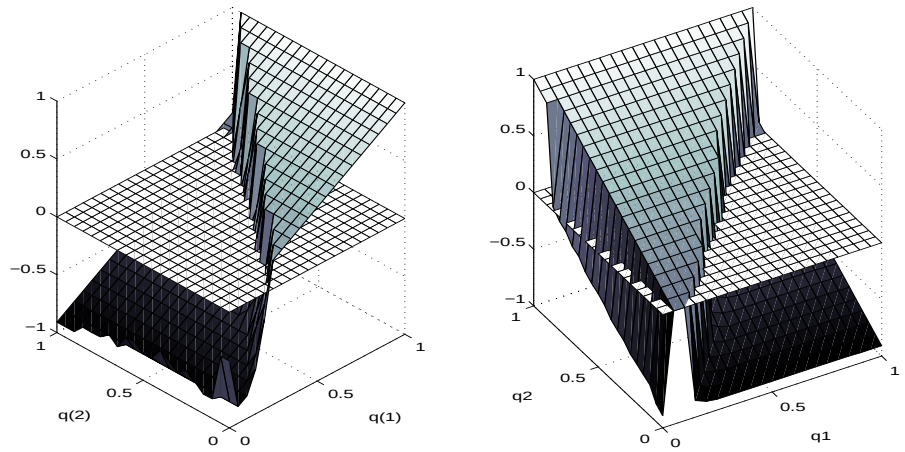

Figure 4.3. In the left plot, $q(1) - \hat{q}_{true}(1)$ is plotted and in the right plot, $q(2) - \hat{q}_{true}(2)$ is plotted, for the POMDP.

In figure 4.2 we can find a point where both graphs intersect the zero-plane. The graphs are smooth and the zeros are easy to find. In figure 4.3 we can see that both graphs contains discontinuities. There is no point where both graphs intersect the zeroplane.

4.2.1 Small location areas

In the next two simulations the POMDP is included to check the performance, even though we can't solve eq. (3.20). For the POMDP the Robbins-Monroe has been running for a longer time (more iterations) and the policy graph that produced the best performance was saved.

Unfortunately, when we're using POMDP we have to keep the number of cells small otherwise the simulations takes to long time to run. In this chapter the number of cells in the LA is 4 and the cells are arranged according to figure 4.4.

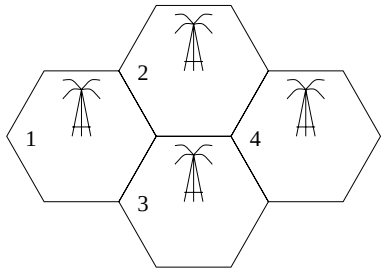

Figure 4.4. Structure of the cells in the LA.

The simulations in this chapter compares *pqup* and POMDP with the conventional method and they all have the following transition probability matrix and initial distribution among the cells

$$P = \begin{pmatrix} 0.9 & 0.05 & 0.05 & 0 \\ 0.0176 & 0.9284 & 0.0206 & 0.0334 \\ 0.0286 & 0.0333 & 0.9047 & 0.0334 \\ 0 & 0.0344 & 0.0213 & 0.9443 \end{pmatrix}$$

$p_1 = (\ 0.12 \quad 0.34 \quad 0.21 \quad 0.33\ )$.

The MS's moves roughly one time out of ten between the cells per PC. The transition probability matrix is chosen so that p_k converge to p_1 when we get observation *blocked*. To get a P-matrix that makes p_k converge like this, the Hastings-Metropolis algorithm has been used. This is described in [9].

Simulation II. Light load

Assume a case where $L > \lambda$. This means that the conventional method is capable of discovering all the incoming PR's. Let the system parameters be

$$\begin{cases} \lambda = 6 \text{ PR's/PC} \\ L = 9 \end{cases}$$

Here, the number of channels per BS is equal to 9 and the conventional method is capable of discovering 9 PR's per PC. The result is given in table 4.2.

F_{conv}	6.0	
S_{conv}	4.0	
	pqup	POMDP
F_{seq}	6.0	6.0
S_{seq}	2.26	2.18
Increase in the discovery rate	0	0
Decrease in the signaling load	43.6 %	45.5 %

Table 4.2. Results from simulation II.

