# SEARCHING FOR MOBILE DATA WITH A PRIORI STATISTICAL KNOWLEDGE OF THEIR WHEREABOUTS UNDER DELAY CONSTRAINTS

by

Yi Feng

A dissertation submitted to the Graduate Faculty in Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy, The City University of New York

2011

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirements for the degree of Doctor of Philosophy.

Amotz Bar-Noy

Date | Chair of Examining Committee

Theodore Brown

Date | Executive Officer

Theodore Brown

Noson S. Yanofsky

Modercai J. Golin

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

ABSTRACT

# SEARCHING FOR MOBILE DATA WITH A PRIORI STATISTICAL KNOWLEDGE OF THEIR WHEREABOUTS UNDER DELAY CONSTRAINTS

BY

YI FENG

Adviser: Professor Amotz Bar-Noy

One or more tokens are hidden into several boxes, and then the boxes are locked. The probabilities of each token being found in each box are known. All the probabilities are independent. A searcher is looking for one, some, or all of the tokens by unlocking boxes in a predetermined number of rounds. In each round, any subset of the boxes can be unlocked and the searcher collects all tokens in them. Each box is associated with a positive unlocking cost. The goal is to minimize the expected cost of unlocking boxes until the desired tokens are found.

The original motivation is to page mobile users in cellular network systems. Mobile users are tokens and cells are boxes. The probabilities of the users in cells can be extracted from historical data. The unlocking costs of boxes reflect the resources that are consumed to page a cell. The predetermined number of rounds ensures that the users will be found within a certain period of time (delay constraint). The goal is to minimize the resources that are consumed to find the users under the pre-determined delay constraint. In addition to the application of paging mobile users, this scheduling problem has broad utilization in finding information in sensor networks, searching for information in distributed data centers, medical decision making, etc.

The special case in which a single token is sought and all the boxes have the same unlocking costs has been studied. Polynomial time optimal algorithms exist. Optimal search strategies can be found in a time which is quadratic with respect to the number of boxes and linear with respect to the number of rounds. We improve this time complexity to linear with respect to both the number of boxes and the number of rounds, and provide a hierarchy of algorithms that trades off optimality for complexity.

In the general case of searching a single token while the boxes can have different unlocking costs, we prove it being strongly NP-hard, and provide various approximation algorithms. We also demonstrate a tradeoff between the time complexity and implementation complexity of our approximation algorithms.

In the case in which we search multiple tokens and all boxes are of the same unlocking costs, we explore the conference call problem and the yellow page problem. In the former we want to find all tokens and in the later we want to find (any) one of the tokens. The conference call problem has been studied. It is NP-hard and approximation algorithms exist. We show a duality between both problems and provide efficient polynomial-time and exponential-time optimal algorithms for specific cases of the problems. We show a tradeoff between the time and space complexity of optimal algorithms.

We implement all of our algorithms and some of the algorithms by other researchers. We conduct a comprehensive experimental study in the context of the paging mobile users application. The experimental study provides further insight of the behavior of algorithms and presents the performance of algorithms in real system.

# Acknowledgments

First of all, I would like to thank my advisor Amotz Bar-Noy. Amotz introduced me into the field of algorithms, inspired me with elegant problems, and provided me solid support both intellectually and financially. He also made me keep a positive attitude through challenges and hardship. My classmate Panagiotis Cheilaris worked closely with me on this work. He guided me in both theoretical computer science research and scientific writing. Without him, a lot of results in this work would still remain unveiled. He deserves my special thanks.

My coauthors, Mordercai Golin and Asaf Levin essentially enriched the contents of this work. Mordercai also served in my committee. Theodore Brown and Florian Lengyel financially supported my PhD study. Ted also served in my committee. Stathis Zachos and Noson Yanofsky, my other committee members, greatly helped me in scientific writing. The Research Computing Group at the Graduate Center of CUNY provided powerful and reliable computing resource for the experimental study. I would like to thank all of them. I would like to thank Ali Assarpour for his useful discussions on this work. I also want to thank Wenbo Cao, Jinzhong Niu, and Yuqing Tang for their help in my PhD study. My thanks also goes to Joe Driscoll, Lina Garcia, Lauren Brust, Kara Heffernan, and Jin Mao, for their help on the administrative affairs.

Finally, special thanks to my wife Stacy, my parents Ti and Jinjian. Because of them, my work becomes meaningful.

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

## 1.1 Motivation

### 1.1.1 Main application

In the last two decades, we have witnessed two revolutions: availability of information and availability of people. The Internet makes it possible to access information independent of physical distances. Cellular phone systems make it possible to talk with people even if they are not residing in predetermined locations.[1].

A cellular network system is composed of many cells, each controlled by a base station. When a call to a user (mobile) arrives, the system must locate the cell in which the mobile resides in order to establish a connection. Therefore, the system must employ "some" location management scheme to efficiently locate mobile users.

On one extreme, the mobile users may report their location changes to the system whenever they cross the boundaries between cells. In this approach, the system could always locate the users by paging the cells where users last report their locations. However, modern cellular network systems are more and more frequently composed of mini-cells and nano-cells for higher bandwidth availability and lower radiation.

[1]This statement is adapted from Bar-Noy et al. [1994]

Moreover, mobile users are moving faster than they used to. Combining both factors, mobile users would frequently need to report their location changes to the system. This scheme would cause a short battery life of mobile devices as well as consume more premium uplink bandwidth. On the other extreme, the mobile users can never report their location changes to the system. When calls arrive to a user, the system has to page the user in all possible cells that he or she may be located in. In a nationwide cellular network system, this is impractical and expensive.

A tradeoff between the two approaches is usually employed in real systems. A cellular network is divided into multiple location areas (also known as, zones). Each zone is composed of some number of cells. Mobile users only report their location changes when they cross the boundaries between zones. When users are being called, the system looks for the users by paging the cells in the zones that they lastly report their locations. Under such a location management scheme (TSG [2009]), the search process of mobile users is reduced to paging mobile users in the zones that they have lastly reported their locations.

If the system has no a priori knowledge about user locations, the system may conduct the paging process in two extreme approaches: In a blanket search, the system pages all the cells in the zone simultaneously (in one round). In this case, the mobile user is found in minimal time (one round) and the resources consumed in the paging process is maximized (all cells are paged). In a sequential search, the system pages the cells one at a time, and stops once the user is found. In this case, the user is expected to be found at a half of the rounds and a half of the cells are expected to be paged.

As a tradeoff between the time and number of paged cells it takes to find the user, the system usually pages the cells in rounds. In each round, the system pages a subset of cells in the zone, and the paging process terminates as soon as the cell that contains the mobile user is paged. In this scheme, the system may reduce the *expected*

number of paged cells. To ensure the quality of service, a delay constraint is usually added to the paging process. That is, the user must be found in a predetermined number of rounds. Because the user can be in any of the cells, it means all cells in the zone must be paged within a predetermined number of rounds. Therefore, a paging strategy, which is an ordered partition of the cells, is desired. In each round, the cells in the corresponding part are paged.

Although there are different approaches of user location management schemes (for a list of schemes, see survey Akyildiz et al. [1999]), such a zone-paging approach remains the fundamental scheme in real practice (TSG [2009]).

In real practice, the system usually has some statistical knowledge on the location of mobile users. For a particular user, the statistical knowledge can be represented as a real vector that records the probability of the user being in each of the cells. This knowledge can either be acquired from the system log or be reported by the user himself or herself.

Given the statistical knowledge of the users' locations, a paging strategy is evaluated by the expected number of paged cells. Our goal is, given the statistical knowledge of users' locations, to design algorithms that generate paging strategies that minimize expected number of paged cells.

**Example:** As illustrated in Figure 1.1, a user is at the library, at the coffee shop, and at home, with probabilities 40%, 10%, and 50% respectively. The paging process must be completed in 2 rounds. Thus, the system must select some cells (locations) to page in the first round and the rest in the second round. In one paging strategy, the system may choose to page the library and coffee shop in the first round, and the home in the second round. In this strategy, with $(40\% + 10\%)$ probability, the user is found in the first round and two cells (library and coffee shop) are paged. With 50% probability, the user is found in the second round, and all three cells (library and coffee shop in the first round, home in the second round) are paged. The expected number

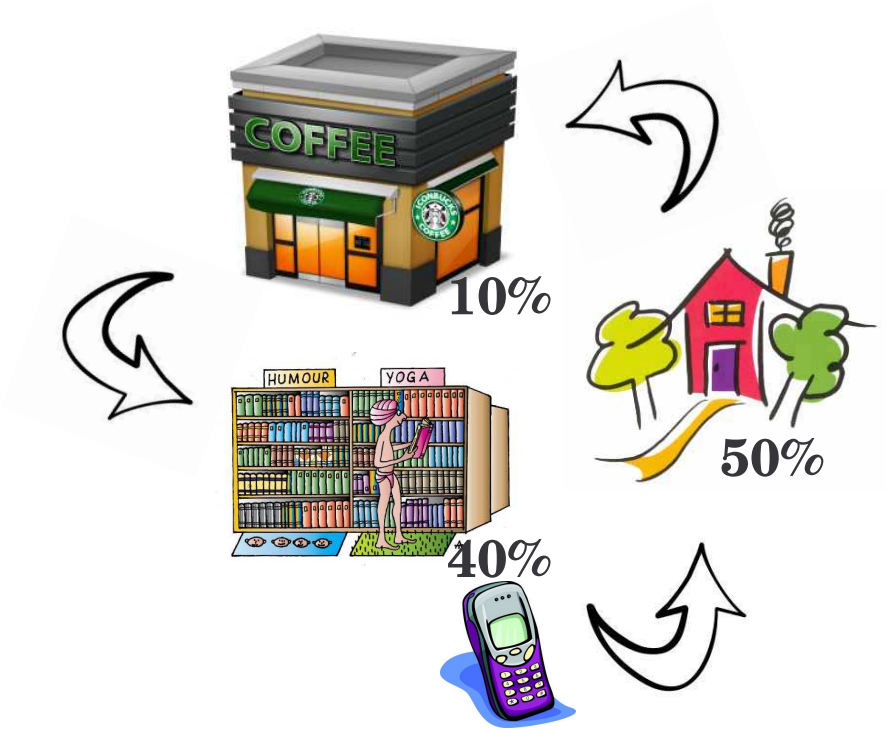



Figure 1.1: Illustration of paging mobile user process

of paged cells on this paging strategy is $(40\% + 10\%) \times 2 + 50\% \times 3 = 2.5$ . In another paging strategy, the system may choose to page the home in the first round, and the library and coffee shop in the second round. In this strategy, with 50% probability, the user is found in the first round and one cell (home) is paged. With $(40\% + 10\%)$ probability, the user is found in the second round, and all three cells (home in the first round, library and coffee shop in the second round) are paged. The expected number of paged cells on this paging strategy is $50\% \times 1 + (40\% + 10\%) \times 3 = 2$ . This strategy is the best we can achieve.

We observe the later paging strategy is better than the former in terms of expected paged cells. Our goal is to compute paging strategy with minimum number of paged cells for any number of cells and rounds.

### 1.1.2 Other applications and the vision of this work

The problems that are studied in this work are very general *scheduling* problems. In additional to the location management application mentioned above, it has broad applications in other fields. For example, it can help doctors decide how to arrange the order of exams in diagnostic process (Kaplan et al. [2005]), as well as how to efficiently find investment alternatives (Weitzman [1979]).

Because of the broad applications, we set our vision in the following paradigm: in today's distributed information systems, we are frequently looking for "some information" or "some entities". However, we do not always know the exact location of the information. Instead, we might have some a priori knowledge about possible locations and the probabilities of the information residing in the locations. We imagine the subject of the search as "mobile data" that may move from one location to another location and thus creating some uncertainty in its exact whereabouts. Our goal is to design, analyze, implement, and evaluate efficient search strategies for the mobile data while taking advantage of any partial knowledge of their whereabouts. Our optimization goals are to find the data on time while querying as few locations as possible. We seek strategies for finding one mobile datum, many mobile data, or one out of many mobile data.

## 1.2 The combinatorial problem

In the rest of this work, we will discuss the problem within the following combinatorial context: A token (mobile user) is hidden in one of several locked boxes (cells). We only know the probability (statistical knowledge) of the token in each of the boxes. A searcher looks for the token by unlocking boxes in a predetermined number of rounds (delay constraints). In each round, any subset of the boxes can be unlocked. A search strategy is an ordered partition of the boxes, such that in each round, the boxes in

the corresponding part are unlocked. The goal is to design algorithms that take the probabilities as input, and output a search strategy with a minimum expected number of unlocked boxes.

## 1.3 Contribution

In this work, we explore the token search problem in three directions. Our major contribution are listed below. The summary of more specific results are reported with greater accuracy in Section 2.5 as soon as all the terminologies are defined.

In the basic search problem as mentioned above, known algorithms can solve it optimally utilizing a dynamic programming scheme. These algorithms take quadratic time with respect to the number of boxes and linear time with respect to the number of rounds. We design two optimal algorithms that take linear time with respect to both the number of boxes and the number of rounds.

In the problem of search with cost, in which different boxes can have different unlocking costs, and the objective is to minimize the expected unlocking cost, we prove the problem being strongly NP-hard and provide a polynomial time approximation scheme (PTAS for abbreviation)[2]. We also design an efficient 8/7-approximation algorithm when the token must be found in two rounds. We provide polynomial time optimal algorithms for other interesting and important cases.

In the searching for multiple tokens problem, in which more than one token is concurrently searched for, we explore the problem of finding one out of many tokens (yellow page) and the problem of finding all of the tokens (conference call). The conference call problem has been proved NP-hard with various approximation algorithms. We show polynomial optimal algorithms for several special cases, and demonstrate a duality between the two problems. We also design a group of twelve heuristics that employ four different criteria. We evaluate their performances through empirical

[2]For the definition of PTAS, see book Vazirani [2001]

experiments and demonstrate preferences of different heuristics on different problem settings.

## 1.4 Organization

Chapter 2 presents the preliminaries of this work. In particular we define the basic search problem, as well as present the dynamic programming scheme employed throughout this work. Then we expand the problem into three generalizations. In chapter 3 we discuss how to improve the efficiency of optimal algorithms for the basic search problem. In chapter 4, we discuss the generalized problem, where the boxes can have different unlocking costs. In chapter 5, we discuss the problem of simultaneously searching for multiple tokens. Chapter 2 is a prerequisite to understand the rest of this work. Chapters 3, 4, and 5 can be read separately after reading Chapter 2. We complement our theoretical results with an experimental study in Chapter 6 in the context of the main application: paging mobile users in cellular networks, in which we evaluate our algorithms on real data and synthetic data. In chapter 7, we summarize our results and present some open problems.

## 1.5 Credit

The results in Chapter 3 has been preliminarily reported in Bar-Noy et al. [2007]. The results in Chapter 4 has been preliminarily reported in Bar-Noy et al. [2010b]. The results in Chapter 5 has been preliminarily reported in Bar-Noy et al. [2010a]. The experiment results in Chapter 6 are sampled from the simulation sections of papers Bar-Noy et al. [2007, 2010b,a] as well as their ongoing journal version.

This work has been partially supported by the NSF program award CNS-0626606, and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this work are those of the author

and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

# Chapter 2

# Preliminaries

## 2.1 The basic search problem

**Example 2.1.** A token is hidden into one of *three* boxes with probabilities $\langle 0.5, 0.4, 0.1 \rangle$, respectively. A searcher is looking for the token by unlocking the boxes in *two* rounds. Assume he or she follows a search strategy that unlocks the first two boxes in the first round and the third box in the second round, then, with probability $(0.5 + 0.4)$, the token is found in the first round by unlocking 2 boxes and with probability 0.1, the token is in the third box and all 3 boxes have to be unlocked. The expected number of unlocked boxes for this search strategy is $(0.5 + 0.4) \cdot 2 + 0.1 \cdot 3 = 2.1$. Another strategy would unlock the first box in the first round and the rest two boxes in the second round. In this case, with probability 0.5 the token is in the first box and 1 box is unlocked. With probability $(0.4 + 0.1)$, the search has to proceed with the second round and all 3 boxes are unlocked. With this search strategy, the expected number of unlocked boxes is $0.5 \cdot 1 + (0.4 + 0.1) \cdot 3 = 2$. It is observed that the second search strategy is better than the first in terms of the number expected unlocked boxes. The second search strategy is actually the best that one can achieve.

**The Basic Search Problem:** A token $T$ is hidden into one of the $N$ boxes $\{B_1, \ldots, B_N\}$.

With positive probability $0 < p_n \leqslant 1$, token $T$ is in box $B_n$. Without lost of generality, we assume $\sum_{n=1}^{N} p_n = 1$. All the probabilities are invariant, that is, the probability $p_n$ remains constant after any set of boxes are unlocked and the token is not in them. A searcher looks for the token by unlocking boxes in $D$ rounds ($1 \leqslant D \leqslant N$). In each round, any subset of the boxes can be unlocked. As soon as the searcher unlocked the box that contains the token, his search process terminates. A *search strategy* is an ordered $D$-partition of the boxes, $\mathcal{S} = \langle S_1, \ldots, S_D \rangle$, such that in the $d$-th round, boxes in part $S_d$ are unlocked. We look for search strategies that minimize the expected number of unlocked boxes.

**Definition 2.2.** An instance $\mathcal{I}$ of the basic search problem is a tuple $\langle N, D, \mathbf{p} \rangle$, where $N$ is the number of boxes, $D$ is the delay constraint measured in number of rounds, and vector $\mathbf{p} = \langle p_1, \ldots, p_N \rangle$ is the probabilities of the token being in boxes $\langle B_1, \ldots, B_N \rangle$.

**Definition 2.3.** A search strategy $\mathcal{S} = \langle S_1, \ldots, S_D \rangle$ on an instance $\mathcal{I} = \langle N, D, \mathbf{p} \rangle$ is an ordered $D$-partition of boxes $\{B_1, \ldots, B_N\}$, such that in the $d$-th round, boxes in part $S_d$ are unlocked.

We denote the probability of the token being in one of boxes of set $S_d$ as $p(S_d)$, formally,

$$p(S_d) = \sum_{B_n \in S_d} p_n \ .$$

**Proposition 2.4.** *The cost of a search strategy $\mathcal{S}$ on an instance $\mathcal{I} = \langle N, D, \mathbf{p} \rangle$ is the expected number of unlocked boxes. It can be computed by*

$$\mathit{cost}(\mathcal{I}, \mathcal{S}) = \sum_{d=1}^{D} \left( \left( \sum_{i=1}^{d} |S_i| \right) \cdot p(S_d) \right) \tag{2.1}$$

*where $|S_i|$ is the number of boxes in set $S_i$.*

*Proof.* If the token is found in the $d$-th round, we would have opened all boxes in sets

$S_1, \ldots, S_d$ with total cost $|S_1| + \cdots + |S_d|$. This event happens with probability of $p(S_d)$. Summing for all rounds we obtain the desired result. □

**Definition 2.5.** Given an instance $\mathcal{I}$ of the basic search problem, an *optimal search strategy* $\mathcal{O}$ is a search strategy such that for all search strategies $\mathcal{A}$, $\text{cost}(\mathcal{I}, \mathcal{A}) \geqslant \text{cost}(\mathcal{I}, \mathcal{O})$.

**Objective:** In the basic search problem, our objective is to design algorithms that take instances as input, and output search strategies, such that their costs are minimized. We look for algorithms that generate good search strategies for all instances belong to the problem model.

We distinguish between search strategies and algorithms: a search strategy is the solution to an instances, while an algorithm is a procedure that computes search strategies from problem instances.

## 2.2 An optimal dynamic programming scheme

Existing algorithms by Rose and Yates [1995], Goodman et al. [1996], Krishnamachari et al. [2004] solve the basic search problem in polynomial time using a dynamic programming scheme. As this scheme serve as a fundamental technique throughout this work, we present it as a preliminary. The dynamic programming scheme is based on the following two observations and lemma.

The first observation is that an optimal search strategy should utilize all the rounds.

**Observation 2.6.** *In the basic search problem, given the delay constraint $D$, it is always good to unlock at least one box in each round.*

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

The second observation relates the location probabilities with the order of boxes in an optimal search strategy.

**Observation 2.7.** *In an optimal search strategy of the basic search problem, the boxes must be unlocked in the non-increasing order of $p_n$. Mathematically, let an optimal search strategy be $\mathcal{O} = \langle O_1, \ldots, O_D \rangle$, if $B_i \in O_d$ and $B_j \in O_{d+1}$, then $p_i \geqslant p_j$, for all $1 \leqslant i, j \leqslant N$ and $1 \leqslant d < D$.*

*Proof.* We prove the lemma by contradiction. Assume there is an optimal search strategy $\mathcal{O} = \langle O_1, \ldots, O_D \rangle$, $B_i \in O_d$ and $B_j \in O_{d+1}$ for some $d$, and $p_i < p_j$. We construct a new strategy $\mathcal{O}' = \langle O'_1, \ldots, O'_D \rangle$, which is the same as $\mathcal{O}$ except $B_i \in O'_{d+1}$ and $B_j \in O'_d$.

$$
\begin{aligned}
&\operatorname{cost}(\mathcal{I}, \mathcal{O}') - \operatorname{cost}(\mathcal{I}, \mathcal{O}) \\
=&\sum_{d=1}^{D} \left( \sum_{i=1}^{d} |O'_i| \cdot \sum_{B_j \in O'_d} p_j \right) - \sum_{d=1}^{D} \left( \sum_{i=1}^{d} |O_i| \cdot \sum_{B_j \in O_d} p_j \right) \text{ (Proposition 2.4)} \\
=&(p_i - p_j)|O_{d+1}| < 0 \text{ (since } p_i < p_j)
\end{aligned}
$$

This is a contradiction to that $\mathcal{O}$ is optimal. □

As a result of the previous observation, without loss of generality we assume that $p_1 \geq p_2 \geq \cdots \geq p_N$. Denote by $\mathcal{O}_d$ the optimal search strategy of searching all boxes in $d$ rounds, where $1 \leq d \leq D$. Denote by $\mathrm{OPT}(d)$ the cost of $\mathcal{O}_d$.

When $D = 1$, all the boxes must be unlocked in the first and only round and the cost of this optimal strategy is $\mathrm{OPT}(1) = N$. When $D = N$, by Observation 2.7, the optimal search strategy unlocks one box in each round in non-increasing order of probability $p_n$, and the cost is $\mathrm{OPT}(N) = \sum_{n=1}^{N} n \cdot p_n$. When $D = 2$, an optimal search strategy unlocks a subset of the boxes in the first round, and if the token is not found, the rest of the boxes must be unlocked in the second round. By Observation 2.7, if we sort the boxes by non-increasing order of $p_n$, there exists a pivot $o$, $1 \leq o < N$,

such that boxes $\{B_1, \ldots, B_o\}$ are unlocked in the first round and the expected search cost is minimized. This pivot can be computed through a sequential search that takes $\Theta(N)$ time.

We now define the recursive formulation to find $\mathrm{OPT}(D)$. We ignore the computation of the optimal partition $\mathcal{O}_D$ that yields the optimal cost. We note that in implementing dynamic programming solutions, it is a standard technique to find the actual solution by first filling in the dynamic programming table with costs and then tracing backwards from the optimal solution to obtain the corresponding strategy. This adds a negligible $\Theta(D)$ overhead in the running time complexity. We also assume that the vector of probabilities is given in a non-increasing order ($p_1 \geq p_2 \geq \cdots \geq p_N$) since the dynamic programming algorithm uses this order. We note that the sorting can be done in $\Theta(N)$ time (using radix-sort for example) if the set of distinct values of probabilities is not too large.

Let $1 \leq n \leq N$ and $1 \leq d \leq D$ be given. Define $h_n^{(d)}$ to be the optimal cost of finding the token in the first $n$ boxes, $\{B_1, \ldots, B_n\}$, in $d$ rounds. By definition, $\mathrm{OPT}(D) = h_N^{(D)}$. For the base of the dynamic programming computation ($d = 1$):

$$\forall n \in \{1, \ldots, N\} \colon h_n^{(1)} = n \cdot \sum_{i=1}^{n} p_i.$$

There are two approaches to formulate the optimal recursion for $h_n^{(d)}$. In the first, by Rose and Yates [1995], Goodman et al. [1996], the size of the first set of boxes to be unlocked in the first round is fixed and then the rest of the boxes are recursively and optimally unlocked in $d - 1$ rounds. In the second, by Krishnamachari et al. [2004], the size of the last set of boxes to be unlocked in the last round is fixed and then the rest of the boxes are recursively and optimally unlocked in $d - 1$ rounds. The two approaches are essentially equivalent. We present and adopt the second approach.

**Definition 2.8.** For fixed $d$, with $2 \leq d \leq D$, let $A = A^{(d)}$ be the $N \times N$ matrix

with entries:

$$a_{n,j}^{(d)} = h_j^{(d-1)} + n \cdot \sum_{i=j+1}^{n} p_i,$$

for $d \leq n \leq N$ and $d-1 \leq j \leq n-1$. In other words, $a_{n,j}^{(d)}$ is the cost of unlocking the first $j$ boxes optimally in $(d-1)$ rounds and the next $(n-j)$ boxes in the $d$-th round.

The following lemma recursively formulates the computation process of an optimal search strategy.

**Lemma 2.9.** $h_n^{(d)} = \min_{j=d-1}^{n-1} a_{n,j}^{(d)}$.

*Proof.* Assume for contradiction purpose, that for some $d$ and $n$,

$$j_o = \arg \min_{j=d-1}^{n-1} a_{n,j}^{(d)}$$

but

$$h_n^{(d)} = a_{n,j_o'}^{(d)}, \text{ and } j_o' \neq j_o \ .$$

By the definition of $j_o$, $a_{n,j_o}^{(d)} < a_{n,j_o'}^{(d)}$ . This is a contradiction to that $h_n^{(d)}$ is the optimal search strategy of unlocking the first $n$ boxes in $d$ rounds. □

With the help of Observation 2.7 and Lemma 2.9, Algorithm 2.1 computes the optimal search strategy for any number of boxes $N$ and any number of rounds $D$ $(1 \leq D \leq N)$.

By analyzing Algorithm 2.1, we have the following observation.

**Observation 2.10.** *The running time of Algorithm 2.1 is $\Theta(DN^2)$ time.*

## 2.3 Notations

The notations in this work remains consistent as follows, unless occasionally, over-written in a section or subsection.

**Algorithm 2.1** Dynamic programming: compute the minimum cost of unlocking $N$ boxes in $D$ rounds in the basic search problem;

**Input:** Probabilities $\langle p_1, \ldots, p_N \rangle$, $N$, $D$
**Output:** $h_N^{(D)}$
  **for** $n = 1 \ldots N$ **do** {base $d = 1$}
    $h_n^{(1)} \leftarrow n \cdot \sum_{i=1}^{n} p_i$
  **end for**
  **for** $d = 2 \ldots D$ **do**
    **for** $n = d \ldots N$ **do**
      $h_n^{(d)} \leftarrow \min_{j=d-1}^{n-1} a_{n,j}^{(d)}$
    **end for**
  **end for**
  **return** $h_N^{(D)}$

A lower case italicized Latin letter, with or without subscript or superscript, e.g., $p$ or $p_n$, denotes a real scalar, usually the probability of the token in a box, or the cost of unlocking a box as in Chapter 4.

A subscript denotes the index of a box, e.g., $_n$. In Chapter 5, double subscripts $_{m,n}$ denotes the index of a token $m$ and the index of a box $n$.

An upper case Latin letter without scripts denotes the scale of the input, usually the number of rounds or number of boxes, e.g. $D$ or $N$. In Chapter 5, $M$ is number of tokens.

A calligraphic letter, e.g, $\mathcal{S}$, denotes a search strategy. In Chapter 5 it is replaced by $\mathcal{A}$ as $S$ is used otherwise. $\mathcal{O}$ usually represents an optimal search strategy.

A boldface letter, e.g., $\mathbf{p}$, usually denotes a vector or matrix of the corresponding lower case letter.

An upper case letter with subscript, is either a set of boxes in a search strategy, or the sum of a property of an element in the set. E.g., $S_d$ is a set of boxes unlocked in the $d$-th round, or $P_d$ is the sum of the probabilities of boxes in $S_d$.

