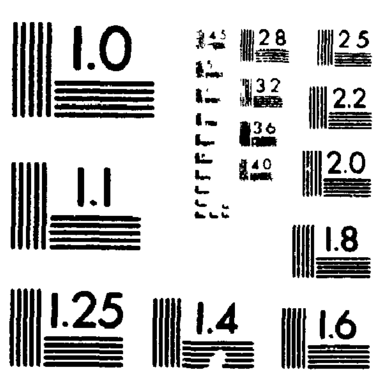
1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT



PRECISION℠ RESOLUTION TARGETS

Efficient Dynamic and On-line Computation with Applications

by

Anthony Spatharis, B. Sc. & M. Sc.

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements for the degree of

Master of Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

May 17, 1995

ISBN 0-612-09016-7

Name ANTHONY SPATHARIS

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided

COMPUTER SCIENCE — 0 9 8 4 — U·M·I

SUBJECT TERM — SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523
Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578
Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996
Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575
Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463

Applied Sciences

Applied Mechanics	0346
Computer Science	0984
Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451

The undersigned hereby recommend to

the Faculty of Graduate Studies and Research

acceptance of the thesis:

Efficient Dynamic and On-line Computation
with Applications

submitted by

Anthony Spatharis, B. Sc. & M. Sc.

in partial fulfillment of the requirements

for the degree of Master of Science

Director, School of Computer Science

Thesis Supervisor

Carleton University

May 17, 1995

Abstract

The best effect of any thesis is that

it excites the reader to self activity.

Thomas Carlyle

Dynamic and *on-line computation* have generated a challenging and theoretically interesting area of research with a wide variety of on-line applications in relevant fields of Computer Science. *Dynamic and on-line algorithms are* concerned with updating the output to a problem as the input is changed incrementally. We use *competitive analysis* to measure the efficiency of an on-line algorithm with respect to the performance of the optimal off-line algorithm.

This thesis studies the design and analysis of efficient on-line algorithms for several combinatorial problems: *list update, paging, weighted caching, the k-server problems, graph coloring* and *on-line matching*. We also consider some specific distributed and geometric computations in on-line setting.

The *goal* of this research is to demonstrate variations of the standard on-line models and develop robust on-line algorithms based on the generalized on-line frameworks using competitive analysis. It is hoped that the maturity of the *theory of on-line algorithms* and the cross-fertilization of dynamic and on-line computation will help in bridging the gap between *theory and practice* in the field of *computer algorithms*.

As for me, all I know is that I know nothing.

Socrates

Acknowledgments

I am deeply grateful to my supervisor Evangelos Kranakis for his guidance and support. He has helped me to begin my research and I might not have finished it without his kindness and trust. Simply, I would like to say that he is a great teacher and a wonderful person! I feel very fortunate to have had him as an advisor.

I would also like to express my gratitude to my thesis defense committee, professors D. Krizanc and J. Urrutia, for their valuable questions, helpful suggestions and useful comments.

I thank all my professors for what they taught me and the administration of the School of Computer Science for their help during my study at Carleton. Several colleagues have been helpful to me during the preparation of this research. Especially, I wish to thank Govindachari Raghunath and Jelber Shirabad for their technical help and assistance on the computers; their knowledge, moral support and friendship were much appreciated. I would also like to acknowledge my office-mate HongLing Chen, who has been supportive.

I am fortunate to have wonderful family and friends who have always been there when I needed their support during this undertaking.

Finally, I thank my loving parents, Spiros and Artemis, who live in the beautiful island of Santorini and who have a crucial influence in my life. Moreover, the tiny "Christoulaki" and the memory of my grandfather, Marcos, have been always in my heart.

Dedication

To my loving parents

and to my dear teachers.

I owe my life to my parents and

my good life to my teachers.

Aristotle

- — *Connectivity* [79,132],
- — *Spanning trees* and *forests* [119,147,149,160,319],
- — *Shortest paths* [17,18,80,187],
- — *Biconnected* and *triconnected components* [119,187,289,340],
- — *Transitive closure* or *reachability* [185,186],
- — *Planarity testing* [118,128,129,130,321,322];

- *Computational Geometry* [266,271,281];
- *Data bases* [1];
- *Syntax-directed editors* and *grammars* [293,294,295,298];
- *Data-flow analysis* [13,53,298]; and
- *Code generation* and *optimization* [187].

There have been parallel incremental algorithms for *minimum spanning trees* and *connected components* [275]. Also, a beautiful research on dynamic data structures and algorithms for graphs can be found in [93,187].

2.1.2 On-line Algorithms versus Off-line Algorithms

An *on-line algorithm* is one that receives a sequence of *requests*, and performs an irrevocable *answer* (*action*) in response to each request before the next request arrives. Each sequence of requests and corresponding actions have an associated *cost*.

Aho, Hopcroft and *Ullman* ([3], pp. 109) define *on-line execution*, for an input sequence **r**, as follows:

> **Definition.** The *on-line execution* of **r** requires that the instructions in **r** be executed from left to right, executing the *i*th instruction in **r** without looking at any following instructions. The *off-line* execution of **r** permits all the **r** to be scanned before answers need to be produced.

List of Figures

List of Tables

Chapter 1

Introduction

ΤΑ ΠΑΝΤΑ ΡΕΙ
Everything Changes

HERACLITUS

The subject of this research is on-line computation with dynamic or changing input data.

1.1 Dynamic Algorithms and Data Structures

The development of *dynamic algorithms* and *data structures* is a fruitful and challenging area that has achieved a great deal of attention in last years. *Dynamic computation* involves updating the solution to a problem when the input changes incrementally. This has generated many new algorithms and data structures for solving dynamic problems efficiently. *Dynamic algorithms* have been considered where a sequence of update and query operations are performed over time and each operation has to be completed before beginning the next.

The *objective* of an efficient dynamic algorithm is to obtain considerable savings over recalculating the solution from scratch. There has been a lot of research especially in the field of graph algorithms motivated by many important applications in *network optimization, VLSI layout, distributed computing and computational geometry*.

1.2 On-line Algorithms

The study of dynamic algorithms is relatively new and hence there is no standard definition in what are known as *dynamic, incremental, update*, or *on-line algorithms*. All above terms refer to algorithms in which a solution is maintained or modified as a result of an incremental change of the input data. Researchers have used variant definitions which were internally restrictive, mostly considering only "atomic" changes, not sequences. In particular, some algorithms were analyzed for numerous updates, while others allowed only one "atomic" change.

In this research, we attempt to carefully define the exact meaning of "*incremental*" and "*on-line*" algorithms. These definitions are based on those in [52,187] and [255,207], respectively. We hope that the presented categorization will be a useful start at understanding its modification in many problems. Our effort has been focused on efficient on-line algorithms, where partial information of the input data is assumed. We leave the more general study of *fully incremental* computation for future work.

There are several important reasons for studying and searching on-line algorithms:

- On-line computation corresponds naturally to the real life situation, where *the future is unknown.*

- On-line algorithms nicely complement many well-studied frameworks of the algorithmic theory (i.e., the dynamic and highly recursive computation).

- The analysis of on-line algorithms forms an elegant model for measuring the performance of algorithms with partial or incomplete access to the input data.

- The theory of on-line algorithms leads towards further research for a unified measure of complexity theory.

1.3 On-line Problems and Applications

A computational problem is said to be *on-line* if it is required to make irrevocable decisions about the output without complete information of the entire input. On-line algorithms attempt to model a real life situation, where the entire input is not known in advance and it is obtained incrementally.

Many problems in computer science are inherently on-line in nature and therefore there have been a lot of on-line applications in relevant fields. Indeed, several advanced computer applications arise such as *real-time manufacturing systems*, *man-machine interfaces*, *robot navigation* and *computer graphics*. Typical applications of on-line algorithms include *resource allocation* in parallel and distributed computer systems, the *stock market*, *bin packing*, *cache management*, *file migration*, *scheduling*, routing, *maintenance of data structures and databases*, *communication networks* and so on. In all these areas and especially for on-line solutions of combinatorial problems, interestingly beautiful mathematical arguments have yielded lower and upper bounds on their complexity.

1.4 Analysis of On-line Algorithms

A fundamental problem of interest in computer science is the analysis of algorithms with the intention of designing an efficient solution of a computational problem. We are interested in making good decisions in on-line computation and find an efficient solution based on the fact that each part of the solution is obtained without *a priori* knowledge of the entire input.

One usual and standard way of solving on-line problems is to re-compute their solution from scratch after each input change (the *off-line* approach). Unfortunately, this is often computationally expensive.

There are more efficient and general approaches (called *incremental* approaches) to maintain some information between subsequent updates so as to react quickly in response to input changes. Many new algorithms and data structures for solving efficiently dynamic graph problems have been generated. Also, some efforts to analyze the concept of incremental computation from a theoretical complexity point of view have been developed [187].

In order to analyze the performance of on-line algorithms, some formal theoretical model is necessary. Traditional worst-case complexity usually fails here, since any algorithm will have an input that gives arbitrarily poor performance for many on-line problems. For example, *List Update* [173,312] and *Paging problems* [141,312][1] can be used to illustrate the shortcomings of the worst-case analysis for measuring the quality (efficiency) of on-line algorithms.

Previous work on on-line algorithms focuses on analyzing the performance of algorithms where the input is generated according to some fixed distribution [144,307]. Most of this work is concerned with analyzing *data structures* [57,187] *and paging algorithms* [141]. Thus, the "quality" of an algorithm is measured by its running time for a fixed distribution which depends on the chosen distribution. This is a useful model for studying specific algorithms, but we cannot use it in the designing and analysing an on-line strategy for the following two reasons:

- Information about the specific input distribution may not be available in advance.

[1] We will also study them in chapter 3.

- We desire to design robust algorithms for which the performance measure does not depend on a particular input distribution.

The problem of evaluating the measure of an on-line algorithm was addressed by *Sleator and Tarjan* [312]. They argued that the traditional approach of measuring the worst-case behavior does not seem appropriate for many on-line algorithms. Therefore, they suggested a different theoretical model to evaluate the performance of an on-line algorithm with respect to the optimal off-line algorithm that knows the entire request sequence in advance. The maximum ratio between their respective performances, taken over all request sequences, is called *competitive ratio (factor) or competitiveness*. This competitive method of analysis is named *competitive analysis* by *Karlin et al.* [204].

1.5 Thesis Outline

This thesis studies the *design and analysis of efficient algorithms for on-line algorithms*. We examine the following fundamental questions:

- How well can an on-line algorithm perform?

- how can we design efficient algorithms that make optimal use of the available information?

The general and interesting problem of whether off-line algorithms can be significantly better (faster) than on-line algorithms arises. Previous efforts to resolve this problem concentrated on *amortized time* [325] and to a lesser extent space. There are *two* reasons for considering this general problem:

- Some situations are off-line ones and we would like to bound the penalty we pay for using on-line algorithms in off-line settings.

- By comparing on-line algorithms to optimal off-line ones, we can indirectly compare on-line algorithms within a constant factor (i.e., the competitive ratio).

We discuss some of the areas in which on-line algorithms have been studied and we present techniques for proving upper and lower bounds on the competitive factors achievable by them in a variety of on-line problems.

The goal of this research is to study some of the areas in which on-line algorithms may apply and design algorithms that are competitively more efficient than the already existing ones under a variety of on-line settings.

The *major contributions* of this thesis are as follows:

- Provide general lower complexity bounds for the on-line algorithms on restricted inputs of some practical problems. Th. se results are often pessimistic, since in practice the input to a problem is not arbitrary.

- Extend the *k-server problem* for *non-resistive* graphs against a *lazy* adversary. In addition, we show that the *strong* competitive factor of the *harmonic* algorithm for the 2-server problem against a lazy adversary is in the interval (1,3] instead of the interval range [3,6] (See [285], *Theorem 8*). We also extend this result for the k-server problem.

- Give a slightly tighter competitive ratio for on-line coloring algorithm *First-Fit* on *d-inductive graphs* with *strong lookahead l*.

- Apply the *dual bounding technique* to simply reanalyze *on-line matching* algorithms.

* Propose a deterministic $\sqrt{3}$-competitive *navigation algorithm* for searching in unknown simple polygons (called *streets*) and show that no randomized algorithm can achieve a better competitive ratio than *ln5* for a visual searching in unknown streets, generally.

In the remainder of this chapter we present an overview of the *thesis' organization* and discuss the above results in further details.

In *Chapter 2* we introduce the terminology and notations used in this thesis. In particular, we give some fundamental definitions for incremental and on-line algorithms and derive lower complexity bounds for on-line strategies under some plausible restrictions.

Chapter 3 considers several general variations of the standard on-line models and some shortcomings of the worst-case analysis to measure the efficiency of on-line algorithms. We study the list update and paging problems in which the theory of on-line algorithms has been applied. We also deal with the competitive analysis of algorithms for managing data in a distributed environment.

Chapter 4 extends the theory of random walks on *resistive* graphs to *non-resistive* spaces. We develop methods for the synthesis of such random walks and we employ them to design randomized competitive on-line algorithms for k-server problems. Additionally, we consider the k-server problem in a more realistic distributed setting, where the transmission of information (messages) to the servers is costly.

Chapter 5 examines some classical *combinatorial optimization* problems in computer science in on-line fashion: the *on-line graph coloring* and *matching*. Furthermore, we apply the *dual bounding technique*, which is a general method for the competitive analysis by using the duality of linear programming (*DLP*) to obtain bounds on the optimal cost, in order to simply reanalyze several on-line matching strategies and show its general applicability.

Chapter 6 deals with the on-line algorithms and applications in *Computational Geometry*. Particularly, we propose an *on-line navigation* strategy in an unknown simple polygon (i.e., a *street*), which achieves the best competitive ratio of $\sqrt{3}$ known in the literature. Moreover, some *geometric* (or *visual*) *routing problems* have been developed for planar graphs under a specific on-line model, the so called *fixed graph scenario*.

Chapter 7 concludes the thesis with several final remarks, points out a few directions for future research and summarizes our results.

The Universe loves nothing so much
as to change the things which are
and to make new things like them.

Marcus Aurelius

Chapter 2

Theory and Complexity of On-line Algorithms

ΤΟ ΜΕΛΛΟΝ ΕΙΝΑΙ ΑΟΡΑΤΟΝ
The Future is Unknown
ΣΟΛΟΝ

But how much of the future is worth knowing?
R. Graham
ACM-SIAM Symposium on Algorithms, 1991

In this chapter we introduce the terminology and new variations on the standard model of competitive analysis for on-line algorithms used in this thesis. Particularly, we discuss the difficulties involved in analyzing the computational complexity of on-line algorithms. We also present new theoretic approaches to derive lower complexity bounds and models for on-line algorithms under some plausible restrictions.

Throughout this thesis, standard theoretical terminology has been used as contained in the algorithms and data structures' references [3,64,100,171,244,258]; e.g., classical definitions on graphs, asymptotic growth notations, computational models, and so on. Sometimes, we restate some definitions and results if needed for our purposes.

2.1 Theory of Dynamic and On-line Algorithms

Classical theory of algorithms deals with computational problems in which an algorithm is assumed to have a complete knowledge of the input data.

Definition 2.1: A *batch* algorithm takes an input and computes an output that is some function of the input. Such algorithms are also called *off-line* or *hindsight algorithms* in the literature.

This setting is not realistic in some algorithms, because sometimes only partial information about data is available, and the algorithm is supposed to compute, or at least approximate, the desired function based on this partial information.

2.1.1 Incremental Algorithms

In contrast to batch algorithms, an *incremental computation* is concerned with updating the output as the input arrives. Let $f\colon I \rightarrow O$ be a function (problem) with domain I being the set of *problem instances* or *inputs*, and range O the set of *answers* or *outputs*. Each $\mathrm{I} \in I$ and $\alpha \in O$ is itself a set, with $|\mathrm{I}| = l$ the length of the problem instance. Given a problem instance I, let $\alpha = f(\mathrm{I})$; in this case, we say that algorithm $\mathcal{A}$ *implements* f. The number of steps required by algorithm $\mathcal{A}$ to compute $f(\mathrm{I})$, in the worst case, is the complexity time of $\mathcal{A}$, denoted by $T_{\mathcal{A}}(l)$.

Definition 2.2: An *incremental algorithm* $\Delta\mathcal{A}$ for computing the function f takes as input the "batch input" I, the "batch output" $f(\mathrm{I})$, possibly some auxiliary information, and a description of the "*change in the batch input*", $\Delta\mathrm{I}$. The algorithm computes the "new *batch output*" $f(\mathrm{I} + \Delta\mathrm{I})$, where $\mathrm{I} + \Delta\mathrm{I}$ denotes the modified input, and updates the auxiliary information as necessary (*see Figure 2.1*). What we refer to as *incremental algorithms* have been called *dynamic algorithms*, *on-line update* (or simply *update*) *algorithms* and *on-line maintenance algorithms* in the literature. These definitions are based on those in [52,187].

A *batch algorithm* for computing a (problem) function *f* can obviously be used in this situation. It is called a *start-over algorithm* in this context (i.e., it starts from scratch). A standard, start-over algorithm can be viewed as an *off-line algorithm*. For example, *Heapsort* [3,175] is an off-line algorithm.