Even though, $q \neq q_{true}$, for the POMDP, it still has a lower S than the *pqup* in this case. This is not true for all models, because we can't solve eq. (3.20) and there doesn't always exist a good q.

Simulation III. Heavy load
In this simulation the performances are compared when $L < \lambda$. This means that the incoming PR's cannot be taken care of with the conventional method.

$$\begin{cases} \lambda = 16 \text{ PR's/PC} \\ L = 9 \end{cases}$$

Since the system is heavily loaded a higher F should be prioritised before a lower S. The result is given in table 4.3.

F_{conv}	9.0	
S_{conv}	4.0	
	pqup	POMDP
F_{seq}	13.3	11.3
S_{seq}	2.71	2.53
Increase in the discovery rate	47.7 %	25.6 %
Decrease in the signaling load	32.3 %	36.8 %

Table 4.3. Results from simulation III.

POMDP has a lower F and a lower S than the *pqup*. When solving POMDP we only try to minimize S. However, since the MS's moves among the cells, a good policy graph should spread out the pagings among the cells. This leads to more available paging channels and a higher F. The F would probably be higher if $q = q_{true}$, for the POMDP.

4.2.2 Larger location areas

When the LA's gets bigger we can't use POMDP because we would have to run the POMDP for weeks for each model. We would also have to run it for more iterations with the Robbins-Monroe because the more cells we have the harder it is to find a good q. In this chapter the performance of the *pqup* is compared with the conventional method in some different situations.

The size of the LA in this chapter is 10 and the cells are arranged as in figure 4.5.

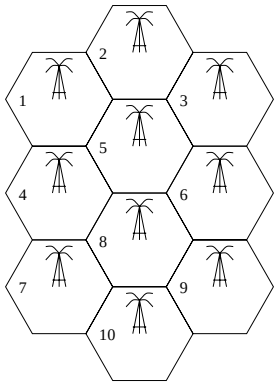

Figure 4.5. Structure of the cells in the LA.

The model used in the simulations in this chapter is

$$P = \begin{pmatrix} 0.900 & 0.033 & 0.000 & 0.033 & 0.034 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.003 & 0.949 & 0.019 & 0.000 & 0.029 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.033 & 0.900 & 0.000 & 0.033 & 0.034 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.002 & 0.000 & 0.000 & 0.946 & 0.023 & 0.000 & 0.011 & 0.017 & 0.000 & 0.000 \\ 0.003 & 0.017 & 0.011 & 0.017 & 0.925 & 0.017 & 0.000 & 0.012 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.017 & 0.000 & 0.025 & 0.919 & 0.000 & 0.018 & 0.021 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.033 & 0.000 & 0.000 & 0.928 & 0.034 & 0.000 & 0.005 \\ 0.000 & 0.000 & 0.000 & 0.017 & 0.017 & 0.017 & 0.011 & 0.920 & 0.016 & 0.002 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.033 & 0.000 & 0.028 & 0.936 & 0.003 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.033 & 0.033 & 0.034 & 0.900 \end{pmatrix}$$

$p_1 = (\ 0.011 \quad 0.12 \quad 0.07 \quad 0.15 \quad 0.21 \quad 0.14 \quad 0.051 \quad 0.15 \quad 0.09 \quad 0.008\ )$.

Once again, the MS's moves roughly one time out of ten between the cells per PC and the transition probability matrix is chosen so that p_k converge to p_1 when we get observation *blocked*.

Simulation IV. Dependence of the incoming paging rate
We start with the case when $L > \lambda$. The system parameters are

$$\begin{cases} \lambda = 6 \text{ PR's/PC} \\ L = 9 \end{cases}$$

The result is given in table 4.4

F_{conv}	6.0
F_{seq}	6.0
Increase in the discovery rate	0
S_{conv}	10
S_{seq}	4.47
Decrease in the signaling load	55.3 %

Table 4.4. Results from simulation IV, light load.