## 2.4 Related works

The problem has been extensively studied in the context of paging mobile users in a cellular network system as well as a few other applications. Polynomial time optimal algorithms that solve the basic search problem has been developed independently in Rose and Yates [1995], Goodman et al. [1996], Krishnamachari et al. [2004]. Wang et al. [2001], Wang and Xue [2006] present a non-optimal heuristic.

In the basic search problem presented in this section, the search strategies are generated subject to worst case constraint. That is, the boxes are allowed to be open in at most $D$ rounds, while taking less than $D$ rounds would not gain extra credit. In contrast to the worse cast delay constrain, Krishnamachari et al. [2004] has discussed the problem of minimize expected number of unlocked box under *average* delay constraint. In their setup, some search strategies can have more than $D$ rounds as long as the amortized delay constraint is no more than $D$ rounds. They have proved the problem is NP-hard and have provided a weakly polynomial time optimal algorithm.

A more general framework of how to trade optimality with efficiency has been discussed in Mullender and Vitányi [1988], Awerbuch and Peleg [1995] in a combinatorial context.

In Chapters 3, 4, and 5, we further review related works on the specific problems.

## 2.5 Our contribution

We explore the basic search problem in three directions.

**Efficient algorithms for the basic search problem:** Existing algorithms compute optimal search strategies in $\Theta(D \cdot N^2)$ time. We design two new algorithms that compute optimal search strategies in $\Theta(D \cdot N)$ time and another algorithm that also computes optimal search strategies in $\Theta(D \cdot N \log N)$ time. We design a $\Theta(D \cdot N)$

time algorithm that computes optimal search strategies on all input instances except a class of non-trivially crafted instances, and this algorithm is implementation-wise simpler than all optimal algorithms. An existing non-optimal heuristic takes $\Theta(N)$ time. We design an $\Theta(N \cdot \log D)$ algorithm that experimentally outperforms the existing heuristic in terms of search cost and runs fast on real computers when $D \ll N$.

**Search with cost problem:** We generalize the basic search problem to the search with cost problem, in which the boxes can have different unlocking costs. We prove that this problem is strongly NP-hard[1]. We provide an efficient 8/7-approximation algorithm for $D = 2$ rounds, and a polynomial time approximation scheme for any constant number of rounds $D$. We also provide polynomial time optimal algorithms for three cases: i) when $D = N$; ii) when all $p_n$s are the same; iii) when there exist an order of boxes such that $p_n$ is monotonically non-decreasing but the corresponding cost are non-increasing.

**Searching for multiple tokens problem:** We generalize the basic search problem to multi-token search problem, in which all the boxes have the same unlocking costs, but more than one token is being searched for simultaneously. We concentrate on two cases: in the *yellow page* problem, our goal is to find any one out of the many tokens, whereas in the *conference call* problem, our goal is to find all tokens. The conference call problem has been studied Bar-Noy and Malewicz [2004], Epstein and Levin [2008]. It is NP-hard and a PTAS exists. We demonstrate a duality between the minimization version of the conference call problem and the maximization version of the yellow page problem. We design polynomial time optimal algorithm for i) monotonic tokens, in which there exist an order of boxes, such that the probabilities of tokens are non-increasing with respects to this order; ii) disjoint tokens, in which no box may contain more than one token. We also design an efficient exponential-time optimal algorithm, improving a factorial-time algorithm, using a tradeoff between

[1]For definition of strongly NP-hard, see book Vazirani [2001]

time complexity and space complexity. This algorithm increase solvable problem size on a real computer from $N = 11$ to $N = 30$. This is important in the application of making investment decisions (Weitzman [1979]). We also design a hierarchy of 12 heuristics and study their behavior and performance experimentally.

**Experimental study:** For all three directions of our research, we have conducted empirical simulation to examine the behavior our algorithms in addition to their theoretical properties. We have obtained user data from a real cellular network system. We verify the user data fellows Zipf distribution and estimate the parameter. According to the model, we generate large amount of synthetic data and data that follows other distribution to observe the behavior of our algorithms. On the basic search problem, we examine the performance of our non-optimal heuristics and measure the running time of all our algorithms on real computers. On the search with cost problem, we measure the performance of our approximation algorithms and observe that they perform remarkably better than their theoretical bounds. On the searching for multiple tokens problem, we test the optimality of our efficient polynomial-time and exponential-time algorithms for special cases. Some of them are very subtle and challenging to implement. We also empirically evaluate the performance of our 12 non-optimal heuristics and offer general guideline of heuristic selection on different problem settings.

It also worth mentioning that, in some cases, experimental study has helped us in generating theoretical result. For example, to determine the upper bound of algorithm FRO in Chapter 4, we have conducted exhaustive search on small instances that subject to fine granularity. We observe that approximation ratio is 1.142. We conjecture it as an 8/7-approximation algorithm and construct a proof.

# Chapter 3

# Improving efficiency for the basic search problem

The basic dynamic programming algorithm has been described in Chapter 2. The algorithm sequentially searches the row-minima of its dynamic programming table and its running time is $\Theta(D \cdot N^2)$. We note that the entries of the dynamic programming table do not have to be pre-computed, and each entry can be computed in $\Theta(1)$ time when needed. Therefore, in this chapter, we explore the basic search problem by improving the efficiency of the known optimal algorithms. We first introduce two optimal algorithms of complexity $\Theta(DN)$ and one optimal algorithm of complexity $\Theta(DN \log N)$, utilizing some properties of the dynamic programming table. We also present a hierarchy of non-optimal heuristics, of complexity from $O(1)$ to $O(DN)$, to demonstrate the tradeoff between optimality and complexity.

## 3.1 Related work

The papers Rose and Yates [1995], Goodman et al. [1996], Madhavapeddy et al. [1996], Krishnamachari et al. [2004] describe how to trade the search cost for the delay constraint. They show how to find a search strategy that uses at most $D$ rounds

($1 \leqslant D \leqslant N$) and that minimizes the expected number of unlocked boxes until the token is found. The running time of their algorithms, that are based on a dynamic programming formulation, is $\Theta(DN^2)$, i.e., on the order of $DN^2$, ignoring multiplicative constants. In papers Rose and Yates [1995], Goodman et al. [1996], Madhavapeddy et al. [1996] the strategy is computed starting from the first round, whereas in paper Krishnamachari et al. [2004] it is computed starting from the last round. In this work we take the later approach, and it has been reviewed in Section 2.2.

In the context of paging mobile users in cellular networks: Modeling uncertainty of locations of mobiles as a probability distribution vector is studied in, e.g., Rose [1999], which discusses a framework for measuring uncertainty. The paper Lyberopoulos et al. [1995] provides a simple strategy for two rounds. The papers Rose and Yates [1995], Goodman et al. [1996], Madhavapeddy et al. [1996], Krishnamachari et al. [2004] 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 Rose and Yates [1995], Krishnamachari et al. [2004] also study how to minimize the expected number of paged cells given the average (as opposed to worst-case) delay constraint using relaxation to a continuous model (Rose and Yates [1995]) or with a weakly polynomial dynamic programming solution (Krishnamachari et al. [2004]). The papers Wang et al. [2001], Wang and Xue [2006] present sub-optimal algorithms which are computationally more efficient than the dynamic programming ones. The papers Bar-Noy and Mansour [2004], Bar-Noy and Klukowska [2007] address the case in which the exact values of the probabilities are not known. In the model studied in Bar-Noy and Mansour [2004], the probabilities are learned online and in the model studied in Bar-Noy and Klukowska [2007], only estimate values of the probabilities that are accessible to the algorithm, due to privacy concerns. The effect of queuing on paging delay when paging requests arrive to the system according to some random

process and each request is to page a single mobile user is studied by Goodman et al. [1996], Rose and Yates [1997], Gau and Haas [2004].

## 3.2 Efficient optimal algorithms

Following the same dynamic programming scheme in Section 2.2, we provide a alternative view of the solution.

**Definition 3.1.** For fixed $d$, with $2 \leq d \leq D$, let $A = A^{(d)}$ be the $N \times N$ matrix with entries:

$$a_{n,j}^{(d)} = h_j^{(d-1)} + n \cdot \sum_{i=j+1}^{n} p_i,$$

for $d \leq n \leq N$ and $d-1 \leq j \leq n-1$. In other words, $a_{n,j}^{(d)}$ is the cost of unlocking the first $j$ boxes optimally in $(d-1)$ rounds and the next $(n-j)$ boxes in the $d$-th round.

Let $N' = N - d + 1$, then matrix $B$ is an $N' \times N'$ sub-matrix of $A$, where $b_{n',j'} = a_{n'+d-1,j'+d-2}^{(d)}$, $1 \leq j' \leq n' \leq N'$.

We remark that the rest of the entries of $A$ are irrelevant since in each round at least one box is unlocked. We can set them to $\infty$ or $a_{n,j}^{(d)} = h_j^{(d-1)}$, when $j \geq n$. $B$ is the submatrix of $A$ that precisely contains relevant entries (see Figure 3.1). We will use interchangeably $A$ and $B$; their index variables are related as follows:

$$n' = n - (d-1) \quad \text{and} \quad j' = j - (d-2). \tag{3.1}$$

Recall Lemma 2.9 and Algorithm 2.1. For a fixed $d$, the innermost loop in Algorithm 2.1 computes the minimum in each of the $N'$ rows of $A^{(d)}$ with relevant entries. Formally, the input to the Row-Minima problem is an $N' \times N'$ lower triangular matrix $B$ and the objective is to find the minimum value in each row. Alternatively, the objective is to compute the following sequence $\{j'(n')\}_{n'=1}^{N'}$:

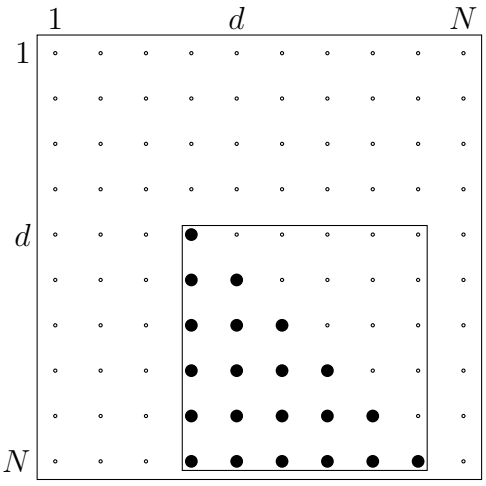



Figure 3.1: Matrix $A^{(d)}$, matrix $B$, and the lower triangular part of the latter

**Definition 3.2.** For $1 \leq n' \leq N'$, let $j'(n')$ be the smallest column index in which the minimum value of row $n'$ appears in matrix $B$. Alternatively, for $d \leq n \leq N$, let $j(n)$ be the smallest column index in which the minimum value of row $n$ appears in matrix $A = A^{(d)}$.

In the following theorem, we state the complexity relationship between the Row-Minima problem of matrix $B$ and the basic search problem.

**Theorem 3.3.** *Let algorithm $\mathcal{X}$ be an algorithm that solves the Row-Minima problem for the matrix $B$ in time $\Theta(T(N'))$ bounded by a polynomial with respect to $N'$. If $T(N') = \Theta(N')$, then the overall running time of algorithm $\mathcal{X}$ on the basic search problem is $\Theta(D \cdot T(N))$.*

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

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

Including the schematic Algorithm 2.1, we now describe four algorithms, respectively named Seq, Bin, SMAWK, and SpeedUp, that are based on the *Dynamic Programming Scheme*. For each of them, we first show how to find the vector $(j(d), \ldots, j(N))$, i.e., solve the *Row-Minima problem* for the matrix $A$ (or $B$), and analyze its time complexity. Then, for each algorithm the overall time complexity is an immediate corollary of Theorem 3.3.

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

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

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

**Corollary 3.4.** *The running time of Seq is $\Theta(DN^2)$.*

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

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

**Lemma 3.5.** *For $d \leq n_1 < n_2 \leq N$, $j(n_1) \leq j(n_2)$.*

*Proof.* Assume for the sake of contradiction that there is some $n > d$ such that

$j_1 = j(n) < j(n-1) = j_2$. Then:

$$a^{(d)}_{n-1,j_1} > a^{(d)}_{n-1,j_2} \quad \text{and} \quad a^{(d)}_{n,j_2} \geq a^{(d)}_{n,j_1}$$

and thus $a^{(d)}_{n-1,j_1} + a^{(d)}_{n,j_2} > a^{(d)}_{n-1,j_2} + a^{(d)}_{n,j_1}$, which, from Definition 3.1, implies

$$(n-1) \cdot \sum_{i=j_1+1}^{n-1} p_i + n \cdot \sum_{i=j_2+1}^{n} p_i > (n-1) \cdot \sum_{i=j_2+1}^{n-1} p_i + n \cdot \sum_{i=j_1+1}^{n} p_i$$

or $\sum_{i=j_1+1}^{j_2} p_i < 0$, which is a contradiction. □

**Row-Min-Bin($A$):** Instead of computing $j(n)$ sequentially, compute it in a binary fashion. Informally (ignoring ceilings and floors), the algorithm has $\log_2 N$ stages. In the first stage, it computes $j(N/2)$; in the second stage, it computes $j(N/4)$ and $j(3N/4)$; and in general, in the $i$th stage it computes $j(kN/2^i)$ for all odd numbers $k$ between 1 and $2^i$. Formally, we can apply Algorithm 3.1 with parameters RowMinBin$(D, N, D, N)$ on matrix $A^{(d)}$.

**Algorithm 3.1** Algorithm Row-Min-Bin: rowMinima = RowMinBin$(r_l, r_r, c_l, c_r)$

**Input:** Probabilities $\langle p_1, \ldots, p_N \rangle$, row indices bounds $r_l$ and $r_r$, column indices bounds $c_l$ and $c_r$.
**Output:** $\langle j(r_l), \ldots, j(r_r) \rangle$

```
if r_l < r_r then
    return
else if r_l = r_r then
    j(r_l) ← arg min_{c_l ≤ j ≤ c_r} a^(d)_{r_l,j}
    return ⟨j(r_l)⟩
else
    pivot ← ⌊(r_l + r_r)/2⌋
    j(pivot) ← arg min_{c_l ≤ j ≤ c_r} a^(d)_{pivot,j}
    left ← RowMinBin(r_l, pivot − 1, c_l, j(pivot))
    right ← RowMinBin(pivot + 1, r_r, j(pivot), c_r)
    return ⟨right, j(pivot), right⟩
end if
```

By Lemma 3.5, each stage can be computed in $\Theta(N)$ time because each one of

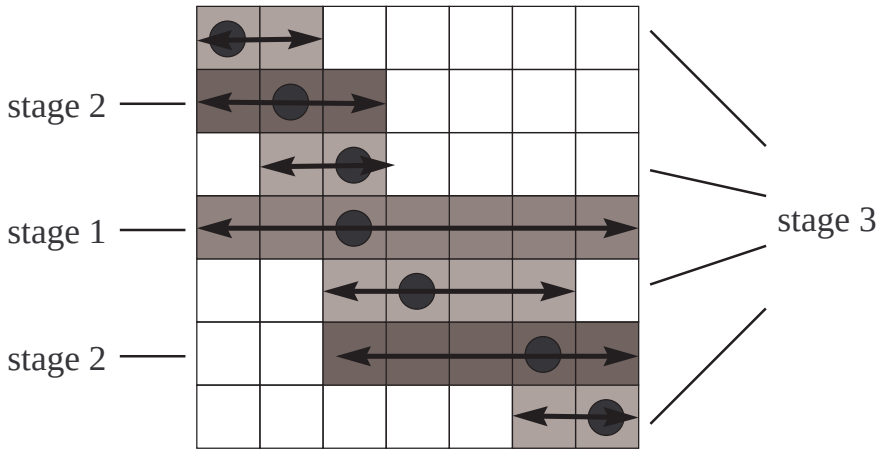



Figure 3.2: Illustration of Row-Min-Bin: $7 \times 7$ matrix. In stage 1, we search the minimum of row 4 ($N/2$) through all entries. In stage 2, we search minima of rows 2 and 6, through the segments divided by the minimum of row 4. In state 3, we search minima of rows 1, 3, 5, and 7 through the entries between the minima indices of adjacent rows. There are 3 ($\log N$) stages, and each stage can be computed in $\Theta(N)$ time.

the $2^{i-1}$ values that are computed in stage $i$ is computed over a unique range and the total size of all these ranges is $N$. Since there are $\log_2 N$ stages, the running time of Row-Min-Bin is $\Theta(N \log N)$. Figure 3.2 illustrates the algorithm with a $7 \times 7$ matrix example.

**Corollary 3.6.** *The running time of Bin is $\Theta(DN \log N)$.*

### 3. The $\Theta$(DN) SMAWK algorithm:

We prove a stronger property for the relevant entries of matrix $A$ (or $B$), named the *Monge* property (in particular, Monge property implies Lemma 3.5).

**Definition 3.7.** An $N' \times N'$ matrix $M$ satisfies the lower triangular *Monge* property Burkard et al. [1996] if for any $1 < j' < n' \leq N'$:

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

**Lemma 3.8.** *Matrix $B$ satisfies the lower triangular Monge property.*

*Proof.* Because of (3.1), it is enough to prove:

$$a^{(d)}_{n-1,j-1} + a^{(d)}_{n,j} \leq a^{(d)}_{n,j-1} + a^{(d)}_{n-1,j}$$

or equivalently:

$$\begin{aligned}
&(a^{(d)}_{n-1,j-1} + a^{(d)}_{n,j}) - (a^{(d)}_{n,j-1} + a^{(d)}_{n-1,j}) \\
=&\left(h^{(d-1)}_{j-1} + (n-1)\cdot\sum_{i=j}^{n-1} p_i + h^{(d-1)}_{j} + n\cdot\sum_{i=j+1}^{n} p_i\right) - \\
&\left(h^{(d-1)}_{j-1} + n\cdot\sum_{i=j}^{n} p_i + h^{(d-1)}_{j} + (n-1)\cdot\sum_{i=j+1}^{n-1} p_i\right) \\
=&(n-1)p_j - np_j = -p_j < 0
\end{aligned}$$

□

In Aggarwal et al. [1987] the authors introduce an algorithm that solves the Row-Minima problem for Monge matrices in $\Theta(N)$ time. The paper Burkard et al. [1996] gives an alternative description of this algorithm and call it *SMAWK*[1].

**Theorem 3.9.** *If an $N \times N$ matrix satisfies the Monge property, then all of its row minima can be found in $\Theta(N)$ time.*

**Row-Min-SMAWK**$(A)$**:** Apply the SMAWK technique from Aggarwal et al. [1987] to solve the Row-Minima problem for the matrix $A$. The SMAWK algorithm reduces an $N \times N$ matrix to an $\frac{N}{2} \times \frac{N}{2}$ matrix with the same row minima as the odd row minima of the original $N \times N$ matrix, computes row minima of the $\frac{N}{2} \times \frac{N}{2}$ matrix recursively, and finally uses them to compute the row minima of the original matrix (see Algorithm 3.2 for a high-level description and Figure 3.3 for an example run of the algorithm).

**Corollary 3.10.** *The running time of SMAWK is $\Theta(DN)$.*

**Algorithm 3.2** Row-Min-SMAWK: Compute indices of row minima of matrix $M_{N\times N}$; rowMinima = SMAWK($M$)

**Input:** $M$
**Output:** $\langle j(D), \ldots, j(N)\rangle$
  **if** $M$ is at most $2 \times 2$ **then** {base: brute force search}
    rowMinima $\leftarrow$ findRowMinima($M$)
    **return** rowMinima
  **else** {$M$ is larger than $2 \times 2$}
    $M^\star \leftarrow$ oddRows($M$); $i \leftarrow 1$
    **while** $M^\star$ has more than $\lceil N/2 \rceil$ columns **do**
      **if** $m^\star_{i,i} \leq m^\star_{i,i+1}$ and $i + 1 < \lceil N/2 \rceil$ **then**
        $i \leftarrow i + 1$
      **else if** $m^\star_{i,i} \leq m^\star_{i,i+1}$ and $i + 1 = \lceil N/2 \rceil$ **then**
        removeColumn($M^\star, i + 1$)
      **else**
        removeColumn($M^\star, i$); $i \leftarrow \max\{i - 1, 1\}$
      **end if**
    **end while**
    oddMinima $\leftarrow$ SMAWK($M^\star$)
    {find row minima of $M$ using oddMinima from $M^\star$}
    rowMinima = rowMinimaUsing($M$, oddMinima)
    **return** rowMinima
  **end if**

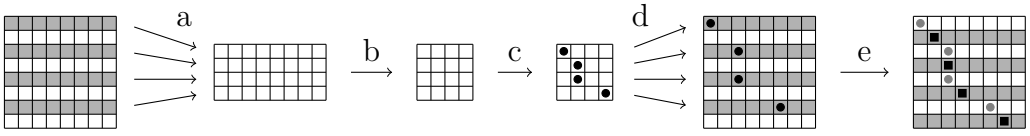



Figure 3.3: A run of the SMAWK algorithm on a $N \times N$ matrix: (a) extracting odd rows of $N \times N$ matrix in order to get a $\frac{N}{2} \times N$ matrix, (b) deleting columns in order to get a $\frac{N}{2} \times \frac{N}{2}$ matrix with the same row minima as the $\frac{N}{2} \times N$ matrix, (c) using SMAWK in a recursion to find row minima of the $\frac{N}{2} \times \frac{N}{2}$ matrix, (d) mapping row minima of the $\frac{N}{2} \times \frac{N}{2}$ matrix to odd row minima of original $N \times N$ matrix, (e) computing even row minima with the help of odd row minima in the $N \times N$ matrix.

**4. The Θ(DN) SpeedUp algorithm:**

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

**Definition 3.11.** An $N' \times N'$ matrix $M$ satisfies the *SpeedUp* property if $m_{n',j'} - m_{n'-1,j'} = u_{n'} + v_{j'}$ for $1 \leq j' < n' \leq N'$, where $u_{n'}$ is a function of $n'$ and $v_{j'}$ is a monotonically decreasing function of $j'$.

**Lemma 3.12.** *Matrix B satisfies the SpeedUp property.*

*Proof.* Because of (3.1), it is enough to show:

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

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

We can show that for matrices that satisfy the SpeedUp property there exists an algorithm, namely *FGZ*, that solves the Row-Minima problem in $\Theta(N)$ time. This algorithm is a modification of the one given in Fleischer et al. [2006] for solving the dynamic program describing placing $K$ medians on a line, and it is shown in Algorithm 3.3. We omit the proof of correctness of the algorithm (details can be found in Fleischer et al. [2006]). We note that the time complexity is $O(N')$, because

[1] After the initials of the authors of Aggarwal et al. [1987]

**Algorithm 3.3** Row-Min-FGZ: Compute indices of row minima of matrix $B_{N' \times N'}$; rowMinima = FGZ($B$)

**Input:** $B$
**Output:** $\langle j'(1), \ldots, j'(N - D + 1) \rangle$

```
z ← empty list
for n' = 1 ... N' do
  while z has at least two elements do
    j'_1 ← first element of z
    j'_2 ← second element of z
    if (b_{n',j'_1} − b_{n',j'_2})/(v_{j'_2} − v_{j'_1}) < 0 then
      remove first element from z
    else
      exit while loop
    end if
  end while
  while z has at least one element do
    j'_1 ← last element of z
    if z has at least two elements then
      j'_2 ← next to last element of z
      x = (b_{n',j'_1} − b_{n',j'_2})/(v_{j'_2} − v_{j'_1})
    else
      x = 0
    end if
    if b_{n',n'} + v_{n'} · x < b_{n',j'_1} + v_{j'_1} · x then
      remove last element from z
    else
      exit while loop
    end if
  end while
  append n' at the end of z
  j'(n') ← first element of z
end for
```

each $b_{n',j'}$ and $v'_j$ can be computed in constant time ($v_{j'} = Q_{j'}$), and each of the possible elements $1, \ldots, N'$, of the list $z$ is appended to and deleted from the list at most one time (more details in Fleischer et al. [2006]).

**Theorem 3.13.** *If an $N \times N$ matrix satisfies the SpeedUp property, then all of its row minima can be found in $\Theta(N)$ time.*

**Row-Min-SpeedUp**($A$)**:** Apply algorithm FGZ to solve the Row-Minima prob-

lem for the matrix $B$.

**Corollary 3.14.** *The running time of SpeedUp is* $\Theta(DN)$.

## 3.3 More efficient sub-optimal heuristics

In the previous section, we show how to reduce the running time complexity of algorithms for finding *optimal* strategies from $\Theta(DN^2)$ to $\Theta(DN \log N)$ and then to $\Theta(DN)$. In highly dynamic systems the probabilities might change frequently and for large $N$ even $\Theta(DN)$ might be too slow. Moreover, in all of the optimal algorithms, we observe that the actual constant factor in the running time is not small enough, e.g., the constant factor of the SMAWK algorithm is 7 and is comparable to $\log N$ when $N$ is at the level of hundreds to thousands. We therefore propose several non-optimal heuristics whose running time is better than $\Theta(DN)$.

We first present three oblivious algorithms 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 has been proposed in Wang et al. [2001] 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 Section 6.2, we will demonstrate that for $D \ll N$, our heuristic outperforms the heuristic of Wang et al. [2001] in both categories: performance and running time. Our last heuristic has a $\Theta(DN)$ running time and is much simpler to describe and implement than the two optimal $\Theta(DN)$ running time algorithms. Moreover, we note that this heuristic provided the optimal solution for most of the instances in our simulations.

### 3.3.1 Oblivious heuristics

The following three heuristics depend only on the value of $N$ and $D$ and ignore the values of $p_1, \ldots, p_N$. All of them can be computed in $\Theta(D)$ time assuming the boxes 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 Bar-Noy and Klukowska [2007].

**LargeSuffix:** In rounds 1 to $(D-1)$ unlock only one box. In the last round unlock the rest of the $(N-D+1)$ boxes.

This heuristic performs very well when the token is likely to be found in $D-1$ boxes. It performs poorly when the values of the $N$ probabilities are almost the same.

**Uniform:** In the first $(D-(N \bmod D))$ rounds unlock $\lfloor N/D \rfloor$ boxes; in the last $(N \bmod D)$ rounds unlock $\lceil N/D \rceil$ boxes.

This heuristic performs very well when the values of the $N$ probabilities are almost the same. It performs poorly when the token is likely to be found in one particular box. In a way, *Uniform* and *LargeSuffix* complement each other.

**Doubling:** Find a parameter $\alpha$ such that $\alpha + \alpha^2 + \cdots + \alpha^D = N$ ($\alpha \approx N^{1/D}$). For $1 \leq d \leq D$, in round $d$ unlock either $s_d = \lfloor \alpha^d \rfloor$ or $s_d = \lceil \alpha^d \rceil$ boxes. The floor and ceiling decisions are made such that $s_1 + \cdots + s_D = N$ and $s_1 \leq s_2 \leq \cdots \leq s_D$.

This heuristic is a compromise between Uniform and LargeSuffix. The Doubling technique (usually $\alpha = 2$) is a well-known and useful technique when some of the parameters are not known in advance. Indeed, in Bar-Noy and Klukowska [2007], it is shown that Doubling has the best worst-case performance against an adversary that may choose maliciously the values for $p_1, \ldots, p_N$.

### 3.3.2 The boundaries heuristic

Wang et al. [2001] describes a non-optimal heuristic. It first sort the boxes by non-increasing order of $p_n$s. Then it starts with some initial partition of the boxes into $D$

sets, e.g., a partition of boxes of almost equal sizes. The authors prove two necessary conditions that an optimal search strategy must satisfy for the boxes at the boundaries between sets. Then boxes at the boundaries of the sets are moved from one set to an adjacent set if some conditions are not satisfied. Such adjustment is continued until both conditions are satisfied. The two conditions are stated in the following definition.

**Definition 3.15.** Let $\{S_1, \ldots, S_D\}$ be a partition of the $N$ boxes, let $n_i$ be the size of the subset $S_i$, let $\ell_i$ be the largest probability in $S_i$, and let $s_i$ be the smallest probability in $S_i$.