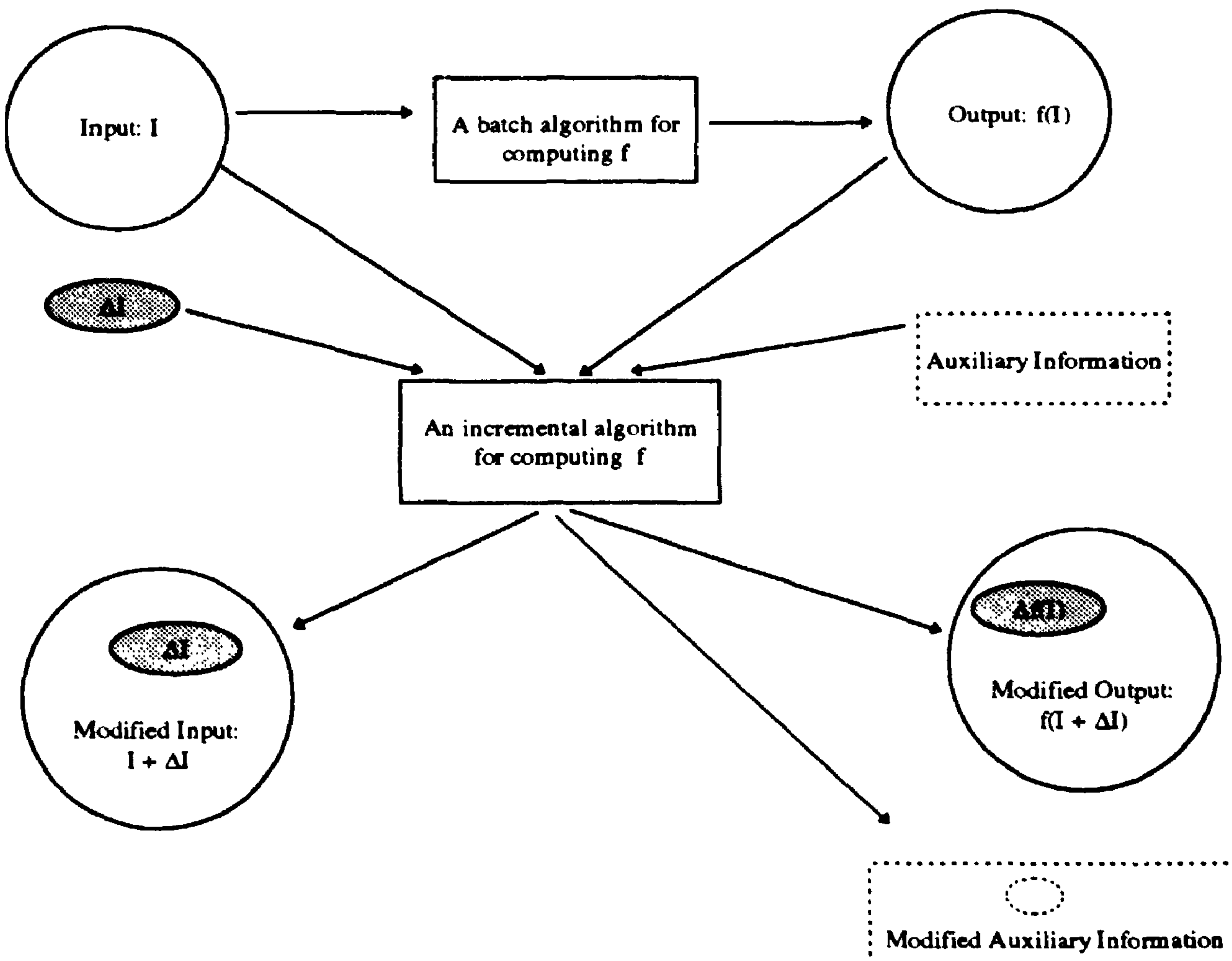


Figure 2.1: The above picture depicts the abstract problem of incremental computation. The shaded regions denote the input and the output. The dotted lines around the auxiliary information signify that it is optional information maintained by the algorithm and it may vary for each incremental algorithm.

There are some problems that can be solved by standard algorithms, but the goal is to find an incremental algorithm with better worst-case complexity than the start-over algorithm. An example of this approach includes *Frederickson's algorithm* [147] for updating minimum spanning trees.

Incremental algorithms, which are faster than the start-over algorithm for single change in the worst case, have been relatively few. For example, the *Incremental Relative Lower Bound (IRLB)* method [52] classify some incremental problems from this point of view. This method is based on a sequence of deletions only (not additions) and gives lower bounds for the incremental algorithms in terms of that for the batch strategies. This approach seems to be more of a theoretical issue than a practical one.

In the typical incremental problem, the applied incremental changes are categorized as *additions* and *deletions*. If only insertions or deletions are permitted, then an incremental algorithm is called *semi-* or *partially-dynamic*, and if both insertions and deletions are allowed, it is called *fully-dynamic*. For example, *Tarjan's* "union-find" algorithm [329] can be viewed as a partially-dynamic incremental algorithm for the problem of *connected components*. On the other hand, *Fredrickson's* algorithm for updating minimum spanning trees [147] is an example of a fully-dynamic algorithm. Also, *Italiano* [187] considers a fully-dynamic algorithm for updating *2-edge connectivity*. In some cases, algorithms may handle both types of change, but the analysis may apply only to a sequence of one type of change (e.g., see [346]). Typically, although not always, on-line algorithms are partially-dynamic; e.g., any *list maintenance strategy* is a notable exception (e.g., see [173,312]).

Research for such *dynamic* or *incremental algorithms* has been focused on the following areas:

- *Graph theoretic algorithms*;

  - *Connectivity* [79,132],
  - *Spanning trees* and *forests* [119,147,149,160,319],
  - *Shortest paths* [17,18,80,187],
  - *Biconnected* and *triconnected components* [119,187,289,340],
  - *Transitive closure* or *reachability* [185,186],
  - *Planarity testing* [118,128,129,130,321,322];
- *Computational Geometry* [266,271,281];
- *Data bases* [1];
- *Syntax-directed editors* and *grammars* [293,294,295,298];
- *Data-flow analysis* [13,53,298]; and
- *Code generation* and *optimization* [187].

There have been parallel incremental algorithms for *minimum spanning trees* and *connected components* [275]. Also, a beautiful research on dynamic data structures and algorithms for graphs can be found in [93,187].

2.1.2 On-line Algorithms versus Off-line Algorithms

An *on-line algorithm* is one that receives a sequence of *requests*, and performs an irrevocable *answer* (*action*) in response to each request before the next request arrives. Each sequence of requests and corresponding actions have an associated *cost*.

Aho, Hopcroft and *Ullman* ([3], pp. 109) define *on-line execution*, for an input sequence **r**, as follows:

> **Definition.** The *on-line execution* of **r** requires that the instructions in **r** be executed from left to right, executing the *i*th instruction in **r** without looking at any following instructions. The *off-line* execution of **r** permits all the **r** to be scanned before answers need to be produced.

The above definition is given in terms of a sequence of "instructions". There is *no* fundamental difference between instructions and requests or any other kind of input data. The description of the input as a sequence of instructions (requests) is typical for an on-line problem.

There is also no difference between "on-line update" and "on-line algorithms". It is a matter of the questions that we choose to ask. On-line algorithms usually refer to a sequence of operations (requests), rather than a solution that needs to be updated.

On-line (resp., off-line) algorithms are often associated with a particular *on-line* (*off-line*, respectively) *data structure* and its corresponding *timing* [308,309]. Generally, we are interested in a sequence of operations for on-line algorithms, rather than a single update. With these notions in mind, we define the on-line setting more rigorously in order to study the design and analysis of the efficient on-line algorithms.

An on-line algorithm is specified by:

- A set **R** of *requests* (*inputs* or problem *instances*);
- A set A of *actions* (*answers* or *outputs*);
- A *cost function* C: $\bigcup_{l=1}^{\infty} \mathrm{R}^{l} \times A \rightarrow \mathbf{R}^{+}$, where $\mathbf{R}^{+}$ denotes the set of non-negative real numbers.

For any request $\mathbf{r} \in \mathrm{R}^{l}$, define $Opt(\mathbf{r})$ as $\min_{a \in A^{l}} C(\mathbf{r}, a)$. An on-line algorithm $\boldsymbol{A}$ is determined by function $f_{\boldsymbol{A}}: \mathrm{R}^{+} \rightarrow A$, where the domain is the set of all finite non-empty sequences of requests. In response to a sequence $\mathbf{r} = r_1, r_2, \ldots, r_l$ the algorithm performs the sequence of actions $\boldsymbol{A}(\mathbf{r}) = f_{\boldsymbol{A}}(r_1), f_{\boldsymbol{A}}(r_1, r_2), \ldots, f_{\boldsymbol{A}}(r_1, r_2, \ldots, r_l)$ and incurs the cost $C(\mathbf{r}, \boldsymbol{A}(\mathbf{r}))$. In the above definition, we note that an on-line algorithm is *deterministic*. In *section 2.2.4*, we will extend the definition to *randomized on-line algorithms*.

On-line algorithms may be contrasted with off-line algorithms, which can use the entire sequence of requests in advance and take an action in response to each request. In other words, an off-line algorithm knows the future, but an on-line does not. On-line algorithms must make decisions based only on past history, which is a more realistic situation in the real world. For example, in the context of a database system each request may be a query or an update, and the corresponding action involves retrieving data from and possibly modifying the database. In an investment situation, a request might consist of a price quotation for a commodity and the action might be to buy or sell some amount of the commodity. It is clear that, in some on-line settings, partial information about the future is a great disadvantage (for example, think of the above situation as the stock market).

Here, the important computational problem of measuring the performance of an on-line algorithm arises. In computer science, we ask the following fundamental questions for an on-line problem:

- How well can an on-line algorithm perform?
- How can we design an algorithm that makes optimal decisions based only on the available partial knowledge of the future?

In order to study these questions, a formal theoretical framework for the performance quality of an on-line algorithm is needed. The analysis of a performance measure of on-line algorithms is more difficult than that in off-line settings, since usually, whatever action an on-line algorithm takes in response to an initial sequence of requests, there will be a sequence of further requests that makes the algorithm look inefficient.

2.2 On-line Models and Complexity Analysis

We consider the development of competitive analysis among amortized analysis. We also take into account the results that distinguish the different types of randomized adversaries which comprise the present theoretical models of on-line algorithms.

2.2.1 Competitive Analysis

Competitive Analysis provides a technique to develop a meaningful worst-case analysis of on-line algorithms without making assumptions about the distribution of the input .

There are many on-line problems for which the traditional worst-case performance of an on-line algorithm gives wrong results about the quality of the algorithm. We use *List Update Problem* [50,51,173,312] to get a poor performance of the worst-case analysis. A common lower-bound technique is to pretend that an algorithm plays against an adversary. The adversary observes the behavior of the algorithm and accordingly chooses a bad input to fail an on-line algorithm. An adversary who plays against a deterministic algorithm for List Update can always choose to access the last item in the algorithm's list. Thus, any deterministic on-line algorithm can be forced to pay the maximum amount ffor every access.

There has been an extensive work on list update problem where the input consists of a sequence of accesses. For each access in the sequence, the item to be accessed is independently chosen according to a fixed distribution over the items [50,57,225,296,312]. Several early studies of the *paging* [141,144,261,307] and *dynamic structures* [57,187] assumed a specific stochastic model of the source of requests. Within such a model, an on-line algorithm may be considered optimal if it chooses its actions so as to minimize the expected cost, where the cost depends on the sequence of requests generated by the stochastic source on the sequence of actions chosen by the algorithm in response to these requests. However, the choice of a stochastic model does not always design efficient on-line algorithms, because it requires data that may not be readily available in advance.

An alternative to stochastic models is *competitive analysis* which evaluates an on-line algorithm in comparison to the optimal off-line algorithm processing the same sequence of requests. This worst-case approach was first presented by *Sleator* and *Tarjan* in analyzing algorithms for *List Update* [312].

Definition 2.3: For a positive constant d, the on-line algorithm $\mathbf{A}$ is said to be *d-competitive* if there exists a constant β such that, for all request sequence $\mathbf{r}$, we have

$$C(\mathbf{r}, \boldsymbol{A}(\mathbf{r})) \leq d \cdot Opt(\mathbf{r}) + \beta,$$

where $Opt(\mathbf{r})$ and $C(\mathbf{r}, \boldsymbol{A}(\mathbf{r}))$ are the costs for servicing the input $\mathbf{r}$ that are charged to the optimal off-line and on-line algorithms, respectively. The *competitive ratio* (or *factor*) of $\mathbf{A}$ is defined as the *infimum* (i.e., *greatest lower bound*) of the set c such that $\mathbf{A}$ is c-competitive.

Some authors use a variant of these definitions, in which β is required to be zero. Since we are comparing an on-line algorithm to the optimal off-line one, we are focusing on what is lost in processing the information in on-line manner. Some sequences are inherently *difficult*; that is, sequences which would access many different items in turn (e.g., *List Update Problem* [173,312]). An on-line algorithm is not expected to perform efficiently on such sequences, because even the optimal off-line algorithm has a high cost on these requests.

The concept of competitive ratio is related to the *minmax* regret concept in *Game Theory* [9,95,111,150,268], and we shall often view the situation as a game between an *on-line player* who selects the on-line algorithm and an adversary who chooses the request sequence, in order to maximize the ratio between the algorithm's cost and that of an optimal off-line algorithm.

Competitive analysis was developed around the same time as *Amortized* analysis [100,325]. Both techniques are often used in conjunction, although the use of one does not necessarily imply the use of the other. In the following subsection we shall illustrate the concept of an on-line algorithm and its competitive analysis.

2.2.2 Rental Ski Problem

An on-line algorithm will be designed for the *rental ski (or equipment rental) problem*, which has been introduced by *L. Rudolph* [207], to illustrate the concept of the competitive ratio.

Supposing we were to decide to try the sport of skiing. Because we don't know how many ski trips we will take, we cannot decide whether to rent a pair each time or to buy a new pair of skis. If we bought skis at the beginning and then decided we did not like the sport after a couple of runs, renting would have been cheaper. On the other hand, if we were always renting and were to like the sport enough to ski many times, the right option would have been to buy the skis in the beginning.

An answer to the ski situation is to rent skis until the cumulative cost of renting first exceeds the cost of buying a new pair, and then to purchase a pair at that point. Suppose the cost of renting a pair of skis for a ski trip is 1, and the cost of buying a pair of skis is s. Here, there is only one possible request ("*take a ski trip*") and three possible actions ("*rent*", "*buy*", and "*use skis already bought*"), with cost 1, t and 0 units, respectively, where the third action can be invoked only if the second action has occurred previously.

A moment's reflection shows that with this strategy, our net cost never exceeds *twice* the minimum possible cost up to the point when we stop skiing, no matter when we stop skiing. Questions such as this come up in the study of on-line algorithms. In the ski rental problem, it is clear that any possible on-line algorithm is of the form "rent for the first k trips, then buy, then use the skis already bought". On a sequence **r** of t requests, the cost increases by the on-line algorithm $\mathcal{A}(\mathbf{r})$ is $C(\mathbf{r}, \mathcal{A}(\mathbf{r})) = \begin{cases} t, \text{ if } t \leq k \\ k + s, \text{ otherwise} \end{cases}$

and the cost incurred by an optimal off-line algorithm is $Opt(\mathbf{r}) = min(s, t)$.

The objective is to choose the parameter k to minimize the competitive ratio. In other words, we want the adversary to continue the ski trips until the on-line algorithm buys a pair of skis, and then stop. The on-line algorithm's cost on such a request sequence is k + *s*, while the optimal off-line cost is *min*(k+1, s). Therefore, the competitive ratio is $\rho(\mathbf{r}, \mathcal{A}(\mathbf{r})) = \frac{k+s}{\min(k+1, s)}$. Assuming that *s* is an integer, the competitive ratio is minimized by setting k = s - 1, thus achieving a competitive ratio of $\frac{2s-1}{s}$.

2.2.3 Amortized Analysis

Amortized Analysis or more specifically a *potential function analysis* (see [100,325]) is a useful tool that is used in analyzing the running time of an algorithm that performs a sequence of operations. Usually, such an analysis is in contrast with a worst-case analysis, which bounds the cost of the sequence by summing the worst-case costs of the individual operations. The idea was initially developed for use in analyzing data structures, although it has been useful in many other on-line contexts (e.g., *Task Systems* [66], *Server Systems* [251,254]; see also *chapter* 3).

The framework consists of a system and a set of operations. Typically, we are mainly concerned with the amount of time required to perform the whole sequence of operations instead of a single operation. Using the worst-case analysis, the cost per operation yields overly pessimistic results on the time to perform the entire sequence of operations (e.g., consider *increment* operations on a k-bit counter [100]). The goal of amortized analysis is to analyze the worst-case over sequences of the average cost per operation [325]. Examples of this type include amortized analysis of *balanced search trees*, the *union-find data structure*, and *splay trees* ([94,147,247,304,325,326]).