We can see that there is a significant decrease in the signaling load compared to the conventional method. Let see what happens when the traffic gets higher. The system parameters are given below and the result is given in table 4.5.

$$\begin{cases} \lambda = 16 \text{ PR's/PC} \\ L = 9 \end{cases}$$

F_{conv}	9.0
F_{seq}	13.8
Increase in the discovery rate	52.9 %
S_{conv}	10
S_{seq}	5.40
Decrease in the signaling load	46.0 %

Table 4.5. Results from simulation IV, heavy load.

As expected, the two performance measures F and S effects each other. In figure 4.6 the increase in discovery rate and the decrease in signaling load are plotted as a function of the incoming paging rate, in the case when the MS is moving.

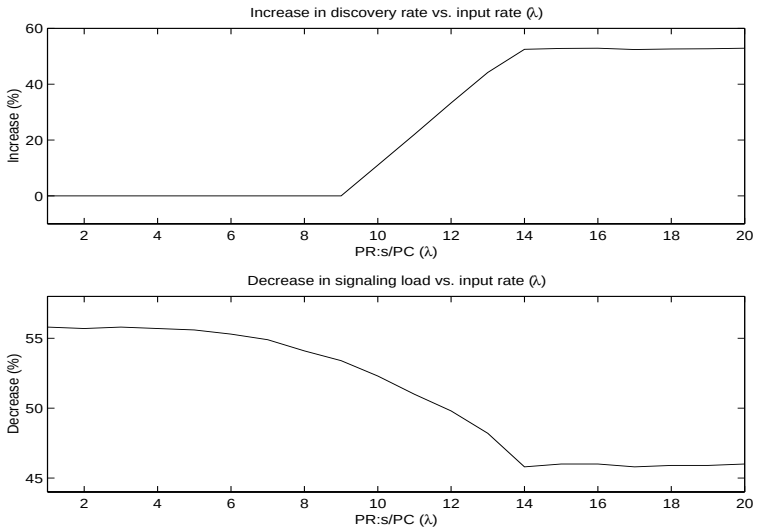

Figure 4.6. Performance as a function of the incoming paging rate

Once again, an increase in F leads to an increase in S and vice versa (compare to figure 4.1).

Simulation V. Estimation error in the p-vector

A natural question is how the algorithm works when there is an estimation error in the model, for example in the p-vector. For a start, let the system parameters be

$$\begin{cases} \lambda = 6 \text{ PR's/PC} \\ L = 9 \end{cases}$$

Call the estimated p_1 for $\hat{p_1}$. In table 4.6, $\hat{p_1}$ is listed. The real probability distribution, p_1, is repeted for convenience.

$\hat{p}_1$	0.05	0.16	0.1	0.28	0.15	0.02	0.08	0.12	0.03	0.01
p_1	0.011	0.12	0.07	0.15	0.21	0.14	0.051	0.15	0.09	0.008

Table 4.6. The estimated and the the real probability distribution vectors used in simulation V.

The result from the simulation is given in table 4.7.

F_{conv}	6.0
F_{seq}	6.0
Increase in the discovery rate	0 %
S_{conv}	10
S_{seq}	4.80
Decrease in the signaling load	52.0 %

Table 4.7. Results from simulation V, light load and estimated p_1.

The fact that the p_1-vector may not be estimated precisely didn't affect the performance drastically in this case. Let's see what happens when the traffic gets higher.

$$\begin{cases} \lambda = 16 \text{ PR's/PC} \\ L = 9 \end{cases}$$

F_{conv}	9.0
F_{seq}	13.6
Increase in the discovery rate	51.2 %
S_{conv}	10
S_{seq}	5.59
Decrease in the signaling load	44.1 %

Table 4.8. Results from simulation V, heavy load and estimated p_1.

Even in this case, the *pqup* is superior to the conventional method. The algorithm may thus be started with a rough estimate of p_1.

Simulation VI. Reduced number of paging channels
An interesting thing to investigate is the performance when fewer channels are allocated for the purpose of paging. Fewer paging channels leaves more radio spectrum available for other transmissions (e.g. voice and data). Therefore, fewer paging channels contributes to the overall improvement of the cellular system.