Then, the *forward* and the *backward* boundary condition are, respectively:

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

**Boundaries:** Partition the $N$ boxes 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 box 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 box to the previous set each time the condition is violated.

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

**Proposition 3.16.** *The running time of Boundaries is $\Theta(N)$.*

We note that the initial partition could be any of the partitions generated by the three oblivious heuristics. Our simulations in Section 6.2 show that the choice made by Wang et al. [2001] was the best for most of the instances tested by the simulation.

### 3.3.3 The divide and conquer heuristic

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

**DQ:** In stage 1, find the best partition of the $N$ boxes into two sets such that each set gets $D/2$ unlocking rounds. For $2 \leq i \leq \log_2 D$, in stage $i$, partition the $N$ boxes into $2^i$ sets such that each set gets $D/2^i$ unlocking rounds. The final partition is the one after stage $\log_2 D$. To divide a set of boxes into two, we look for a pivot box $B_m$ in the set, such that boxes up to $B_m$ are unlocked in the previous round(s), boxes after $B_m$ are unlocked in the later round(s), and the expected cost of unlocking the two sets is minimized. Formally, we apply Algorithm 3.4 by $\mathrm{DQ}(\langle p_1, \ldots, p_N \rangle, 1, D, 1, N)$.

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

**Proposition 3.17.** *The running time of DQ is $\Theta(N \log D)$.*

### 3.3.4 The first local minimum heuristic

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

The search for values $j(d), \ldots, j(N)$ takes $\Theta(N')$ time since the minimum search ranges overlap only at their extremes, and therefore immediately from Theorem 3.3:

**Corollary 3.18.** *The running time of FirstLocalMin is $\Theta(DN)$.*

**Algorithm 3.4** Divide and conquer: Recursively divide boxes $\{B_{n_1} \dots B_{n_2}\}$ that contain rounds $\{d_1 \dots d_2\}$ into two parts. parts = DQ($\mathbf{p}, d_1, d_2, n_1, n_2$)

**Input:** $\mathbf{p}, d_1, d_2, n_1, n_2)$
**Output:** $\langle S_{d_1}, \dots, S_{d_2} \rangle$
  **if** $d_1 = d_2$ or $n_1 = n_2$ **then**
    **return** $[n_1 \dots n_2]$
  **else**
    $d \leftarrow \lfloor (d_1 + d_2)/2 \rfloor$
    $m \leftarrow \underset{n_1+d-d_1 \le i \le n_2+d_2-d}{\arg\min} \{i \cdot \sum_{n_1 \le j \le i} p_j + n_2 \cdot \sum_{i < j \le n_2} p_j\}$
    $\mathcal{S}_{\text{left}} = \text{DQ}(\mathbf{p}, d_1, d, n_1, m)$
    $\mathcal{S}_{\text{right}} = \text{DQ}(\mathbf{p}, d+1, d_2, m+1, n_2)$
    **return** $\mathcal{S}_{\text{left}}, \mathcal{S}_{\text{right}}$
  **end if**

A heuristic solution is usually compared against an optimal solution algorithm by the ratio of the cost computed by the heuristic to the cost of an optimal solution. Although our heuristic has computed the optimal solution in all simulations we have tested, we prove that for some specially crafted inputs it can compute solutions quite far from optimal, settling an open problem in Bar-Noy et al. [2007].

**Proposition 3.19.** *There are inputs for which FirstLocalMin finds a solution worse than optimal, even for $D = 3$, and the ratio of the two solutions is unbounded.*

*Proof.* Consider the following probability vector with $N$ probabilities:

$$p = (x, x, \overbrace{sy, \dots, sy}^{t}, \overbrace{y, \dots, y}^{N-2-t}),$$

where $x < 1/2$, $s$, and $t$ are parameters, to be specified later. Since $\sum_{i=1}^{N} p_i = 1$, the value of $y$ can be computed from the parameters as follows:

$$y = \frac{1 - 2x}{N - 2 - t + st}.$$

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

$$\mathcal{S}_{\text{FLM}} = \langle \{1\}, \{2\}, \{3, \ldots, N\} \rangle ,$$
$$\mathcal{S} = \langle \{1, 2\}, \{3, \ldots, t+2\}, \{t+3, \ldots, N\} \rangle .$$

The respective costs of the two strategies are:

$$C_{\text{FLM}} = 3x + (1 - 2x)N,$$
$$C = 4x + syt(t+2) + (N - 2 - t)yN.$$

In order for FirstLocalMin to output strategy $\mathcal{S}_{\text{FLM}}$, it must be the case that: for all $n \in [4, N]$, $a^{(3)}_{n,2} \leq a^{(3)}_{n,j}$ or equivalently,

$$\forall n \leq N : a^{(3)}_{n,3} - a^{(3)}_{n,2} = x + 3sy - syn \geq 0,$$

or, since $n \leq N$,

$$x + 3sy - syN \geq 0. \tag{3.2}$$

We need to choose the parameters $s$, $t$, $x$, so that condition (3.2) is true and the ratio $C_{\text{FLM}}/C$ goes to infinity as a function of $N$. We set $s = N^{\sigma}$, $t = N^{\tau}$, and $x = \frac{1}{2} - \frac{1}{2}N^{-\alpha}$, where $\sigma$, $\tau$, $\alpha$ are positive reals. With these new parameters, condition (3.2) is satisfied whenever

$$\alpha + \sigma + \tau > 1 + \alpha \quad \text{and} \quad \alpha + \sigma + \tau > 1 + \sigma. \tag{3.3}$$

If $1 - \alpha > 0$, the cost of each partition is:

$$C_{\text{FLM}} = \Theta(N^{1-\alpha}) \text{ and } C = \Theta(N^{\tau-\alpha} + N^{2-(\alpha+\sigma+\tau)}),$$

and by choosing $\alpha = 1 - \varepsilon/2$, $\sigma = 2$, and $\tau = \varepsilon$, where $\varepsilon$ is a positive real arbitrarily close to 0, condition (3.3) is satisfied and the ratio becomes $C_{\text{FLM}}/C = \Theta(N^{1-\varepsilon})$. □

## 3.4 Summary and open problems

In this chapter, we have concentrated on developing more efficient algorithms for the basic search problem. We first reduce the basic search problem to the problem of finding row-minima of matrices. We then observe that the matrices satisfy the Monge property and the Speed-Up property, and provide two non-trivial linear-time optimal algorithms: SMAWK and SpeedUp. We also design a series of straightforward heuristics of complexity hierarchy $\Theta(1)$, $\Theta(D)$, $\Theta(N)$, $\Theta(N \log D)$, and $\Theta(DN)$. We experimentally study the performance of all algorithms in terms of optimality and running time. The comprehensive experimental study is presented in Section 6.2.

Some open problems remain: The tradeoff between the running time and optimality can be refined and possibly improved. In particular, we would like to see a faster than $\Theta(DN)$ optimal algorithm or a proof showing that $\Theta(DN)$ is the fastest optimal algorithm that one can achieve. The two $\Theta(DN)$ optimal algorithms, SMAWK and SpeedUp, are both implementation-wise non-trivial. Implementation-wise straightforward $\Theta(DN)$ optimal algorithms are desired. Algorithm FirstLocalMin is close to such goal – it is non-optimal only on the class of instances that we have subtly crafted. We look for a better tradeoff between complexity and optimality. In particular, we would like to design heuristics that are either faster or of less cost than our current heuristics. We also look for proved upper bounds for our non-optimal heuristics. In dynamic systems the values of box probabilities might change 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 $\Theta(TDN)$.

# Chapter 4

# Search with cost problem

In Chapter 2, we have introduced the basic search problem, in which the unlocking cost of all boxes are the same. In real world applications, including the paging mobile users problem as well as searching for information in distributed data centers problem, the unlocking costs of different boxes can be different. In this chapter, we study this generalized problem that the boxes can have different unlocking costs. Our objective is to minimize the expected unlocking cost of boxes (in contrast to minimizing the expected number of unlocked boxes) under a delay constraint.

## 4.1 Related work

We have reviewed the literature closely related basic search problem in Chapter 3. Here we only review the literature that are directly related to the search with cost problem. The paper Chang and Liu [2004] explores a similar problem in which the order of boxes is dictated in the context of time-to-live flood searching in sensor networks. Papers Chandra and Wong [1975], Cody and Coffman [1976], Alon et al. [1998] discuss a task scheduling problem on multiple machines with the objective of minimizing the $L_2$ norm of completion time on all machines. They serve as a theoretical foundation on the search with cost problem.

## 4.2 Preliminaries

**The search with cost problem:** Denote the $N$ boxes by $\{B_1, B_2, \ldots, B_N\}$. Let $\mathbf{p} = \langle p_1, p_2, \ldots, p_N \rangle$ be the vector of independent probabilities of the token being respectively placed in these boxes. Let $\mathbf{w} = \langle w_1, w_2, \ldots, w_N \rangle$ be the vector of costs of unlocking these boxes respectively. Denote the delay constraint for finding the token by $D$, $1 \leq D \leq N$.

An *instance* to the search with cost problem is the quadruple $\mathcal{I} = (N, D, \mathbf{w}, \mathbf{p})$. An instance $\mathcal{I}$ is a *uniform cost instance* if $w_1 = \cdots = w_N = 1/N$ and a *uniform probability instance* if $p_1 = \cdots = p_N = 1/N$.

A *search strategy* $\mathcal{S} = \langle S_1, \ldots, S_D \rangle$ is an ordered $D$-partition of the boxes, such that in the $d$-th round, all the boxes in the set $S_d$ are unlocked. The search process terminates in round $d$ if the token is found in one of the boxes of the set $S_d$. For a given search strategy $\mathcal{S}$ and a round $1 \leq d \leq D$, the *round probability* is $P_d = \sum_{B_i \in S_d} p_i$ and the *round cost* is $W_d = \sum_{B_i \in S_d} w_i$.

**Definition 4.1.** The expected unlocking cost of a search strategy $\mathcal{S}$ on an instance $\mathcal{I} = (N, D, \mathbf{w}, \mathbf{p})$ is denoted by $\text{cost}(\mathcal{S}, \mathcal{I})$. When the definition of $\mathcal{I}$ is clear it is denoted by $\text{cost}(\mathcal{S})$.

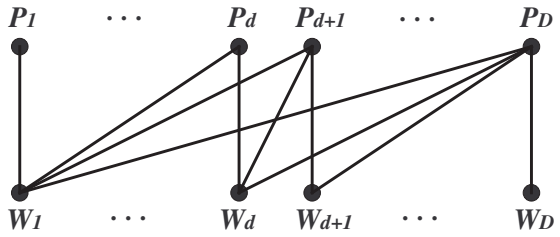
**Proposition 4.2.** *The following are two different but equivalent ways to compute the* search cost *of a search strategy $\mathcal{S} = \langle S_1, \ldots, S_D \rangle$ on the instance $\mathcal{I} = (N, D, \mathbf{w}, \mathbf{p})$:*

$$\textit{cost}(\mathcal{S}, \mathcal{I}) = \sum_{d=1}^{D} \left( P_d \sum_{i=1}^{d} W_i \right) \tag{4.1}$$

$$\textit{cost}(\mathcal{S}, \mathcal{I}) = \sum_{d=1}^{D} \left( W_d \sum_{i=d}^{D} P_i \right) \tag{4.2}$$

*Proof.* Eq. (4.1) follows since with probability $P_d$ the token is found during the $d$-th round and the strategy pays the cost of the first $d$ sets $S_1, \ldots, S_d$ of the partition. Eq. (4.2) follows since the strategy pays the cost of the $d$-th round only if the token

is in a box belonging to the last $(D - d + 1)$ sets $S_d, \ldots, S_D$ of the partition.




The cost is the sum of the products of every pair of connected terms in the figure above. □

An *optimal search strategy* is a search strategy $\mathcal{O}$ such that cost($\mathcal{O}$) is the minimum among all possible search strategies. An *optimal algorithm* is an algorithm that generates optimal search strategies for all possible instances. Let OPT be an optimal algorithm and let algorithm ALG be another search algorithm. ALG is a $(1 + \varepsilon)$-approximation, if

$$\frac{\text{cost(ALG)}}{\text{cost(OPT)}} \leq 1 + \varepsilon \ .$$

for any instance.

**Normalizing the cost and the probability vectors:** We observe the following basic fact that allows us to assume without loss of generality that $\sum_{i=1}^{N} w_i = 1$ and that $\sum_{i=1}^{N} p_i = 1$.

**Proposition 4.3.** *Let $\mathcal{O}$ be an optimal search strategy on boxes with probabilities $p_i$ and costs $w_i$, $1 \leq i \leq N$. $\mathcal{O}$ is also an optimal search strategy on boxes with probabilities $p'_i$ and costs $w'_i$, if $w'_i = c_w \cdot w_i$ and $p'_i = c_p \cdot p_i$, where $1 \leq i \leq N$ and $c_w$ and $c_p$ are positive constants. The approximation ratio of any non-optimal solution is retained, too.*

The case when the sum of the probabilities is strictly less than 1 can model situations in which with some positive probability the token "disappeared" (in Cellular Networks, the user is *off*). Obviously, if the token is not found in any of the $N$ boxes

then the system can deduce that the token disappeared without any additional cost. This happens with probability $1 - \sum_{i=1}^{N} p_i$. Without loss of generality, we assume that the token is in one of the boxes (i.e., $\sum_{i=1}^{N} p_i = 1$) and that the searcher must unlock the box with the token. With this requirement, even if the searcher knows that the token is in box $B_N$ because it was not found in boxes $B_1, \ldots, B_{N-1}$, the searcher still must pay the unlocking cost $w_N$. It is not hard to see that with this assumption, we cover both cases when the token is in one of the boxes and when the token disappeared.

**Types of tokens:** We classify tokens by their *typicality*. In one extreme, the probability vector of a *typical token* is proportional to its cost vector. By Proposition 4.3, without loss of generality, for a typical token, $p_i = w_i$ for any box $B_i$. In the other extreme, an *atypical token* is more likely to be located in lower cost boxes. Formally, the costs and the probabilities are in opposite order. A token with no typicality association is called an *arbitrary* token. Such a token may be located in any box $B_i$ with arbitrary cost $w_i$ and arbitrary probability $p_i$.

**Optimal polynomial time algorithms:** We say that an ordered partition $\mathcal{S} = \langle S_1, \ldots, S_D \rangle$ *respects the order of boxes* $\langle B_1, B_2, \ldots, B_N \rangle$ if there are no $S_i$, $S_j$ with $i < j$, such that $B_{i'} \in S_i$ and $B_{j'} \in S_j$ with $i' > j'$. Given an order of the boxes, one can find a minimum cost partition that respects that order in polynomial time by slightly modifying the dynamic programming methods described Section 2.2 to include costs of boxes. A naive implementation implies an $O(D \cdot N^2)$ algorithm. We next show a more efficient implementation.

**Theorem 4.4.** *The dynamic programming scheme in Section 2.2 can be implemented in $\Theta(D \cdot N)$ time to find a minimum cost partition that respects a given order of the boxes.*

*Proof.* An analog of Lemma 3.5 holds with $Y - X = w_{n+1} \sum_{k=i+1}^{j} p_k \geq 0$, which implies a $O(D \cdot N \log N)$ implementation. An analog of lemma 3.8 holds with $Y - X =$

$p_{j+1}w_{n+1} \geq 0$, which implies a $O(D \cdot N)$ algorithm (Monge property, see Aggarwal et al. [1987], Burkard et al. [1996] ). An analog of lemma 3.12 holds with:

$$B[n+1,j] - B[n,j] = \underbrace{p_{n+d}\Big(\sum_{l=1}^{n+d} w_l\Big) + w_{n+d}\Big(\sum_{k=1}^{n+d-1} p_k\Big)}_{C(n)} + \underbrace{\Big(-w_{n+d}\Big(\sum_{k=1}^{j+d-2} p_k\Big)\Big)}_{D(j)}$$

which implies a simpler $O(D \cdot N)$ algorithm (SpeedUp property, see Fleischer et al. [2006] ). □

Unfortunately, for the general problem, as we will prove later, it is impossible to find in polynomial time the order of boxes in an optimal search strategy unless P = NP.

**Algorithm FRO:** Consider the $N$ boxes ordered by the non-increasing order of the $p_i/w_i$ ratio. That is, $p_1/w_1 \geq p_2/w_2 \geq \cdots \geq p_N/w_N$. We call the algorithm that computes the minimum cost partition that respects the above order Algorithm FRO (follow ratio order).

## 4.3 Special cases with polynomial time optimal algorithms

We first present two lemmas that reveal some properties an optimal searching strategy must retain. Based on these lemmas, we show three special cases in which optimal search strategies can be found by polynomial time algorithms.

**Lemma 4.5.** *If there is an order of the boxes such that $p_1 \geq \ldots \geq p_N$ while $w_1 \leq \ldots \leq w_N$, then algorithm FRO follows this order and generates an optimal search strategy.*

*Proof.* Assume towards a contradiction that there exists an optimal solution $\mathcal{S} = \langle S_1, \ldots, S_D \rangle$ that does not respect the non-decreasing order of the ratios $p_i/w_i$. This

implies the existence of indices $j < i$ (and therefore $p_i < p_j$ and $w_i > w_j$) such that $B_i \in S_d$ and $B_j \in S_{d+1}$ for some $1 \leq d < D$. Define a new partition $\mathcal{S}'$ which is almost identical to $\mathcal{A}$ except that $B_i$ and $B_j$ are swapped. To get a contradiction, we show that the cost of $\mathcal{S}'$ is smaller than the cost of $\mathcal{S}$.

Let $P_d = P'_d + p_i$, $W_d = W'_d + w_i$, $P_{d+1} = P'_{d+1} + p_j$, and $W_{d+1} = W'_{d+1} + w_j$. A careful examination of the terms in Eq. (4.1) from Proposition 4.2 reveals that both $\text{cost}(\mathcal{S})$ and $\text{cost}(\mathcal{S}')$ share some identical terms while $\text{cost}(\mathcal{S})$ has unique terms $P'_d w_i + p_j w_i + p_j W'_{d+1}$ and $\text{cost}(\mathcal{S}')$ has unique terms $P'_d w_j + p_i w_j + p_i W'_{d+1}$

$$\text{cost}(\mathcal{S}) = \text{identical terms} + P'_d w_i + p_j w_i + p_j W'_{d+1}$$
$$\text{cost}(\mathcal{S}') = \text{identical terms} + P'_d w_j + p_i w_j + p_i W'_{d+1}$$

where by "identical terms" we mean the terms that appear in both costs. Now, since $p_i < p_j$ and $w_i > w_j$, it follows that $\text{cost}(\mathcal{S}') < \text{cost}(\mathcal{S})$. □

**Lemma 4.6.** *Let $\mathcal{S} = \langle S_1, \ldots, S_D \rangle$ be an optimal search strategy for an arbitrary instance $\mathcal{I}$. Then*

$$\frac{P_d}{W_d} \geq \frac{P_{d+1}}{W_{d+1}}$$

*for $1 \leq d < D$.*

*Proof.* Fix $d$, $1 \leq d < D$. Define another search strategy

$$\mathcal{S}' = \langle S_1, \ldots, S_{d-1}, S_{d+1}, S_d, \ldots, S_D \rangle$$

that is obtained from $\mathcal{S}$ by swapping $S_d$ and $S_{d+1}$. By Proposition 4.2, it follows that

$$\text{cost}(\mathcal{S}') - \text{cost}(\mathcal{S}) = P_d W_{d+1} - P_{d+1} W_d \ .$$

This is because in both strategies all the costs incurred by $W_i$ for $i < d$ and $i > d+1$ are the same and in both strategies the terms $P_d W_d$ and $P_{d+1} W_{d+1}$ are part of the cost. Now, since $\mathcal{S}$ is optimal, it follows that

$$P_d W_{d+1} - P_{d+1} W_d \geq 0 \ ,$$

which is equivalent to

$$\frac{P_d}{W_d} \geq \frac{P_{d+1}}{W_{d+1}} \ .$$

□

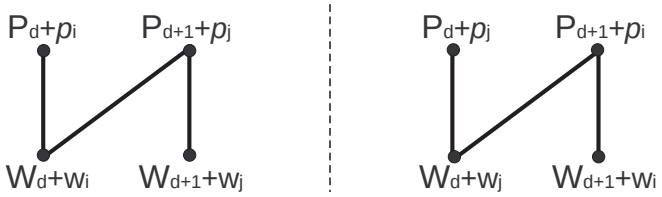



Figure 4.1: Proof illustration of Lemma 4.5

**Uniform Probabilities:** The uniform probability case is when all the probabilities are $1/N$ (in contrast to uniform cost case in Chapter 3). This is the case when the searcher has no clue for the whereabouts of the token but still knows the unlocking costs associated with the boxes. We show that the algorithm FRO yields an optimal solution if the order of optimal searching strategy is known for any number of rounds $2 \leq D \leq N$. In the uniform probabilities case, Lemma 4.5 provides this optimal order by sorting the boxes in *non-decreasing* order of unlocking cost $w_i$, by which FRO generates optimal polynomial time solutions.

**Atypical token:** Recall that an atypical token is one that appears in boxes of lower unlocking costs with higher probabilities. For an atypical token, the non-increasing order of the probability vector corresponds to the non-decreasing order of the cost vector. Lemma 4.5 directly implies that an optimal search strategy can be

found in polynomial time for such tokens.

**D = N rounds:** The following corollary is a direct consequence of Lemma 4.6, which implies that the polynomial time algorithm FRO generates an optimal search strategy for the case $D = N$.

**Corollary 4.7.** *For $D = N$, the optimal search strategy unlocks the boxes in a non-increasing order of $p_i/w_i$, one box per round.*

## 4.4 Typical tokens

We discuss and analyze search strategies for the special case of typical tokens. Recall that a typical token is more likely to be found in high cost boxes and therefore, after normalizing the cost and probability values, we assume that $p_i = w_i$ for all $1 \leq i \leq N$. We prove that the problem is strongly NP-Hard even in this special case. In particular, we show that the problem is essentially a variation of a known load balancing problem. This enables us to apply a known *polynomial time approximation scheme* (PTAS) solution for the load balancing problem to our problem of searching for typical tokens. In addition, we analyze the performance of a natural load balancing greedy algorithm applied to the search problem. We first show how to compute the cost for a typical token by simplifying the equations from Proposition 4.2.

**Proposition 4.8.** *Let $\mathcal{I} = (N, D, \mathbf{p}, \mathbf{p})$ be an instance of the typical token problem and $\mathcal{S} = \langle S_1, \ldots, S_D \rangle$ be a search strategy. Then, cost$(\mathcal{S}, \mathcal{I}) = 1/2 + \sum_{d=1}^{D} (P_d)^2/2$*

*Proof.* For all $1 \leq d \leq D$, denote by $P_d = W_d$ the probability as well as the cost of the set $A_d$. By Proposition 4.2,

$$\text{cost}(\mathcal{S}, \mathcal{I}) = \sum_{d=1}^{D} \left( P_d \sum_{i=1}^{d} W_i \right) = \sum_{1 \leq i \leq j \leq D} P_i P_j = \frac{1}{2} \left( \left( \sum_{d=1}^{D} P_d \right)^2 + \sum_{d=1}^{D} (P_d)^2 \right)$$

The proposition follows since $\sum_{d=1}^{D} P_d = 1$. □

The above proposition implies that for a typical token and a given search strategy $\mathcal{S}$, the cost is the same regardless of the order of the $D$ sets in $\mathcal{S}$. That is, the task of finding an efficient $D$-round search strategy becomes the task of partitioning the $N$ boxes into $D$ sets while minimizing a particular cost function. One can view the sets as machines and the probabilities as the processing times of tasks. Then the problem is almost identical to the known *load balancing* problem from Chandra and Wong [1975], Cody and Coffman [1976] whose goal is to minimize the $L_2$ norm of the tasks' completion time on all the machines. Formally,

**Definition 4.9.** Let $\{T_1, \ldots, T_N\}$ be $N$ tasks and $\{M_1, \ldots, M_D\}$ be $D$ machines. The processing time for $T_i$ on any machine is $p_i$. The *load balancing problem* is to find an allocation of tasks to machines that minimizes the $L_2$ norm of the completion time on all machines $\sum_{d=1}^{D}(\sum_{T_i \in M_d} p_i)^2$.

If $X$ is the optimization goal of the load balancing problem and $Y$ is the optimization goal of the search problem, then $X = (1 + Y)/2$ by Proposition 4.8 and Definition 4.9. As a result, we now show that any approximation algorithm to the load balancing problem is also an approximation algorithm to the searching for typical tokens problem with an even smaller approximation factor.

**Lemma 4.10.** *Let ALG be a $(1 + \varepsilon)$-approximation algorithm to the load balancing problem where $(1+\varepsilon)$ is a tight bound. Then ALG is a $(1+\varepsilon')$-approximation algorithm to the searching for typical tokens problem, where $\varepsilon' < \frac{\varepsilon}{2}$ for all instances and $\varepsilon' \geq \frac{\varepsilon}{D+1}$ for some instance.*

*Proof.* Let OPT be an optimal solution to the load balancing problem. Let $\mathcal{O}$ and $\mathcal{S}$ be the $L_2$ norm of solutions OPT and ALG, respectively. Let OPT$'$ be an optimal solution to the search typical tokens problem. Let ALG$' =$ ALG be a $(1 + \varepsilon')$-approximation solution to the search problem. Let $\mathcal{O}' =$ cost(OPT$'$) and $\mathcal{S}' =$ cost(ALG$'$). By definition, we have $\mathcal{S} \leq (1 + \varepsilon)\mathcal{O}$ for all instances in the load

balancing problem. Therefore,

$$1+\varepsilon' = \frac{\mathcal{S}'}{\mathcal{O}'} = \frac{0.5+0.5\mathcal{S}}{0.5+0.5\mathcal{O}} \leq \frac{1+(1+\varepsilon)\mathcal{O}}{1+\mathcal{O}} = 1+\frac{\mathcal{O}}{1+\mathcal{O}}\varepsilon \text{ for all instances.}$$

Since $(1+\varepsilon)$ is a tight bound to the load balancing problem, we have $\mathcal{S} = (1+\varepsilon)\mathcal{O}$ for some instance. Therefore,

$$1+\varepsilon' = \frac{\mathcal{S}'}{\mathcal{O}'} = \frac{0.5+0.5\mathcal{S}}{0.5+0.5\mathcal{O}} = \frac{1+(1+\varepsilon)\mathcal{O}}{1+\mathcal{O}} = 1+\frac{\mathcal{O}}{1+\mathcal{O}}\varepsilon \text{ for some instance.}$$

Since all $P_i \geq 0$ and $\sum_{i=1}^{D} P_i = 1$, it follows that the cost $\sum_{i=1}^{D} P_i^2$ (Definition 4.9) of any solution to the load balancing problem is greater than $1/D$ and smaller than 1. In particular $1/D \leq \mathcal{O} \leq 1$ and therefore $\mathcal{O}/(1+\mathcal{O}) \leq 1/2$ for all instances and $\mathcal{O}/(1+\mathcal{O}) \geq 1/(D+1)$ for some instance. □

The paper Alon et al. [1998] has pointed out that the load balancing problem is strongly NP-Hard. The next theorem follows since Lemma 4.10 gives a lower bound to the approximation ratio to the searching for typical tokens problem without $N$ as a parameter.

**Theorem 4.11.** *For any $N > D \geq 2$, the search problem is* strongly NP-Hard *even for a typical token.*

Constant approximation ratio algorithms to the load balancing problem have been introduced in Chandra and Wong [1975], Cody and Coffman [1976] and a PTAS to this problem has been presented in Alon et al. [1998]. The following theorem is another corollary of Lemma 4.10.