A stronger type of result for data structures uses amortized analysis together with competitive analysis to bound the amortized cost of an operation with respect to an optimal off-line algorithm. A typical competitive analysis with a *potential function* is of the following form:

Competitive Analysis with a Potential Function

Given an on-line algorithm $\boldsymbol{A}$ producing a solution in response to $\mathbf{r} = r_1, r_2, \ldots, r_l$:

1. Define a *potential* Φ which is a function of the states of $\boldsymbol{A}$ and OPT.
2. Show that, in response to each request
$$a \cdot x_i \leq b \cdot c_i + \Phi_i - \Phi_{i-1},$$
where $a > 0$, x_i and c_i are the costs incurred by $\boldsymbol{A}$ and OPT, respectively, in response to the ith request, and Φ_i is the value of the potential function after $\boldsymbol{A}$ and OPT have responded.
3. Sum the inequalities, showing that the cost incurred by $\boldsymbol{A}$ is bounded by
$$(b \cdot \text{opt} + \Phi_l - \Phi_0) \,/\, a$$
where "opt" is OPT's cost.
4. Show that $\Phi_l - \Phi_0$ is appropriately bounded.

Figure 2.2: An Amortized Analysis together with Competitive Analysis.

A *potential function* is defined to represent the "*distance*" of the on-line algorithm's configuration to the optimal off-line algorithm's configuration; its name stems from a natural interpretation of the *physical system* [100,325].

We may think of a potential function analysis as transforming OPT's costs: In response to the ith request, OPT's cost is changed to $c_i' = c_i + (\Phi_i - \Phi_{i-1}) / b$. The analysis then shows that OPT's overall cost is not substantially increased under the transformation, and gives a worst-case (per operation) bound on the transformed costs. That is, $b \cdot c_i' \geq a \cdot x_i$ is shown for each i.

In *section 3.2*, we consider ***List Update*** to illustrate the use of amortized analysis in conjunction with competitive analysis and show that the *Move-to-Front* (*MTF*) algorithm for ***List Update Problem*** [173,312] is 2-competitive.

2.2.4 Randomization in the On-line Model

In playing against an arbitrary deterministic on-line algorithm $\boldsymbol{A}$, adversary constructions are the principal means of proving lower bounds on the competitive ratio achievable for a given problem. The constructions usually depend on the ability to simulate $\boldsymbol{A}$. Thus, we would expect consideration of randomized on-line algorithms, which toss coins in the course of their execution, to improve the performance of on-line algorithms.

It seems that the unpredictability of such randomized algorithms should make it more difficult for an adversary to construct bad sequences. The amount of information available to the adversary will determine the strength of the adversary. We measure the strength of a randomized on-line algorithm in terms of the strength of the adversary it plays against and the competitiveness it achieves.

Ben-David et al. [47] introduce the most general framework for on-line algorithms, ***request-answer*** games. We may view a randomized algorithm as playing a game against a deterministic adversary. In each play of the game, the adversary chooses the request sequence $\mathbf{r} = r_1, r_2, \ldots, r_l$ and its own sequence of actions $\mathbf{b} = b_1, b_2, \ldots, b_l$, ***and*** the on-line algorithm chooses the sequence of actions $\mathbf{a} = a_1, a_2, \ldots, a_l$. We have the following three types of adversaries in increasing order of power of randomized on-line algorithms:

- The *Oblivious* Adversary:

  $r_1(b_1)(a_1)r_2(b_2)(a_2)\ldots$;

that is, an oblivious adversary chooses a complete request sequence before the on-line algorithm begins to process it. It is also called a *weak adversary*. An algorithm which is c-competitive against such a weak adversary is called *weakly c-competitive*.

- The *Adaptive On-line Adversary*:

  $r_1(b_1)a_1r_2(b_2)a_2\ldots$;

that is, an adaptive on-line adversary is allowed to watch the on-line in action, and generate the next request based on all previous moves made by the on-line algorithm. This adversary is also called a *medium adversary*. However, how to answer the present sequence has to be decided without knowing how the algorithm answers the present and future requests.

- The *Adaptive Off-line or Strong Adversary*:

  $r_1a_1r_2a_2\ldots r_la_lb_1b_2\ldots b_l$;

that is, an adaptive off-line adversary may adapt the produced sequence of requests to the random choices made to date by the on-line algorithm, and then pay for the entire sequence optimally. However, it can wait to see the entire sequence before deciding how to answer any request. An algorithm, where is c-competitive against such a strong adversary, is called *strongly c-competitive*. Note that in the above notations, the left-to-right sequence indicates the time sequence of the requests and actions, and parentheses indicate actions that are kept secret.

Suppose an adversary plays against a randomized algorithm $\mathcal{A}$ and presents a sequence $\mathbf{r}$ to the algorithm. Let $C(\mathbf{r}, \mathbf{a})$ denote the cost of the adversary's answers to the sequence $\mathbf{r}$ and let $E[C(\mathbf{r}, \mathbf{a})]$ be the expected cost of algorithm $\mathcal{A}$ on $\mathbf{r}$. The randomized on-line algorithm is said to be *c-competitive* if, for every adversary and for a positive constant c, we have

$$E[C(\mathbf{r}, \mathbf{a}) - c \cdot C(\mathbf{r}, \mathbf{b})] \leq \beta,$$

where β is a constant independent of the length $|\mathbf{r}| = l$.

All three adversary types have the same power against deterministic on-line algorithms. Against randomized on-line algorithms, the adaptive off-line adversary type is clearly the most powerful, and the oblivious adversary type is the least powerful. The competitive ratio that is achieved depends on the adversary type considered (see [47,141,261]).

Ben-David et al. [47] prove the following two very powerful theorems about the relative strengths of these adversaries:

Theorem 2.1. *If there exists a randomized c-competitive algorithm against any adaptive off-line adversary, then there also exists a deterministic c-competitive algorithm.*

Theorem 2.2. *If there exist a c-competitive algorithm against oblivious adversaries and a randomized d-competitive algorithm against adaptive on-line adversaries, then there is a (c·d)-competitive algorithm against any adaptive off-line adversary.*

The two theorems together imply that if there exists a best randomized c-competitive algorithm against an on-iine adversary, then there exists a deterministic *c^2-competitive* algorithm.

Unfortunately, *Theorem 2.1* is *not* constructive generally. In [111], an *infinite request-answer game* is shown such that there is a randomized 1-competitive strategy, but there is no computable c-competitive strategy for any c > 1. *Theorem 2.2* is important, as it lets us show the existence of a deterministic competitive algorithms by constructing randomized competitive algorithms. *Irani* and *Karp* show that *Theorem 2.2* is tight for request-answer games (see an example in [47]).

Finally, there is much greater difference between an oblivious adversary and an adaptive adversary than that between adaptive on-line and off-line adversaries. This is best illustrated in the *Paging Problem* [312][1]. Similar work has been done on *List Update* [173,312], although the results are less dramatic.

2.3 Complexity Bounds and Models for On-line Algorithms

In this section, we consider the issue of restricting the computational sources for on-line algorithm. This is a practical idea which is addressed by *Borodin et al.* [65] and underlines the philosophy behind *Paging problem* with *locality of reference*. The *goal* is to find an on-line algorithm that computes the new answer *faster* than an off-line algorithm, when a small change in the input is given. We give relative lower bounds of an on-line algorithm in terms of that for the off-line performance available. We also show that *no* on-line algorithm can be better than l times the hypothetical optimal off-line algorithm, where l is the length of the request sequence $\mathbf{r}$.

2.3.1 Lower Complexity Bounds

Typically, an efficient on-line algorithm uses an additional amount of establishing supplementary data structures and preprocessing cost that is required to produce a good (efficient) initial solution. This amount is often referred to as *preprocessing cost (time)* of the algorithm.

Generally, an on-line algorithm returns the tuple (a, T_0), where a is the current answer and T_0 is any preprocessing time. The computational model used determines the

[1] Also see *chapter 3*.

form of preprocessing cost. In a *Random Access Machine* (RAM) [3], T_0 consists of the complete context of memory and registers after each step of the algorithm. Clearly, T_0 depends both upon the computational model and the particular algorithm being used; while the answer a depends only upon the definition of the problem being solved. Additionally, the preprocessing cost that is required to be an output does not necessarily increase the cost by more than a constant factor, since the algorithm must already maintain the state internally.

Proposition 2.1. Given any on-line or off-line algorithm $\mathcal{A}$, problem instance I, answer a, and preprocessing cost T_0, we have that $T_{\mathcal{A}}(I) = \Omega(|a| + |T_0|)$, where $T_{\mathcal{A}}(I)$ is the time complexity of the algorithm $\mathcal{A}$.

Proof: Since $\mathcal{A}$ outputs a and T_0, it has to write them to some memory device. $\square$

Definition 2.3. Let $f: I \rightarrow O$ be a function (problem).

If $(\forall l \in \mathbf{R}^+)(\exists I_0^l \in I \text{ with } |I_0^l| = l)\ [a = f(I_0^l)]$ and a can be determined in time dominated by T_f, then the function f for which such a procedure exists is said to be a *good function* or a *function with a good initialization value.*

Clearly, a good initialization value problem can be found in complexity time $O(l +$ size of the output$)$ by inspection for many functions. For example, the ***sorting by comparisons***, ***maximum*** and ***minimum*** problems have good initialization values. We see that the existence of a good initialization value is a property of the function (problem) f, and not of any particular algorithm which implements it. Also, there is no direct relation between preprocessing time and good initialization values.

Definition 2.5. An (on-line or off-line) algorithm $\boldsymbol{A}$ is said to be *bounded* if it implements a function f with a preprocessing time T_0 such that $|T_0| \le T_f$.

Again, we see that the notion of preprocessing cost has only a relationship with the algorithm that implements it. Particularly, an algorithm that has no preprocessing cost is a special case of a bounded algorithm for some computational problems (e.g., on-line coloring partial graphs). We can use the above two definitions to limit the power of an on-line algorithm, although all of the on-line algorithms are not restricted. Furthermore, if an efficient on-line algorithm exists for a problem f, we can design an efficient off-line algorithm by using the on-line algorithm repeatedly.

This would be developed by the following procedure:

1. Compute the output value for some initial *dummy* input.
2. Apply the on-line algorithm repeatedly and compute the real value for the problem instance by changing the initial values, one-by-one, successively.

More formally, we get the following algorithm which is called an *effective algorithm:*

---

0. $I \leftarrow I_0^l$; {* I_0^l is an initial dummy input of length $l = |I_0^l|$. *}
1. $a \leftarrow f(I)$;
2. $T \leftarrow T_0(l)$; {* The initial preprocessing cost. *}
3. **for** $i \leftarrow 1$ **to** l **do**;
   $(a, T_0) \leftarrow \boldsymbol{A}_{on}(a, I, T, I_i)$; {* $\boldsymbol{A}_{on}$ denotes an on-line algorithm. *}
   $I \leftarrow I_i$; {* I is successfully modified to I_i, where the Hamming distance $|I - I_i| \ll \varepsilon$, for each $\varepsilon > 0$, under a suitable encoding. *}

Figure 2.3: *Effective Algorithm* $\boldsymbol{A}'$ for Updating an Initial Solution.

Theorem 2.3. *A good function f can be implemented by both bounded off-line and on-line algorithms if and only if* $T_f \le l \cdot T_{A_{on}}(l)$.

Proof: Let $T_{A'}(l)$ be the complexity time of the effective algorithm $\boldsymbol{A'}$ for computing $f(I_0^l)$ with $|I_0^l| = l$ (see *figure 2.3*). Since f is a good function, the complexity time of *steps* 0 and 1 is less than T_f. Also, the complexity time of *step* 2 is greater than that of step 1. The time complexity of *steps* 3 and 4 is $l\cdot$(complexity of step 4) $\leq l \cdot T_{A_\infty}(l)$, because otherwise the effective algorithm $\boldsymbol{A'}$ for computing f would be better (faster) than the optimal off-line algorithm; which is a contradiction. Thus, $T_f \leq l\ T_{A_\infty}(l)$.

Conversely, we assume that the theorem is *not* true; that is, $T_{A_\infty}(l) < \frac{T_f}{l}$. Then the complexity time of the effective algorithm $\boldsymbol{A'}$ is less than T_f, which is a contradiction again. Therefore, our theorem is true. □

Now, we consider the problem "sort by comparisons" to illustrate the above proof. Sorting by comparisons has a good initialization value with time complexity $O(l\cdot \log l)$ by applying our *effective algorithm*. Clearly, this is a contradiction, because "sorting" cannot be that fast (e.g., see [175], pp. 350-352). So the best bounded on-line algorithm *cannot* be faster than $O(l\cdot \log l)$.

2.3.2 Amortized Complexity Bounds

In some environments, an amortized performance of an on-line algorithm may be better than its worst-case complexity time, even if some steps have a poor worst-case performance. We apply *Theorem 2.3* to the amortized case analysis and we have the following result:

Corollary 2.1. *If a good function f can be implemented by both bounded off-line and on-line algorithms, then $T^{l}_{A_{on}}(l) \leq \frac{T_f}{l}$, where $T^{l}_{A_{on}}(l)$ denotes the time complexity per operation required by an on-line algorithm* $\boldsymbol{A}_{on}$ *amortized over l operations.*

Proof: By *contradiction*, using the same argument as in the proof of the *Theorem 2.3.* □

2.3.3 On-line NP-Completeness

We apply *Theorem 2.3* to an *NP*-complete problem [156,209] and show the effect this problem has on the complexity time of on-line algorithms.

Theorem 2.4. *There is no bounded on-line algorithm with time complexity less than $T_{k\text{-}SAT}(l)/l$ for the k-SAT problem, which is an NP-complete for $3 \leq k \leq l$.*

Proof: Let us construct a good initialization value for k-SAT problem with l clauses such that each clause contains k literals $x_1, x_2, \ldots, x_k$, where x_i = T (*true value*) for every $1 \leq i \leq k$. Clearly, the theorem is true by using *Theorem 2.3.* □

Additionally, we show that no *NP-complete* problem can have a polynomial time, bounded on-line algorithm unless *P = NP*.

Corollary 2.2. *If there is a bounded on-line algorithm* $\boldsymbol{A}_{on}$ *that implements any NP-complete problem (function) f in polynomial time, then P = NP.*

Proof: Suppose that there is such an on-line algorithm $\boldsymbol{A}_{on}$, then we can construct a polynomial transformation to use it to update k-SAT instances, where $3 \leq k \leq l$. By *Theorem 2.4*, we get that the complexity time $T_{A_{on}}(l)$ cannot be polynomially better (faster) than $T_{k\text{-}SAT}(l)/l$. This implies that every *NP-complete* problem must be in *P* (i.e., $NP \subseteq P$). Therefore, $P = NP$. □

We would like to note that a statement (*without a proof* !) similar to *Corollary 2.2* was made in [9,80] for incremental graph algorithms. Also, the above complexity results for on-line algorithms can be easily extended to incremental algorithms as well.

We have seen that the lower bound of an on-line algorithm depends strictly upon the function (problem) and not upon the implementation. The above *Theorem 2.3* holds for on-line algorithms that cannot be performed in time faster than that required for a good initialization of the problem. It is an interesting *open problem* whether we can find any function for which no *good initialization* exists to such on-line algorithms. It would also be interesting to determine a set of necessary conditions for a *good function (problem)*. Generally, the techniques to derive lower bounds for an on-line problem in terms of that for the off-line problem limit the preprocessing cost available and therefore it is a specific problem, unfortunately.

2.3.4 On-line Complexity Models

Delcher and *Kasif* [264] proposed other *notions of completeness* for on-line algorithms, but they completely ignored the preprocessing issue. Although their results are interesting, they are somewhat weak in that the issues of dynamic date structures and preprocessing which are overlooked. They tried to overcome this drawback by showing the following *conjecture*: *the on-line versions of all P-complete problems are P-complete.*

Reif [290] presented another on-line complexity model to analyzing on-line algorithms and sketched an interesting *notion of completeness*. He showed that some problems are unlikely to have efficient on-line algorithms, but he did *not* develop a comprehensive theory or consider the necessary details of preprocessing. He pointed out that there exists a number of problems for which it is difficult to develop an on-line algorithm with linear time complexity. Some of such problems for which a deterministic on-line algorithm can be designed in polynomial time with respect to sequential *LOGSPACE Turing Machine* reductions include:

1. *Acceptance of a linear time Turing Machine;*
2. *Path system problems;*
3. *Boolean circuit evaluation;*
4. *Unit resolution, and*
5. *Depth-first search numbering of a graph.*

There exist linear time reductions in the sequential RAM model of computation for all these problems. Also, they are *constant-time updatable*; that is, they can be reducible to each other in constant time under a suitable encoding. This result implies that if a sublinear on-line algorithm can be found for updating one of these problems, it can be applied to update any of them.