In this simulation the number of paging channels per BS are reduced to 4. We start with an incoming paging rate of 6.

$$\begin{cases} \lambda = 6 \text{ PR's/PC} \\ L = 4 \end{cases}$$

The result is given in the table below.

F_{conv}	4.0
F_{seq}	5.96
Increase in the discovery rate	49.1 %
S_{conv}	10.0
S_{seq}	5.41
Decrease in the signaling load	45.9%

Table 4.9. Results from simulation VI, light load and reduced number of paging channels.

There is an increase in S, compared with the case when we had 9 paging channels per BS. The increase is from 4.47 pagings/MS to 5.41 pagings/MS. It's still much better than the conventional method, who pages each MS 10 times.

Let us investigate what happens when the traffic gets even higher.

$$\begin{cases} \lambda = 16 \text{ PR's/PC} \\ L = 4 \end{cases}$$

F_{conv}	4.0
F_{seq}	6.08
Increase in the discovery rate	51.9 %
S_{conv}	10.0
S_{seq}	5.69
Decrease in the signaling load	43.1 %

Table 4.10. Results from simulation VI, heavy load and reduced number of paging channels.

The maximum number of MS per PC that we can discover with *pqup* is 6.08. With only 4 paging channels per BS, the conventional method can only discover 4 MS's per PC. This is an increase with 51.9 percent.

4.3 Conclusions

The simulations showed that both algorithms was superior to the conventional method. The *pqup* proved to be a reliable method and requires little computational power in the MSC's. The method is robust and works well both under low and high traffic, when the number of paging channels is reduced and when there is an estimation error in the occupancy probability vector. Since we couldn't solve eq. (3.20) for the $POMDP$, this method is unreliable and doesn't behave as well as expected. If we somehow could solve eq. (3.20) for the $POMDP$ then this would give us the optimal solution to the given problem.

5 Conclusions

It has been shown that there are several ways of improving the conventional scheme. The *pqup* improved the performance significantly. The simulations showed that the method

- reduces the signaling load between the MS's and the MSC's during the paging process
- is able to found more MS per PC
- work well both under light and heavy load
- work well with reduced number of paging channels
- is insensitive to the choice of occupancy vector.

The last point has, of course, practical significance since obtaining an occupancy probability vector that captures the occupancy probabilities of all the MS's is a practical impossibility.

The POMDP-algorithm didn't work as well as expected. Even though it worked better than the *pqup* in some cases, we can't use the POMDP as a paging scheme, as long as we can't solve eq. (3.20). If we could, then this algorithm would have been optimal for the given model.

5.1 Future work

The research about using POMDP as a paging scheme is going to be continued. The main research is going to be about how we can specify the model in a way that we can solve eq. (3.20). Maybe we can use different rewards or different observations?

A Rewards

Here it will be shown that giving the agent a reward that equals the probability of detection, given that the MS resides in the searched cell a, (i.e. $1 - q(a)$), is equal to give the agent a reward of 1, each time he finds the object.

To begin with, we write the reward function g as a function of *action*, *state* and *observation*, i.e. $g(a_k, x_k, \theta_k)$. This is the reward the agent recieves when he searches cell a_k, the object is in cell x_k and the agent recieves the observation θ_k at time step k. We start with the 2 observation case so the observations are either *found* or *not found*. If the agent recieves reward R_i each time he finds the object in cell i and the penalty for paging cell i is C_i, then the reward functions will be

$$\begin{array}{ll} g(x_k = i, a_k = i, \theta_k = notfound) = -C_i & \\ g(x_k = i, a_k = i, \theta_k = found) = R_i - C_i & \\ g(x_k = i, a_k = j, \theta_k = notfound) = -C_i, & i \neq j \\ g(x_k = i, a_k = j, \theta_k = found) = R_i - C_i, & i \neq j. \end{array} \tag{A.1}$$