**Theorem 4.12.** *For any $N > D \geq 2$, there exists a constant approximation ratio algorithm and a PTAS to the searching for typical tokens problem.*

Algorithm $\mathcal{S}$ in Chandra and Wong [1975] is a natural greedy algorithm. Essentially, it allocates boxes one by one in a non-increasing order of $p_i$ to the set $A_d$

currently with the smallest $P_d$. In some cases, like when $D = 2$, one can compute exactly the approximation ratio of $\mathcal{S}$ for both the load balancing and the searching for typical tokens problems. We omit the technical details. The specific approxima-

| Case | Load Balancing | Searching Typical Tokens |
|---|---|---|
| $N \leq 4$, any $D$ | 1 | 1 |
| $D = 2$, $N \geq 5$ | tight $\approx 1.0285$ | tight $\approx 1.0143$ |
| $D = 3$, $N \geq 5$ | $[83/81, 25/24]$ | $[326/324, 49/48]$ |
| even $D \geq 4$, $N \geq 5$ | $[37/36, 25/24]$ | $[1 + 1/36(D+1), 49/48]$ |
| odd $D \geq 5$, $N > 5$ | $[1 + (D-1)/36D, 25/24]$ | $[1 + (D-1)/36D(D+1), 49/48]$ |

Table 4.1: Approximation ratio of $\mathcal{S}$-algorithm

tion ratios for different values of $N$ and $D$ of Algorithm $\mathcal{S}$ for the load balancing problem and for the searching for typical tokens problem (by lemma 4.10) are shown in Table 4.1. In the table, the meaning of $[x, y]$ is that there exists a $y$ approximation factor and there cannot be an approximation factor smaller than $x$.

## 4.5 Algorithm FRO

In this section, we analyze the performance of Algorithm FRO for two rounds on arbitrary tokens whose cost vector has no correlation to the probability vector. Note that the problem for an arbitrary token is strongly NP-hard when $2 \leq D < N$ because it is strongly NP-hard already for a typical token. Therefore, our goal is to prove a guaranteed approximation factor. We show that the approximation ratio of FRO is $8/7 \approx 1.143$ for $D = 2$ rounds. For typical tokens, we slightly improve the ratio to $\frac{7-2\sqrt{7}}{28-10\sqrt{7}} \approx 1.108$. For both cases, we provide instances that show that these bounds are tight.

**Theorem 4.13.** *Algorithm FRO has an approximation ratio* $8/7$ *when* $D = 2$ *and* $N > 2$*, and the ratio* $8/7$ *is attainable.*

*Proof.* First, we show that an upper bound on the approximation ratio of FRO for $N = 3, 4$ is also an upper found for all $N > 2$ (Lemma 4.14). Next, we prove that the

upper bound on the approximation ratio of FRO for $N = 3, 4$ is $8/7$ (Lemmas 4.16, 4.17). Finally, for every $N$, we construct an instance with approximation ratio $8/7$ (Lemma 4.23). □

**Lemma 4.14.** *For each instance with $N \geq 4$ for which the approximation ratio of FRO is $\rho$, there exists an instance, with either $N = 3, 4$, for which the approximation ratio of FRO is at least $\rho$.*

*Proof.* Let the instance with $N \geq 4$ consist of boxes $\{B_1, \ldots, B_N\}$. Let the OPT partition be $\langle X, Y \rangle$ and let the FRO partition be $\langle X', Y' \rangle$. Define the following four subsets of the $N$ boxes: $A = X \cap X'$, $B = X' \setminus X$, $C = Y' \setminus Y$, $D = Y \cap Y'$. Hence, by definition $\text{OPT} = \langle A \cup C, B \cup D \rangle$ and $\text{FRO} = \langle A \cup B, C \cup D \rangle$.

If all four sets $A, B, C, D$ are not empty, we construct an instance with $N = 4$ boxes as follows. The boxes are $\{B_A, B_B, B_C, B_D\}$, such that $p_I = \sum_{B_i \in I} p_i$ and $w_I = \sum_{B_i \in I} w_i$, where $i = 1 \ldots N$ and $I \in \{A, B, C, D\}$. Let $\text{OPT}_4$ be the optimal solution for $N = 4$. It follows that $\text{cost}(\text{OPT}_4) = \text{cost}(\text{OPT}_N)$. This is because; (i) $\text{cost}(\text{OPT}_4)$ cannot be less than $\text{cost}(\text{OPT}_N)$, otherwise $\text{OPT}_N$ would not be optimal (taking the corresponding $\text{OPT}_4$ partition); (ii) $\text{cost}(\text{OPT}_4)$ can reach $\text{cost}(\text{OPT}_N)$ by taking $\text{OPT}_4 = \langle \{B_A B_C\}, \{B_B B_D\} \rangle$ (by definition of partition $\{A, B, C, D\}$). It also follows that $\text{FRO}_4 = \langle \{B_A B_B\}, \{B_C B_D\} \rangle$ because otherwise, $\text{FRO}_N$ will not be $\langle A \cup B, C \cup D \rangle$ by definition. Thus, $\text{cost}(\text{FRO}_4) \geq \text{cost}(\text{FRO}_N)$. Therefore,

$$\text{cost}(\text{FRO}_N)/\text{cost}(\text{OPT}_N) \leq \text{cost}(\text{FRO}_4)/\text{cost}(\text{OPT}_4) \ .$$

If either of the four sets $A, B, C, D$ is empty, we can similarly construct an instance of $N = 3$ with at least the same approximation ratio. Finally, if there are at least two empty sets among $A,\ B,\ C,\ D$, then OPT *is* FRO, and thus the approximation ratio is 1. □

**Corollary 4.15.** *An upper bound $\rho$ on the approximation ratio of FRO for all instances with $N = 3, 4$ is also an upper bound on the approximation ratio for all instances with $N > 4$.*

It remains to prove the following two lemmas to complete the proof of Theorem 4.13.

**Lemma 4.16.** *For all instances of $N = 3$, the approximation ratio $\rho = cost(FRO)/cost(\text{OPT}) \leq 8/7$.*

For $N = 3$, $D = 2$, assume boxes $B_1$, $B_2$, $B_3$, are sorted in decreasing order of $p_i/w_i$. The only possible FRO ordered partitions are $\langle\{B_1, B_2\}, \{B_3\}\rangle$ and $\langle\{B_1\}, \{B_2, B_3\}\rangle$. We use the shorthand notation 12|3 and 1|23, respectively, for the above ordered partitions. The only possible OPT partitions that are not of the FRO form are 2|13 and 13|2, because 3|12 has cost at least as much as 12|3, and 23|1 has cost at least as much as 1|23.

The costs for the above mentioned partitions are:

$$\text{cost}(12|3) = 1 - p_1w_3 - p_2w_3 \qquad \text{cost}(1|23) = 1 - p_1w_2 - p_1w_3$$
$$\text{cost}(2|13) = 1 - p_2w_1 - p_2w_3 \qquad \text{cost}(13|2) = 1 - p_1w_2 - p_3w_2$$

Therefore we have to compute the worst ratio for four possible cases of FRO and OPT ordered partitions:

$$\text{FRO : } 12|3 \text{ and OPT : } 2|13 \qquad \text{FRO : } 12|3 \text{ and OPT : } 13|2$$
$$\text{FRO : } 1|23 \text{ and OPT : } 2|13 \qquad \text{FRO : } 1|23 \text{ and OPT : } 13|2$$

We prove the first of the four cases. The rest can be proved by similar methods.

*Proof.* Assume FRO: 12|3 and OPT: 2|13. Since FRO outputs 12|3, the cost of 12|3

is not worse than the cost of the other FRO ordered partition 1|23, which implies

$$p_1 w_2 \leq p_2 w_3. \tag{4.3}$$

The ratio cost(FRO)/cost(OPT) is:

$$\rho = \frac{1 - p_1 w_3 - p_2 w_3}{1 - p_2 w_1 - p_2 w_3}.$$

Let $d = (p_1 w_2 - p_2 w_1)/(w_1 + w_2) \geq 0$. Consider a new input, by substituting $p_1$ with $p_1 - d$ and $p_2$ with $p_2 + d$. In the new input, equation (4.3) still holds and therefore FRO outputs the same partition. Moreover, in the new input, OPT outputs the same partition too. The ratio $\rho'$ of the new input is at least the ratio of the original input, because:

$$\rho' \geq \rho \iff (p_1 w_2 - p_2 w_1)(w_1 + w_3)(1 - p_1 w_3 - p_2 w_3) \geq 0$$

and each factor of the product $(p_1 w_2 - p_2 w_1)(w_1 + w_3)(1 - p_1 w_3 - p_2 w_3)$ is non-negative.

The above transformation of $p_1$ to $p_1 - d$ and $p_2$ to $p_2 + d$ has the effect of creating a new input in which the $p/w$ ratio of boxes 1 and 2 is the same. It is possible to solve exactly the optimization problem of maximizing the ratio of FRO over OPT under this additional constraint. The solution of the optimization problem gives a maximum ratio of 8/7 for input: $\mathbf{p} = \langle 1/4, 3/4, 0 \rangle$, $\mathbf{w} = \langle 1/5, 3/5, 1/5 \rangle$. In the following we sketch a proof of $\rho \leq 8/7$. Since boxes 1 and 2 have the same $p/w$ ratio, we write $p_1 = r w_1$ and $p_2 = r w_2$. We want to prove that the approximation ratio is:

$$\frac{1 - r w_1 w_3 - r w_2 w_3}{1 - r w_1 w_2 - r w_2 w_3} \leq \frac{8}{7}$$

or equivalently $r(8 w_1 w_2 - 7 w_1 w_3 + w_2 w_3) \leq 1$. Therefore, it is enough to study the

maximization problem:

$$\text{maximize } r(8w_1w_2 - 7w_1w_3 + w_2w_3) \text{ under:}$$
$$w_1 + w_2 + w_3 = 1 \wedge w_3 \geq w_1 \wedge w_2 \geq w_3 \wedge rw_1 + rw_2 \leq 1.$$

The constraint $w_3 \geq w_1$ is implied by $p_1w_2 \leq p_2w_3$, the constraint $w_2 \geq w_3$ is implied by $p_2w_1 \leq p_1w_3$ (which is implied by $\text{cost}(2|13) \leq \text{cost}(12|3)$), the constraint $rw_1 + rw_2 \leq 1$ is implied by $p_1 + p_2 \leq 1$. Writing $w_3 = 1 - w_1 + w_2$ and by the fact that $r \leq 1/(w_1+w_2)$ we solve the following maximization problem (which might have a bigger solution):

$$\text{maximize } \frac{8w_1w_2 - 7w_1(1 - w_1 - w_2) + w_2(1 - w_1 - w_2)}{w_1 + w_2}$$
$$\text{under: } w_1 \geq 0 \wedge 1 - w_2 \geq 2w_1 \wedge 2w_2 \geq 1 - w_1.$$

The value of the maximization function depends on the values of $w_1$, $w_2$. The range of $w_1$, $w_2$ is a triangle in the plane $\mathbb{R}^2$. We will compute the maximum in each of the segments of the form $w_2 = w_2^0 - \frac{1}{2}w_1$, with $w_1 \in [0, \frac{2}{3}(w_2^0 - \frac{1}{2})]$, and $w_2^0 \in [\frac{1}{2}, 1]$ (each value of $w_2^0$ defines a segment in the triangle, parallel to the $1 - 2w_2 = 2w_1$ side of the triangle). The value of the maximization function is:

$$\frac{w_2^0(1 - w_2^0) + (18w_2^0 - 9)w_1 - 25w_1^2}{w_2^0 - w_1}.$$

For $w_2^0 \leq 25/47$, the maximum is achieved at $w_1 = \frac{2}{3}(w_2^0 - \frac{1}{2})$ and it is $(2 - w_2^0)/3 < 1$. For $w_2^0 \geq 25/47$, the maximum is achieved at $w_1 = \frac{1}{5}(5w_2^0 - 2\sqrt{2}\sqrt{w_2^0 + (w_2^0)^2})$ and it is:

$$-\frac{20\sqrt{2}(w_2^0)^2}{\sqrt{w_2^0(w_2^0 + 1)}} + \left(32 - \frac{20\sqrt{2}}{\sqrt{w_2^0(w_2^0 + 1)}}\right) w_2^0 + 9.$$

The above as a function of $w_2^0 \in [1/2, 1]$ is convex and therefore its maximum is

attained at one of the extremes of the range $[1/2, 1]$, namely $w_2^0 = 1$ for a maximum value of 1, as expected. Therefore $w_1 = 1/5$, which implies $w_2 = 3/5$. We choose the maximum possible $r$ to satisfy the constraint $r \leq 1/(w_1 + w_2)$, which is $r = 5/4$. □

**Lemma 4.17.** *For all instances of $N = 4$, the approximation ratio $\rho = cost(FRO)/cost(\text{OPT}) \leq 8/7$.*

For $N = 4$ and $D = 2$, if there is one round with three boxes in one of the FRO and OPT solutions, then there are two boxes in this round that are in the same round in the other solution (for example, boxes 1 and 3 when FRO: 123|4 and OPT: 14|23). By combining the above two boxes to one (summing the probabilities and weights), we get an instance of $N = 3$, $D = 2$, with the same approximation ratio, i.e., we have reduced the problem of finding the worse approximation ratio to the case $N = 3$ that we studied before.

Therefore, we only have to consider the following cases:

| | |
|---|---|
| FRO : 12\|34 and OPT : 13\|24 | FRO : 12\|34 and OPT : 24\|13 |
| FRO : 12\|34 and OPT : 23\|14 | FRO : 12\|34 and OPT : 14\|23 |

We will consider inputs in which the probabilities or weights do not sum up to 1 and therefore need the following observation, whose proof is based on the fact that a ratio of two costs is oblivious to a scaling of $p_i$s or $w_i$s and we omit the details. We denote the $p/w$-ratio of box $i$ with $r_i$.

**Observation 4.18.** *Scaling $p_i$s and $w_i$s by any positive factor will not affect the approximation ratio.*

Because of space considerations, we only analyze the first case, where FRO is 12|34 and OPT is 13|24. The other cases are similar and we will give a sketch later.

**Lemma 4.19.** *For instances of $N = 4$ in which* OPT $= (13|24)$ *and FRO* $= (12|34)$*, the approximation ratio is at most* $8/7$*.*

In order to prove the above, we transform gradually any instance of the above form to instances that have worse and worse approximation ratio. For the final instance, it is possible to solve an optimization problem and show that the worse approximation ratio is $8/7$, like the $N = 3$ case.

**Observation 4.20.** *$\rho$ is maximized if $w_1 = 0$.*

*Proof.* Consider the input $\mathbf{p} = \langle p_1, p_2, p_3, p_4\rangle$ and $\mathbf{w} = \langle w_1, w_2, w_3, w_4\rangle$ and assume it is ordered according to $p/w$ ratio and that FRO:12|34 and OPT:13|24. Set $\rho =$ cost(FRO)/cost(OPT). We have the following costs:

$$\begin{aligned}
\text{cost(FRO)} = \text{cost}(12|34) &= (p_1 + p_2)(w_1 + w_2) + (p_3 + p_4)(w_1 + w_2 + w_3 + w_4),\\
\text{cost}(1|234) &= p_1 w_1 + (p_2 + p_3 + p_4)(w_1 + w_2 + w_3 + w_4),\\
\text{cost}(123|4) &= (p_1 + p_2 + p_3)(w_1 + w_2 + w_3) + p_4(w_1 + w_2 + w_3 + w_4),\\
\text{cost(OPT)} = \text{cost}(13|24) &= (p_1 + p_3)(w_1 + w_3) + (p_2 + p_4)(w_1 + w_2 + w_3 + w_4).
\end{aligned}$$

Moreover,

$$\text{cost}(12|34) \le \text{cost}(1|234), \quad \text{cost}(12|34) \le \text{cost}(123|4).$$

Now change the input by setting $w_1 = 0$. Then, for that input, the order of boxes does not change, and it is still the case that FRO: 12|34. Moreover the new approximation ratio is:

$$\begin{aligned}
\rho' = \text{cost}'(\text{FRO})/\text{cost}'(\text{OPT}) &\ge (\text{cost(FRO)} - w_1)/(\text{cost(OPT)} - w_1)\\
&= (\rho - w_1/\text{cost(OPT)})/(1 - w_1/\text{cost(OPT)}) \ge \rho
\end{aligned}$$

This implies setting $w_1 = 0$ will not decrease the approximation ratio $\rho$. □

Similarly, we can prove the following.

**Observation 4.21.** *$\rho$ is maximized if $p_4 = 0$.*

Given $w_1 = 0$ and $p_4 = 0$, we now prove:

**Observation 4.22.** *For $\frac{p_2}{w_2} = \frac{p_3}{w_3}$, $\rho$ is maximized.*

*Proof.* Let $r_2 = p_2/w_2$, and $r_3 = p_3/w_3$. Initially $r_2 \geq r_3$. Since FRO is 12|34, we have cost(12|34) $\leq$ cost(1|234) and cost(12|34) $\leq$ cost(123|4), which implies $w_2 \leq p_2(w_3 + w_4)/p_1$ and $p_3 \leq (p_1 + p_2)w_3/w_4$. Observe that for the extreme values $w_2 = p_2(w_3 + w_4)/p_1$ and $p_3 = (p_1 + p_2)w_3/w_4$, we have $r_2 \leq r_3$ and therefore there are values of $p_3$ and $w_2$ both greater or equal to the initial respective values, for which the ratios of the two boxes become the same. Since for

$$\rho = \frac{\text{cost(FRO)}}{\text{cost(OPT)}} = \frac{(p_1 + p_2)w_2 + p_3(w_2 + w_3 + w_4)}{(p_1 + p_3)w_3 + p_2(w_2 + w_3 + w_4)}$$

we have $\frac{\partial \rho}{\partial p_3} \geq 0$ and $\frac{\partial \rho}{\partial w_2} \geq 0$, the approximation ratio when $r_2 = r_3$ is greater or equal than the initial approximation ratio. □

Now, with a similar optimization analysis like the one done for $N = 3$, it can be proven that the worst possible ratio is 8/7 (we omit the tedious details). Otherwise, one can use a symbolic optimization program like Mathematica and get the same result.

When OPT outputs 23|14, we transform the input as follows: We set $p_4$ to 0, decrease $w_2$ until the first two boxes have the same ratio ($r_1 = r_2$), and increase $p_3$ until the first of the following events happens: a) $r_2 = r_3$ or b) cost(12|34) = cost(123|4). It can be proven by taking derivatives of the approximation ratio $\rho$ that the above transformations never make $\rho$ smaller. Now what remains is to study the two cases with a) $p_4 = 0$, $r_2 = r_3$ and b) $p_4 = 0$, cost(12|34) = cost(123|4). With an

optimization analysis like above, it is possible to prove that 8/7 is an upper bound for $\rho$.

When OPT outputs 14|23, we transform the input as follows: We set $w_1$ to 0, decrease $p_3$ until $r_3 = r_4$, and increase $w_2$ until the first of the following events happens: a) $r_2 = r_3$ or b) cost(12|34) = cost(1|234). Again it can be shown by taking derivatives of $\rho$ that the above transformations never make $\rho$ smaller and what remains is to study the two cases with a) $w_1 = 0$, $r_2 = r_3$ and b) $w_1 = 0$, cost(12|34) = cost(1|234). Again for both of the above cases, it can be proven that 8/7 is an upper bound for $\rho$.

Finally, when OPT outputs 14|23, we do the following transformations: We decrease $w_2$ until $r_1 = r_2$ and decrease $p_3$ until $r_3 = r_4$. Again it can be shown by taking derivatives of $\rho$ that the above transformations never make $\rho$ smaller, and (by an optimization analysis) that the maximum possible $\rho$ for an input of the last form is 8/7.

In the next lemma we show that the 8/7 upper bound on the approximation ratio is tight.

**Lemma 4.23.** *The approximation ratio* 8/7 *is attainable for any* $N > 2$.

*Proof.* Consider the instance with $p_1 = 1/4$, $p_2 = 3/4$, $p_3 = \cdots = p_N = 0$ and $w_1 = 1/5$, $w_2 = 3/5$, $w_3 = \cdots = w_N = 1/(5(N-2))$. FRO outputs the partition $(1|23\ldots N)$ the cost of which is 4/5 while the optimal partition is $(2|13\ldots N)$ the cost of which is 7/10. The ratio is 8/7. □

**Typical tokens:**

For a typical token, FRO does not distinguish among the boxes since all the ratios are 1. Therefore, we assume that an adversary picks the worst permutation on the boxes that is respected by FRO. Still, we can prove a smaller guaranteed approximation factor for this case.

**Theorem 4.24.** *Algorithm FRO has an approximation ratio* $\frac{7-2\sqrt{7}}{28-10\sqrt{7}} \approx 1.108$ *when* $D = 2$ *and* $N > D$*, and this ratio is attainable.*

*Proof.* The proof is very similar to the proof of Theorem 4.13. First, the reduction lemma, Lemma 4.14, is correct for any type of token. Then the proofs of the equivalent Lemmas (but with a different ratio) to Lemma 4.16 and Lemma 4.17 are easier since there are fewer variables. Finally, the next lemma demonstrates the instance for which the ratio is attainable. □

**Lemma 4.25.** *The approximation ratio* $\frac{7-2\sqrt{7}}{28-10\sqrt{7}}$ *is attainable for any* $N > 2$*.*

*Proof.* For simplicity, assume that $0/0 = 1$ since one can replace each zero value with a small $\varepsilon$. Consider the instance with $p_1 = w_1 = x$, $p_2 = w_2 = 1 - 2x$, $p_3 = w_3 = x$, and $p_4 = w_4 = \cdots = p_N = w_N = 0$. FRO, that respects this order, outputs the partition $(1|23 \ldots N)$ the cost of which is $1 - x + x^2$ while the optimal partition is $(2|13 \ldots N)$ the cost of which is $1 - 2x + 4x^2$. The maximum of the ratio is achieved for $x = (3 - \sqrt{7})/2$ and is $\frac{7-2\sqrt{7}}{28-10\sqrt{7}}$. □

## 4.6 A general PTAS for $D = 2$

We present a PTAS for the problem of finding a strategy of optimal expected cost, when $D = 2$. If $D$ is constant, we still have a PTAS, with similar methods as the ones described in this section; we omit the details because of space considerations.

Fix an optimal solution OPT. Denote by $F_1 = 1$ and $F_2$ the probability that OPT will get to first and second round respectively (i.e., $F_2$ is the probability that the token is not found by OPT in the first round). Similarly, let $W_1$ and $W_2$ be the total weight of the boxes that OPT probes in the first and second round, respectively. Therefore, the cost of OPT is $C = F_1W_1 + F_2W_2$.

Consider a positive constant $\varepsilon < 1$. The first step of the PTAS is to guess an approximation within a factor of $(1+\varepsilon)$ of each one of the values of $F_2$, $W_1$, and $W_2$ in

the optimal solution. We denote these approximations with $F_2'$, $W_1'$, $W_2'$, respectively. More precisely, if the minimum of a probability of a box is $p_{\min} = \min_{n=1}^{N} p_n$, then $F_2$ can be approximated by a value in $\{(1+\varepsilon)^{-K}, \ldots, (1+\varepsilon)^{-2}, (1+\varepsilon)^{-1}, 1\}$, where $K = \lceil \log_{1+\varepsilon} p_{\min}^{-1} \rceil$. If the minimum of a weight of a box is $w_{\min} = \min_{n=1}^{N} w_n$, and the sum of weights of all boxes is 1, then each of $W_1$, $W_2$ can be approximated by a value in $\{(1+\varepsilon)^{-L}, \ldots, (1+\varepsilon)^{-2}, (1+\varepsilon)^{-1}, 1\}$, where $L = \lceil \log_{1+\varepsilon} w_{\min}^{-1} \rceil$. Without loss of generality we assume that $p_{\min}, w_{\min} \neq 0$.

In total, we have $KL^2$ possible triples $(F_2', W_1', W_2')$, which is polynomial in $1/\varepsilon$, $\log(1/p_{\min})$ and $\log(1/w_{\min})$, that is a polynomial number of triples. For every triple, we consider the cost $C' = F_1 W_1' + F_2' W_2'$. For one of the triples, each of $F_2'$, $W_1'$, $W_2'$, is $\varepsilon$-close to the corresponding value in the optimal solution, and thus,

$$C' \leq F_1(1+\varepsilon)W_1 + (1+\varepsilon)F_2(1+\varepsilon)W_2 \leq (1+\varepsilon)^2(F_1W_1 + F_2W_2) \leq (1+3\varepsilon)C. \quad (4.4)$$

We apply the following algorithm for each possible triple: With the help of a linear program, we compute a feasible solution (if it exists) with corresponding values close enough to the values of the triple and record the solution's cost. Then, we return as an output the feasible solution with minimum cost. In the analysis of the scheme it suffices to consider the iteration of the algorithm in which we tried the values of $F_2'$, $W_1'$, $W_2'$ which are $\varepsilon$-close to the corresponding values in OPT. In particular, for each triple of values $(F_2', W_1', W_2')$, we consider the following linear program, over variables $x_{1,1}, \ldots, x_{1,N}, x_{2,1}, \ldots, x_{2,N}$ (an integral feasible solution to this linear program has the following meaning, $x_{i,n} = 1$ if box $B_n$ is probed in round $i$):

minimize $f = W_1' + \sum_{n=1}^{N} W_2' p_n x_{2,n}$ such that:

(a) $\sum_{n=1}^{N} p_n x_{2,n} \leq F_2'$

(b) $\sum_{n=1}^{N} w_n x_{1,n} \leq W_1'$ and $\sum_{n=1}^{N} w_n x_{2,n} \leq W_2'$

(c) $x_{1,n} + x_{2,n} = 1$ for $n = 1, \ldots, n$

(d) $x_{d,n} \geq 0$ for $d = 1, 2$ and $n = 1, \ldots, n$

We say that a box $B_n$ is a *large box* if $p_n W_2' > \varepsilon C'$; otherwise we say it is a *small box*. We denote by $Y$ the set of small boxes, and by $Z$ the set of large boxes. Intuitively, the large boxes have a major influence on the value of the goal function $f$, so we will treat them separately. Assume $B$ large boxes are assigned to round 2 in the optimal solution. Then,

$$C \geq F_2 W_2 \geq F_2 W_2'(1+\varepsilon)^{-1} > B\varepsilon C'(1+\varepsilon)^{-1} \geq B\varepsilon(1+\varepsilon)^{-1} C.$$

(The last inequality holds, because $C' \geq C$.) Therefore, it must be the case that $B\varepsilon(1+\varepsilon)^{-1} < 1$, or $B < 1 + \varepsilon^{-1}$. i.e., we have a constant upper bound on the cardinality $B$ of the set $X$ of large boxes that are assigned to round 2 in the optimal solution. Thus, the number of such sets is $O(N^B)$, i.e., polynomial in $N$.

Our second guessing step would be to guess the set $X$. That is, for every subset $X \subseteq Z$, such that $|X| < 1 + \varepsilon^{-1}$, we set the values of $x_{i,n}$ for $i = 1, 2$ and $n \in Z$ as follows. For all $n \in X$, we set $x_{1,n} = 0$ and $x_{2,n} = 1$, and for all $n \in Z \setminus X$ we set $x_{1,n} = 1$ and $x_{2,n} = 0$. The value of the other decision variables are determined by the solution of the following linear program, on the set of variables $\{x_{1,n} \mid n \in Y\} \cup \{x_{2,n} \mid n \in Y\}$:

minimize $f = W_1' + \sum_{n=1}^{N} W_2' p_n x_{2,n}$ such that:

(a) $\sum_{n=1}^{N} p_n x_{2,n} \leq F_2'$

(b) $\sum_{n=1}^{N} w_n x_{1,n} \leq W_1'$ and $\sum_{n=1}^{N} w_n x_{2,n} \leq W_2'$

(c) $x_{1,n} + x_{2,n} = 1$ for $n \in Y$

(d) $x_{d,n} \geq 0$ for $d = 1, 2$ and $n \in Y$

The last linear program has $2|Y|$ variables and four types of constraints: (a) One constraint with $p_n$ coefficients (constraint of type (a)), (b) Two constraints with $w_n$

coefficients (constraints of type (b)), (c) $|Y|$ constraints with coefficients equal to 1 (constraints of type (c)), (d) $2|Y|$ non-negativity constraints.

We compute an optimal basic solution to the linear program Matousek and Gärtner [2006]. Such a solution exists for the correct value of $F_1', W_1', W_2'$ and $X$ because for such values OPT corresponds to a feasible solution for the above linear program whose goal function value is

$$W_1' + W_2' F_2 \leq W_1' + W_2' F_2' = C'.$$

Therefore, the cost of the basic optimal solution which we find, is at most $C'$.