Finally, *Miltersen et al.* [264] consider a new and more general complexity approach to *incremental computation*. They defined some new complexity classes for *incremental algorithms* and studied their relation to existing ones (e.g., *sequential* and *NC parallel classes*). Particularly, they show that some problems exist that belong to the *incremental* versions of *P-complete* problems (e.g., the *comparator circuit-value* problem and the *comparator network-stability* problem) and prove that some important special dynamic solutions imply parallel ones. It has also been shown that problems with sequential space complexity have small *incremental* time complexity. According to the authors, the classes *incremental TIME and SPACE* are very important for getting a better understanding of the relationship between *incremental* and *parallel computation.*

In conclusion, we would like to point out that there exist many challenging *open problems* in this area. The following are the most interesting for further research on the complexity models for *on-line* or *incremental computation*:

- *On-line* or *incremental* versions of *P-complete* problems are *P-complete* problems.

- How is the *incremental* version of the class *POLYLOGTIME* related to the class *LOGSPACE* ?

- What is the relation between the *incremental* version of the class *POLYLOGTIME*[1] and *NC parallel* class of problems which have *optimal parallel* algorithms?

When it is not necessary to change,
it is necessary not to change.

Lord Falkland

[1] See [264] for the definitions and more details.

Chapter 3

On-line Models and Applications

The mathematician's patterns must be harmonious.
Beauty is the first step: there is no permanent
place in the world for ugly mathematics.

G. H. Hardy
A mathematician's apology.

This chapter first outlines some general theoretical models followed by applications for the *List Update* and *Paging on-line problems*. Additionally, we present some new results and simple extensions of the above problems for variant on-line models. This study, along with *Chapter 5*, is intended to illustrate the importance of the field and to provide a context for the work in this research.

3.1 On-line Theoretical Models

We introduce two general on-line theoretical models, the *on-line Games* and *Metrical Task Systems* (*MTS*).

3.1.1 On-line Game Theory

Game theory is a mathematical discipline dealing with multiperson decision problems, which is often called the *theory of conflict* without or with cooperation between several parties. In a *non-cooperative* (resp., *cooperative*) game, the players are (resp., are not) inclined to cooperate and to form coalitions. A *non-cooperative game* which gives rise to opposite claims is called a *zero-sum game.*

The history of game theory is generally accepted to start with *John von Neumann's* article "Zur Theorie der Gesellschaftsspiele" (1928) [128]. However, the development of game theory gradually started to appear after the book "Theory of Games and Economic Behavior" [269] was published and was inspired by economic problems rather than problems from physics or other areas.

Games and game-like phenomena occur naturally in computational settings. There are many applications of *game theory* in computer science. For example, in distributed computing and cryptography, researchers have tried to develop models that reflect the competitive nature of distributed and cryptographic protocols.

It is believed that on-line games capture most of the on-line problems in which competitive analysis is applicable. An interesting attempt is to describe on-line problems in terms of games and develop general techniques of constructing competitive algorithms [47,66,285]. Such results are still in progress [9,89].

An *on-line game* is a triple y = (Q, R, f), where

- Q is called a set of *states*.
- R is a *set of requests.*
- f: $Q \times R \times Q \rightarrow \mathbf{R}$ is a *cost function*, where $\mathbf{R}$ is the set of real numbers. We also assume that the sets Q, R are finite and the function f must satisfy certain topological properties to ensure that some *minima* and *maxima* are actually achieved.

In [89], an interesting theory of on-line games and its relationship to the fixed point theory for functional spaces has been developed, which is useful for the proofs of some properties of on-line problems. It is *not* in our intention to restate it here. Instead, we consider some on-line games and applications.

For example, let us first consider a simple *bit-matching game*: both the input and output consist of one bit for each one and the *cost* is 1 if the output matches the input, 2 if it does not. Clearly, any algorithm for the bit-matching game described above is 2-competitive, with zero additive constant. Moreover, if *P* is an on-line problem which consists of repeated plays of the bit-matching game, then *P* has an optimal competitiveness of 2. This simple on-line game simply shows that it is impossible generally to compute the optimal solution of an on-line optimization problem without the notion of competitiveness.

We now describe another simple two-person game which is at the core of many other on-line algorithms and that is a special case of the *server problem*[1] [255]. Also, some special cases of this game have been studied by *Baeza-Yates et al.* [34,35].

> The *cat-and-mouse* or *hide and seek a mouse game* [285] is a game between two players, one of whom we call the (blind) *cat* and the other the *mouse*. The game proceeds in a series of *rounds* and is played on an undirected n-vertex graph G whose edges have positive real *costs* in the form of a *n x n* cost matrix C = (C_{ij}). The *i*th round begins with both players at the same vertex u_i of the graph. The mouse then moves in a new vertex $u_{i+1} \neq u_i$, not known to the cat, incurring a cost equal to the distance between the vertices for this round. Each move of the mouse may depend on all previous walks of the cat. The cat may use a *memoryless*[2] randomized algorithm and choose its next move probabilistically, as a function of its previous walks. The game stops after a fixed number of rounds.

[1] We will study it in chapter 4.

[2] Each cat move depends only on the current position and not on the previous walks.

We will see in *Chapter 4* that the competitiveness of any cat that uses a random walk is at least n - 1 on any graph, no matter what transition probabilities the cat uses. This result is true for *resistive* and *non-resistive graphs* (i.e., *with* and *without* symmetry of the edge weights, respectively).

As we have already mentioned, on-line games arise in connection with the on-line problems. Several on-line games have been referred to in the literature and some of them are: *tree game* [86], *on-line game G* for *LUP* of length 2 [89], *on-line (off-line) continuous pebble games* on graphs [120], *on-line dynamic game* [120], *on-line infinite games* [111], *layered graph traversal game* [288] and the *financial games* for financial decision making [127,357].

3.1.2 Task Systems and On-line Algorithms

In practice, almost all dynamic computer systems perform any given task in on-line fashion, that is, without full knowledge for their future impact on the systems.

Borodin et al. [66] introduce a general model for a system on which a processing sequence of tasks must be performed and develop a general on-line decision algorithm. A number of on-line applications that are special cases of their model, include *operations of dynamic data structures, paging, processor scheduling* and *server systems.*

Specifically, a *task system* (S, d) for processing sequences of tasks consists of a set S with $|S| = n$ states and a $n \times n$ cost matrix $d = (d_{ij})$ where the distance $d_{ij} = d(i, j)$ is the cost of moving from state i to state j. We assume that the distance matrix is non-negative, has zero entries along the diagonal and satisfies the triangle inequality. The cost of processing a given task depends on the state of the system. The input is a sequence $T = (T_1, T_2, \ldots, T_n)$ of *tasks* where each task T_i is the cost of performing the task in the ith state. A *schedule* for a sequence T of tasks is a function $\Phi: \{1,2,\ldots,n\} \rightarrow S$, where $\Phi(i)$ is the state in which the *i*th task is performed. The *cost of schedule* Φ on task sequence T is the sum of all state transition costs plus the sum of the task processing costs:

$$C(T; \Phi) = \sum_{i=1}^{n} d(\Phi(i-1), \Phi(i)) + \sum_{i=1}^{n} T_i(\Phi(i)).$$

The objective is to minimize the cost of the schedule when the tasks are arriving in on-line manner.

An (*off-line*) *scheduling algorithm* for a task system (S, d) is a function f that associates to each task sequence T a schedule $\Phi = f(T)$. It is easy to construct a dynamic programming algorithm that gives an optimal (minimum cost *Opt*(T)) schedule for any task sequence T. On the other hand, an *on-line scheduling algorithm* must determine in which state to perform a given task (S, d) without any knowledge of the future tasks (i.e., $\Phi(i)$ depends *only* on $T_1, T_2, \ldots, T_{i-1}$). The *cost of algorithm* $\mathbf{A}$ on sequence T, denoted by $C_A(T)$, is defined to be $C(T; \mathbf{A}(T))$.

We measure the efficiency of an on-line algorithm $\mathbf{A}$ as compared to the optimal off-line algorithm and we say that algorithm $\mathbf{A}$ is c-competitive (or it has *waste factor* at most c), if for any finite task sequence T, $C_A(T) - c \cdot Opt(T)$ is bounded by a constant. The *waste factor* $W(\mathbf{A})$ of algorithm $\mathbf{A}$ is the infimum of all such c and the *waste factor* W(S, d) of the task system is the infimum of $W(\mathbf{A})$ over all on-line algorithms $\mathbf{A}$.

Borodin et al. [66] give an optimal $(2\cdot|S|-1)$-competitive algorithm for any ***metrical task system*** (***MTS***) (i.e., a task system (S, d) in which the cost matrix d is *symmetric*), and an $O(|S|^2)$-competitive traversal algorithm for every task system. However, for many useful special cases of task systems, $2\cdot|S|-1$ is a very weak bound and there are on-line algorithms whose competitive ratio is independent of the number of states.

Karlin et al. [204] show that there exists a randomized $2 \cdot H_n$-competitive algorithm with a lower bound of H_n (i.e., the *n*th *harmonic number*) for the *snoopy caching problem* in the special case where the task system is *uniform* (i.e.,$\forall i \neq j$, $d(i, j) = 1$). On the other hand, the competitive upper bound for task systems does not give very strong results, since the number of states in a system is often very large when it is applied to particular special cases. For example, if we consider *paging problem*[1] [255,312] with k slots in the fast memory and m virtual memory pages, the number of states is $\binom{m}{k}$. Therefore, a deterministic paging algorithm that has a competitive ratio of k exists.

In the following section, we shall consider some on-line computational problems which are special cases of the task systems and for which we can design on-line algorithms with competitive factors independent of the number of states in the system.

3.2 List Update Problem

We consider the *List Update Problem* (*LUP*) or *sequential search problem* [51,173,256,292,296,312,330] which has been extensively studied in the literature under several formulations and different aspects. Many on-line heurestics have been devised for the LU problem. We investigate them and furthermore, we attempt to simply extend some of these on-line algorithms to handle *successful* and *unsuccessful searches*, as well as *insertions* and *deletions*.

3.2.1 Problem Motivation

List update problem consists of maintaining a *dictionary*[2] as an unsorted linear list of items. The input is a sequence of operations, where each operation accesses, inserts or

[1] Also see paragraph 3.3.

[2] An abstract data structure that involves a collection of words and requests for *insertions*, *deletions* and *membership* operations.

deletes an item. The cost of performing the searched item depends on its position in the current list. Searching is done sequentially starting from the front of the list. The list is maintained according to rearrangement rule called an *update* or *self adjusting heuristic*, which is applied as part of every operation. This list is referred to as *self-adjusting list*, since eventually it converges to the optimal static adjusting list. After an item is accessed, it can be moved anywhere closer to the front of the list in constant time (i.e., with *no* extra cost) using a *paid exchange*. Th , the total cost of moving the item via a paid exchange is the distance the item is moved. As the term "self-adjusting" suggests, our goal is to arrive at the optimal static adjusting of the list.

List update techniques have a lot of applications in practice since they are simple to use. They have been used to design *data compression algorithms* [49] and efficient simple algorithms for computing *point maxima* and *convex hulls* [48].

3.2.2 Self-adjusting Linear List Algorithms

Several *heuristics* for the *LUP* have been considered in the literature. The first three most common list heuristics, the *Frequency Count* (*FC*), the *Transpose* (*TR*) and the *Move-To-Front* (*MTF*), were proposed by *McCabe* [256]. A broader survey of self-adjusting data structures and linear list algorithms can be found in [173,325].

Definition of *FC* Heuristic: We maintain one counter per item to keep a count of the current number of requests for that item. After every operation, the counter for the accessed item is incremented and the list order is updated so that the items are arranged in decreasing order of request frequency.

In *Figure* 3.1, we see an example of an operation using the *FC* heuristic.

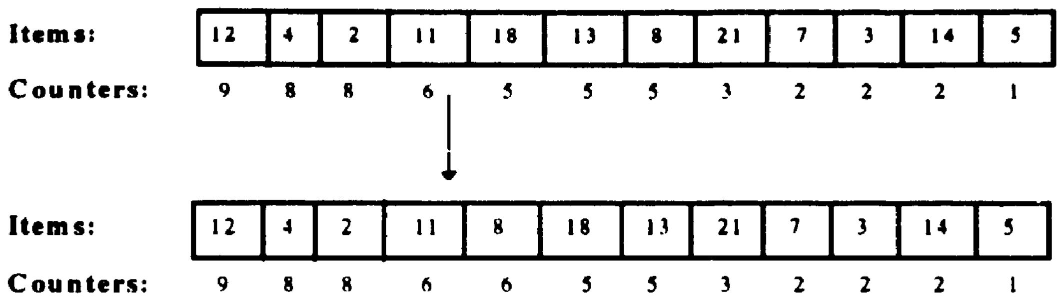


Figure 3.1: Frequency Count Example.

Definition of the *TR* Heuristic: Every time an accessed item moves forward one position at the front of the list (unless the item is already there) by interchanging cost i, where x is the ith item.

Bentley and *McGeoch* [50] showed that transposition heuristic is not competitive.

Definition of the *MTF*: Every time an item is accessed it is moved to the front of the list with the intervening items being shifted back one position in the list to make room at the front. (If the item is already at the front of the list, the list is not changed.)

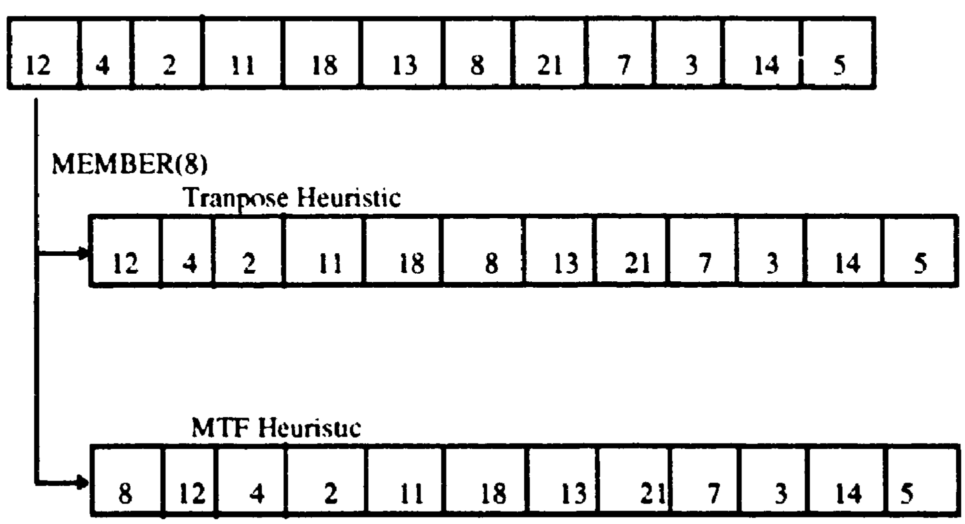


Figure 3.2: List Update Heuristics.

In practice, it is easy to check that the expected cost of *TR* converges to a better asymptotic value, but that the convergence of *MTF* is faster. Unfortunately, there are no analytic results about the behavior of *TR* in the literature, except for some simple cases. Experimental evidence (e.g., [50,51,57]) has shown that in real life situations, the lists using *MTF* do quite well compared to lists using *FC* as well *TR*. It is a much more difficult problem to prove it mathematically. It should be noted that both the *TR* and *MTF* heuristics are instances of a more general heuristic called the *Move-ahead-k-heuristic*, first studied by *Rivest* [296]; that is, TR (resp., MTF) heuristic is equivalent to *move-ahead-1* (resp., *move-ahead-∞*) heurestic.

Sleator and *Tarjan* [330] analyzed the competitiveness of list update heuristics and proved that *MTF* is 2-competitive. The proof is inductive, because it uses the important concept of a *potential function* in the *amortized analysis* as we have seen. A *proof idea* follows:

> At any step, let p be *MTF*'s list and let q be *OPT's* list. The potential function $\Phi(p, q)$ is chosen to be the number of pairs (called *inverted pairs*) of items which appears in a different order in *MTF*'s list than in OPT's list. It is then easy to show that, at each step, $C_{on} + \Delta\Phi \leq C_{off}$, where C_{on} and C_{off}, respectively, denote the cost incurred by *MTF* and by OPT at the step and $\Delta\Phi$ denotes the change of the potential function at this step. Since Φ is non-negative and initially zero, it follows that *MTF*'s amortized cost is less than twice the *OPT's* cost for the access. The analysis for paid exchanges, insertions and deletions is similar.

Furthermore, we can also prove that *no* deterministic on-line algorithm can achieve a competitive ratio less than 2 against a *strong* adversary (i.e., *MTF* is an optimal among all the deterministic heuristics for *LUP* [173,312] with competitiveness $\Theta\left(\frac{2L}{L+1}\right)$, where L is the size of the list).

Recently, *Lai* and *Wood* [233] have presented two new randomized list update heuristics:

- The first heuristic, the *randomized transposition* (*RT*) heuristic, performs at most one *transposition* (i.e., interchange any two adjacent items) on each access and its expected search time is 4-competitive against an *adaptive* adversary that manages a *static* list (i.e., we say that *RT* is 4-*pseudocompetitive* or *statically* 4-*competitive*). We note that *RT* heuristic works, if a request can be only a search; that is, insertions and deletions are not allowed. Although *RT* is statically competitive, it is not competitive in the class of singe-exchange heuristics against an adaptive adversary (e.g., *RT* has a competitive ratio of $\Omega(L)$ against *TR* heuristic).