Of course, the last reward will never be obtained, because if the agent page the wrong cell it will never find the object, but it's written here anyway for the sake of concretness. Now we can write the reward function as a function of *action* and *state* (i.e. as defined on page 7)

$$\begin{aligned} & g(x_k = i, a_k = i) \\ & = g(x_k = i, a_k = i, \theta_k = notfound) * Pr(\theta_k = notfound | x_k = i, a_k = i) \\ & \quad + g(x_k = i, a_k = i, \theta_k = found) * Pr(\theta_k = found | x_k = i, a_k = i) \\ & = -C_i * q_i + (R_i - C_i) * (1 - q_i) = R_i * (1 - q_i) - C_i \end{aligned}$$

and

$$\begin{aligned} & g(x_k = i, a_k = j) \\ & = g(x_k = i, a_k = j, \theta_k = notfound) * Pr(\theta_k = notfound | x_k = i, a_k = j) \\ & \quad + g(x_k = i, a_k = j, \theta_k = found) * Pr(\theta_k = found | x_k = i, a_k = j) \\ & = -C_i * 1 + (R_i - C_i) * 0 = -C_i. \end{aligned}$$

This gives us g^a as defined in eq. (3.6).

$$g^a = (-C_a, -C_a, ..., -C_a, R_a * (1 - q_a) - C_a, -C_a, ..., C_a, C_a)'. \tag{A.2}$$

Putting this in eq. (3.6) yields,

$$\begin{aligned} V(p_k) &= \max_{a\in A}(p_k' g^a + \beta \sum_{\theta\in\Theta} \sigma(\theta|p_k,a)V(T(\theta,p_k,a))) \\ &= \max_{a\in A}(p_k' * (-C_a,...,-C_a,R_a*(1-q(a))-C_a,...,C_a)' \\ &\quad +\beta \sum_{\theta\in\Theta} \sigma(\theta|p_k,a)V(T(\theta,p_k,a))) \\ &= \max_{a\in A}(p_k(a)*R_a*(1-q(a))-C_a \\ &\quad +\beta \sum_{\theta\in\Theta} \sigma(\theta|p_k,a)V(T(\theta,p_k,a))) \end{aligned}$$

Comparing this result with Eagle's (eq. (3.7)) gives that the reward R_i should be chosen to 1 for $i = 1, ..., N$ and the penalty C_i should be choosen to 0 for $i = 1, ..., N$. However, since C_i is just a constant and it is in a maximisation expression, we can choose C_i to any constant we want (as long as it the same for all i). Eagle just chosed it to be 0. As a matter of fact, we can actually choose the reward to be any constant ≥ 0 as long as it's the same for all cells. This is a bit harder to see and it won't be discussed further here.

B The POMDP-file

B.1 POMDP-file specification

POMDP solve version 4.0 written by A. R. Cassandra.

This describes the POMDP file format.

There are some semantics to the format and these are discussed here. All floating point number must be specified with at least one digit before and one digit after the decimal point.

Comments: Everything from a '#' symbol to the end-of-line is treated as a comment. They can appear anywhere in the file.

The following 5 lines must appear at the beginning of the file. They may appear in any order as long as they preceed all specifications of transition probabilities, observation probabilities and rewards.

discount: %f
values: [reward, cost]
states: [%d, <list of states>]
actions: [%d, <list of actions>]
observations: [%d, <list of observations>]

The definition of states, actions and/or observations can be either a number indicating how many there are or it can be a list of strings, one for each entry. These mnemonics cannot begin with a digit. For instance, both:

actions: 4
actions: north south east west

will result in 4 actions being defined. The only difference is that, in the latter, the actions can then be referenced in this file by the mnemonic name. Even when mnemonic names are used, later references can use a number as well, though it must correspond to the positional numbering starting with 0. The numbers are assigned consecutively from left to right in the listing starting with zero. The mnemonics are discarded once the whole file has been read in.

When listing states, actions or observations one or more whitespace characters are the delimiters (space, tab or newline). When a number is given instead of an enumeration, the individual elements will be referred to by con-

secutive integers starting at 0.

After the preamble, there is the optional specification of the starting state. For the current POMDP-solve code, this is completely ignored. It allows it to parse it to be compatible with newer versions of the specification. There are a number of different formats for the starting state, but these won't be discussed here. You won't need them for this program, and if you are running them on a file that has them, then you'll know what they look like.