Moreover, a basic solution of the above linear program has the property that $2|Y|$ linearly independent constraints of the linear program are set to equality (i.e., as many constraints as the number of variables). This means that at least $|Y| - 3$ of the (d)-constraints are set to equality, i.e., at least $|Y| - 3$ variables are set to 0. However, if $x_{d,n} = 0$ for some $d = 1, 2$ and some $n$, then (because of the corresponding (c)-constraint) $x_{(3-d),n} = 1$. This further implies that the variables that correspond to at most 3 boxes (i.e., 6 variables) are set to fractional (non-integral) values in a basic optimal solution. A similar method of bounding the number of fractional values of a basic solution was first employed in Lenstra et al. [1990], in the context of scheduling unrelated parallel machines.

Let $x^*$ be a basic optimal solution to the linear program. We will associate with it a *cost*:

$$C^* = W_1^* + W_2^* P_2^*,$$

where $P_2^* = F_2^* = \sum_{n=1}^{N} p_n x_{2,n}^*$, $W_1^* = \sum_{n=1}^{N} w_n x_{1,n}^*$, and $W_2^* = \sum_{n=1}^{N} w_n x_{2,n}^*$. Since $W_1^* \leq W_1'$, $W_2^* \leq W_2'$, and $P_2^* = F_2^* = F_2'$, we have

$$C^* \leq C' \tag{4.5}$$

Assume without loss of generality that the non-integral values are at a subset

of the boxes $B_1$, $B_2$, and $B_3$ (that is, we assume that boxes $B_4, B_5, \ldots, B_N$ have integral solution). We round the non-integral variables of the linear program so that $x^{\mathrm{r}}_{11} = x^{\mathrm{r}}_{12} = x^{\mathrm{r}}_{13} = 0$ and $x^{\mathrm{r}}_{21} = x^{\mathrm{r}}_{22} = x^{\mathrm{r}}_{23} = 1$, i.e., we assign the non-integral boxes to the second round, and $x^{\mathrm{r}}_{d,n} = x^*_{d,n}$, for every other variable. Consider the cost $C^{\mathrm{r}}$ of the rounded solution; we intend to compare $C^{\mathrm{r}}$ and $C^*$. We define:

$$\Delta p = x^*_{11}p_1 + x^*_{12}p_2 + x^*_{13}p_3$$

and

$$\Delta w = x^*_{11}w_1 + x^*_{12}w_2 + x^*_{13}w_3.$$

Then,

$$\begin{aligned}\Delta C &= C^{\mathrm{r}} - C^* \\ &= (W^*_1 - \Delta w) + (W^*_2 + \Delta w)(P^*_2 + \Delta p) - (W^*_1 + W^*_2 P^*_2) \\ &= \Delta w(P^*_2 + \Delta p - 1) + \Delta p \cdot W^*_2. \end{aligned} \tag{4.6}$$

Define $P^-_2 = P^*_2 - (x^*_{21}p_1 + x^*_{22}p_2 + x^*_{23}p_3)$, i.e., $P^-_2$ is the sum of the probabilities of the integral boxes of round 2, i.e., it does not contain any probability for boxes $B_1$, $B_2$, and $B_3$. But then, $P^*_2 + \Delta p = P^-_2 + p_1 + p_2 + p_3 \leq 1$, which implies

$$\Delta w(P^*_2 + \Delta p - 1) \leq 0. \tag{4.7}$$

On the other hand,

$$\begin{aligned}\Delta p \cdot W^*_2 \leq \Delta p \cdot W'_2 &= x^*_{11}p_1W'_2 + x^*_{12}p_2W'_2 + x^*_{13}p_3W'_2 \\ &\leq x^*_{11}\varepsilon C' + x^*_{12}\varepsilon C' + x^*_{13}\varepsilon C' \leq 3\varepsilon C'. \end{aligned} \tag{4.8}$$

Then, using inequalities (4.7) and (4.8), equation (4.6) implies

$$C^{\mathrm{r}} \leq C^* + 3\varepsilon C' \leq C' + 3\varepsilon C' = (1+3\varepsilon)C' \leq (1+3\varepsilon)^2 C \leq (1+15\varepsilon)C,$$

where the second inequality is true because of (4.5) and the fourth inequality because of (4.4). Therefore the rounded solution is an $(1+\varepsilon')$-approximation, if we choose $\varepsilon = \varepsilon'/15$ above.

Although the PTAS we described above is of theoretical interest, it might not be very attractive for use in a real system, because it is quite complicated to implement compared to the simplicity of the FRO algorithm, and it relies on computational tools like linear program solvers. We supply theoretical bounds on the worst-case performance for both the PTAS and the FRO algorithm, so that a designer of a real system can choose what best suits the application. As we have seen, the running time of the PTAS depends on the minimum values of probabilities and costs. If we have a guarantee that these values are not too small, we can avoid solving a linear program, for every triple $(F_2', W_1', W_2')$, and instead reduce to a version of the knapsack problem, that can be solved with simpler methods, and still have a PTAS.

## 4.7 Summary and open problems

We have explored the problem of search with cost, in which each box is associated with a positive unlocking cost $w_n$ in contrast to the basic problem in which all $w_n$s $= 1$. We first apply the dynamic programming scheme of the basic search problem to the search with cost problem with respect to a given order of boxes, and show that the dynamic program scheme can be implemented as efficient as it is in the basic search problem. We introduce two necessaries conditions for an optimal search strategy and provide polynomial time optimal solutions to three special cases: i) for $D = N$; ii) for atypical tokens; iii) for uniform probabilities. We prove the problem is strongly NP-

hard even for typical tokens. We show an equivalence reduction from searching for a typical token problem to a known load balancing problem, and provide a PTAS. For arbitrary tokens, we provide an 8/7-approximation algorithm for $D = 2$ and a general PTAS for any constant $D$. We supplement our theoretical results with experimental studies showing that our algorithms perform remarkably better than their theoretical bound in real applications. The details of our simulation are presented in Section 6.3.

There are open problems left on this topic. We conjecture that the 8/7 bound for the FRO algorithm, from Section 4.5, holds also for $D > 2$. Similarly to the $D = 2$ case, we can reduce the problem instances with any $N$ to instances with $D \leq N \leq D^2$. However, we do not know how to handle all these cases, even for $D = 3$. We only know how to resolve an instance with $N = 4$ and $D = 3$ showing a 8/7 lower bound for algorithm FRO.

In Subsection 4.6, we present a PTAS for $D = 2$, running in polynomial time with respect to the size of the input (minimum encoding of small least probability and cost), that can be generalized to a PTAS for a constant number of searching rounds $D$. It would be interesting to find a PTAS running in polynomial time with respect to the number of boxes and for an arbitrary number of rounds $D \leq N$ (i.e., not necessarily constant $D$).

The uniform cost case is investigated also in many settings in the context of paging multiple users in cellular networks. These settings of finding more than one hidden token could be addressed in the non-uniform case as well.

# Chapter 5

# Searching for multiple tokens

In this chapter we study the searching for multiple tokens problem with *uniform cost*. Same as the basic search problem, our objective is to minimize the expected number of unlocked boxes. However, in the searching for multiple tokens problem, more than one token is hidden in the boxes. After unlocking a box, we collect all tokens residing in it. We are particularly interested in two extreme cases of the problem. In the first case, we stop after finding one out of the many tokens, and this token can be any token. We name this problem the *yellow page problem* because when we search information in yellow page books, we stop searching after we find the first useful information. In the second case, we stop after finding all the tokens. We name this problem the *conference call problem* because in the context of paging mobile users in a cellular network system, this is equivalent to establishing a conference call among a group of mobile users.

The searching for multiple tokens problem is complicated in the computation of the search cost. Given an instance of the problem and a search strategy, computing the search cost is not as straight forward as searching for a single token. We present three methods of computing search cost given a problem instance and a search strategy. We provide a hierarchical classification of tokens, and provide polynomial time

optimal algorithms for certain types of tokens. The conference call problem has been proved NP-hard in Bar-Noy and Malewicz [2004]. We show some duality between the conference call problem and the yellow page problem. A straight forward implementation of optimal algorithms for both problems takes factorial running time and linear space with respect to the number of boxes. We demonstrate a dynamic programming scheme that computes an optimal search strategy in exponential time and exponential space with respect to the number of boxes. In addition to optimal algorithms, we present a group of 12 heuristics based on the classic dynamic programming scheme. These heuristics belong to 4 criteria and 3 techniques. We conduct a substantial experimental study on the heuristics with a conclusion of their performance. The experimental study is presented in Section 6.4.

## 5.1 Related work

The conference call problem has been introduced in Bar-Noy and Malewicz [2004]. The authors have proved NP-hardness of the problem, and have showed a natural greedy algorithm is an $\frac{e}{e-1}$-approximation. In Epstein and Levin [2008], the authors have introduced a PTAS to the conference call problem for any constant number of search rounds. In Bar-Noy and Naor [2006], the authors have studied the conference call problem with an additional constraint, such that in each round, a limited number of tokens in each box can be found. The NP-hardness of the problem has been shown, and a fast heuristic that minimized both delay constraint and search cost has presented.

In Epstein and Levin [2006], the authors have explored another version of the conference call problem: instead of unlocking a box and collecting all tokens in it, the system queries a box by asking if a specific token is in it, and gets a boolean answer. The authors have shown the complexity of the problem and have provided

approximation algorithms in this setting.

The yellow page problem has not been studied to the best of our knowledge in our problem setting. In Kaplan et al. [2005], the authors have explored a more general dynamic version of the problem. They have proved it is NP-hard and have provided a 4-approximation algorithm. In Gau and Haas [2004], the authors have studied the conference call problem by minimizing the amortized cost per search. In Cohen et al. [2003], the authors have studied a similar problem. The problem differs from the yellow page problem in the parameter of number of tokens $M$. They have proved its NP-hardness and have provided an efficient approximation algorithm. In Weitzman [1979], the author has discussed the yellow page problem in the context of efficiently finding alternative investment and has provided heuristical solutions in a continuous model (in contrast to our discrete model).

## 5.2 Preliminaries

**Example 5.1.** Let 2 tokens $\{T_1, T_2\}$ be hidden into 3 boxes, $\{B_1, B_2, B_3\}$ with probabilities $\{0.5, 0.3, 0.2\}$ (token $T_1$) and $\{0.4, 0.1, 0.5\}$ (token $T_2$), respectively.

For the search strategy $\langle B_1, B_2|B_3\rangle$, in the conference call problem, the probability that both tokens are in boxes $\{B_1, B_2\}$ (i.e., search stops after the first round and only 2 boxes are unlocked) is $(0.5+0.3)\cdot(0.4+0.1) = 0.4$. Otherwise, all 3 boxes are unlocked. The expected cost is $0.4\cdot 2+(1-0.4)\cdot 3 = 2.6$. For the same strategy, in the yellow page problem, both tokens are in $\{B_3\}$ with probability $0.2\cdot 0.5 = 0.1$, in which case no token has been found in the first round and we unlock all 3 boxes. Otherwise, we only unlock the first 2 boxes. The expected cost is $0.1 \cdot 3 + (1 - 0.1) \cdot 2 = 2.1$.

Another search strategy could be $\langle B_1|B_2B_3\rangle$. In the conference call problem, with the probability of both tokens reside in $\{B_1\}$, which is $0.5 \cdot 0.4 = 0.2$, we unlock 1 box; otherwise, we unlock all 3 boxes. The search cost is $0.2 \cdot 1 + (1 - 0.2) \cdot 2 = 2.6$.

In the yellow page problem, with probability $(0.3 + 0.2) \cdot (0.1 + 0.5) = 0.3$ that both tokens are not in $\{B_1\}$, we unlock 3 boxes; otherwise, we unlock $B_1$ only. The search cost is $0.3 \cdot 3 + (1 - 0.3) \cdot 1 = 1.6$, which is better than the previous search strategy.

**Definition 5.2.** Let $M$ tokens $\{T_1, \ldots, T_M\}$ be hidden in $N$ boxes $\{B_1, \ldots, B_N\}$ and $p_{m,n}$ be the probability of token $T_m$ being in box $B_n$. An instance of the search for multiple tokens problem is a quadruple $\mathcal{I} = \langle M, N, D, \mathbf{p} \rangle$, where $M$ is the number of tokens, $N$ is the number of boxes, $D$ is the number of rounds, and $\mathbf{p}$ is the $M \times N$ probability matrix $\{p_{m,n}\}$.

The definition of a search strategy $\mathcal{A} = \langle A_1, \ldots, A_D \rangle$ remains the same. Given an instance $\mathcal{I}$ and a search strategy $\mathcal{A}$, we denote $\mathrm{YP}(\mathcal{I}, \mathcal{A})$ and $\mathrm{CC}(\mathcal{I}, \mathcal{A})$ be the expected cost of the yellow page problem and conference call problem, respective. When there is no ambiguity, we omit parameter $\mathcal{I}$.

## 5.3 Cost computation

We describe three methods that compute the search cost for the yellow page problem and the conference call problem. The first two are used in our proofs while the third is used by our simulations since it is computationally the most efficient.

Given a search strategy $\mathcal{A} = \langle A_1, \ldots, A_D \rangle$, let $P_{m,d} = \sum_{B_n \in A_d} p_{m,n}$ be the probability of token $T_m$ in boxes of part $A_d$. Denote the suffix probability by $R_{m,d} = \sum_{i=d+1}^{N} P_{m,i}$ ($R$ for Remainder) and the prefix probability by $Q_{m,d} = \sum_{i=1}^{d} P_{m,i}$. ($Q$ for (cu)Qmulator). Let $S_d$ be the number of boxes in the first $d$ parts, $A_1$, …, $A_d$. By convention, $S_0 = 0$. Let $\mathrm{YP}(\mathcal{A})$ be the cost of the yellow page problem and $\mathrm{CC}(\mathcal{A})$ be the cost of the conference call problem, on search strategy $\mathcal{A}$. Consider the vector $\mathbf{d} = (d_1, \ldots, d_M) \in \{1, \ldots, D\}^M$, which encodes in which part each token is (i.e., token $T_m$ is in part $d_m$, for $1 \leq m \leq M$).

**Combinatorial computation:** For a part location vector $(d_1, \ldots, d_M)$, which occurs

with probability $\prod_{m=1}^{M} P_{m,d_m}$, the strategy pays a cost of $S_{\min\{d_1,\ldots,d_M\}}$ for the yellow page problem (i.e., it unlocks parts until it finds the first part that contains some token) and therefore:

$$\mathrm{YP}(\mathcal{A}) = \sum_{\mathbf{d}\in\{1,\ldots,D\}^M} \left( S_{\min\{d_1,\ldots,d_M\}} \cdot \prod_{m=1}^{M} P_{m,d_m} \right). \tag{5.1}$$

Similarly, the cost of the conference call problem is:

$$\mathrm{CC}(\mathcal{A}) = \sum_{\mathbf{d}\in\{1,\ldots,D\}^M} \left( S_{\max\{d_1,\ldots,d_M\}} \cdot \prod_{m=1}^{M} P_{m,d_m} \right), \tag{5.2}$$

where the only difference is that for each part location vector, the strategy pays a cost of $S_{\max\{d_1,\ldots,d_M\}}$, because it has to unlock also all boxes in the last part that contains a token.

Time Complexity: $\Theta(MN + (M+D)D^M)$.

**Recursive computation:** In the yellow page problem, we extend the definition of cost, so that $\mathrm{YP}(\langle A_d,\ldots,A_D\rangle)$ is the expected cost of unlocking box sets $\langle A_d,\ldots,A_D\rangle$ given the condition that no token is found in sets $A_1,\ldots,A_{d-1}$. If no token has been found in the first $(D-1)$ rounds, we must unlock all boxes in in $A_D$, i.e., $\mathrm{YP}(\langle A_D\rangle) = |A_D|$. The recursion step is

$$\mathrm{YP}(\langle A_d,\ldots,A_D\rangle) = |A_d| + \frac{\prod_{m=1}^{M} R_{m,d}}{\prod_{m=1}^{M} R_{m,d-1}} \mathrm{YP}(\langle A_{d+1},\ldots,A_D\rangle)\ , \tag{5.3}$$

because to search for tokens in $A_d,\ldots,A_D$ given that no tokens are in $A_1,\ldots,A_{d-1}$, we must unlock boxes in $A_d$ by paying a cost of $|A_d|$ and if no token is found there (an event with probability $\prod_{m=1}^{M} R_{m,d}/\prod_{m=1}^{M} R_{m,d-1}$) we pay an extra cost of $\mathrm{YP}(\langle A_{d+1},\ldots,A_D\rangle)$.

Let $\mathrm{CC}(\langle A_1,\ldots,A_d\rangle)$ be the conference call cost of searching all tokens in box sets $A_1,\ldots,A_d$. The recursion base is $\mathrm{CC}(\langle A_1\rangle) = \prod_{m=1}^{M} Q_{m,1}|A_1|$, since if all tokens

are in boxes of part $A_1$, we unlock $|A_1|$ boxes. The recursion step is

$$\mathrm{CC}(\langle A_1, \ldots, A_d \rangle) = \mathrm{CC}(\langle A_1, \ldots, A_{d-1} \rangle) + \left( \prod_{m=1}^{M} Q_{m,d} - \prod_{m=1}^{M} Q_{m,d-1} \right) |A_d|, \quad (5.4)$$

because to search all tokens in box sets $A_1, \ldots, A_d$, we must unlock sets $A_1, \ldots, A_{d-1}$ first and pay a cost of $\mathrm{CC}(\langle A_1, \ldots, A_{d-1} \rangle)$, and with probability that at least one token is in $A_d$, we pay an extra cost of $|A_d|$.

Time Complexity: $\Theta(MN + MD)$.

**Exclusive computation:** With probability $\prod_{m=1}^{M} R_{m,d-1}$, all tokens are in parts $\{A_d \ldots A_D\}$; with probability $\prod_{m=1}^{M} R_{m,d}$, all tokens are in parts $\{A_{d+1} \ldots A_D\}$. Thus, with the difference of the above two probabilities, at least one token is in part $A_d$ but no token is in parts $A_1 \ldots A_{d-1}$, in which case we need to unlock exactly $S_d$ boxes. Summing through $d = 1, \ldots, D$, we have the cost for the yellow page problem.

$$\mathrm{YP}(\mathcal{A}) = \sum_{d=1}^{D} \left( S_d \cdot \left( \prod_{m=1}^{M} R_{m,d-1} - \prod_{m=1}^{M} R_{m,d} \right) \right) \quad (5.5)$$

Similarly, in the conference call problem, with probability $\prod_{m=1}^{M} Q_{m,d}$, all tokens are in parts $\{A_1 \ldots A_d\}$ and with probability $\prod_{m=1}^{M} Q_{m,d-1}$, all tokens are in parts $\{A_1 \ldots A_{d-1}\}$. Thus, with the difference of the above probabilities, at least one token is in part $A_d$ and all tokens are in parts $A_1 \ldots A_d$, in which case we need to unlock exactly $S_d$ boxes.

$$\mathrm{CC}(\mathcal{A}) = \sum_{d=1}^{D} \left( S_d \cdot \left( \prod_{m=1}^{M} Q_{m,d} - \prod_{m=1}^{M} Q_{m,d-1} \right) \right) \quad (5.6)$$

Time Complexity: $\Theta(MN + MD)$

## 5.4 Types of tokens

In Bar-Noy and Malewicz [2004], the authors proved that the conference call problem is NP-hard. We conjecture that the yellow page problem is also NP-hard. In Section 5.5, we observe some "duality" between the two problems. Since the general setting is hard to tackle, we study some interesting restricted classes of instances, for which we provide more efficient optimal solutions – some have polynomial running time and some have improved exponential running time. Toward that goal, we present a hierarchical classification of types of tokens.

We define a few properties for a set of $M$ tokens according to their probabilities in the set of boxes. A set of tokens is *identical* if for any box $B_n \in \{B_1, \ldots, B_N\}$, $p_{1,n} = \cdots = p_{M,n}$. A set of tokens is *uniform* if for each token $m = 1 \ldots M$, $p_{m,n}$ is either 0 or $1/k$, where $k$ is the number of non-zero entries in $\{p_{m,1}, \ldots, p_{m,N}\}$. A set of tokens is *similar*, if for all tokens $m = 2, \ldots, M$, $\{p_{m,1}, \ldots, p_{m,N}\}$ is some permutation of token $T_1$'s probabilities $\{p_{1,1}, \ldots, p_{1,N}\}$. A set of tokens is *disjoint* if for each box $B_n \in \{B_1, \ldots, B_N\}$, there is exactly one non-zero entry $p_{m,n}$, for some $m = 1, \ldots, M$. We present the combinations of these properties in a hierarchy in Fig. 5.1.

## 5.5 Optimal solutions

In this section, we first adapt the dynamic programming scheme in Section 2.2 to compute the search strategy for a predetermined order of the boxes. Based on these algorithms, we describe relatively efficient optimal solutions for some types of tokens (and their ancestor types in the hierarchy described in Fig. 5.1.

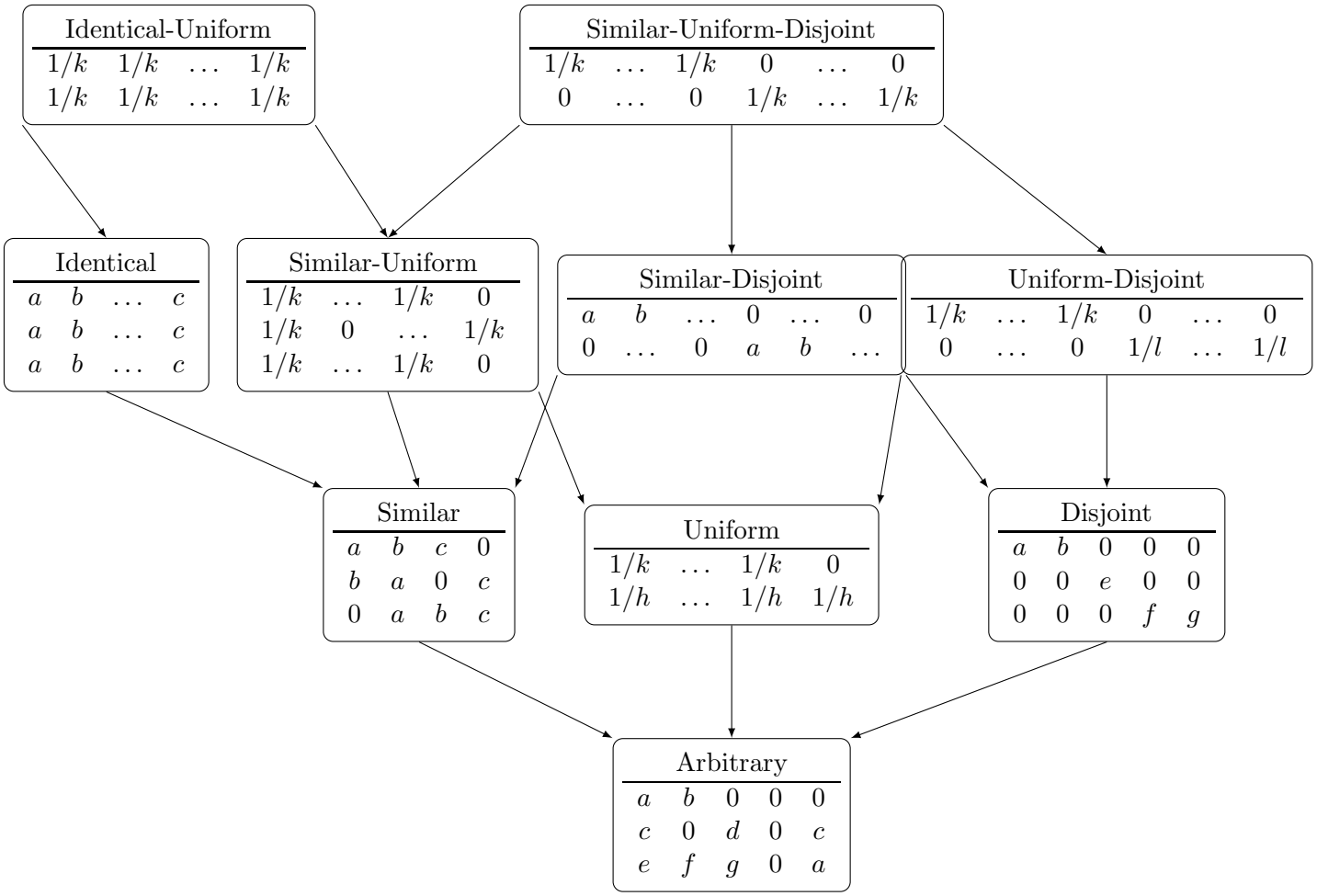



Figure 5.1: Hierarchy among different token types; $a, b, c, d, e, g, h \leq 1$ are positive real numbers, $h, k, l \leq N$ are positive integers.

Given an order of the boxes, say $\langle B_1, \ldots, B_N \rangle$ (without loss of generality), a search strategy $\mathcal{A} = \langle A_1, \ldots, A_D \rangle$ is said to *respect the above order* if for any boxes $B_i$, $B_j$ with $i < j$, we have $B_i \in A_{d_i}$ and $B_j \in A_{d_j}$ with $d_i \leq d_j$. Given an order of the boxes, Algorithms 5.1 (for the YP problem) and 5.2 (for the CC problem) compute the optimal search cost and corresponding strategy that respects this order, in polynomial time.

In Algorithm 5.1, let $h^{\text{yp}}_{n,d}$ denote the optimal cost of unlocking boxes $\{B_n, \ldots, B_N\}$ in $d$ rounds given the condition that no token resides in boxes $\{B_1, \ldots, B_{n-1}\}$. Our objective is to find $h^{\text{yp}}_{1,D}$. It is not difficult to see that $h^{\text{yp}}_{n,1} = N - n + 1$. To compute $h^{\text{yp}}_{n,d}$, we need to search through all possible $j$s with $n + 1 \leq j \leq n - d + 1$ for the strategy of minimum cost that unlocks boxes $\{B_j, \ldots, B_N\}$ in the last $(d-1)$ rounds and unlocks boxes $\{B_n, \ldots, B_{j-1}\}$ in the previous round; the inner loop $(*)$ follows equation (5.3)).

**Algorithm 5.1** Dynamic programming algorithm, respect order $\langle B_1, \ldots, B_N \rangle$, yellow page, cost = DPYP($p[M][N], D$)

**Input:** $\mathbf{p}_{M \times N}$, $D$
**Output:** $h^{\text{yp}}_{1,D}$
  **for** $n = 1 \ldots N$ **do**
    $h^{\text{yp}}_{n,1} \leftarrow N - n + 1$
  **end for**
  **for** $d = 2 \ldots D$ **do**
    **for** $n = 1 \ldots (N - d)$ **do**
      $h^{\text{yp}}_{n,d} \leftarrow \min_{j=n+1}^{n-d+1} |j - n| + (\prod_{m=1}^{M} R_{m,n} - \prod_{m=1}^{M} R_{m,j}) / \prod_{m=1}^{M} R_{m,n} \cdot h^{\text{yp}}_{j,d-1}$ $(*)$
    **end for**
  **end for**
  **return** $h^{\text{yp}}_{1,D}$

In Algorithm 5.2, let $h^{\text{cc}}_{n,d}$ denote the optimal cost of unlocking boxes $\{B_1, \ldots, B_n\}$ in $d$ rounds. Our objective is to find $h^{\text{cc}}_{N,D}$. It is not difficult to see that $h^{\text{cc}}_{n,1} = n$. To compute $h^{\text{cc}}_{n,d}$, we need to search through all possible $j$s with $(d-1) \leq j < n$ for the strategy of minimum cost that unlocks boxes $\{B_1, \ldots, B_j\}$ in the first $(d-1)$ rounds and unlock boxes $\{B_{j+1}, \ldots, B_n\}$ in the $d$th round; the inner loop $(*)$ follows

equation (5.4).