- The second heuristic, the *randomized-exchange* (*RE*) heuristic, performs at most one *exchange* (i.e., interchange any two items) on each access and is 4-pseudocompetitive (resp., 8-competitive) against an adaptive, *static* (resp., *on-line*; e.g., *MTF* heuristic) adversary. Thus, one obvious open problem is to improve the analyses of *RT* and *RE*, or to show that their analyses are tight.

Sleator and *Tarjan* introduced a very simple method of maintaining a set of linearly ordered items in a *Splay Tree;* that is, a dynamic binary search tree. The objective is to maintain a tree, using *only* tree rotations so as to minimize the total running time of a sequence of dictionary operations. A splay tree performs tree rotations according to a simple procedure called *splaying*. They proved that the amortized cost to access an item in their scheme is $O(\log n)$ by using a similar way to that in the proof of *MTF* algorithm. The famous and still unresolved *Splay Tree Conjecture*[1] states that splay trees have a constant competitive ratio against a dynamic optimal off-line strategy. Such a result and properties would be analogous to *LUP*'s ones, whereas *splaying* has been proved to be competitive *only* when compared with *static* algorithms.

[1] A more complete definition may be found in the mice *surveys* of applications of *amortized analysis* [173,325].

Recently, *Sherk* [303,304] generalized splay trees defining the *k-ary Search Trees* for some fixed k ≥ 2 and he extended the heuristics, *splay tree conjecture* and *Sleator - Tarjan*'s splay tree results. With k = 2 and splay trees used in place of 2-splay trees, his *Dynamic Optimality Conjecture* (k-*DOC*) for k-splay trees is *Tarjan*'s *Dynamic Optimality Conjecture* for splay trees (*DOC*: on all sufficiently long request sequences, splay trees are as fast as any implementation using a binary search tree (not just those using a static tree); see [313]). In addition, it is *not* clear that any doubly optimal off-line strategy exists for dynamic binary trees. If such a strategy exists, then it is sufficient to prove that *DOC* maintains a balanced tree in a restricted class of data structures. This may be an important step towards resolving these conjectures.

3.2.3 Randomized Competitive List-Update Algorithms

Reingold et al. [292] used the power of randomization to improve the deterministic previous results and the competitiveness of the *MTF* algorithm. We consider a randomized version of *MTF* for *LUP* as follows:

Algorithm *BIT*

Let b(x) be one random bit for each item x, which is randomly initialized. From then on *BIT* runs completely deterministically: after finding x, *BIT* first complements b(x) and then moves x to the front of the list if b(x) = 1.

Figure 3.3: A Randomized *MTF* Algorithm for *LUP*.

Roughly speaking *BIT* is "move-to-front every other access" and it is 1.75-competitive against an *oblivious* adversary.

BIT can be generalized to a family of *COUNTER* algorithms, which are a slightly more complicated.

Algorithm *COUNTER* (s, S)

Let s be a positive integer and let S be any non-empty subset of $\{0,1,...,s-1\}$. The algorithm keeps a *mods* counter for each item and each value chosen independently with equal probability. At a request to item x, *COUNTER* decrements the x's counter *mods* and then moves x to the front of the list via *free* exchange if x's counter is in S.

Figure 3.4: A Generalized Randomized *MTF* Algorithm for *LUP*.

BIT is *COUNTER* ``.{1}). In fact, *COUNTER* algorithm can be modified to obtain a competitive ratio of $\sqrt{3}$ [292]. It has been recently proved [330] that *no* randomized algorithm can achieve a competitive ratio better than 1.5, while the lower competitive bound of any algorithm for a list update problem cannot be better than 1.27 against such an *oblivious* adversary in a standard model (*Reingold et al.* [292] have improved the lower bounds for three- and four-item lists to 1.2 and 1.25, respectively).

All the above algorithms for the List update problem share the same drawback as *MTF*; that is, they do *not* efficiently handle *unsuccessful* searches, *additions* and *deletions* as well. In fact, it is possible to modify the algorithm *BIT* to handle successful and unsuccessful searches as well as insertions, but *not* deletions.

Algorithm *BIT-UA*

Deterministic step: Let p and s be two *bits* for each list item x. If x_p is ahead of x in the list, then $p = 0$; otherwise $p = 1$. Similarly x_s is defined. Find x by finding both x_p and x_s, in order to finish an unsuccessful search.

Random step: Let a third bit, b(x), be the random *bit*. Initially, b(x) is set to 0 or 1 with equal probability. After a successful *find(x)*, we toggle b(x). If b(x) changes to 1, we move x to the front, otherwise the list remains unchanged. For an unsuccessful search, for each of the two boundary keys

(x_p and x_s), we toggle the random bit; if a key's random bit changes to 1, we move it to the front (i.e., *BIT-UA* preserve their relative order). The *p, s* bits are maintained with constant extra time using the techniques described in [292].

Figure 3.5: A Randomized Algorithm for *LUP* to handle Successful and Unsuccessful Searches as well as Insertions.

By dividing the expected change of the potential function into three parts and using similar techniques as in [180,181,292], we find that the algorithm *BIT-UA* achieves a competitive factor of 2.5 (i.e., 2·1.75-1) by summing up all successful and unsuccessful searches. In addition, if *BIT-UA* allows insertions as well, this does not affect its competitive analysis. In fact, *BIT-UA* algorithm can be improved using similar techniques as for *COUNTER* algorithms [292] to achieve a competitive ratio of $2\cdot\sqrt{3}-1$ (<2.46142) against an oblivious adversary.

Recently, *Hui and Martel* [179] have presented an improved version of *BIT-UA* which was also able to handle *deletions* efficiently. They also proved that their new modified algorithm *BIT-UAD* for the list update problem is 6-competitive against an *oblivious* adversary when considering successful and unsuccessful searches, insertions and deletions as well. It is also interesting to see whether we can extend the amortized analysis, which uses both the *accounting method* and the *potential factors approach* [100], for the list update algorithms to include deletions as well.

3.2.4 Weighted List Update Problem

The traditional model [50,173,312] for *LUP* may be generalized if we change the cost of the operations that can be performed on the list. We study two further generalizations, the *weighted list* [104,105] and the *paid exchange* (P^d) models [312,292].

For these generalizations, several algorithms with different competitive ratios have been considered.

In the *weighted list update problem* (*WLUP*), any item of the list has an associated cost that depends on the sum of the costs of the preceding items. The goal of the problem is to minimize the overall cost of processing a request sequence and design efficient on-line algorithms.

Two *MTF* versions for *WLUP* are the following:

- The *Counting MTF* (*CMTF*).
  This is a deterministic greedy strategy which uses one real counter per item to decide whether moving the accessed items to the front.
- The *Random MTF* (*RMTF*), which is a randomized version of *CMTF* using biased coins instead of a counter.

It has been shown in [104] that both of the greedy on-line strategies are 2-competitive against a *lazy*[1] adversary (i.e., an adversary that uses an (optimal) *static* arrangement of the list, without resorting the list after each request).

A simple application of the *WLUP* is the *tree update problem* (*TUP*), where items are to be found in the tree instead of in a sequential list. The tree is represented by a list of successors and is searched by a left-to-right depth-first search. Thus, any instance of the *WLUP* can be transformed into an instance of *TUP* using a tree of depth 1. Therefore, *AND-OR trees* and *Directed acyclic graphs (DAGs)* under several visiting algorithms

[1] In the context of *server problems* [254], a *lazy* strategy for the adversary consists in moving as few as possible items (servers) to service requests [104,285]. Clearly, the *lazy* adversary for LUP does *not* move any item of the list.

could exploit efficient solutions for the *WLUP* and should lead to the design of competitive algorithms [103,105].

Furthermore, another generalization has been studied, where the list searches to retrieve *sets* of elements rather than just one item at a time. *D'Amore* [103] presented the following deterministic algorithm *Move-Sets-Front* (*MSF*, for short) for the list update problem, which generalizes the well-known *MTF*.

Algorithm *MSF*

This algorithm moves to the front of the list any accessed set of items, without changing either their relative ordering or that of the other items.

Figure 3.6: A Deterministic On-line Algorithm for *LUP* with *Retrieval Sets*.

It has been shown that *MSF* algorithm is $(1+\beta)$-competitive, both in the *standard* and in the *wasted work models* [103], where β is the unknown maximum size of the sets that will be requested. A randomized version of *MSF* is developed as follows:

Algorithm *BITS* (i.e., *BIT*-for-Sets)

It associates a bit with each element in the list and the n bits are initialized uniformly and independently at random. Whenever one accesses a retrieval set r_j, the bit of the last element of r_j in *BITS*'s list is complemented and if it changes to 1, the accessed set is moved to the front of the list, otherwise it remains unchanged.

Figure 3.7: Algorithm *BITS* for *WLUP* with Retrieval Sets.

Algorithm *BITS* is $(1 + \frac{3}{4}\beta)$-competitive against an *oblivious* adversary both in the *standard* and in the *wasted work* models [103]. Again, both *MSF* and *BITS* algorithms have the same drawback; that is, they handle only successful searches. We can easily modify *MSF* (resp., *BITS*) to get the randomized algorithms *MSF-U* (resp., *BITS-U*), in order to handle successful and unsuccessful searches as well as insertions, but *not* deletions. Easily, the algorithm *MSF-U* (resp., *BITS-U*) has a competitive ratio of $1 + 2\beta$ (resp., $1 + \frac{3}{2}\beta$) against the same models as in the successful case. For these generalizations of the traditional list update problem, some properties of the optimal (off-line) algorithm do *not* hold any more and hence, they provide negative results as well as some general interesting open problems [103,104]. For example: *can we design randomized algorithms that allow us to overcome the difficulties of the deterministic ones?*

Finally, *Luccio* and *Pedrotti* [359] have considered *LUP* in *parallel* computation (*PLUP*), using the *EREW-PRAM model* [210]. The *MTF* strategy has been adopted to solve *LUP* using n processors, one for each list allowing to move items from one list to another. This *parallel MTF* strategy (*PMTF*) is a deterministic (n^2+1)-competitive, while a lower bound is 2n. They showed that randomization helps for *PLUP* drastically reducing the competitive ratio to $\frac{9}{2}n$, versus a lower bound $\frac{3}{2}n$. As a side result, the same competitive ratios (i.e., 2 for the deterministic and $\frac{3}{2}$ for the randomized case) are derived for the *sequential LUP* when n = 1 as we have already known (see [292,312] as well). Thus, it would be interesting to design efficient competitive algorithms for other on-line problems in parallel computation.

3.3 Paging Problem

We consider the *Paging problem* [141,255,261,312,347] which is of fundamental interest among on-line problems. We especially examine three variants on the standard model for the competitive analysis of paging algorithms which allow *randomization*, *weak* and *strong lookahead*.

3.3.1 Problem Motivation and Complexity Results

The *paging problem* is defined as follows: Consider a computer system which has two-level memory, a *fast memory* (or equivalently a *cache* or *hit*) with capacity for k items (representing *pages* or *servers*) and a *slow memory* with unlimited capacity. A set of pages is to be kept in storage at all times where n > k. In response to each request, the requested k pages must be moved into the fast memory and the other n - k pages (*faults*) will reside in the slow memory.

When a program requests access to a page that lies in the slow memory, we say that a *page fault* occurs. It is typically expensive to handle such a page, because some page (or pages) must be evicted from the fast memory to make room for the new page. The *goal* of the paging problem is to choose which pages have to be evicted in order to minimize the *fault rate* (i.e., the number of page faults) that occurs.

In terms of our formulation, a *page replacement strategy* or a *paging algorithm* is *on-line* (resp., *off-line*) if the algorithm chooses which page to evict *without* (resp., *with*) knowledge of future requests.

Here, a *schedule* is the appropriate request sequence of evictions and the number of evictions is the *cost of the schedule*. The cost of strategy S on a sequence **r** for a given size k of fast memory is the cost of the schedule produced by the deterministic algorithm and it is denoted by C_r (S, k). In the case of a randomized paging strategy, the cost of the schedule is a random variable and the cost of the strategy refers to the expected cost of the schedule.

Now, the issue is how we can analyze such on-line algorithms. The classical *worst-case analysis* is useless, because if arbitrary reference sequences are allowed, then an adversary that always references the last discarded page can force any paging algorithm to fault on each reference.

Average-case analysis is also problematic, since it requires a statistical model o˜ the reference sequences. It is extremely difficult to produce a realistic model, since the pattern of access changes dynamically with time and with different applications. Nonetheless, several of the early analyses of paging algorithms were performed in the *independent reference model*, which assumes a fixed probability distribution on the reference sequences [144,307].

Sleator and *Tarjan* [330] used the competitive analysis, which avoids the assumptions of probabilistic analysis and has the power of differentiating paging algorithms. Before we develop and analyze specific paging algorithms using competitive analysis, it would be useful to know that the synthesis of optimal on-line algorithms is, at least theoretically, achievable.

Proposition 6.1. *Generally, it is undecidable if a given paging algorithm* $\mathbf{A}$ *achieves a given competitive ratio.*

Proof: For every i, we could design an algorithm $\mathbf{A}_i$ which follows a known competitive algorithm on the jth request if the *Turing machine* on input i halts in at most j steps, or else it follows a known algorithm with no finite competitive ratio.

The above undecidability result holds as well as for any extended on-line paging problem (e.g., the *k-server* problems).

3.3.2 Paging Algorithms

We consider the following *paging algorithms*:

OPT **or** ***MIN*** **Algorithm:** *Belady's algorithm* [45], which yields an optimal (minimum cost) *off-line scheduling for paging* by evicting the *page* (*item*) whose next request is further in the future.

LRU: *Least-Recently-Used*, which evicts the page that has been requested least recently.

RAND: Whenever a miss occurs, a cache location is chosen at random and the page (item) in it is evicted. The algorithm is *memoryless*[1] but uses *logn* bits of randomness per miss.

FIFO: *First-In-First-Out*, which evicts the page (item) that has been in the fast memory the longest.

FWF: *Flush-When-Full*, which evicts all pages (items) when space is needed.

RFWF: *Random-Flush-When-Full* [254]. Same as *FWF*, except that a random invalid entry is selected for eviction. The algorithm uses n memory bits and up to *logn* random bits per miss.

MARK: *The Marking Algorithm* [141,347], a randomized paging algorithm which evicts a page chosen uniformly at random from the set of pages *not* in the fast memory of *FWF* when memory is needed.

[1] An algorithm with zero memory is deemed *memoryless*.

We can easily verify that all of the above presented strategies, except *OPT*, are on-line and all are *conservative*; a paging algorithm is conservative if the following holds:

(i) *no* evictions before k + 1 distinct pages have been requested, and

(ii) *at most* k evictions have been incurred during any subsequence of requests to at most k distinct pages.

Unfortunately, the above facts are *not* practical and variant paging algorithms that have the same competitive ratio may behave very differently in practice. On the other hand, good paging algorithms, such as *FIFO* and *LRU* are k-competitive, and hence best possible in their model. They have been observed to achieve a page fault rate, on reference sequences that arise in practice, while *LRU* has been almost always superior to *FIFO* [348].

3.3.3 Randomized Paging

Randomization can help on-line paging algorithms. Let k (resp., h) be the fast memory size of an on-line strategy (resp., the *OPT*). Generally, the competitive ratio of an algorithm depends on k and h, where $h \leq k$. For the special case h = k, deterministic on-line algorithms are at best k-competitive, whereas *MARK* is $2 \cdot H_k$-competitive [347].

McGeoch and Sleator [261] have presented a more complicated randomized paging algorithm which has a competitive factor of H_k (the *k-th harmonic number*). On the other hand, *no* randomized on-line algorithm is less than H_k-competitive.

Young [347,348] generalized the above results showing that, when h < k, *MARK* algorithm is $2 \cdot (ln\frac{k}{k-h} - lnln\frac{k}{k-h} + \frac{1}{2})$-competitive if $\frac{k}{k-h} > e$ and 2-competitive otherwise. He also showed that the competitive ratio of any randomized on-line paging algorithm is at least H_k, if h = k[1], and at least $ln\frac{k}{k-h}$ - $lnln\frac{k}{k-h}$ - $\frac{2}{k-h}$, if h < k and $\frac{k}{k-h} \geq e$ [2]. Here, we note that when $\frac{k}{k-h} \leq e$ the analysis of *MARK* shows that its competitive ratio is at most 2.

3.3.4 Paging with Weak and Strong Lookahead

We introduce *two* new *on-line models* of *lookahead* for on-line paging problems and we study their influences on competitive paging algorithms.

According to *Young* [347], a paging strategy is on-line with a *resource-bounded lookahead of size l* (i.e., the intuitive *weak lookahead of size l*). if it sees the present request and the maximal sequence of future requests for which it never incurs more than l evictions on any such request subsequence, where $l \geq 1$ is an integer.