After the initial five lines and optional starting state, the specifications of transition probabilities, observation probabilities and rewards appear. These specifications may appear in any order. Any probabilities or rewards not specified in the file are assumed to be zero.

You may also specify a particular probability or reward more than once. The definition that appears last in the file is the one that will take affect. This is convenient for specifying exceptions to a more general specification.

To specify an individual transition probability:

T: <action> : <start-state> : <end-state> %f

Anywhere an action, state or observation can appear, you can also put the wildcard character '$*$' which means that this is true for all possible entries that could appear here. For example:

T: 5 : $*$: 0 1.0

is interpreted as action 5 always moving the system state to state 0, no matter what the starting state was (i.e., for all possible starting states.)

To specify a single row of a particular actions transition matrix:

T: <action> : <start-state>
%f %f ... %f

Where there is an entry for each possible next state. This allows defining the specific transition probabilities for a particular starting state only. Instead of a list of probabilities the mnemonic word 'uniform' may appear. In this case, each transition for each next state will be assigned the probability 1/#states. Again, an asterick in either the action or start-state position will indicate all possible entries that could appear in that position.

To specify an entire transition matrix for a particular action:

T: <action>
%f %f ... %f
%f %f ... %f
⋱
%f %f ... %f

Where each row corresponds to one of the start states and each column specifies one of the ending states. Each entry must be separated from the next with one or more white-space characters. The state numbers goes from left to right for the ending states and top to bottom for the starting states. The new-lines are just for formatting convenience and do not affect final matrix results. The only restriction is there must be NxN values specified where N is the number of states.

In addition, there are a few mnemonic conventions that can be used in place of the explicit matrix:

identity
uniform

Note that uniform means that each row of the transition matrix will be set to a uniform distribution. The identity mnemonic will result in a transition matrix that leaves the underlying state unchanged for all possible starting states (i.e., the identity matrix).

The observational probabilities are specified in a manner similiar to the transition probabilities. To specify individual observation probabilities:

O : <action> : <end-state> : <observation> %f

The asterick wildcard is allowed in any of the positions.

To specify a row of a particular actions observation probability matrix:

O : <action> : <end-state>
%f %f ... %f

This specifies a probability of observing each possible observation for a particular action and ending state. The mnemonic short-cut 'uniform' may also appear in this place.

To specify an entire observation probability matrix for an action:

O: <action>
%f %f ... %f
%f %f ... %f
⋱
%f %f ... %f

The format is similiar to the transition matrices except the number of entries must be NxO where N is the number of states and O is the number of observations. Again, the 'uniform' mnemonic can be substituted for an entire matrix. In this case it will assign each entry of each row the probability 1/#observations.

To specify individual rewards:

R: <action> : <start-state> : <end-state> : <observation> %f

For any of the entries, an asterick for either <state>, <action>, <observation> indicates a wildcard that will be expanded to all existing entities.

There are two other forms to specify rewards:

R: <action> : <start-state> : <end-state>
%f %f ... %f

This specifies a particular row of a reward matrix for a particular action and start state. The last reward specification form is

R: <action> : <start-state>
%f %f ... %f
%f %f ... %f
⋱
%f %f ... %f

which lets you specify an entire reward matrix for a particular action and start-state combination.

B.2 A POMDP-file example

Here is a POMDP-file example running on the paging system. There are 3 observations and the reward 1 is obtained when the agent receives observation *found*. The number of states is 3, the transition probability matrix is

$$P = \begin{pmatrix} 0.9 & 0.05 & 0.05 \\ 0.05 & 0.9 & 0.05 \\ 0.05 & 0.05 & 0.9 \end{pmatrix}$$

and the blocking probabilities are

$q = (\ 0.25 \quad 0.4 \quad 0.1\ )$.