**Algorithm 5.2** Dynamic programming algorithm, respect order $\langle B_1, \ldots, B_N \rangle$, conference call, cost = DPCC($p[M][N], D$)

**Input:** $\mathbf{p}_{M \times N}$, $D$
**Output:** $h^{\text{cc}}_{1,D}$
  **for** $n = 1 \ldots N$ **do**
    $h^{\text{cc}}_{n,1} \leftarrow n$
  **end for**
  **for** $d = 2 \ldots D$ **do**
    **for** $n = d \ldots N$ **do**
      $h^{\text{cc}}_{n,d} \leftarrow \min_{j=d-1}^{n-1} h^{\text{cc}}_{j,d-1} + \left( \prod_{m=1}^{M} Q_{m,n} - \prod_{m=1}^{M} Q_{m,j} \right) |n - j|$ $(*)$
    **end for**
  **end for**
  **return** $h^{\text{cc}}_{N,D}$

The correctness of Algorithms 5.1 and 5.2 follows from the fact that, under the particular order constraint, any sub-partition of an optimal search strategy must be local-optimal within itself; otherwise, replacing the sub-partition with the alternative local-optimal search strategy would gain a better search strategy than optimal.

**Lemma 5.3.** *The running time of Algorithms 5.1 and 5.2 is* $\Theta(MDN^2)$

### 5.5.1 Monotonic tokens

**Definition 5.4.** A set of tokens is called *monotonic* if there is a permutation of boxes, without lost of generality, say $\langle B_1, \ldots, B_N \rangle$, such that $p_{m,1} \geq \cdots \geq p_{m,N}$ for every $m \in \{1, \ldots, M\}$. Let this permutation be the monotonic order of the boxes for the monotonic tokens.

**Lemma 5.5.** *For monotonic tokens, the optimal search strategies for both the yellow page and conference call problems follow the monotonic order of the boxes.*

*Proof.* In the yellow page problem, denote the per-box probability suffix with $r_{m,n} = \sum_{i=n+1}^{N} p_{m,i}$. Assume for the sake of contradiction, that in the optimal search strategy

$\mathcal{O}$, there are two boxes, $B_i$, $B_j$, for which $B_i \in O_d$ and $B_j \in O_{d+1}$, but $p_{m,i} < p_{m,j}$. Using equation (5.5), the search cost of $\mathcal{O}$ is:

$$\mathrm{YP}(\mathcal{O}) = \text{cost before round } d + S_d \cdot \left( \prod_{m=1}^{M} r_{m,S_{d-1}} - \prod_{m=1}^{M} r_{m,S_d} \right) + S_{d+1} \cdot \left( \prod_{m=1}^{M} r_{m,S_d} - \prod_{m=1}^{M} r_{m,S_{d+1}} \right) + \text{cost after round } (d+1). \quad (5.7)$$

Consider the strategy $\mathcal{O}'$ where we switch the round assignments of boxes $B_i$ and $B_j$. Then,

$$\mathrm{YP}(\mathcal{O}') = \text{cost before round } d + S_d \cdot \left( \prod_{m=1}^{M} r_{m,S_{d-1}} - \prod_{m=1}^{M} r'_{m,S_d} \right) + S_{d+1} \cdot \left( \prod_{m=1}^{M} r'_{m,S_d} - \prod_{m=1}^{M} r_{m,S_{d+1}} \right) + \text{cost after round } (d+1). \quad (5.8)$$

Besides identical terms, for every $m$, $r'_{m,S_d}$ contains $p_{m,i}$ whereas $r_{m,S_d}$ contains $p_{m,j}$, i.e., $r'_{m,S_d} - r_{m,S_d} = p_{m,i} - p_{m,j}$, which implies $r'_{m,S_d} < r_{m,S_d}$, because $p_{m,i} < p_{m,j}$. Subtracting (5.8) from (5.7), we get

$$\mathrm{YP}(\mathcal{O}) - \mathrm{YP}(\mathcal{O}') = (S_d - S_{d+1}) \cdot \left( \prod_{m=1}^{M} r'_{m,S_d} - \prod_{m=1}^{M} r_{m,S_d} \right) > 0$$

because both factors are negative. This is a contradiction to the optimality of $\mathcal{O}$.

Similarly, we can prove the lemma for the conference call problem. □

Applying Algorithms 5.1 and 5.2 on the monotonic order yields:

**Corollary 5.6.** *The optimal search strategies for both the yellow page problem and the conference call problem for monotonic tokens can be computed in polynomial time for any $D$, $M$, and $N$.*

*Proof.* By Lemma 5.5, applying Algorithms 5.1 and 5.2 on the monotonic order yield the optimal strategies. □

### 5.5.2 D=N, duality

An interesting case is when $D = N$, i.e., *sequential* search, in which a search strategy is a permutation of the boxes. The conference call problem has been proved NP-Hard in Bar-Noy and Malewicz [2004]. Although we have not yet proved the NP-hardness for yellow page problem, we show that there is some kind of *duality* between the two problems.

**Lemma 5.7.** *Let $\mathcal{A}$ be a search strategy that searches $M$ tokens in $N$ boxes in $D = N$ rounds. Let an instance of yellow page with probabilities $p_{m,n}$. Let another instance for the conference problem with probabilities $q_{m,n} = p_{m,N+1-n}$. Then, $YP(\mathcal{A}, p_{m,n}) + CC(\mathcal{A}, q_{m,n}) = N + 1$.*

*Proof.* From (5.2), we have (the summation is over all possible location vectors $\mathbf{d} = (d_1, \ldots, d_M) \in \{1, \ldots, N\}^M$):

$$\begin{aligned}
\mathrm{CC}(\mathcal{A}, q_{m,n}) &= \sum_{\mathbf{d}} \left( \max_{m=1}^{M} d_m \cdot \prod_{m=1}^{M} q_{m,d_m} \right) \\
&= \sum_{\mathbf{d}} \left( (N + 1 - \min_{m=1}^{M}(N + 1 - d_m)) \cdot \prod_{m=1}^{M} p_{m,N+1-d_m} \right) \\
&= \sum_{\mathbf{d}} \left( (N+1) \cdot \prod_{m=1}^{M} p_{m,N+1-d_m} \right) - \sum_{\mathbf{d}} \left( \min_{m=1}^{M}(N + 1 - d_m) \cdot \prod_{m=1}^{M} p_{m,N+1-d_m} \right) \\
&= N + 1 - \sum_{\mathbf{d}'} \left( \min_{m=1}^{M} d'_m \cdot \prod_{m=1}^{M} p_{m,d'_m} \right) \\
&= N + 1 - \mathrm{YP}(\mathcal{A}, p_{m,n})
\end{aligned}$$

where $d'_m = N + 1 - d_m$ and $\mathbf{d}' = (d'_1, \ldots, d'_m)$, because the mapping $d_m \mapsto d'_m$ is a bijection in $\{1, \ldots, N\}$, after using equation (5.1). □

**Corollary 5.8.** *When $D = N$, the maximization problem of yellow page is equivalent to the minimization problem of conference call, and vice versa.*

### 5.5.3 D=N, arbitrary tokens

When $D = N$, a brute-force approach to find the optimal strategy is to test all permutations of the boxes. This method requires $\Theta(N!)$ time. We present lemmas, which allow us to give instead a $\Theta(2^N)$ algorithm that generates the optimal permutation.

**Lemma 5.9.** *In the yellow page problem, if $\langle B_{i_1}, \ldots B_{i_n}\rangle$ is an optimal search strategy of unlocking boxes $\{B_{i_1}, \ldots B_{i_n}\}$ given the condition that no token is located in the rest of boxes, then any suffix of it, $\langle B_{i_k}, \ldots B_{i_n}\rangle$, for $1 \leq k \leq n$, is an optimal search strategy that unlocks boxes $\langle B_{i_k}, \ldots B_{i_n}\rangle$ given no token is located in any other boxes (except in $\{B_{i_k}, \ldots B_{i_n}\}$).*

*Proof.* Assume for contradiction that, $\langle B_{j_k}, \ldots B_{j_n}\rangle$ ($\neq \langle B_{i_k}, \ldots, B_{i_n}\rangle$) is the optimal search strategy for the suffix, then

$$\mathrm{YP}(\langle B_{j_k}, \ldots B_{j_n}\rangle) < \mathrm{YP}\,\langle B_{i_k}, \ldots B_{i_n}\rangle \ .$$

Using equation (5.3),

$$\mathrm{YP}(\langle B_{i_1}, \ldots, B_{i_{k-1}}, B_{j_k}, \ldots, B_{j_N}\rangle) < YP(\langle B_{i_1}, \ldots, B_{i_{k-1}}, B_{i_k}, \ldots, B_{i_N}\rangle) \ ,$$

which is a contradiction to the optimality of $\langle B_{i_1}, \ldots, B_{i_{k-1}}, B_{i_k}, \ldots, B_{i_N}\rangle$. □

Similarly, we have the following lemma for the conference call problem:

**Lemma 5.10.** *In the conference call problem, if $\langle B_{i_1}, \ldots, B_{i_n}\rangle$ is an optimal search strategy of unlocking boxes $\{B_{i_1}, \ldots, B_{i_n}\}$ in the first $i_n$ rounds, then any prefix of it, $\langle B_{i_1}, \ldots, B_{i_k}\rangle$, for $1 \leq k \leq n$, is an optimal search strategy that unlocks boxes in $\{B_{i_1}, \ldots B_{i_k}\}$ in the first $i_k$ rounds.*

In light of Lemma 5.9, Algorithm 5.3 computes an optimal search strategy for the yellow page problem. A dedicated array Best[$2^N$] is used in the algorithm. Best[$k$]

records the optimal sub-strategy of unlocking boxes $\{B_{i_1}, \ldots, B_{i_l}\}$ where $i_1, \ldots, i_l$ are the bits of 1 after converting $k$ into binary. In the first for loop, we initialize the optimal search strategy of a single box given no token is found in other boxes which is to unlock the box itself in the only around. For unlocking $l$ boxes in $l$ rounds, we search through all possible cases that unlock one of the $l$ boxes in the first round, and unlock the other $(l-1)$ boxes optimally in the remaining $(l-1)$ rounds. The data structure Best is set up for random access any optimal sub-strategy that has been already computed.

**Algorithm 5.3** $D = N$, arbitrary tokens: compute the optimal cost and strategy for yellow page using dynamic programming; opt = YPDP($A$)

**Input:** $A$
**Output:** Best[$A$]
  **for** $\forall A$, that $|A| = 1$ **do**
    $\text{Best}[A]_{\text{cost}} \leftarrow 1$
    $\text{Best}[A]_{\text{strategy}} \leftarrow \langle A \rangle$
  **end for**
  **for** $\forall A$ that $|A| = 2 \ldots N$ **do**
    $\text{Best}[A]_{\text{cost}} \leftarrow \min_{B_i \in A}\{1 + \frac{\prod_{m=1}^{M} \sum_{B_n \in A \setminus B_i} p_{m,i}}{\prod_{m=1}^{M} \sum_{B_n \in A} p_{m,i}} \cdot \text{Best}[A \setminus B_i]_{\text{cost}}\}$
    $\text{Best}[A]_{\text{strategy}} \leftarrow \langle \arg\min\{B_i | \text{Best}[A \setminus B_i]_{\text{cost}}\}, \text{Best}[A \setminus B_i]_{\text{strategy}} \rangle$
  **end for**
  **return** $\{\text{Best}[A] | \text{ that } |A| = N\}$

Similarly, we construct Algorithm 5.4 for the conference call problem. The only difference with the yellow page algorithm is the recursive computation of the cost according to (5.4).

**Theorem 5.11.** *The time complexity of Algorithms 5.3 and 5.4 is* $\Theta(MN \cdot 2^N)$. *The space complexity is* $\Theta(N \cdot 2^N)$.

*Proof.* Fig. 5.2 demonstrates an example of running process of Algorithm 5.3. We note that there are $\Theta(2^N)$ nodes, for general $N$, in the above figure. To compute each node, on average, one need to randomly access $\lceil N/2 \rceil + 1$ nodes from the previous column. Each random access can be executed in $\Theta(M)$ time after some linear time

**Algorithm 5.4** $D = N$, arbitrary tokens: compute the optimal cost and strategy for conference call using dynamic programming; opt = CCDP($A$)

**Input:** $A$
**Output:** Best[$A$]
  **for** $\forall A$, that $|A| = 1$ **do**
    $\text{Best}[A]_{\text{cost}} \leftarrow \prod_{m=1}^{M} P_{m,A}$
    $\text{Best}[A]_{\text{strategy}} \leftarrow \langle A \rangle$
  **end for**
  **for** $\forall A$ that $|A| = 2 \dots N$ **do**
    $\text{Best}[A]_{\text{cost}} \leftarrow \min_{B_i \in A}\{\text{Best}[A \setminus B_i]_{\text{cost}} + \left(\prod_{m=1}^{M} P_A - \prod_{m=1}^{M} P_{A \setminus B_i}\right) \cdot |A|\}$
    $\text{Best}[A]_{\text{strategy}} \leftarrow \langle \arg\min\{\text{Best}[A \setminus B_i]_{\text{cost}}, \text{Best}[A \setminus B_i]_{\text{strategy}}, B_i\}\rangle$
  **end for**
  **return** $\{\text{Best}[A] | \text{ that } |A| = N\}$

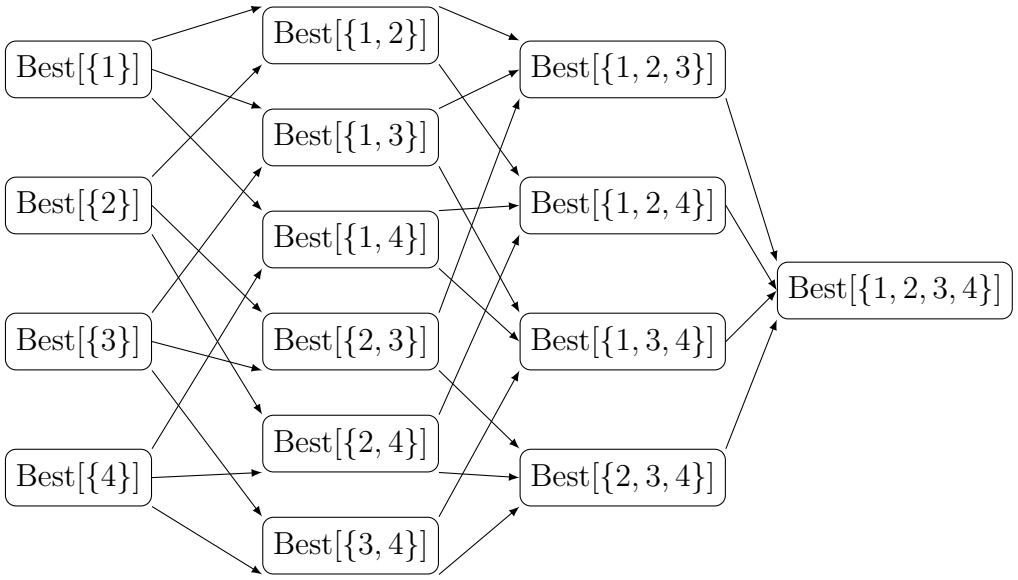



Figure 5.2: Illustration of Algorithm 5.3 for $D = N = 4$.

preprocessing. The overall time complexity of the algorithm is $\Theta(MN \cdot 2^N)$.

To randomly access from one column to the previous column, we need to maintain at least $\binom{N}{\lfloor N/2 \rfloor} + \binom{N}{\lfloor N/2 \rfloor + 1}$ nodes in the memory, and the average size of a node is $\lceil N/2 \rceil + 1$. Thus, the overall space complexity of the algorithm is $\Theta(N \cdot 2^N)$. □

### 5.5.4 D=N, disjoint tokens

For disjoint tokens we can reduce the running time of optimal algorithm to $O(N^M)$ based on the following lemma.

**Lemma 5.12.** *For disjoint tokens, in an optimal search strategy, for every token, the boxes where the token is located must be unlocked in the order of non-increasing probability.*

*Proof.* By contradiction. Without loss of generality, suppose in the optimal search strategy $\mathcal{O}$, for token $T_1$, box $B_i$ is unlocked in $d_i$th round and $B_j$ is unlocked in $d_j$th round, with $d_i < d_j$, but $p_{1,i} < p_{1,j}$. By swapping $B_i$ and $B_j$, we obtain another search strategy, $\mathcal{O}'$. We have

$$\mathrm{YP}(\mathcal{O}) = \text{terms group 1} + p_{1,i} \cdot d_i \cdot (\text{terms group 2}) + p_{1,j} \cdot d_j \cdot (\text{terms group 3})$$
$$\mathrm{YP}(\mathcal{O}') = \text{terms group 1} + p_{1,j} \cdot d_i \cdot (\text{terms group 2}) + p_{1,i} \cdot d_j \cdot (\text{terms group 3})$$

We observe that $p_{1,i}d_i + p_{1,j}d_j > p_{1,j}d_i + p_{1,i}d_j$. Thus $\mathrm{YP}(\mathcal{O}) > \mathrm{YP}(\mathcal{O}')$, which is a contradiction to $\mathcal{O}$'s optimality. (A similar proof can be applied to the conference call problem.) □

**Optimal algorithm [for disjoint tokens]:** We show the algorithm for $M = 2$ tokens in the yellow page problem. It can be extended to any number of tokens by constructing a $M$ dimensional dynamic programming table. The algorithm also works with slight modifications for the conference call problem.

Let two tokens hidden in $(k + l)$ boxes. Token $T_1$ only resides in the first $k$ boxes with probabilities $p_1, \ldots, p_k$ and token $T_2$ only resides in the other $l$ boxes with probabilities $q_1, \ldots, q_l$. Without loss of generality, we assume $p_1 \geq \cdots \geq p_k$ and $q_1 \geq \cdots \geq q_l$. Define matrix $\mathrm{Best}_{k\times l}$ such that $\mathrm{Best}[i, j]$ is the best search strategy within boxes with probabilities $p_i, \ldots, p_k, q_j, \ldots, q_l$ given the condition that no token is located in the other $(i + j - 2)$ boxes. By defining $p_{k+1} = q_{l+1} = 0$, we observe

that $\text{Best}[k+1, l+1]_{\text{cost}} = 0$, $\text{Best}[k+1, j]_{\text{cost}} = 1 + \frac{\sum_{t=j+1}^{l} q_t}{\sum_{t=j}^{l} q_t} \text{Best}[k+1, j+1]_{\text{cost}}$ for $1 \leq j \leq l$, and $\text{Best}[i, l+1]_{\text{cost}} = 1 + \frac{\sum_{t=i+1}^{k} p_t}{\sum_{t=i}^{k} p_t} \text{Best}[i+1, l+1]_{\text{cost}}$ for $1 \leq i \leq k$. We recursively compute other entries in Best, according to Lemma 5.12:

$$\text{Best}[i, j]_{\text{cost}} = \min \begin{cases} 1 + \frac{p_{i+1} + \cdots + p_k}{p_i + \cdots + p_k} \text{Best}[i+1, j]_{\text{cost}} \\ 1 + \frac{q_{j+1} + \cdots + q_l}{q_j + \cdots + q_l} \text{Best}[i, j+1]_{\text{cost}} \end{cases} \tag{5.9}$$

$\text{Best}[1, 1]$ is the optimal search strategy for all boxes.

For $M > 2$ tokens, a similar algorithm can be designed by using an $M$-dimensional Best array. The size of data structure Best is $O(N^M)$. Computing an entry of Best, takes $\text{Poly}(M, N)$ time. The overall complexity of the algorithm is $\Theta(N^M \text{Poly}(M, N))$, which is polynomial with respect to the number of boxes.

We implement the algorithm and use it in our simulation. It runs very fast for small $M$.

## 5.6 Non-optimal heuristics

We design a family of 12 heuristics that compute search strategies in practice. All our heuristics are of the following form: First, we compute an order of the boxes (according to some greedy method) and then, we apply Algorithms 5.1 or 5.2 to find the best strategy that follows this order of the cells. We have four criteria to order the boxes. Define $X_n = \prod_{m=1}^{M} p_{m,n}$ (the probability all tokens are in box $B_n$). Define $Y_n = \prod_{m=1}^{M} (1 - p_{m,n})$ (the probability no token is in box $B_n$). Define $S_n = \sum_{m=1}^{M} p_{m,n}$ (the sum of token probabilities being in box $B_n$). Define $Z_n = \max_{m=1}^{M} p_{m,n}$ (the maximum probability of a token in box $B_n$). Heuristics $X$, $Y$, $S$ and $Z$ unlock the boxes in the orders $X_1 \geq \ldots \geq X_N$, $Y_1 \leq \ldots \leq Y_N$, $S_1 \geq \ldots \geq S_N$[1] and $Z_1 \geq \ldots \geq Z_N$, respectively.

[1] Bar-Noy and Malewicz [2004] showed $Y$ is an $\frac{e}{e-1}$-approximation for any $D \leq N$.

We use the above four basic heuristics to compute search strategies for both problems. For each basic greedy heuristic $\mathcal{G} \in \{X, Y, Z, S\}$, we design two adaptive versions, namely $BF\mathcal{G}$ and $WL\mathcal{G}$. In the *Best First* (BF) version, each time we select a box to form an order, we select the next available box that has the best value (maximum for $X$, $S$, $Z$ and minimum for $Y$), and then normalize the probabilities among unselected boxes. In the *Worst Last* (WL) version, we select the available box that has the worst value (minimum for $X$, $S$, $Z$ and maximum for $Y$) as the last box in the order, and then normalize probabilities among unselected boxes.

## 5.7 Summary and open problems

For the searching for multiple tokens problem, we have shown three methods of computing search cost given a search strategy. We also have shown a duality between the yellow page problem and conference call problem. We have provided polynomial time algorithm for: 1) monotonic tokens 2) disjoint tokens. We have reduced the complexity of computing optimal sequential search strategy from $\Theta(M \cdot N!)$ to $\Theta(M \cdot 2^N)$ utilizing a tradeoff between time and space. We also have designed a family of 12 heuristics based on 4 criteria.

We conclude by presenting some open problems. Our search strategies are static since they are predetermined before the search process starts. In a dynamic setting, we may select the boxes to be paged in the next round according to the tokens found in previous rounds. The dynamic yellow page problem remains the same while a solution to the dynamic conference call problem can outperform the optimal solution for the static call conference problem. How to compute a good dynamic search strategy remains open.

A natural generalization of searching for multiple tokens problems is to efficiently find $k$ out of $M$ tokens. In the yellow page problem $k = 1$ and in the conference call

problem $k = M$. The motivation for this general case could be the task of finding a team of $k$ doctors out of a pool of $M$ doctors. A further generalization of the problem can be searching for a subset of tokens such that some disjunctive normal formulae (DNF) are satisfied. For example, we search for three tokens $\{A, B, C\}$. We must find token $B$, and either of the tokens $A$ or $C$. The DNF of the search condition is $(A \wedge B) \vee (B \wedge C)$.

In this chapter, we have assumed that the probabilities are independent among tokens and among boxes. The cases where the probabilities are dependent among tokens or among boxes are to be studied. For example, if the probabilities are box-wise dependent, a token can be more likely to be in box $B_1$ if it is not in box $B_2$, i.e., $\Pr(T_1 \text{ in } B_1 | T_1 \text{ not in } B_2) > \Pr(T_1 \text{ in } B_1)$; if the probabilities are token-wise dependent, it is possible that $\Pr(T_1 \text{ in } B_1 \text{ and } T_2 \text{ in } B_1) \neq \Pr(T_1 \text{ in } B_1) \cdot \Pr(T_2 \text{ in } B_1)$.

In the searching for multiple token problem, we have assumed the unlocking costs of all the boxes are the same as the basic search problem in Chapter 3. We want to combine the searching for multiple tokens problem with the search with cost problem in Chapter 4. In such a combination, the boxes can have different unlocking costs and the goal becomes minimizing the expected search cost. This makes both the yellow page problem the conference call problem more difficult and more general.

# Chapter 6

# Experimental study

In this chapter, we present our experimental study of existing algorithms and our own algorithms on the problem of searching for mobile data in the context of its main application: paging mobile users in cellular networks. We start with the general configuration of our experiments, and then we present our simulation for three different settings – the basic search problem, the search with cost problem, and the searching for multiple tokens problem.

## 6.1 Experiment setup

### 6.1.1 Data

In Buchanan [2008], the authors have revealed that people appear in $N$ different places with a frequency that follows the *Zipf* distribution, which is defined as $\mathbf{p} = \langle p_1, \ldots, p_N \rangle$ with $p_i = i^{-\alpha} / \sum_{i=1}^{N} i^{-\alpha}$ where $\alpha \geq 0$ is the Zipf parameter. Zipf distribution obeys a power law. When $\alpha = 0$, it is a uniform distribution; as $\alpha$ grows, the distribution becomes more and more skewed.

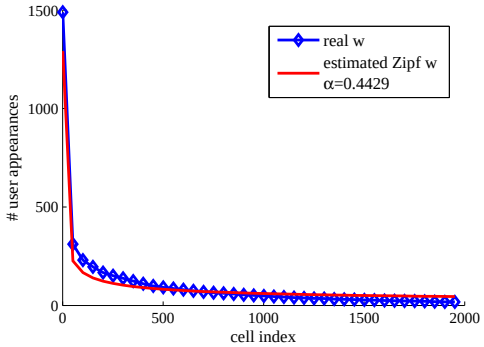
For our problem of searching mobile users in cellular networks, we conjecture that both probability vector $\mathbf{p}$ and cost vector $\mathbf{w}$ follow the Zipf distribution. We now

model and verify our conjecture by analyzing real user data. Based on this model, we generate substantial amount of synthetic data to conduct our simulation.

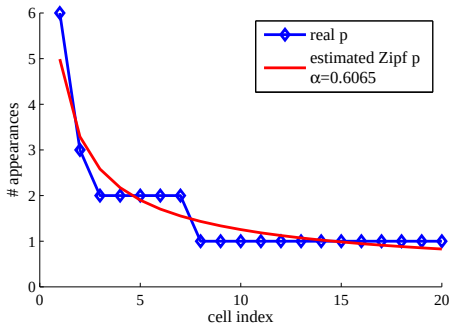
**Real data:** We obtain 171929 appearances of 996 users in 5625 cells on 31 consecutive days from the boundary of a metro and a suburban region. The data is anonymously provided by Shenzhen division of China Unicom. For every user appearance, we record pair ⟨user ID, cell ID⟩ that denotes the user and the cell in which it appears.

Cellwise, we model the cell congestion vector $\mathbf{w}$ of the 5625 cells. We estimate exponent $\alpha$ using the maximum likelyhood method with cross validation and we obtain $\alpha = 0.4429$. The cross validation indicates that the estimates for $\alpha$ differ less than 1% when using the odd entries and even entries of the data separately. This further validates our assumption of a Zipf distribution. The cell congestion $\mathbf{w}$ and the estimated Zipf $\mathbf{w}$ is shown in Figure 6.1(a).

Userwise, we model the user probability vector $\mathbf{p}$ of all 996 users. Since each user only appears in a limited number of cells, techniques like cross validation cannot be used because we do not have enough samples. We plot many user probability vectors and their corresponding estimated Zipf distribution, and almost all of the plots appear like in Figure 6.1(b). This indicates that $\mathbf{p}$ also follows a Zipf distribution.




(a) Cell congestion $\mathbf{w}$ and estimated Zipf distribution; $x$-axis: cell indices, $y$-axis: number of user appearances




(b) A random user's $\mathbf{p}$ and the estimated Zipf distribution; $x$-axis: cell indices, $y$-axis: number of appearances

Figure 6.1: Data analysis

Based on these real data: for each cell, we calculate a cost, which is the number of user appearances in the particular cell; for each user, we generate a probability vector $\mathbf{p}$ that records the probability of it being in different cells, and a corresponding cost vector $\mathbf{w}$, which records the cost of paging different cells that it might locate in. Vectors $\mathbf{p}$ and $\mathbf{w}$ respect the same order of cells and are normalized so that all the elements each vector sum to 1.

**Synthetic data:** To conduct empirical study on the problem, we generate a large amount of simulated data in addition to the real data. The simulated data are recorded in pair of $\langle \mathbf{p}, \mathbf{w} \rangle$. Both $\mathbf{p}$ and $\mathbf{w}$ follows Zipf distribution. The Zipf parameter $\alpha$ varies from 0 to 3 in increment of 0.1. The number of cells of each user varies from 10 to 20000 in progressive steps.