In this model, the paging algorithm is a given *lookahead queue* with known contents which may *either* service the request at the head of the queue (provided there is one) *or* add an additional request (if there is one) to the end of the queue.

Young presented the following randomized on-line algorithm *MARK*(l) with a weak lookahead of size l:

Algorithm *MARK*(l)

At the beginning of each phase execute an *initial step*: Add requests to the end of the queue until either k distinct items or l new requests are in the queue (or there are no more requests). Choose pages (items) uniformly at random from among the pages in fast

---

[1] An alternate proof of the lower bound when h = k is given by *Fiat et al.* [141]. The advantage of this proof is that it generalizes nicely to h < k.
[2] e is the base of natural logarithms.

memory which are not contained in the current lookahead queue and evict these pages. Finally, apply the *MARK* algorithm after this initial step.

Figure 3.8: A Randomized Paging Algorithm with a Weak Lookahead.

MARK(l) algorithm is $max\{\frac{2k}{l}, 2\}$-competitive, while its deterministic version *DMARK(l)*, which only allows arbitrary choices of items, has a competitive ratio of $2\cdot(ln\frac{k}{l}+1)$ [347].

However, the model of *weak lookahead* is *not* realistic in practice, but it is theoretically interesting and leads to reduced competitive ratios. The goal is to find a new model which has both realistic as well as theoretical interest and can significantly improve the competitive ratios of on-line paging algorithms.

A paging algorithm is *on-line with strong lookahead*[1] *l* if it sees the present request and a sequence **r** = (r(1), r(2),...,r(m)) of m future requests that contains *l* *pairwise distinct* pages, where r(t) denotes the request at time t.

Now, all on-line *lazy*[1] paging algorithms can be easily extended to a new model of *strong lookahead* of size $l \le$ k-2, where k $\ge$ 3 is an integer. For example, the deterministic *LRU(l)* (or the randomized *MARK(l)*) paging algorithm with *strong lookahead* $l \le$ k-2 is (k- *l*)-competitive (resp., $2\cdot H_{(k-l)}$-competitive) *only* against the *oblivious* adversary.

Furthermore, all *lazy* on-line algorithms with *strong lookahead* can be simply generalized, if the algorithms do *not* use full lookahead but rather serve the request

[1] More practical on-line models might be considering *loose competitiveness* of strategies with regular lookahead [347], assuming an *average* (rather than consistent) weak lookahead of size *l*, or assuming that the sequence is fixed by an adversary of the lookahead.

sequence in a series of *blocks* instead of list items only. For example, the new obtained *LRU(l)-B* and *RANDOM(l)-B lazy* paging algorithms, with *strong lookahead l* ≤ k-2 using blocks, are (k-*l*+1)-competitive. The above result shows that *LRU(l)* (resp., *LRU(l)*-B) is optimal (resp., nearly optimal).

Clearly, if *l* = k-1 and the total number of different pages in the memory system equals k+1, then *LRU(l)* is 1-competitive because it behaves like *Belady*'s optimal paging algorithm *MIN*. On the other hand, the competitive ratio of the *(lazy) RANDOM(l)-B* algorithm does not achieve any improvement upon the previously presented *RANDOM(l)* algorithm with *strong lookahead l* ≤ k-2.

Finally, *Raghavan and Snir's* results [285] can be extended as follows:

Theorem 3.1. *Let $l \geq k-2$ with $k \geq 3$, $k \in \mathbb{Z}^{+}$ and let A be a deterministic (or randomized) on-line lazy paging algorithm with strong lookahead l. If A is c-competitive, then $c \geq (k-l)$ (resp., $c \geq H_{k-l}$) against only the oblivious adversary.*

Proof: The proof is similar to *Raghavan*'s proof [285] by also applying *Yao's min-max principle* [344].

The proofs of all above generalized results are almost similar to those of the *weak lookahead* and are hence omitted. Actually, these upper bounds can be slightly generalized using *Young's extension* results [347], but they are weak and therefore their corresponding on-line paging algorithms seem not to take full advantage of the strong lookahead. It is surprising that the advantage of lookahead was not simply a *tradeoff* in *l*, but rather produced a threshold effect. Furthermore, the point at which lookahead becomes an advantage is quite high.

3.3.4 Competitive Distributed Paging

In this section we study the competitive analysis of algorithms for *on-line paging problems in* a *distributed environment.* Especially, we deal with the *file* (or *page*) *migration* and *replication* problems, as well as the more extended abstract *data file allocation* (or *assignment*) problem.

The *file allocation problem* (*FAP*) [40] is the distributed memory management problem for a globally addressed share memory of large multiprocessor systems, which typically limited local memory capacity. A global shared memory in multiprocessor system is modeled by distributing the indivisible blocks such as *physical files* (*pages*) among the local memories. However, a full file may be replicated in various processors throughout the network at a cost equal to the distance traveled times the *page size factor D* and discarded over time under the following *assumptions*:

- *At least* one copy of every file must be stored somewhere in the network; and
- the multiple copies must be kept in *consistency* (i.e., files cannot be split among processors).

The objective is to device *residency* on-line strategies (i.e., in the presence of on-line and unpredictable access pattern) that decide which *local* memory should have the copy of a *readable* and *writable* file requests so as to optimize the total communication cost in processing a sequence of file-accesses.

The *file allocation* problem is the simultaneous solution to two partial problems, the *page migration* and *page replication* problems [58]. *FAP* collapses to *page migration* (resp., *replication*) problem if *only* **writes** (resp., **reads**) occur. Clearly, the page replication problem is a fundamental on-line problem whose simplest case corresponds to *the ski rental* problem.

The problem of designing *efficient file allocation algorithms* has been studied from both the practical and theoretical point of view [41,47,58,85,226,354]. We study the transformation of some *standard* or *centralized model* [204,313] (i.e., using only global information of the system) into the more realistic *distributed model.*

Black and Sleator [58] have considered an *optimal* deterministic 3-competitive algorithm for the *migration* problem on *trees, uniform* networks and *metric* spaces. They have also showed that *no* deterministic on-line strategy could be better than 3 (resp., 2)-competitiveness for the *page migration* (resp., *replication*) problem on any metric space of three points.

Chrobak et al. [85] have proposed the following *randomized, migration algorithm* when the *uniform* metric space M has *only* two points, x and y.

Algorithm *RAND-MIGRATION*

Suppose the current *offset function*[1] $(w(x), w(y)) = (0, \beta)$ (symmetrically, if the offset function is $(\beta, 0)$), where $0 \le \beta \le D$. This algorithm uses the probability distribution that places mass $p_\beta = \dfrac{D+\beta}{2D}$ on x and $1 - p_\beta = \dfrac{D-\beta}{2D}$ on y.

Figure 3.9: A randomized Page Migration Algorithm for any Metric Space of Two Points.

[1] An *work function* [58,89] whose infimum value is zero.

Algorithm *RAND-MIGRATION* achieves a competitive ratio of $C_D = 2 + \frac{1}{2D}$ which is *optimal* for the *page migration* problem on a metric space of *two* points. This algorithm can also be extended to a randomized C_D-competitive strategy for a *uniform* metric space and any *tree* [85].

The following *Table 3.1* summarizes the *randomized, distributed competitive, migration* algorithms against *oblivious* adversaries (otherwise, it is specified) and their performance ratios.

Network topology	Competitive ratio	Reference
Any network	$1 + \phi = 2.61$ [1]	[85]
Uniform networks	$((5 + \sqrt{17})/4) = 2.28$	[85]
Metric spaces of 2 points	C_D [2]	[85]
Continuous trees	C_D	[85]
Hypercube and *meshes* (in L_1 metric)	C_D	[85]
Metric space of *3* points	3	[58,85]

Table 3.1: Randomized Page Migration Algorithms and their Competitive Ratios.

Next we describe a randomized ρ_D-competitive algorithm for replication problem on *trees* and *uniform* networks, which is optimal for all values of *page size D*, where $\rho = \frac{D+1}{D}$ and $\rho_D = \frac{\rho^D}{\rho^D - 1}$.

[1] ϕ is the *golden ratio* and the on-line algorithm is against an *adaptive* adversary.

[2] $C_D = 2 + \frac{1}{2D}$, where D is the *page size factor*

Algorithm *RAND-GEOMETRIC*

Choose a random number i from the set $\{1,2,\ldots\ldots, D\}$ with probability $p_i = \alpha\ \rho^{i-1}$, where $\alpha = \frac{\rho - 1}{\rho^D - 1}$. Process the request sequence and maintaine a *count* (initially zero) on each edge of the tree. If there is a request at node υ that does not have the page, then all counts along the path from υ to the closest node with the page are increased by 1. When a count reaches the value of the randomly chosen number, the page is replicated to the child node of the corresponding edge.

Figure 3.10: A Randomized Page Replication Algorithm for Trees and Uniform Networks.

The above *RAND-GEOMETRIC* algorithm can easily be extended to a $2\rho_D$-competitive strategy for a *ring* by cutting it at the point *opposite* (or *uniformly at random*) to the starting node of the ring which initially has the page. We observe that $\lim_{D\to\infty} \rho_D = \frac{2e}{e-1} \approx 3.16$ (i.e., e is the *natural logarithmic base*). Moreover, if we don't use the *only* one random number which is used during the initialization step, then the above algorithm becomes a completely *deterministic*, *4*-competitive strategy for replication problem. *Koga* [226] has also presented another interesting on-line replication algorithm *COINFLIP* which achieved a competitive ratio of 2 for *trees* and 4 for *rings*.

In the following *Tables 3.2* and *3.3*, we summarize the competitive performances of the *replication algorithms* against an *oblivious* adversary.

Network topology	Competitive ratio	Reference
Trees and *uniform* networks	2	[58]
Trees and *uniform* rings	$\frac{e}{e-1} \approx 1.58$	[4,356]
Ring	4	[47]
Ring[1]	3.16	[4]
Any network topology	7	[29]

Table 3.2: Deterministic Page Replication Algorithms and their Performances.

Network topology	Competitive ratio	Reference
Trees	$(1+\frac{\sqrt{2}}{2}) \approx 1.71$	[226]
Rings	$2\cdot(2+\sqrt{3})$	[41]
Circles	4	[92,226]
Rings[2]	4	[226]

Table 3.3: Randomized Replication Algorithms and their Competitive Ratios.

Awerbuch et al. [29] have proposed a various centralized, deterministic migration algorithm on arbitrary network:

Algorithm *MTM* (i.e., Move-To-Min.)

Divide the request sequence into phases. Each phase consists of D consecutive **write** requests at processors $p_1, p_2, \ldots\ldots, p_D$. During a phase the algorithm doesn't move the copy of the file. At the end of phase, migrate the copy to processor p_m in the network such that

$$\sum_{i=1}^{D} d(p_m, p_i) \text{ is minimized.}$$

Figure 3.11: A Centralized Migration Algorithm on Arbitrary Networks.

[1] Either a *deterministic* or *memoryless* algorithm.

[2] Against an *adaptive* adversary.

Theorem 3.2. *Algorithm MTM is 7-competitive on arbitrary network topologies.*

Proof sketch [29]*:* We show that $\Delta\Phi \leq 7\cdot Cost_{Adv} - Cost_{MTM}$ using the potential function $\Phi = 2D\cdot d(\alpha_0, \beta)$, where α_0 (resp., β) denotes the position of the *adversary's* (resp., the *on-line*) copy at the beginning of a phase.

The same authors [29] extended the above algorithm *MTM* for the *FA* problem which is the simultaneous solution to both *migration* and *replication* problems. They proposed a *centralized FA* (i.e., *CFA* for short) (resp., a *distributed FA* (*DFA*)) algorithm which is $O(log\,n)$-competitive (resp., $O(log^4 n)$-competitive).

Recently, *Bartal et al.* [40] presented a simple distributed version of the deterministic *FWF* [204] and those of the randomized *MARK* algorithm [141] for the file *allocation* problem on specific network topologies (e.g., *trees* and *uniform* networks).

Furthermore, *Awerbuch et al.* [28] proposed a new randomized competitive distributed *paging* algorithm (so called *Heat & Dump*) against *oblivious* adversaries for *uniform* networks, whose competitive ratio was logarithmic in the local storage capacity.

We observe that all results on the performance ratios of the *distributed paging algorithms* demonstrated the *power of randomization* for the *page migration, replication* and *file allocation problems*. Additionally, some important *open questions* for these problems are the following:

- Consider *various assumptions* for these problems in order to address real-life concerns (e.g., issues regarding *delay* and *congestion*); and

- Close the gaps left in the *upper* and *lower* competitive bounds for arbitrary networks.

In conclusion, we like to point out that the general structure of combining deterministic and randomized algorithms, with a minimum competitiveness, is a promising tool for designing new efficient on-line strategies.

3.3.5 Recent Related Results of the Paging Problem

Recently, considerable work has been done to competitive analysis of on-line algorithms in order to extend the *Paging Problem* and improve their lower competitive bounds.

Borodin et al. [65] considered the paging problem on *restricted classes* of inputs that occur in practice. In their work "*Competitive paging with locality of reference*", they assume that an on-line algorithm knows in advance if the input it will receive falls in a particular class. In this sense, the problem is less "*on-line*", because it restricts the arbitrariness of the adversary in generating a sequence of requests. The *access graph*, a model of a program's reference patterns, has been developed to determine a restricted class of inputs. Many classical algorithms (e.g., *LRU*, *FIFO* and *marking algorithms*) of paging problem on the *access graphs*, (also, on their further extension to *directed* and *structured graphs* [205]), have been reanalyzed deriving useful properties and nice lower bounds on their competitiveness.

Feuerstein et al. [137] studied another extension of the paging problem to graph problems. This includes the *Path paging* and *Connectivity paging* problems in graphs, which, besides their theoretical interest, have significant applications to the *memory* *management problem* of data structures for graphs. An important issue would be to extend these results to *weighted versions* of paging problems making them more applicable in practice.

Finally, in the next section, we will study two other extensions of the paging problem, the *problem of maintaining caches* in a multiprocessor system [254] and the *k-server problem* [255]. These problems are based on more general and complex models, but they share essentially the same fundamental *serving* (*paging*) property (i.e to serve (page) the request).

A great truth is a truth whose
opposite is also a great truth.

Christopher Morley

Chapter 4

The k-Server Problem and Algorithms

The interest of science lies in the art of making science.
Paul Valéry

In this chapter we introduce two generalizations of the *paging problem*, the *weighted caching* and *k-server problems*. Particularly, we enumerate the weighted caching and k-server algorithms summarizing relevant previous work.

Furthermore, we present some new results about the *strong competitiveness* of the 2-server problem against a *lazy* adversary and we extend *Coppersmith et al.* theory on resistive graphs [98,99] to *non-resistive spaces* (i.e., *no symmetry* of the edge weights (costs)). We develop methods for the synthesis of the random walks, and use them to design competitive randomized on-line algorithms for the *k-server problem* and its well-known related problems (i.e., task systems and cat-mouse game) on non-resistive spaces.

4.1 The Weighted Caching and k-Server Problems

4.1.1 The Statement of the Problems

The *weighted caching problem* is a generalization of the paging problem in which the cost of evicting an item (page) r_i is a non-negative function $W(r_i)$ of the item (i.e., the costs of moving different items into the cache differ). This scheduling problem as well as the *disk-head motion planning problem* [254,255] were first introduced by *Sleator and Tarjan* [312].

The *k-server problem* is a further generalization and was first formulated by *Manasse et al.* [255]. In this problem, the cost is a non-negative function $d(r_i, r_j)$ of the item r_i evicted and the item r_j requested, and the fast memory is assumed to be initially full. Except for the special case of weighted caching, the distance d is assumed to be *metric* (i.e., symmetric, satisfying both the triangle inequality and $d(r_i, r_j) = 0$ for every $i \neq j$).

The famous k-server problem is an appealing special case of metrical task systems and has been one of the most extensively studied on-line problems in the past several years. A reason for the interest is that the k-server problem is a natural abstraction of *paging*, *weighted caching* and *planning the movement of diskheads*, where k mobile servers reside in a metric space [254]. In addition, this problem is both practical and simple to be defined.

The k-server problem may be transformed into the following *network problem*. There are k servers which are free to move around from point ("request") to point in a metric space, and each request must be serviced by some server moving to cover the corresponding point in the space. For simplicity, we can assume all servers to be on some arbitrary points initially.

The cost of a k-server (on-line or off-line) algorithm is the total distance traveled by the servers. A dynamic programming algorithm can be used to compute the cost of the optimal off-line algorithm handling a request sequence [255].

If the metric space is the *uniform (or unit) metric space* U_n with *n* points (i.e., the distance between any two points is 1), then the k-server problem reduces to the paging problem with points in the space corresponding to pages (items) of slow memory and servers corresponding to page slots in fast memory.

4.1.2 Weighted Caching and k-Server Algorithms

Many natural algorithms for the k-server problem fail to achieve a bounded competitive ratio. For example, consider the following *greedy algorithm*: "Answer each request by moving the closest server". In any metric space where arbitrary small positive distances occur, the greedy algorithm can be defeated by placing requests alternately at two points that are sufficiently close together. The greedy algorithm will construct an unbounded cost by shuttling the same server back and forth forever. On the other hand, the performance ratio of optimal off-line algorithms can be bounded by stationing a server permanently at each of the two points on the same request sequence.