One way to specify the POMDP-file for this model is:

discount: 0.25
values: reward
states: 3
actions: 3
observations: found missed blocked

T: *
0.9 0.05 0.05
0.05 0.9 0.05
0.05 0.05 0.9

O: 0
0.75 0 0.25
0 0.75 0.25
0 0.75 0.25

O: 1
0 0.6 0.4
0.6 0 0.4
0 0.6 0.4

O: 2
0 0.9 0.1
0 0.9 0.1
0.9 0 0.1

R: * : * : *
1 0 0

C The result-files after a POMDP simulation

After running a POMDP simulation, the POMDP program will give you the solution in two files. How to use these files are described here.

C.1 The alpha-file

This file lists the alpha vectors after a POMDP simulation. The vectors fully specifies the solution after running a simulation. Each vector has an action associated with it and, at time step k, we should pick the vector according to eq. (C.1), (compare to eq. (3.6)).

$$\arg\max_j(p_k\alpha_j), \quad j = 1, ..., J, \tag{C.1}$$

where J is the number of vectors. After picking an alpha vector we choose the action associated with it. This is the action for time step k. The format of the alpha-file is simply:

<action> <list of vector components>

<action> <list of vector components>

etc...

The number of alpha vectors is equal to the number of states in the POMDP.

C.2 The pg-file

Another (and more convenient) way to specify the solution after a POMDP simulation is to use a policy graph. With a policy graph we don't have to calculate p_k at each time step. We only have to know the initial location distribution probability vector, p_1. Each line in the pg-file represents one node of the policy graph and its contents are:

N A Z1 Z2 Z3 ...

Here 'N' is a node ID giving the node a unique name. 'A' is an integer and specifies the action associated with this node. These are followed by a list of node ID's, one for each observation. Thus the list will have a length equal to the number of observations in the POMDP. This list specifies the transitions in the policy graph. If the observation recieved is n, then the $n : th$ element

in the list will specify the next node in the policy graph. The start node in the policy graph is specified (from the alpha-file) by eq. (C.2).

$$\arg\max_j(p_1\alpha_j), \quad j = 1, ..., J. \tag{C.2}$$

D Abbreviations

This abbreviations are all defined earlier in the paper. They are written here for convinience.

BS: Base Station, each cell has one base station.

HLR: Home Location Register, contains information about the whereabouts of the mobile stations but is usually bigger than a visitor location register and it caches portions of information to these.

LA: Location Area, a geographic area covered by a group of cells.

MS: Mobile Station, a mobile communication device, e.g. a mobile phone.

MSC: Mobile Switching Center, manages the base stations within a location area.

PC: Paging Cycle, the time epoch at which a page request can be sent to a base station.

PR: Page Request, a request to page a mobile station in a specific cell.

SD: Smart Distributor, distributes the page requests among the cells.

VLR: Visitor Location Register, contains information about the whereabouts of the mobile stations. It isolates the small-scale movements.

References

[1] T.S. Rappaport, *Wireless Communications*. Prentice Hall, 1996.

[2] W.S. Lovejoy, "A survey of alghorithmic methods for partially observed Markov decision processes". *Annals of Operations Research 28*, pp. 47-65, 1991.

[3] J.N. Eagle, "The optimal search for a moving target when the search path is constrained". *Operations Research*, vol. 32, pp. 1107-1115, 1984.

[4] A.R. Cassandra, L.P. Kaelbling, M.L. Littman, "Acting optimally in partially observable stochastic domains". *http://www.cs.duke.edu/ mlittman/topics/pomdp-page.html*, link: "POMDP's as a model of planning".

[5] A.R. Cassandra, M.L. Littman, N.L. Zang, "Incremental Pruning, a simple, fast, exact method for partially observable Markov decision processes". *http://www.cs.duke.edu/ mlittman/topics/pomdp-page.html*, link: "POMDP algorithms".

[6] R. Rezaiifar, A.M. Makowski, "From optimal search theory to sequential paging in cellullar networks". *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 17, pp. 1253-1264, september 1997.

[7] M.T. Wasan, *Stochastic Approximation*. New York: Cambridge University, 1969.

[8] S.M. Ross, *Introduction to Stochastic Dynamic Programming*. Academic press, 1983.

[9] S.M. Ross, *Simulation*. San Diego: Academic Press, 1997, 2:nd ed.