**Special data:** In order to research the behavior of different algorithms, we also conduct our simulate on crafted type of data. Such data will be introduced in the specific sections as needed.

### 6.1.2 Hardware, operating system, and programming language

**Hardware:** Our simulation has been conducted on the Research Computing Cluster at the Graduate Center of City University of New York. Two types of computing nodes are used through the process of this work. When compare performance of algorithms, we ensure the results are acquired from the same type of computing nodes. All our algorithms are implemented single processed, single threaded, thus, only one core of each computing node are utilized.

- Quad Node: Intel Q9550 Quad Core CPU with 16GB of Memory
- Octo Node: Two Intel Xeon 5240 Quad Core CPU with 64GB of Memory

**Operating System:** Red Hat Enterprise Linux 5.5 x64, or its open source equivalent counterpart CentOS 5.5 x64.

**Programming languages:**

- C++: used to implement our search algorithms, compiler g++ 4.3, default options
- GNU Plot: used to plot figures, GNU plot 4.3 for Linux
- Matlab: used to plot figures, Matlab 2008 and 2009 under Windows

## 6.2 The basic paging problem

### 6.2.1 Overview

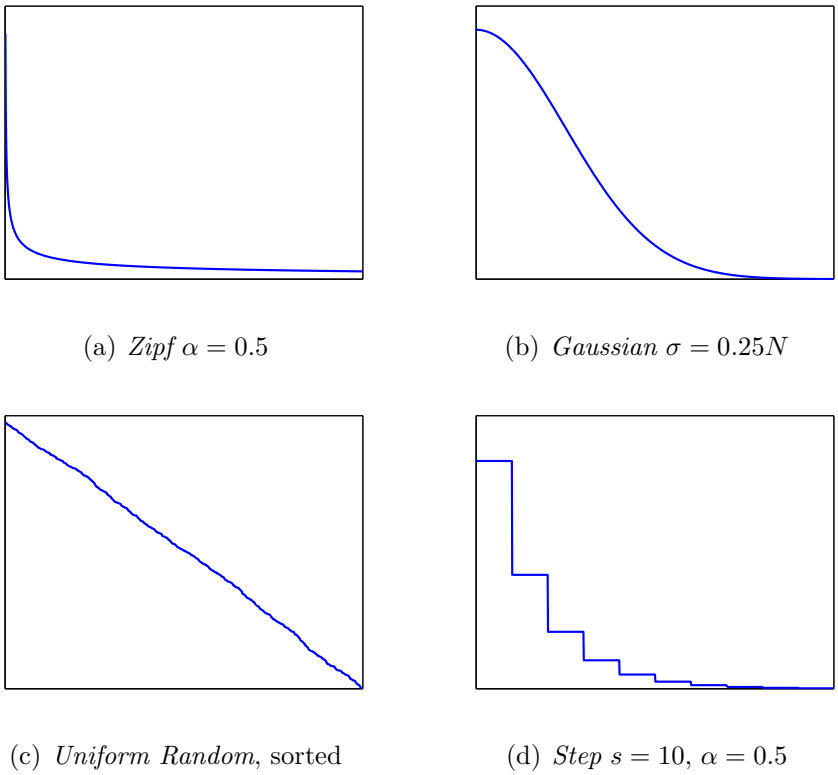
**Objective:** We conduct experimental study on the basic paging problem for three purposes: 1. Experimentally verify and evaluate the running time of our algorithms. 2. Demonstrate a hierarchy of the optimality, time complexity, and implementation complexity of our algorithms. 3. Experimentally evaluate the performance of our non-optimal heuristics.

**Algorithms Implemented:** We implement all the algorithms in Chapter 3, including the optimal algorithms Seq, Bin, SMAWK, and SpeedUp, and the non-optimal algorithms LargeSuffix, Uniform, Doubling, Boundaries, DQ, and FirstLocalMin.

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

**Data:** In addition to the standard Zipf data, we test our algorithm in the following three types of special data that is generated from the Gaussian, Uniform Random, and

Step distributions. Some data types, e.g., the Step distributed data, have potentially assisted us in the optimality analysis of algorithm FirstLocalMin.



(a) *Zipf* $\alpha = 0.5$ (b) *Gaussian* $\sigma = 0.25N$

(c) *Uniform Random*, sorted (d) *Step* $s = 10$, $\alpha = 0.5$

Figure 6.2: Distributions of data model: $x$-axis: the box index $B_i$, $y$-axis: $p_i$.

The *Gaussian* distribution is a continuous distribution commonly used to model the notion of ranking based on some frequency of occurrence Wang et al. [2001]. The *Uniform Random* distribution is a natural distribution, in which each $p_i$ is a uniform random number between 0 and 1 that is normalized by the sum of all the $N$ original random numbers. The *Step* distribution is a local-uniform distribution in which the cells are partitioned into groups such that all the cells in a group are associated with the same location probability. We note that probability vectors that are similar to the Step distribution are used as counter examples to some stronger claims regarding the structure of the matrix $A$. We now define the various distributions formally. See Figure 6.2 for some typical plots. Without lost of generality, we order the probability vector in non-increasing order of $p_i$s because in the optimal search strategy, boxes

much be unlocked with respect to this order.

**The Zipf distribution:** Given $\alpha \geqslant 0$, for $1 \leq i \leq N$,

$$p_i = \frac{i^{-\alpha}}{\sum_{i=1}^{N} i^{-\alpha}} \ .$$

**The Gaussian distribution:** Given $\sigma > 0$, for for $1 \leq i \leq N$,

$$p_i = \frac{q_i}{\sum_{i=1}^{N} q_i},\ 1 \leq i \leq N, \text{ where } q_i = \frac{2}{\sigma\sqrt{2\pi}} \exp(-\frac{i^2}{2\sigma^2}) \ .$$

**The Uniform Random distribution:**

$$p_i = \frac{q_i}{\sum_{i=1}^{N} q_i},\ 1 \leq i \leq N,\ q_i = \text{rand}(0,1) \ .$$

**The Step distribution:** Given positive integer $s$ and real step ratio $\alpha \geqslant 1$ the probability vector $\mathbf{p}$ is divided in to $s \geqslant 1$ equal segments. $p_i$ is uniform within the same segment. For adjacent segments, the probabilities in left segment is $\alpha$ times of the probabilities in the right segment.

$$p_i = \frac{\alpha^k}{\sum_{k=1}^{s} \sum_{j=1}^{N/s} \alpha^k},\ 1 \leq i \leq N \ .$$

**Implementation:** For the algorithms of time linear in $N$, there are differences in the multiplicative factor in front of the linear term. For example, SMAWK is using recursion and an upper bound on the number of comparisons is $6DN - 3D(D+1)$ (see Aggarwal et al. [1987]), whereas Boundaries is doing about $2N$ comparisons, because it needs to check the conditions from $B_1$ to $B_N$ and then back to $B_1$. The query function $a_{n,j}^{(d)}$, as defined in Section 2.2 ,is implemented as an *inline*[1] function.

[1] An inline switch in C++ replaces each call to the function with its actual code. This accelerates the execution of the function and increases the executable size. It is often applied to frequently called small functions.

Table 6.1: Competitive ratios of algorithms DQ and Boundaries on four types of data: $D = 10$, $N = 100, 200, \ldots, 1000$

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

### 6.2.2 Experiments and results

**First test:** We test the running time of all optimal algorithms and non-optimal heuristics using all four types of data. We measure the running time with respect to problem scale ($N$). Figure 6.3(a) illustrates the results for the Zipf user location data, where $N$ varies from 100 to 1000. Indeed, the plot coincides with our complexity analysis. We note that similar behavior occurs for all the distributions and for many other parameters. A direct observation is that all our optimal algorithms and heuristics run much faster than the original $\Theta(DN^2)$ optimal algorithm Seq.

Beyond the complexity reduction, we observe two facts. First, that Bin is efficient when $N$ is in a reasonable range (hundreds to thousands). For this range, $\log_2 N$ is comparable to the constant 6 coefficient in the complexity of SMAWK. Moreover Bin only involves simple algorithmic operations and is faster than SMAWK in this test. Only when $N \geq 8\,000$, SMAWK outperforms Bin in our simulation. Second, both FirstLocalMin and SpeedUp are of complexity $DN$. Since FirstLocalMin has a faster row minima operation, it is slightly faster than SpeedUp. Even though theoretically

FirstLocalMin is not optimal, in all of our simulations it found the optimal strategy. Moreover, it is also implementation-wise simpler.

We also note that the running time of SMAWK fluctuates as $N$ grows. This is because during the recursion of SMAWK, the process of eliminating entries differs greatly among different instances of the same length $N$, which affects the actual running time.

We also test the complexity of the three $\Theta(DN)$ algorithms w.r.t. the delay constraint $D$. Figure 6.3(b) illustrates the results for a Zipf distribution where $D$ varies from 2 to 10. The plot coincides with the worst case analysis by showing the linearity of $\Theta(DN)$ algorithms as a function of $D$. For all parameters we checked, SpeedUp outperforms SMAWK, as was expected from the exact details of both algorithms. From now on, we adopt SpeedUp as the optimal implementation.

**Second test:** We test the performance of the two non-optimal heuristics DQ and Boundaries. Both heuristics, as expected, are faster than SpeedUp. This indicates a trade-off between optimality and efficiency. For this set of parameters and for all instances we checked, DQ runs slower than Boundaries. We demonstrate the superiority in performance of DQ to Boundaries in Table 6.1. The table shows the ratio of computed cost over the optimal cost for $D = 10$, where $N$ varies from 100 to 1000 with a step of 100, and for data following the four distributions. We note that the performance of Boundaries heavily relies on the initial partition of cells and the user location distribution because of its limited ability to adjust cells among partitions. Thus, for all tests we run on Boundaries, we try three initial partitions (large-suffix, uniform, doubling) and then pick the best search strategy. We observe that the doubling initial partition performs better on uneven distributions, while the uniform initial partition performs better on even distributions.

In the table, we observe that DQ remarkably outperforms Boundaries on most distributions, but is slightly worse in the uniform random distribution. This is because

for uniform random data, an even initial partition is already very close to optimal. Given an almost optimal initial partition, Boundaries performs very well since it does not need to adjust. Indeed, for uniform random data, oblivious heuristic Uniform already performs close to optimal. The advantage of DQ is more impressive on those inputs that no good oblivious initial partition could be provided for Boundaries.

**Third test:** To demonstrate the trade-off between search cost and delay constraint, we test the actual search cost against the delay constraint $D$ on all four types of data. Figures 6.3(c), 6.3(d), 6.3(e) and 6.3(f) show the results on all four different types of input data.

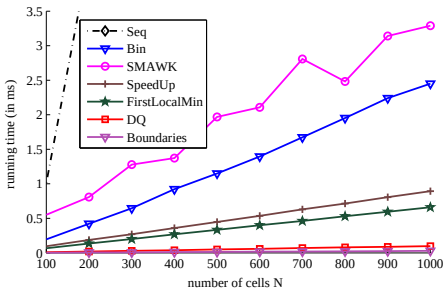
We find that search cost drops remarkably by increasing the delay constraint at small values, but when we increase the delay at larger values the drop in search cost is not as dramatic.

**Fourth test:** Figure 6.4(a) coupled with Figure 6.4(b) depict a clear trade-off between optimality and running time efficiency. Let *linear* class of algorithms be of complexity $\Theta(N)$. We select the best algorithm in each of the following complexity classes: the *above linear* class (SpeedUp), the *almost linear* class (DQ), the linear class (Boundaries) and the *sub-linear* class (Doubling).

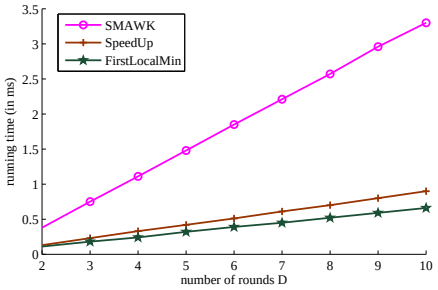
Figure 6.4(a) illustrates the results for the performance and Figure 6.4(b) illustrates the results for the running time, on data following the Zipf distribution. We emphasize that this trade-off between time complexity and performance repeated itself for all the instances tested by our simulations.

**Fifth test:** Finally, we test the performances and running time of the algorithms on real user data. We extract the user location probability vector from the 30 most frequent cells for each of the 996 users as mentioned in Subsection 6.1.1 and normalize it. We measure the search cost divided by the optimal search cost.

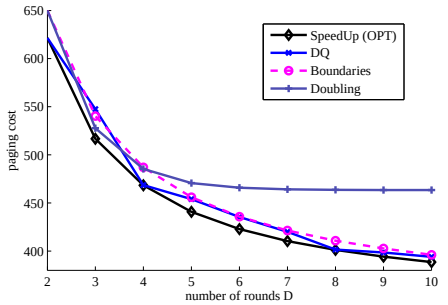
In Table 6.2, we demonstrate the *average*, *worst* and *best* competitive ratio of four classes of non-optimal algorithms. We observe three facts from the table. First,



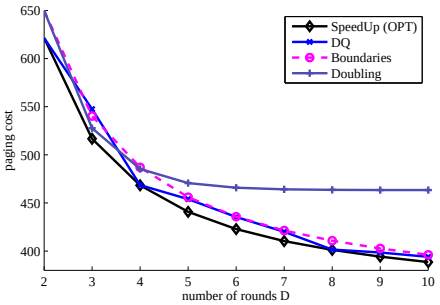

(a) **Running time** versus **number of cells**: $x$-axis: number of cells $N$, $y$-axis: running time; $D = 5$, Zipf, $\alpha = 0.5$.



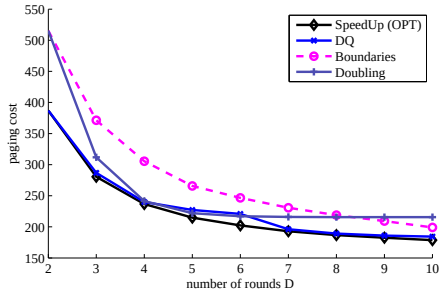

(b) **Running time** versus **number of rounds**: $x$-axis: number of rounds $D$, $y$-axis: running time; $N = 1000$, Zipf, $\alpha = 0.5$.



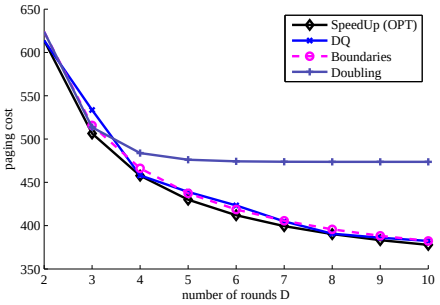

(c) **Cost** versus **delay constraint**: $x$-axis: delay constraint $D$, $y$-axis: search cost; **Zipf** $\alpha = 0.5$, $N = 1000$.




(d) **Cost** versus **delay constraint**: $x$-axis: delay constraint $D$, $y$-axis: search cost; **Gaussian** $\sigma = 0.25N$, $N = 1000$.



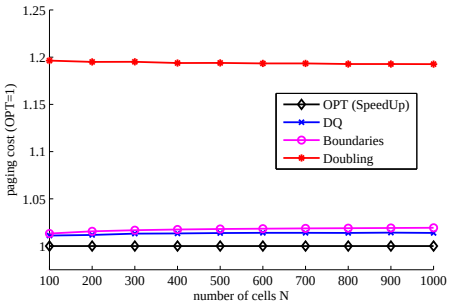

(e) **Cost** versus **delay constraint**: $x$-axis: delay constraint $D$, $y$-axis: search cost; **Step** $s = 10$, $\alpha = 0.5$, $N = 1000$.



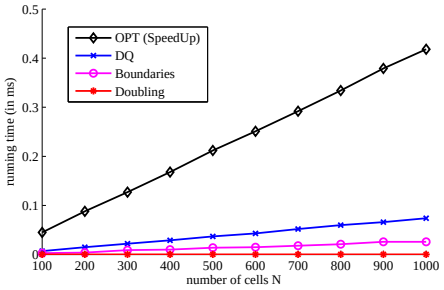

(f) **Cost** versus **delay constraint**: $x$-axis: delay constraint $D$, $y$-axis: search cost; **Uniform Random** $N = 1000$.

Figure 6.3: Optimality, complexity and other properties of different algorithms

FirstLocalMin always generates an optimal search strategy for real data. Second, DQ outperforms the later two heuristics on average optimality in most of the cases. Only when $D$ is comparable to $N$, the two heuristics performs close to DQ, since there is less flexibility on partitioning the cells. Third, DQ has a significantly better worse




(a) **Optimality** hierarchy: Zipf $\alpha = 0.5$ and $D = 10$.




(b) **Complexity** hierarchy: Zipf $\alpha = 0.5$ and $D = 10$.

Figure 6.4: Optimality and complexity hierarchy of four classes of algorithms

case performance than the two faster heuristics. This implies applying DQ in real system offers great optimality and efficiency.

Table 6.2: Competitive ratios of algorithms on real user data

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

In Table 6.3, we present the running time of all algorithms on real user data. We

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

Table 6.3: Running time of algorithms on real user data: $D = 4$, $N = 30$, in milliseconds

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

## 6.3 Paging with congestion cost problem

### 6.3.1 Overview

**Objective:** We conduct simulation on the paging with congestion cost problem in order to experimentally evaluate the performance of our theoretically competitive algorithm and heuristics without guaranteed bound.

**Algorithms and benchmarks:** We implement four algorithms: $\mathcal{S}$ and FRO are being evaluated; OPT and OPT-$N$ are metrics to evaluate algorithm performance.

- FRO: Algorithm FRO uses the dynamic programming scheme in Section 2.2. The most efficient implementation takes $\Theta(ND)$ time.
- $\mathcal{S}$: $\mathcal{S}$ is the greedy algorithm introduced in Chandra and Wong [1975]. It is implemented based on $\mathbf{p}$ without using information on $\mathbf{w}$. It sorts cells by a

non-increasing order of $p_i$ and allocate cells in this order, one at a time, to the partition $d$ with smallest sum of allocated cells $P_d$. Its complexity is $\Theta(ND)$. It is a 49/48-competitive algorithm for typical users.

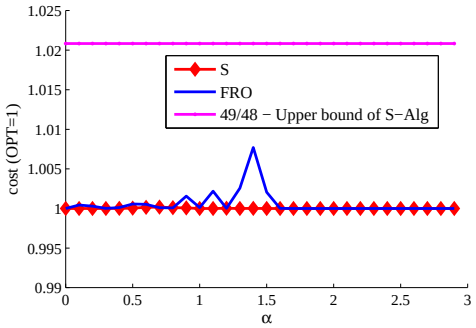
- OPT: Since the problem is NP-Hard, obtaining optimal solution takes exponential time. Our most efficient implementation of OPT computes all permutations of $D$-partition on the $N$ cells and takes $\Theta(D^N)$ time. On current computers, this implementation allows us to run instances up to $N = 20$ and $D = 2$, $N = 15$ and $D = 3$, or in general $D^N \leq 2^{20}$.

- OPT-$N$: For larger values of $D$ and $N$, computing an optimal solution may takes a prohibitively long time. OPT-$N$ is the polynomial time algorithm that optimally pages cells in $D = N$ rounds. According to Corollary 4.7, this can be implemented in $\Theta(N \log N)$ time by sorting cells by $p_i/w_i$. OPT-$N$ gives a lower bound for the cost of OPT.

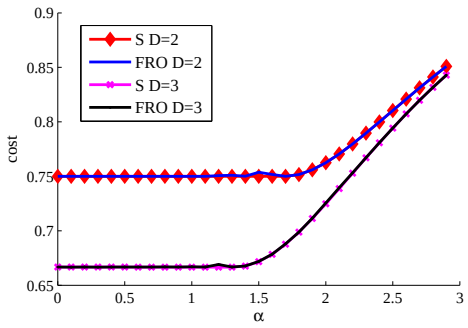
### 6.3.2 Experiments and results

**First test**: We test the performance of $\mathcal{S}$ and FRO on a typical user, that is, $\mathbf{p} = \mathbf{w} \sim \text{Zipf}(\alpha)$. The results are shown in Figure 6.5. We can see that on typical users, both algorithms perform very well, and $\mathcal{S}$ slightly outperforms FRO because it is smarter in breaking ties (of $p_i/w_i$). We run the test on different $N$s and $D$s and the results are very similar. We also run the FRO algorithm with different initial order of cells and the results are similar.

**Second test**: We test the performance of algorithm FRO on uniform random $\mathbf{p}$ and $\mathbf{w}$ for larger number of rounds $D$ to complement our theoretical results. Figure 6.6(a) shows the first result. We can see that FRO performs stably for larger $D$ because the worst-case cost is very close to average cost. We also see that when $D \geq \sqrt{N}$, FRO is almost optimal. This is because when $D$ is larger, there is less flexibility for the



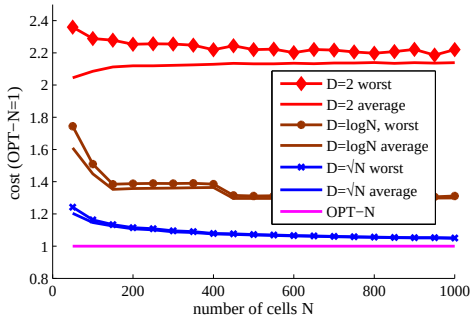

(a) $N = 20$, $D = 2$, $y$-axis: cost (OPT $= 1$)



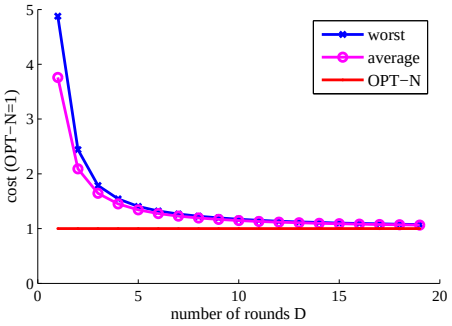

(b) $N = 100$, $y$-axis: cost (absolute value)

Figure 6.5: Performance of $\mathcal{S}$ and FRO on a typical user, $\mathbf{p} = \mathbf{w} \sim \text{Zipf}(\alpha)$, $x$-axis: $\alpha$

optimal solution to not obey the cellwise order. To evaluate the performance of FRO on not very large $D$, we run the test shown in Figure 6.6(b). It indicates that FRO performs reasonably well when compared with OPT-$N$ (recall that when $D = 2$, FRO is a 8/7-approximation). FRO also performs very stably because the worst-case and average costs are very close and the curve is very smooth. We also run this test on random Zipf data. We create random Zipf data by shuffling $\mathbf{w}$ on a typical Zipf user. To do such a shuffle we take a random permutation of the elements of the vector $\mathbf{w}$. The results are very similar.




(a) Average and worst case cost (OPT-$N$ = 1) of algorithm FRO, $N = 50 \ldots 1000$, $D = 2, \log N, \sqrt{N}$, *uniform* random $\mathbf{p}$ and $\mathbf{w}$, 10,000 runs.




(b) Average and worst case cost (OPT-$N$ = 1) of algorithm FRO, $N = 100$, $D = 1 \ldots 20$, uniform random $\mathbf{p}$ and $\mathbf{w}$, 10,000 runs.

Figure 6.6: Tests on uniform data (similar results for Zipf $\mathbf{p}$ and $\mathbf{w}$ after random shuffle)

**Third test**: We test FRO on atypical users. We generate atypical users by randomly

generating a sorted $\mathbf{p}$, reversing its order and randomly generating another sorted $\mathbf{w}$. Both $\mathbf{w}$ and $\mathbf{p}$ are normalized. We test 1,000,000 instances for which ($N = 20$, $D = 2$), and ($N = 15$, $D = 3$) and the solutions are always optimal. This coincides with our theoretical results.

**Fourth test**: Finally, we test FRO for a general user. A general user is a user that follows a Zipf location distribution. However, it may or may not follow the massive behavior. Therefore, we define a general user to be one with $\mathbf{w} \sim \text{Zipf}(\alpha = 0.4429)$ and $\mathbf{p} \sim \text{Zipf}(\alpha' = 0.5)$, while $\mathbf{p}$ and $\mathbf{w}$ are independently shuffled. Lemma 4.23 shows that the FRO solution can be 8/7 times worse than the optimal. We measure the average and worst approximation ratio in 1,000,000 runs and the results are shown in Table 6.3(a). We conclude that for the above random user the FRO algorithm has almost optimal performance on average and worst case performance much better than 8/7.

(a) Approximation ratio FRO, general user, $D = 2$

| metric | $N = 4$ | $N = 5$ |
|---|---|---|
| worst | 1.00868 | 1.00194 |
| average | 1.00199 | 1.00009 |
| metric | $N = 8$ | $N = 10$ |
| worst | 1.00489 | 1.00557 |
| average | 1.00021 | 1.00016 |

(b) Approximation ratio, real user

| Setting | Algo. | Average | Worst |
|---|---|---|---|
| $N = 20$ $D = 2$ | $\mathcal{S}$ | 1.13475 | 9.60367 |
| | FRO | 1.00000 | 1.00001 |
| $N = 15$ $D = 3$ | $\mathcal{S}$ | 1.07308 | 5.94067 |
| | FRO | 1.00000 | 1.00036 |
| $N = 10$ $D = 4$ | $\mathcal{S}$ | 1.01648 | 2.52683 |
| | FRO | 1.00007 | 1.00080 |

Table 6.4: Approximation ratios

We also try to tune up algorithm FRO not only taking consideration the $p_i/w_i$ ratio but also $p_i$ and $w_i$ values. Our preliminary results indicate this does not provide noticeable improvement.

**Real user data:** We test the performance of FRO and $\mathcal{S}$ on real user data. In order to obtain the optimal solution, for each user, we only take their top 20 or 15 most frequent cells. This is justifiable because no user appears more than once beyond their top 20 favorite cells. Vector $\mathbf{p}$ is acquired directly from the user and

**w** is a universal vector. Table 6.3(b). shows the results. We can see that algorithm FRO performs almost optimally with respect to both average and worst-case cost. Algorithm $\mathcal{S}$ performs fine with respect to average cost but poorly for worst-case cost. This indicates there is a certain portion of not-very-typical users, of which some are very atypical.

## 6.4 Paging multiple users problem

### 6.4.1 Overview

**Objective:** We set up the experiment study of paging multiple users problem in order to: 1. Evaluate the actual running time of our algorithm on real computers; 2. Evaluate the performance of our 12 heuristics on different types of data.

**Implementation:** The implementation of paging multiple users problem is substantial and complicated. We describe the general results in Subsection 6.4.2. We describe the detailed setup and implementation in Subsection 6.4.3.

### 6.4.2 Experiments and results

**Select the best heuristics:**

We test our heuristics for both the yellow page and the conference call problem. For each problem, we check arbitrary tokens and disjoint tokens. We also conduct our simulation on small instances and large instances with respect to number of boxes. For small instances, we try all possible inputs (exhaustive search) up to some granularity (values of probabilities are integer multiples of some small value), then we compute strategies for Zipf and Uniform distributed user location data. We check the cases of $D = 2$ and $D = N$ (for efficient optimal algorithms). We compare the search costs between different greedy heuristics, and between greedy heuristics and OPT. For large instances, we conduct simulation on Zipf and Uniform distributed

Table 6.5: Performance ranking of heuristics. $\mathcal{G} > \mathcal{H}$: heuristic $\mathcal{G}$ is consistently better than $\mathcal{H}$; $\mathcal{G} \geq \mathcal{H}$: $\mathcal{G}$ is no worse than $\mathcal{H}$ (sometimes better, sometimes comparable); $\mathcal{G} \sim \mathcal{H}$: $\mathcal{G}$ is comparable to $\mathcal{H}$.

| Yellow Page | Conference Call |
|---|---|
| $Y \geq S > Z \geq X$ | $Y \geq S > Z \geq X$ |
| $BFY > Y \geq WLY$ | $Y > BFY \sim WLY$ |

data, for several values of $D$, and then we compare the search costs between the heuristics. We measure the average and worst case ratio of search costs between each pair of heuristics, and between each heuristic and OPT, for applicable instances in each group of test with a combination of parameters and for real data. We use a kind of "voting system" to generate a ranking of heuristics. We describe details of the setup and results of our experiments in Subsection 6.4.3.