We consider the following *weighted caching* and k-*server algorithms*:

OPT: The algorithm that produces an optimal k-server or weighted caching schedule.

BALANCE: The *Balance algorithm* or *BAL* [254,255,347] for k-servers:

Algorithm *Balance*

For each server the algorithm maintains the total distance it has moved, since the start of the request sequence.
If the server is currently at point i, the distance traveled by i so far is denoted by W_i.

Now consider a request at a vertex j.

- **If** j is already covered by a server, ***then*** *BAL* does nothing.
- **If** j is not covered, ***then*** *BAL* moves the server i to the point j, where i is chosen to minimize $W_i + d(i, j)$.

Figure 4.1: The Algorithm Balance (*BAL*) for k-Server Problem.

In other words, *BAL* moves any server that would have the smallest cumulative cost after moving. As indicated by its name, the balance algorithm tends to use all of its servers equally.

> ***GREEDYDUAL***: The *greedy dual algorithm* [347] for *weighted caching*:
>
> The algorithm maintains values (*credits*) on the servers. Initially the value of server is the weight of the node it serves. When an unserved point ("request") is requested, the server values are decreased by the minimum server value, some zero-valued server is moved and its value is raised to the weight of its new point. When a served point is requested, the server value is reset anywhere between its current value and the weight of its point.

The *GREEDYDUAL algorithm* may be described as follows:

Algorithm *GREEDYDUAL*

Each server has a varying amount of credit. In response to request 0, all servers are placed on r_0 with no credit. In response to each subsequent request j to node r_j,

1. **If** node r_j has no server:
   a) Each server's credit is increased equally until some server has enough credit to move to r_j. (If a server is currently on r_i, it must have $d(r_i, r_j) = w(r_i)$ credit to move to r_j.)
   b) One such server serves request j, giving up all its credit.
2. **If** node r_j has at least one server:
   a) One such server serves request j.
   b) Unless the server has not yet moved, it gives up an arbitrary amount (possibly none) of its credit.

Figure 4.2: The *GREEDYDUAL* Algorithm for Weighted Caching Problem.

The performance of on-line algorithms for weighted caching and k-server problems have been analyzed using *competitive analysis*. *Manasse et al.* [255] show that *no* deterministic on-line k-server algorithm is better than ($\frac{k}{k-h+1}$)-competitive[1] in *any metric space* (or graph with symmetric edge weights satisfying the triangle inequality) with at least k+1 points. *Chrobak et al.* [87] have shown independently, that balance algorithm (*BAL*) is at least k-competitive (when h = k) for the general k-server problem in *any* metric space with at least k+1 (distinct) points. The proof uses a nice averaging technique:

Proof idea:
For every on-line algorithm $\mathcal{A}$, the adversary constructs a sequence such that there are k different algorithms, which have a total cost equal to $\mathcal{A}$'s cost. Thus $\mathcal{A}$'s cost is at least k times greater than the cost for one of these algorithms. Since the lower bound holds for any metric space with at least k+1 distinct points, the proof is similar to the proof that the competitiveness of any deterministic paging algorithm is at least k. $\Box$

The proof can easily be extended for randomized algorithms against an adaptive on-line adversary.

GREEDYDUAL is a new algorithm that generalizes *LRU*, *FWF*, *MARK* and *BAL* with optimal ($\frac{k}{k-h+1}$)-competitiveness for *weighted caching*. This algorithm is of practical interest and gives the *first result* we know of showing reduced competitiveness when h < k for any problem other than paging. *GREEDYDUAL* is a primal-dual, deterministic, on-line weighted caching algorithm of theoretical interest as well, because the motivation by the discovery of a general technique (so called the *Primal-dual bounding technique* [347]) is implicit in the analysis of the algorithms it generalizes [347].

[1] Remember that *h* refers to the fast memory size of the optimal off-line algorithm OPT.

A *sketch of the primal-dual bounding technique* is the following:

> "We formulate the problem as an integer linear program (*ILP*), so that each solution to the problem of *ILP* yields a linear program (*LP*) (which, incidentally, has *optimal* integer solutions) of equal cost. The cost of any feasible solution to the dual of this *LP* is a bound of the optimal cost".

The *GREEDYDUAL* implicitly generates a solution to the dual program (*DP*) of *LP*. The *goal* of the dual solution is actually two-fold:

- *GREEDYDUAL* uses the structural information that the solution provides about the problem instance to guide its choices, and
- the cost of the dual solution of this linear program can be used as a lower bound and also correlates it with the on-line algorithm to show competitiveness.

Primal-dual technique is also important for *approximation problems* [272], including on-line problems, because it helps reveal combinatorial structure, especially how to bound optimal costs. This approach has been explicitly used for finding approximate solutions to *NP-hard connectivity problems* [160].

Generally, duality has been used to obtain lower bounds on the complexity of randomized algorithms [100], on randomized communication complexity [247] and in other contexts; for example *Von Neumann's min-max Theorem* for zero-sum games may be viewed as a special case of linear programming duality [344]. In particular, we shall use this technique to reanalyze the *weighted matching on-line algorithms* (see section 5.2).

4.1.3 More Related Work

There has been considerable work on *k-server problems*. *Manasse et al.* posed the following famous conjecture:

The k-Server Conjecture: *In any metric space there is an on-line algorithm which is k-competitive.*

They also gave an elegant proof that *no* deterministic algorithm can be better than k-competitive.

Much excellent work has been done (e.g., see [82,191]) in attempt to solve the k-server conjecture which has been verified *only* for k = 2 by the present time. It has been open for some time to find a general algorithm for k-server problems such that there is a function of k which bounds the competitiveness of this algorithm in any metric space. As a result, researchers mostly turn to special cases (e.g., to restricted metric spaces).

Manasse et al. [255] presented optimal k-competitive algorithms for k-server problems in any metric space with n points, when k = 2 or n-1. However, implementing their algorithms requires space linear in, and time quadratic in, the minimum of the number of requests seen so far and the number of points in the metric space. It is more desirable to have an algorithm the time and space complexity per request of which is a function of the number of servers.

Irani et al. [191] and later *Chrobak et al.* [82,84] showed two algorithms that only maintain one variable and only perform a constant number of operations to decide which server has to service a particular request. Both algorithms have a constant competitiveness for the 2-server problem.

Chrobak et al. [87,285] showed an optimal k-competitive algorithm when the metric space is the *real line*. The algorithm is very simple and follows:

Algorithm *Real-line*

Upon a request to a point i, if i is to the left or to the right of all the servers, just move the closest server. Otherwise move one server directly to the left and another directly to the right of i at the same speed. When one of the servers reaches i, then servers stop.

Figure 4.3: A k-Server Algorithm for a Real Line

This k-competitive on-line algorithm can naturally be extended for k servers on *trees* as well [86]. Algorithm *GREEDYDUAL* for the weighted caching problem appears to be closely related to the above algorithm.

Fiat et al. [142] first showed a randomized algorithm, so called *expand-contract*, whose the competitiveness is bounded by an $O(k \cdot \mathit{log} k)$ exponential function in a metric space. This algorithm is defined recursively in terms of l-server problem for $l < k$, whose base case is simply the greedy l-server problem. Later on, they used an interesting technique [141,142], which is essentially a *MIN generator* over on-line server algorithms, to prove the upper bound on the competitiveness of *expand-contract* algorithm.

Next, *Raghavan* and *Snir* [285][1] presented a very simple and practical algorithm *harmonic* for k-servers in any metric space., while *Grove* [163] proved that the competitiveness of this algorithm is $(\frac{5}{4} k \cdot 2^k - 2 \cdot k) \in O(k \cdot 2^k)$. This result is the best competitive bound of any algorithm for k-server problem in a general space by the present time. It is also *conjectured* that the correct competitive ration of the *harmonic* algorithm $O(2^k)$ (e.g., see [56,285]).

Finally, *Coppersmith et al.* [98,99] obtained a randomized k-competitive algorithm for the k-server problem in finite *resistive spaces*. It is interesting for us to extend their

[1] See paragraph 4.2.2 as well.

results and show that randomized $k\cdot\Psi'(C)$-competitive algorithms exist against the adaptive on-line adversaries on finite *non-resistive spaces*.

The following two tables summarize all the competitive *upper* bounds of on-line algorithms for special cases of the k-*server problem* known in the present literature.

Competitive Upper Bounds for k-Server Problem

Deterministic On-line Algorithms

Competitive ratio	Special Case	Sources
2	$k = 2$	[82,84,191,255]
k	$k = n-1$	[254,255]
k	Points on a *line*	[88,285]
$4k^2$	Points on *discrete circle*[1]	[39,285]
$12k^3 + 4k^2 + 4 \in O(k^3)$	Points on a *(continuous) circle*[2]	[139]
k	*Weighted Cache*	[88,254]
k	Points on a *tree*	[86]

Table 4.1: Deterministic On-line Algorithms for k-Server Problem

Randomized On-line Algorithms

Competitive ratio	Special Case	Sources
3^{17000}	$k = 3$	[5 [illegible]
3	$k = 2$	[82,84,191]
k	*Resistive* graphs	98,99,285]
$k\cdot\Psi'(C)$ [3]	*Non-resistive* graphs	[This paper]
2k	Points on a *discrete circle*[4]	[98,285]
$O(k\cdot \log k)$	Any *metric* space	[142,347]
$k\cdot(\frac{5}{4}\cdot 2^k - 2) \in O(k\,2^k)$	Any *metric* space	[163]

Table 4.2: Randomized On-line Algorithms for k-Server Problem

---

[1] A *discrete circle* is a metric space that consists of a *finite* subset of the circle points.

[2] On the contrary, the *(continuous) circle* consists of an *infinite* set of points.

[3] $\Psi'(C)$ is the edge *offset ratio*, while the on-line algorithm is *memoryless* against a *lazy* adversary.

[4] In this case, the algorithm is against an *adaptive on-line* adversary

Theorem 4.2. *There exists a unique*[1] *randomized c-competitive on-line algorithm against any adversary for any 2-server problem with $c \leq 2$. Furthermore, if the adversary is lazy, then the equality holds (i.e., $c = 2$).*

***Proof*:** First, we compute the transition probabilities for any random walk of a 3-node graph (i.e., without loss of generality for a n-node graph). These probabilities are *unique*. We conclude that the expansion factor of the determined random walk against a lazy adversary is 2. Note that on the larger cycle of the n-node graph, the expansion factor is always bounded by 2 (i.e., also by *Corollary 4.1*: a generalization).

Let P be a 3×3 matrix of transition probabilities and let H be a 3×3 matrix of hitting times. Assuming edge weight symmetry, elementary probability theory yields the following three 2×2 linear system of the commute times

$$(H_{ij} + H_{ji}) \;=\; 2\cdot(d_{ij} + d_{ji}) \;=\; 4\cdot d_{ij} \text{ for } 1 \leq i, j \leq 3 \text{ and } i \neq j,$$

which have a *unique* non-negative solution in terms of the transition probabilities.

In addition, we find the following equations of hitting times circles on the 3 nodes:

$$(H_{12} + H_{23} + H_{31}) \;=\; 2\cdot(d_{12} + d_{23} + d_{13})$$

$$(H_{13} + H_{32} + H_{21}) \;=\; 2\cdot(d_{13} + d_{23} + d_{12})$$

Clearly, if the adversary is lazy, the expansion factor over all cycles, not just 2-cycles (or commutes) or 3-cycles, is exactly 2. This occurs, because the hitting time H_{ij} from i to j is composed of 2 parts and equals $2\cdot d_{ij}$ (i.e., exactly what we want!).

Next, we show that the random walk has an expansion factor of 2 even if the adversary is *non-lazy*. Let us denote $S_i(j)$ as the set of all adversary algorithms where the adversary makes no more than i moves, at most j of which are *non-lazy* moves. We can easily see that for each $l \in S_i(j)$, there exists an $l \in S_i(j-1)$ such that the adversary can always replace an ith non-lazy move with a lazy move without decreasing the expansion factor of its sequence. Thus, we can replace the strategy from $S_i(j)$ with a strategy from $S_i(j-1)$ without loss to the adversary.

[1] The uniqueness is in terms of an *only one* probability matrix.

Given a network R = (r_{ij}) of resistors (a network C = (C_{ij}) of conductances where edge weight $C_{ij} = \frac{1}{r_{ij}}$), we can define the probability matrix for the random walk by $P_{ij} = C_{ij} / C_i$ where $C_i = \sum_j C_{ij}$ and $1 \le i, j \le n$.

Let R_{ij} denote the *effective resistance* between vertices *i* and j (i.e., a *unit voltage* ϕ_{ij} between *i* and *j* in this network of resistors results in an electric current of $\frac{1}{R_{ij}}$). We require that the support graph to be connected so that the effective resistances will be finite.

Definition 4.1. A cost matrix C = (C_{ij}) is *resistive* if it is the matrix of effective resistances obtained from a connected non-negative *symmetric* real matrix (G_{ij}) of conductances. The matrix (G_{ij}) is the *resistive inverse* of C.

Definition 4.2. A stochastic cost matrix P = (P_{ij}) is *ergodic* if any state can be reached from any other state; we call the corresponding random walk an *ergodic walk*. Then a non-negative real cost matrix P is *reversible* if for all *i*, *j*, we have $W_i \cdot P_{ij} = W_j \cdot P_{ji} = C_{ij} / \Delta$ (i.e., *symmetry* of the edge weights), where W_i is the stationary probability of being in the *i*th state (node) and $\Delta = \sum_{i,j} C_{ij}$.

Conversely, given a *Markov chain* defined by a reversible *ergodic* probability matrix P, we can get the corresponding electrical network by taking $C_{ij} = W_i \cdot P_{ij}$.

Chandra et al. [71] extended the above work to arrive at new bounds for *commute* and *cover times* for random walks. They used the *harmonic* probability distribution, which was defined by *Doyle and Snell* [121], to get a system of linear equations of the *hitting times* H_{ij} (i.e., the expected length of a walk that starts at node *i* and ends on the first reaching node j). These linear systems have ***unique*** solutions and turn out to be identical if we identify the voltages ϕ_{ij} with hitting times H_{ij}.

Using the same argument twice we can easily get the following equation:

$$H_{ij} + H_{ji} = 2m \cdot R_{ij} = \Delta \cdot R_{ij} \qquad \textbf{(4.1)},$$

where $H_{ij} + H_{ji}$ is the commute time, m the number of edges and R_{ij} is the ***effective resistance*** between nodes i and j. This result establishes the close relation between commute times for the simple random walk on G and effective resistances in the electrical network R.

If we think of edge weights d_{ij} (i.e., the distance between nodes i and j) as vectors, then the ***harmonic*** random walk as defined in [71] has transition probabilities $P_{ij} = \frac{1/d_{ij}}{\sum_{j \neq i} 1/d_{ij}}$ by making use of the notion of the *expansion factor* or *stretch*[1] H_{ij} / d_{ij} of the random walk from i to j. Clearly, if we use the commute time of $2m \cdot R_{ij}$ as an upper bound on the hitting times H_{ij}, we conclude that the ***harmonic*** random walk has an ***expansion factor*** of at least $2m$.

Let P denote the transition probability matrix of size $n \times n$ of an ***ergodic markov chain*** with stationary distribution W. Let $P_{ii} = 0$ [2] for all i, and let $H = (H_{ij})$ denote the expected first-passage-matrix of hitting times for the above chain.

Lemma 4.1.[3] $\sum_{i,j} W_j P_{ji} H_{ij} = n - 1,$ *for* $1 \leq i, j \leq n.$

---

[1] Similarly, we can define the *stretch* of a random walk over a *path* or a *cycle*.
[2] This condition is *not* needed in the case of *non-resistive spaces*.
[3] This lemma also holds for *non-resistive spaces*.

Proof: $\sum_{i,j} W_j P_{ji} H_{ij} = \sum_j W_j (\sum_i P_{ji} H_{ij}) = \sum_j W_j (H_{jj} - 1) = \sum_j W_j (\frac{1}{W_j} - 1) = n - 1$, since $H_{jj} = \frac{1}{W_j}$.

Foster's Theorem [99] suggests that $\sum_{i \leftrightarrow j} \frac{R_{ij}}{r_{ij}} = n - 1$, where $i \leftrightarrow j$ denotes that the nodes are connected by a finite r_{ij}. We can very easily show, using the *formula (4.1)* and because P is reversible, that $\sum_{i,j} W_j P_{ji} H_{ij} = \sum_{i<j} \frac{R_{ij}}{r_{ij}}$. Thus, *Lemma 4.1*, implies *Foster's Theorem.*

Given P as above, we define $\overline{P}$ to be the following (n-1) x (n-1) matrix. Let $\overline{P}_{ii} = W_i$ ($= \sum_{j=1, j \neq i}^{n} W_i P_{ij}$), and $\overline{P}_{ij} = - W_i \cdot P_{ij}$ for $1 \le i, j \le n-1$. Further, let $\overline{H}_{jj} = H_{jn} + H_{nj}$, and $\overline{H}_{jk} = H_{jn} + H_{nk} - H_{jk}$, for $1 \le j, k \le n-1$. We then claim the following generalization of the *resistive inverse identity* which is well known in electrical network theory (e.g., see in [71,121]).