Our results show that the ranking of heuristics is the same for large and small instances, when doing exhaustive search, for Zipf and uniform data, for average case and worst case measurement, and when comparing heuristics among themselves or when comparing heuristics with OPT. Table 6.5 shows the ranking of heuristics and of the three versions of the best heuristic ($Y$, $BFY$, $WLY$) for the two problems.

**Performances of best greedy heuristics:** We evaluate the performances of the best heuristics ($BFY$ for yellow page and $Y$ for conference call) on small instances, large instances. We measure their average case and worst case cost ratios over OPT for small instances and real data. We measure the average and worst case cost ratios of other heuristics over $BFY$ or $Y$ for large instances.

Tables 6.6(a), 6.6(c) and 6.6(d) show the results. We observe that our selected heuristics perform well in the worst case and average performance for all types of input data. We also present the results in bar chart in Fig. 6.8.

**Simulation on real data:** We have no knowledge about affiliations among users (so as to make them a group) because of privacy constraint. Thus, we randomly pick two ($M = 2$), three ($M = 3$), and four ($M = 4$) users and compute the probabilities

Table 6.6: Cost ratios of $BFY$ for yellow page (YP) and $Y$ for conference call (CC)

(a) Cost ratios over OPT: small instances

| Problem | Average | Worst |
|---|---|---|
| YP (BFY) | 1.00638 | 1.19415 |
| CC (Y) | 1.00173 | 1.03609 |

(b) Cost ratios over OPT: real data

| Problem | Average | Worst |
|---|---|---|
| YP (BFY) | 1.03352 | 1.54384 |
| CC (Y) | 1.00643 | 1.05930 |

(c) Cost ratios over $BFY$: large instances, YP

| Heuristic | Average | Worst |
|---|---|---|
| $BFX$ | 1.19102 | 2.23126 |
| $BFZ$ | 1.28360 | 2.51777 |
| $BFS$ | 1.00003 | 1.00961 |

(d) Cost ratios over $Y$: large instances, CC

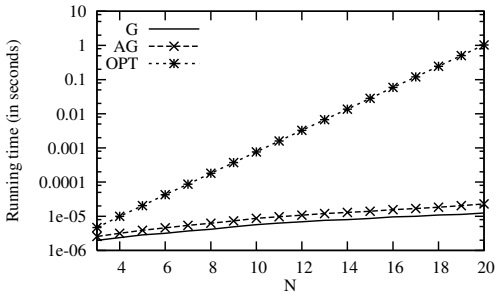
| Heuristic | Average | Worst |
|---|---|---|
| $X$ | 1.15884 | 1.62346 |
| $Z$ | 1.00626 | 1.03000 |
| $S$ | 1.00000 | 1.00316 |

$p_{m,n}$s. For each $M = 2, 3, 4$, we pick 10,000 instances as above. We conduct the same simulations as in the non-real data. We observe that each real user is close to Zipf distributed and users are almost disjoint. The results coincide with the results from non-real data as shown in Tables. 6.5 and 6.6(b).

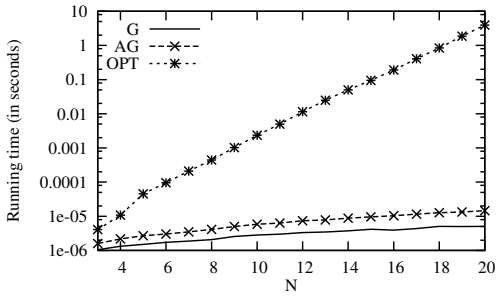
**Running time:** We compare the running time of our greedy heuristics and efficient optimal algorithms. For accurate time measurement, for each $N$, we run our algorithms for many iterations and compute the average running times.

For $D = 2$, we measure the running time of the static version of the greedy heuristics $G$ (i.e., $Y$) of complexity $\Theta(MDN \log N)$, the adaptive version of the greedy heuristics $AG$ (i.e., $BFY$ and $WLY$) of complexity $\Theta(MDN^2 \log N)$, and the $D^N$ optimal algorithm of complexity $\Theta(MND^N)$. The result is in Figure 6.7(a) For $D = N$, we test the running time of the static greedy heuristic $G$, the adaptive greedy heuristic $AG$, and the fast optimal algorithm of complexity $\Theta(MN2^N)$. The result is in Figure 6.7(b) In the two experiments, we observe a complexity hierarchy of the algorithms and a tradeoff between complexity and optimality. For $D = N$, we also compare the running time of the straight forward optimal $N!$ algorithm and fast $2^N$ algorithm. The result is in Fig. 6.7(c). As expected, the $N!$ algorithm is super exponential and the $2^N$ algorithm is comparably much faster, which allows us

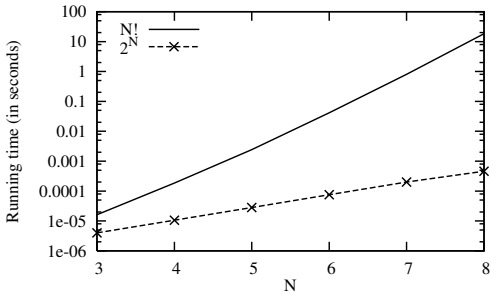
do experiments on larger problem instances.




(a) $D = 2$, heuristics and optimal algorithm




(b) $D = N$, heuristics and optimal algorithm




(c) $D = N$, two optimal algorithms

Figure 6.7: Running time

### 6.4.3 Detailed experiment setup*

**Algorithms structure:** Input: Probability matrix $P_{M\times N}$, delay constraint $D$. Output: Paging cost and paging strategy array $part[N]$, where $part[n]$ is the part index of box $B_n$.

**Algorithms implemented:** 1) OPT: A simple optimal algorithm for arbitrary users on any $D$: It tests all assignments of cells $\langle C_1, \ldots, C_N\rangle$ to parts $\langle A_1, \ldots, A_D\rangle$ and selects the best. The complexity is $\Theta(MN \cdot D^N)$. For $D = 2$, its complexity is $\Theta(MN \cdot 2^N)$. 2) FAST OPT: Algorithms 5.3 and 5.4. Optimal algorithms for arbitrary users when $D = N$. Their complexity is $\Theta(MN \cdot 2^N)$. 3) DISJOINT OPT: Optimal algorithm for disjoint users when $D = N$. 4) All 12 greedy heuristics for $1 \leq D \leq N$.

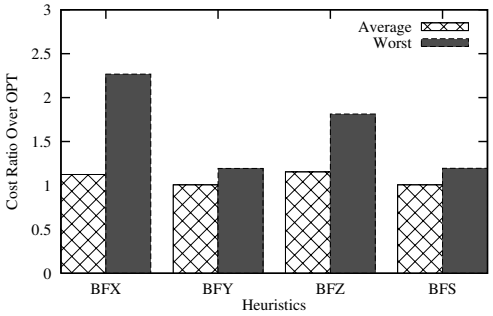
Their general complexity is $O(MD \cdot N^4)$. When $D = N$ their complexity becomes $\Theta(MN^2)$. The static version of heuristics has complexity $\Theta(MDN^2)$ when $D < N$.

**User Types:** 1) Regular users: directly generate $p_{m,n}$s according to data type. 2) Disjoint users: select non-zero entries $p_{m,n}$s for each user such that all users have (almost) same number of non-zero $p_{m,n}$s, then generate $p_{m,n}$ values within non-zero entries of each user.

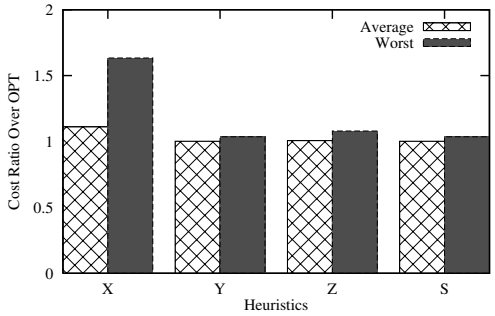
**Instance scale:** 1) Small instances: For Zipf and random user location data, we select $M = 2, 3, 4, 5$ and $N = 10, 12, 14, 16$, $D = 2$ and $D = N$. 2) Large instances: For Zipf and random user location data, we select $M = 2, 3, 4, 5$ and $N = 10, 20, \ldots, 100$, $D = 2, 4, 8, \ldots, 2^{\lfloor \log_2 N \rfloor}$.

**Data type:** 1) EXT: Exhaustive search on small instances: for each set of parameters $\{M, N\}$ and granularity $K$, we enumerate all possible $p_{m,n}$s in increment of $1/k$, where $2 \leq k \leq K$. The number of instances is $\Theta((K^2)^{MN})$. We choose $N = 3, 4, 5, 6$, $M = 2$ and $K = 13$ so that the experiments can be completed in a reasonable time frame (hours to days). 2) RANDOM: Random data: for each set of parameters $\{M, N\}$, we generate 1000 instances of random data, such that $p_{m,n} \sim \mathrm{U}(0, 1)$. We normalize $p_{m,n}$s such that $\sum_{n=1}^{N} p_{m,n} = 1$ for $1 \leq m \leq M$. 3) ZIPF: Zipf data: Empirical study Buchanan [2008] on user location distribution shows that $p_{m,n}$ follows *Zipf* distribution, i.e., $p_{m,i} = i^{-\alpha} / \sum_{n=1}^{N} n^{-\alpha}$. Zipf distribution is a power-law distribution and $\alpha \geq 0$ is its parameter. When $\alpha = 0$, Zipf distribution is uniform. As $\alpha$ grows, it becomes more uneven. Using the method of least squares on real data, we acquire the following estimate of the parameter $\alpha = 0.4429$. We generate Zipf data $p_{m,n}$s for each user $m$, and randomly shuffle for each user (otherwise we would just create monotonic users). For each set of parameters $\{N, M, \alpha\}$, we generate 1000 instances. We select $\alpha = 0.25, 0.4429, 0.5, 0.75, 1.0$.

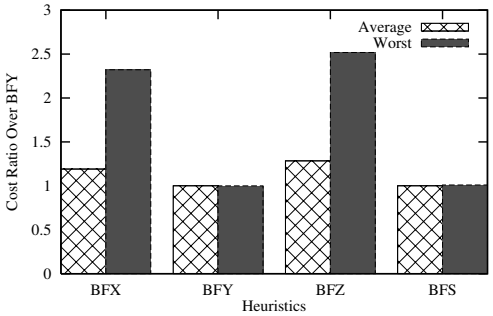
**Variation:** We set up a group of instances for each combination of the following seven criteria:



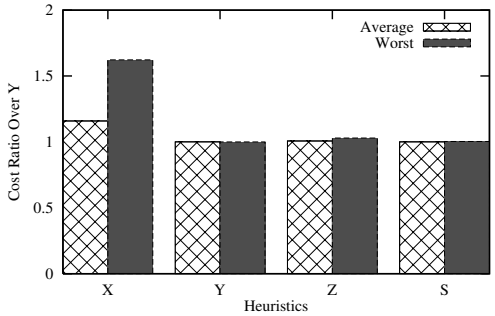

(a) Cost ratios over OPT: small instances, YP



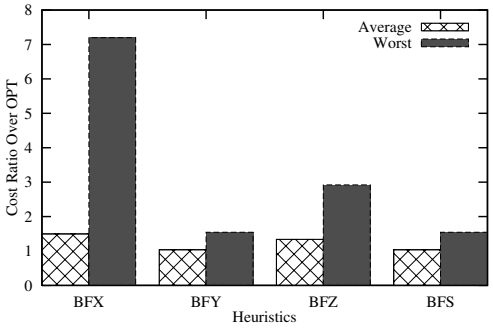

(b) Cost ratios over OPT: small instances, CC



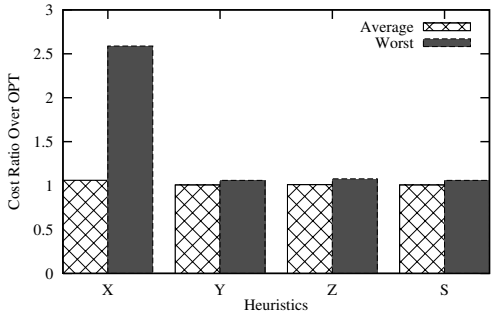

(c) Cost ratios over $BFY$: large instances, YP




(d) Cost ratios over $Y$: large instances, CC




(e) Cost ratios over OPT: real data, YP




(f) Cost ratios over OPT: real data, CC

Figure 6.8: Cost ratios of greedy heuristics

- Problem: yellow page versus conference call
- User type: regular users versus disjoint users
- Instance scale: large versus small

- Data: EXT versus RANDOM versus ZIPF
- Delay constraint: $D = 2$ versus $D = N$ for small instances only
- Measurement: worst case (lower bound) versus average ratios
- Evaluation criteria: between heuristics versus between heuristic and OPT (to be explained next).

**Example of a group of tests:** For the *yellow page* problem, we create 1,000 *small* random instances ($N = 10, 12, 14, 16$, $M = 2, 3, 4, 5$) of *regular users* with probabilities that follow *Zipf* distribution. We measure the *average* cost ratios between each *pairs of heuristics*. We examined all paths of combination from top level to bottom level in Figure 6.9.

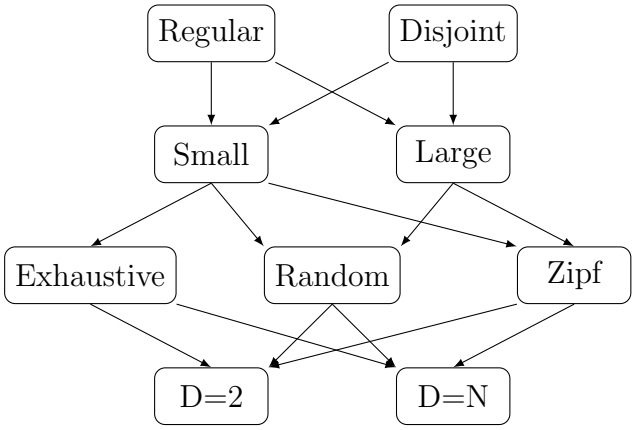



Figure 6.9: Experiment data: every path in the graph

**Measurements:** 1) For each instance in each group of data, we measure the cost ratio of one heuristic over another heuristic, as well as each heuristic over OPT. We have $12 \times 12$ such ratios for each instance ($12 \times 11$ between pairs of heuristics and 12 between all heuristics and OPT). 2) For each group of instances, we measure the worst case and average case of ratios.

**Evaluation Criteria:** 1) Between heuristics: Let $\mathcal{G}$ and $\mathcal{H}$ be two greedy heuristics. Let $\rho_{\mathcal{G}/\mathcal{H}}$ be the cost ratio of $\mathcal{G}$ over $\mathcal{H}$ on a group of instances; let $\rho_{\mathcal{H}/\mathcal{G}}$ be the

ratio of $\mathcal{H}$ over $\mathcal{G}$ on the same group of instances. Define the competitive factors $\varepsilon_{\mathcal{G}/\mathcal{H}} = \rho_{\mathcal{G}/\mathcal{H}} - 1$ and $\varepsilon_{\mathcal{H}/\mathcal{G}} = \rho_{\mathcal{H}/\mathcal{G}} - 1$. Let $\mu \in [2, 4]$ be a constant that we will choose and that will be used to distinguish performances of different algorithms. If $\varepsilon_{\mathcal{G}/\mathcal{H}}$ and $\varepsilon_{\mathcal{H}/\mathcal{G}}$ are of same sign and $1/\mu \leq \varepsilon_{\mathcal{G}/\mathcal{H}}/\varepsilon_{\mathcal{H}/\mathcal{G}} \leq \mu$, we say heuristics $\mathcal{G}$ and $\mathcal{H}$ have comparable performance. Otherwise, if $\varepsilon_{\mathcal{G}/\mathcal{H}} < \varepsilon_{\mathcal{H}/\mathcal{G}}$, we say $\mathcal{G}$ outperforms $\mathcal{H}$. 2) Between a heuristic and optimal: Let $\mathcal{G}$ be a heuristic. Let $\rho_{\mathcal{G}} \geq 1$ be the cost ratio of $\mathcal{G}$ over OPT. Define competitive factor $\varepsilon_{\mathcal{G}} = \rho_{\mathcal{G}} - 1$. For any heuristics $\mathcal{G}$ and $\mathcal{H}$, we say their performances are comparable if $1/\mu \leq \varepsilon_{\mathcal{G}}/\varepsilon_{\mathcal{H}} \leq \mu$; otherwise, we say $\mathcal{G}$ outperforms $\mathcal{H}$ if $\varepsilon_{\mathcal{G}}/\varepsilon_{\mathcal{H}} < 1/\mu$. 3) $\mu = 2, 3, 4$.

We observe that the two evaluation criteria above coincide very well in our experiments. The disagreement rate varies from 2.78% to 26.7%, with a mean of 9.37%. Varying $\mu$ from 2 to 4 does not remarkably affect results.

**Statistics Collection:** For each group of instances, we create a $12 \times 12$ table that records the superiority of one heuristic over another and OPT for each variation of tests. Each table entry can be of three values, $-1$, 0 and 1 denoting the row heuristic is worse, comparable or better than the column heuristic, respectively. The value is determined by both evaluation criteria.

We measure the superiority among four groups of heuristics: $X$, $Y$, $Z$ and $S$ in their corresponding versions (e.g., $WLX$, $WLX$, $WLZ$ and $WLS$), and among the three versions within each group (e.g., $Y$, $BFY$ and $WLY$).

We conduct a voting on the corresponding table entries of one heuristic over another. If heuristic $\mathcal{G}$ has a 2/3 majority of superiority over $\mathcal{H}$, we denote this by $\mathcal{G} \geq \mathcal{H}$. If heuristic $\mathcal{G}$ has a 100% majority of superiority over $\mathcal{H}$, we denote this by $\mathcal{G} > \mathcal{H}$. Otherwise, we say $\mathcal{G} \sim \mathcal{H}$

**Results:** We first observe that a few factors do not affect the performance of heuristics; these are: delay constraint: $D = 2$ or $D = N$; Data type: EXT, RANDOM OR ZIPF; Measurement: average or worst case; Evaluation criteria: between heuristics

or between a heuristic and OPT.

A summary of the results is shown in Table 6.5.

Besides Table 6.5, we observe a few facts: 1) All heuristics perform quite well, on average with ratios of less than 1.15 competitive to OPT. In the worst case ratios, only heuristics $X$, $BFX$ and $WLX$ perform badly with a lower bound ratio of $N-1$ for the yellow page problem and $(N+1)/2$ for the conference call problem. All other worst case ratios are less than 2. For the conference call problem we record better cost ratios than for the yellow page problem because the latter has smaller value in paging cost for the same instance. 2) The average case and worst case performance of heuristics coincides with each other, which suggests that their performance is consistent and stable.

# Chapter 7

# Conclusion and open problems

In this work, we explore problem of searching for mobile tokens in boxes with statistical knowledge of their locations. Table 7.1 summarized the major theoretical results before and after this work.

We also conduct comprehensive simulation on all three problems on real and synthetic data to examine: i) experimental running time of the algorithms; ii) experimental performances of the algorithms; iii) behavior of the algorithms.

We have discussed open problems in related chapters independently (Sections 3.4, 4.7, and 5.7). We first repeat a few important open problems that worth future effort:

- In the basic search problem, we look for both faster optimal algorithms and better non-optimal heuristics
- In the search with cost problem, we look for simple approximation algorithms
- In the searching for multiple tokens problem, we want determine the hardness of the yellow page problem, and provide approximation algorithms for the yellow page problem

| Problem studied | Knowledge before this work | Knowledge after this work |
|---|---|---|
| The basic search problem | $\Theta(D \cdot N^2)$ optimal algorithms<br><br>$\Theta(N)$ Heuristic | Two $\Theta(D \cdot N)$<br>one $\Theta(D \cdot N \log N)$ optimal algorithms<br>a better $\Theta(N \log D)$ heuristic<br>A hierarchy of algorithms with<br>tradeoff between optimality and complexity |
| Search with cost problem | None | strong NP-hardness<br>8/7-approximation for $D = 2$<br>PTAS for constant $D$<br>Optimal algorithms for special cases |
| Searching for multi-token problem | CC is NP-hard<br>APX and PTAS for CC<br>Factorial time OPT | Duality between CC and YP<br>P-time OPT for special token types<br>Efficient exponential time OPT |

Table 7.1: Major results developed in this work

In a more general framework, we describe the problem in the following paradigm.

- $N$ boxes, $M$ tokens, $D$ rounds
- Look for $k$ out of the $M$ tokens ($1 \leqslant k \leqslant M$)
- Probabilities $\mathbf{p}_{M\times N}$ and unlocking cost $\mathbf{w}_{1\times N}$
- Minimize the expected search cost

In this generalized framework, we describe a few working directions:

In certain applications, probabilities $\mathbf{p}$ and costs $\mathbf{w}$ are either frequently changing, or learned through the process of searching. We look for search algorithms, such that the amortized search cost is minimized after a number of search expectation. Preliminary work has been conducted in Bar-Noy and Mansour [2004] and Gau and Haas [2004].

Sometimes, the exact values of probabilities $\mathbf{p}$ and costs $\mathbf{w}$ may not be known because of privacy issues. Partial knowledge, e.g., the order of $p_n$s in $\mathbf{p}$ is known. How to compute search strategies utilizing such partial knowledge are to be explored. Preliminary work on the basic search problem has been presented in Bar-Noy and Klukowska [2007].

Throughout this work, we have demonstrated tradeoffs among optimality, time complexity, and implementation complexity. We look for better tradeoffs among the three factors. That is, keeping some factor(s) unchanged, improve the other factor(s).

Throughout this work, we have assumed the probabilities and costs are independent both box-wise and token-wise. This may not be true in some real applications. We look for algorithms that solve the problems while the probabilities and costs are box-wise and / or token-wise dependant. With such dependency, the problems may become harder or easier.

# Bibliography


Aggarwal, Alok, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2: 195–208, 1987.

Akyildiz, Ian F., Janise Mcnair, Joseph Ho, Huseyin Uzunalioglu, and Wenye Wang. Mobility management in next-generation wireless systems. In *Proc. IEEE*, pages 1347–1384, 1999.

Alon, Noga, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *J. Scheduling*, 1(1):55–66, 1998.

Awerbuch, Baruch and David Peleg. Online tracking of mobile users. *J. ACM*, 42(5): 1021–1058, 1995.

Bar-Noy, Amotz and Joanna Klukowska. Finding mobile data: efficiency vs. location inaccuracy. In *Proc. 15th Annual European Symposium on Algorithms (ESA)*, pages 111–122, 2007.

Bar-Noy, Amotz and Grzegorz Malewicz. Establishing wireless conference calls under delay constraints. *J. Algorithms*, 51(2):145–169, 2004.

Bar-Noy, Amotz and Yishay Mansour. Competitive on-line paging strategies for mobile users under delay constraints. In *Proc. 23rd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 256–265, 2004.

Bar-Noy, Amotz and Zohar Naor. Efficient multicast search under delay and bandwidth constraints. *Wireless Netw.*, 12(6):747–757, 2006.

Bar-Noy, Amotz, Ilan Kessler, and Moshe Sidi. Mobile users: To uptdate or not to update? In *Proc. 13th IEEE Conference on Computer Communications (INFOCOM)*, pages 570–576, 1994.

Bar-Noy, Amotz, Yi Feng, and Mordecai J. Golin. Paging mobile users efficiently and optimally. In *Proc. 26th Annual IEEE Conference on Computer Communications (INFOCOM)*, pages 1910–1918, 2007.

Bar-Noy, Amotz, Panagiotis Cheilaris, and Yi Feng. Paging multiple users in cellular network: Yellow page and conference call problems. In *Proc. 9th International Symposium on Experimental Algorithms (SEA)*, pages 361–372, 2010a.

Bar-Noy, Amotz, Panagiotis Cheilaris, Yi Feng, and Asaf Levin. Finding mobile data under delay constraints with searching costs. In *Proc. 29th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 297–304, 2010b.

Buchanan, Mark. Ecological modelling: the mathematical mirror to animal nature. *Nature*, 453:714–716, 2008.

Burkard, Rainer E., Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discrete Appl. Math.*, 70(2):95–161, 1996.

Chandra, Ashok K. and C. K. Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM J. Comput.*, 4(3):249–263, 1975.

Chang, Nicholas B. and Mingyan Liu. Revisiting the TTL-based controlled flooding search: optimality and randomization. In *Proc. 10th Annual International Conference on Mobile Computing and Networking (MOBICOM)*, pages 85–99, 2004.

Cody, R. A. and E. G. Coffman. Record allocation for minimizing expected retrieval costs on drum-like storage devices. *J. ACM*, 23(1):103–115, 1976.

Cohen, Edith, Amos Fiat, and Haim Kaplan. Efficient sequences of trials. In *Proc. 14th Annual ACM-SIAM Symposium of Discrete Algorithms (SODA)*, pages 737–746, 2003.

Epstein, Leah and Asaf Levin. The conference call search problem in wireless networks. *Theor. Comput. Sci.*, 359(1-3):418–429, 2006.

Epstein, Leah and Asaf Levin. A PTAS for delay minimization in establishing wireless conference calls. *Discrete Optimization*, 5(1):88–96, 2008.

Fleischer, Rudolf, Mordecai J. Golin, and Yan Zhang. Online maintenance of k-medians and k-covers on a line. *Algorithmica*, 45(4):549–567, 2006.

Gau, Rung-Hung and Zygmunt J. Haas. Concurrent search of mobile users in cellular networks. *IEEE/ACM Trans. Netw.*, 12(1):117–130, 2004.

Goodman, David J., P. Krishnan, and Binay Sugla. Minimizing queuing delays and number of messages in mobile phone location. *Mobile Netw. and Appl.*, 1(1):39–48, 1996.

Kaplan, Haim, Eyal Kushilevitz, and Yishay Mansour. Learning with attribute costs. In *Proc. 37th ACM Symposium on Theory of Computing (STOC)*, pages 356–365, 2005.

Krishnamachari, Bhaskar, Rung-Hung Gau, Stephen B. Wicker, and Zygmunt J. Haas. Optimal sequential paging in cellular wireless networks. *Wireless Netw.*, 10 (2):121–131, 2004.

Lenstra, Jan Karel, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

Lyberopoulos, G. L., J. G. Markoulidakis, D V. Polymeros, D. F. Tsirkas, and E. D. Sykas. Intelligent paging strategies for third generation mobile telecommunication systems. *IEEE Trans. Veh. Tech.*, 44(3):543–553, 1995.

Madhavapeddy, Seshu, Kalyan Basu, and Allison Roberts. *Adaptive paging algorithms for cellular systems*, volume 1, pages 83–101. Kluwer Academic Publishers, Norwell, MA, USA, 1996. ISBN 0-7923-9694-4.

Matousek, Jiri and Bernd Gärtner. *Understanding and Using Linear Programming.* Springer, 2006.

Mullender, Sape J. and Paul M. B. Vitányi. Distributed match-making. *Algorithmica*, 3:367–391, 1988.

Rose, Christopher. State-based paging/registration: a greedy technique. *IEEE Trans. Veh. Tech.*, 48(1):166–173, January 1999.

Rose, Christopher and Roy D. Yates. Minimizing the average cost of paging under delay constraints. *Wireless Netw.*, 1(2):211–219, 1995.

Rose, Christopher and Roy D. Yates. Ensemble polling strategies for increased paging capacity in mobile communication networks. *Wireless Netw.*, 3(2):159–167, 1997.

*3GPP Tech. Spec. 23.012 Location management procedures, version 8.2.0, Section 2.* TSG/WG, 2009.

Vazirani, Vijay V. *Approximation Algorithms.* Springer, 2001.

Wang, Wenye and Guoliang Xue. A cost-minimization algorithm for fast location tracking in mobile wireless networks. *Comput. Netw.*, 50(15):2713–2726, 2006.

Wang, Wenye, Ian F. Akyildiz, and Gordon L. Stüber. An optimal paging scheme for minimizing signaling costs under delay bounds. *IEEE Commun. Lett.*, 5(2):43–45, 2001.

Weitzman, Martin L. Optimal search for the best alternative. *Econometrica*, 47(3): 641–654, 1979.