Lemma 4.2. $\overline{P} \cdot \overline{H} = I_{n-1}$, *where* I_{n-1} *is the identity matrix of size* $(n - 1) \times (n - 1)$.

Proof: By elementary *theory of linear algebra* and using the triangle inequality for hitting times (see [143] for more details).

4.2.2 The Harmonic Algorithm for the k-Server Problem

We have seen that the simple greedy algorithm, which always chooses the closest server, is easily failed by an adversary because of its predictability, and fails to achieve a bounded competitive ratio. On the other hand, an efficient competitive on-line algorithm for the k-server problem should be obtained if we choose our servers to be close to the request points.

Raghavan and Snir [285] presented a very natural, *memoryless*[1] algorithm, called *Harmonic algorithm*, which is defined as follows:

Algorithm *Harmonic*

Let $d_1, d_2, \ldots, d_k$ be the distance of each server from the current request. Send server *i* with probability $(\frac{1}{d_i}) / (\sum_{j=1}^{k} \frac{1}{d_j})$, which is inversely proportional to that server's distance from the request point.

Figure 4.4: Harmonic Algorithm for k-Server Problem

Raghavan and Snir also showed that harmonic algorithm is 2-competitive and $(n-1)^2$-competitive against a non-adaptive adversary when k = 2 and k = n-1, respectively. They did *not* see the usefulness of the analysis of the relationship between random walks and server problems (or expansion factors and competitive factors) as being restricted to k-node graphs. They showed that harmonic algorithm is $2 \cdot \binom{k}{2}$-competitive against a *lazy* adversary in any metric space with k points. A *lazy* adversary is relatively simple and it is restricted to requesting a point that is occupied by an off-line server but *not* by an on-line server if such a point exists. It can easily be proven that the competitiveness of the harmonic algorithm is $\binom{k}{2}$, because the competitiveness of the server algorithm is clearly bounded above by the largest expansion factor of all phases.

---

[1] As the name suggests, tl algorithm does not maintain any state information.

Furthermore, *Raghavan and Snir* conjectured the following:

> ***Lazy Adversary Conjecture* (*LAC*)**: *The following (strong) adversary strategy results in the poorest performance for memoryless algorithms: Whenever there is a point in the space at which the adversary has a server but we have none, the adversary presents a request at that point (instead of making a move and incurring a cost).*

They also claimed (see [285], pp. 701, *Theorem 18*: its *proof is omitted!*) that even without *LAC* they could bound the competitive performance of the harmonic on-line algorithm in an arbitrary metric space for the *2-server* problem in the *interval* [3,6].

Manasse et al. [254,255] gave a deterministic 2-competitive on-line algorithm for the 2-server problem against any adversary and therefore the above claim is wrong even in the randomized case.

Theorem 4.1. *The strong competitiveness ratio of the harmonic algorithm for the 2-server problem is in the interval (1,3] (not in the interval range* [3,6]).

Proof: Clearly, the harmonic algorithm against a lazy adversary has a competitive ratio (or an *expansion factor*) bounded above by 3 (also, by *Lemma 4.1*). As we have seen, a game against a lazy adversary always proceeds in a series of phases. Assuming that a phase starts with k = 2 servers and those of the adversary overlapping on node 2. The adversary requests node 1 and moves there with his server from node 3. Thus, the server pays the cost d_{31}.

We proceed using the harmonic algorithm against the lazy adversary, until finally we answer a request with the server from node 3, and those of two servers overlapping on nodes 2 and 1, then end this phase. The amount that has been paid is the expected cost H_{13} of a random walk from 1 to 3.

We get the following 2 x 2 system of equations:

$$\begin{cases} H_{13} = P_{13} \cdot d_{13} + P_{12} \cdot (d_{12} + H_{23}) \\ H_{23} = P_{23} \cdot d_{23} + P_{21} \cdot (d_{21} + H_{13}) \end{cases}$$

By solving the above system of equations and using the symmetry of the transition probabilities p_{ij} (where $1 \le i, j \le 3$) which are given by the harmonic algorithm, we have that

$$H_{13} = \frac{2d_{13} \cdot (2d_{23} + d_{12})}{d_{12} + d_{13} + d_{23}}.$$

The above formula can be rewritten as

$$H_{13} = 2d_{31} \cdot \left(\frac{d_{12} + d_{13} + d_{23}}{d_{12} + d_{13} + d_{23}} + \frac{d_{23} - d_{31}}{d_{12} + d_{13} + d_{23}}\right).$$

Thus, the expansion factor for the random walk and hence the competitiveness of the algorithm is

$$H_{13} / d_{13} = 2 \cdot \left(1 + \frac{d_{23} - d_{31}}{d_{12} + d_{13} + d_{23}}\right).$$

Using the triangle inequality, we get that

$$\lim_{d_{23} \to 0} \frac{2 \cdot (2d_{23} + d_{31})}{d_{12} + d_{13} + d_{23}} = 1$$

and

$$\lim_{d_{31} \to 0} \frac{2 \cdot (2d_{23} + d_{31})}{d_{12} + d_{13} + d_{23}} = 3.$$

Therefore, the lazy adversary always forces the competitive factor to be in the *interval* (1,3]. □

Now, the question arises whether there exists a randomized 2-competitive on-line algorithm for the 2-server problem.

We repeat the process one-by-one, until all the non-lazy moves have been eliminated. Since the algorithm we consider is 2-competitive against a lazy adversary, it has to be 2-competitive against any adversary as well. □

4.2.3 Resistive Spaces in the k-Server Problem

Recently, *Coppersmith et al.* [98,99] very cleverly used *Raghavan and Snir's* interesting technique [285] to treat the edge weights in the graph where our servers are moving as *effective resistances* in some electrical network, and calculate the transition probability using the harmonic algorithm in this "*inverse*" electrical network. The designed reversible random walks are useful for certain randomized competitive on-line algorithms.

Coppersmith et al. designed randomized k-competitive algorithms against any adaptive adversary on *resistive spaces* with resistive inverses. *Resistive spaces* include every metric space for which a k-competitive algorithm has been proven and many more resistive graphs as well. For example, some of these graphs include:

- *3 node graphs satisfying the triangle inequality* [56],
- *distances on a line* [285],
- *tree closure* [71],
- *uniform graphs* [99,142,254,255].

Note that the *Euclidean plane* has *no resistive approximation* (see [99], pp. 442) and that *no H_k-competitive algorithm* exists for the *k-server problem* when the metric space is *non-uniform* (e.g., it can be shown by establishing lower and upper bounds on the competitive ratio for the 2-server problem on certain triangles; see [203] for more details).

Coppersmith et al. also showed how to compute a value for each pair of points in a resistive space such that on a request to a node υ, if there is no server on node υ, then the server sitting on node υ_i services the request with probability proportional to the value of the edge (υ_i, υ). This algorithm is *simple* and *memoryless*.

The same authors proved the following tight bound for all symmetric cost matrices:

Any random walk on an weighted (undirected) graph with n-vertices has stretch factor (or simply *stretch*) *at least n-1, and every weighted (undirected) graph has a random walk with stretch at most n-1.*

They also justified the above results for the *cat and mouse game* [99], *metrical task systems* [66] and *k-server problem* [255]. Additionally, they derive algorithms for some *non-resistive* [98] spaces *by approximating* the original metric space by a resistive metric space. The approximation technique yields a randomized $2k$-competitive algorithm for points on the periphery of a circle (i.e., a *discrete circle*). This is the *first* on-line algorithm for the k-server problem on a metric space.

4.2.4 Asymmetric 2-Server Problem

Symmetry of the edge weight of an electrical network is very crucial to the basic technique used in designing the appropriate random walk. All the work previously done on the k-server problem dealt with *resistive graphs*, where symmetry of the edge weights (costs) could be assumed.

The following question arises: *Can we design competitive on-line algorithms for the k-server problem on non-resistive graphs* (i.e., *no* symmetry of the edge weights)? The answer seems to be that we can *no* longer find algorithms with a competitive ratio in terms of the number of servers alone.

In the *symmetric case*, we have seen that there exists a randomized competitive, *memoryless* algorithm for any 2-server problem against any adversary (*Theorem 4.2*) and we found a competitive ratio of exactly 2 against a *lazy adversary*. In the *non-symmetric case*, if we adopt the same strategy, we cannot achieve an expansion factor of exactly 2 against a *lazy* adversary for large cycles.

Definition 4.2. Let C = (C_{ij}) be the given cost matrix of size *n x n*. The *cycle offset ratio* $\Psi(C)$ is defined as the maximum over all cycles $(\upsilon_1, \upsilon_2, \ldots, \upsilon_k = \upsilon_1)$ of the ratio

$$\frac{\sum_{j=1}^{k-1} C_{\upsilon_j,\, \upsilon_{j+1}}}{\sum_{i=1}^{k-1} C_{\upsilon_{i+1},\, \upsilon_i}}.$$

If we assume the edge costs satisfy the *triangle inequality*, then $1 \le \Psi(C) \le (n - 1)$. Moreover, we have that $\Psi(C) = 1$, when C is *symmetric*.

Theorem 4.3. *There exists a randomized, memoryless, $2 \cdot \Psi(C)$-competitive algorithm for the asymmetric 2-server problem against a lazy adversary.*

Proof **(sketch):** It is similar to the proof of *Theorem 4.2* for the symmetric case against a lazy adversary. We find that there are exactly two sets of probabilities (hence, two solutions) yielding an expansion factor of 2 for all commute times against a lazy adversary on a 3-node graph.

On larger n-cycles ($n \ge 3$), the competitive factor is greater than 2 and it is bounded above by two times the cycle offset ratio of the n-node graph, which can be as high as n-1. Since the number of vertices is doubled (in the *worst case*), our algorithm can be at most $2 \cdot n$-1 rather than $2 \cdot (n-1)$. □

It is not our intention to give the completed proof here, because it is straight forward and similar to that of the symmetric case by substituting the hitting times H_{ij} of the old symmetric analysis with the expected "closing distances" D_{ij}.

We observe that asymmetry of that edge weight on graphs gives us randomized on-line algorithms of higher competitiveness (i.e., *upper* bounds) for 2-server problem.

4.2.5 Non-resistive Graphs and Server Problem

Coppersmith et al. [98] have given an approach for resistive graphs that can be extended for non-resistive graphs.

We use *ergodic random walks* (see *definitions 3.1*) with the advantage that reversibility of the walk (i.e., symmetry of the edge weights) is not needed to design randomized competitive on-line algorithms for some previous well known problems (i.e., task systems and cat-mouse game).

Definition 4.3. An *M-matrix* is simply an $n \times n$ matrix A of the form $A = \alpha \cdot I_n - P$ in which P is a non-negative matrix and α is at least as big as the largest *eigenvalue* of P.

Clearly, the matrix $\overline{P}$ defined in *section 4.2* is an *M-matrix.* The following *Theorem of Fiedler et al.* [143] is an interesting trace-inequality.

Theorem 4.4. *For a non-singular M-matrix A of size n x n, we have that* $tr(A^{-1}A^T) \leq n$, *with equality holding if and only if A is symmetric.*

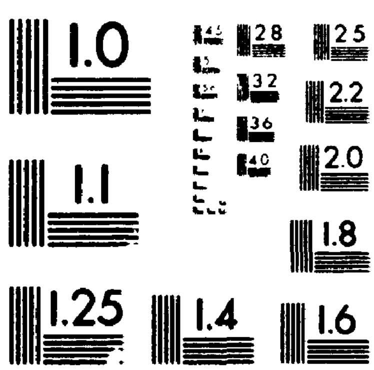
Now let us state a stronger result which generalizes *Lemma 4.1.*

Corollary 4.1. $\sum_{i,j=1}^{n} W_i P_{ij} H_{ij} \leq n-1$, *for any ergodic graph, with equality holding if and only if the graph is resistive (i.e., lemma 4.1).*

Proof: Using *Theorem 4.4* with $\overline{P}$ in the place of A, we have

2 OF/DE 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT



PRECISION℠ RESOLUTION TARGETS

$$tr(\overline{H}\,\overline{P}^{T}) = \sum_{i,j=1}^{n} W_i P_{ij} H_{ij} \leq n-1. \quad \square$$

Now we are ready to show all the results of *Coppersmith et al.* [99] for k-server problems on resistive graphs can easily be extended in the case of non-resistive spaces. We do *not* intend to state and prove all the results here, because most of them, including their proofs, are identical to those in [99]. We clearly justify this claim by arguing that the designed reversible *markov chains* are the same as that of resistive graphs, if we use the new technique on non-resistive spaces.

Theorem 4.5. *Any random ergodic walk over a directed weighted graph has competitive factor at least $(n-1)/\Psi(C)$, where $C = (C_{ij})$ is an $n \times n$ cost matrix.*

***Proof*:** Indeed, the proof is identical to that of *Theorem 1* of [99], wherein the symmetry is assumed. $\square$

Theorem 4.6. *For any $n \times n$ cost matrix C and any transition probability matrix P, the stretch of the ergodic walk by P on a non-resistive graph with weights given by C is at least $n-1$.*

***Proof*:** It suffices to bound the competitive factor over all cycles. This can be extended to all paths, with an additive constant such as $\max_{i,j} C_{ij}$. The expected cost per move is

$$E = \sum_{i,j} W_i P_{ij} C_{ij} = \sum_{i,j} W_i P_{ij} H_{ij} \leq n-1$$

by *Corollary 4.1*.

Now, the expected cost of a sequence of walks (or a walk) through vertices $\upsilon_1, \upsilon_2, \ldots, \upsilon_k = \upsilon_1$ is simply

$$E \cdot \sum_{i=1}^{k} H_{\upsilon_i,\upsilon_{i+1}} = E \cdot \sum_{i=1}^{k} C_{\upsilon_i,\upsilon_{i+1}} \leq (n-1) \cdot \sum_{i=1}^{k} C_{\upsilon_i,\upsilon_{i+1}} \cdot \quad \square$$

Note that the lower bound is n-1 under symmetry, since $\Psi(C) = 1$. The proof of the lower bound is essentially the proof of *Theorem 1* of [99].

The following theorem about the *cat and mouse game* is an immediate consequence of *Theorem 4* of [99]:

Theorem 4.7. *Let G be any weighted ergodic graph with n nodes. There exists a randomized strategy with a competitive ratio of at least (n-1) / $\Psi(C)$ for the cat-mouse game on G, and the ergodic walk by the cat achieves a stretch factor (ratio) of at least (n-1).*

Definition 4.3. Let the *edge offset ratio* $\Psi'(C)$ be $\max_{i,j} \frac{C_{ij}}{C_{ji}}$. Note that $\Psi(C) \leq \Psi'(C)$, and when C is symmetric $\Psi(C) = \Psi'(C) = 1$.

An interesting theorem for the k-server problem on *ergodic* non-resistive graphs follows:

Theorem 4.8. *Let C be a non-resistive cost matrix on n nodes. If every submatrix on (k+1)-nodes is ergodic, then there exists a randomized $k \cdot \Psi'(C)$-competitive strategy for the k-server problem against an adaptive on-line adversary.*

Proof: It is similar to the proof of *Theorem 8* of [99] in the case of resistive graphs. □

Lemma 4.3. *Any random ergodic walk on a graph with self-loops has stretch of at least 2n-1, where the costs C_{ii} are not necessarily zero.*

The proof is omitted, because it is similar to that of *Theorem 7* of [99].

Additionally, *Theorem 4.3* can easily be generalized as follows:

Theorem 4.9. *There exists a randomized, memoryless $k \cdot \Psi'(C)$-competitive algorithm for the k-server algorithm against a lazy adversary on non-resistive spaces. Moreover, there exists 2k-competitive algorithm (called the k-center algorithm) which is optimal up to a factor of 2 among all on-line algorithms for the k-server problem on a bounded non-resistive space.*

The first part of the above theorem is straight forward to prove from the proof of Theorem 4.3 for the non-resistive spaces in this case. For the second part, we can extend the proof of Theorem 5.1 [350] for the k-server problem on any bounded non-resistive space.

An interesting and immediate consequence is the lower bound of $(2n-1) / \Psi(C)$ for any deterministic or randomized on-line algorithm for *task systems* on *non-resistive n-node graphs*. Although the proof is straight forward and similar to that on resistive graphs, it seems to be considerably simpler when we use ideas from the proof of *Theorem 4.7*. Specifically, for the deterministic (resp., randomized) case the proof is essentially that of *Theorem 2.2* of [98] (resp., *Theorem 11* of [99]).

We have seen that the *Coppersmith et al. approach* [99] works for *non-resistive* graphs and it can be used to find competitive solutions of the k-server problems against a lazy adversary. However, the following open question arises: *Can a general metric space be always changed slightly, in a predictable and useful fashion, so that it becomes non-resistive?*

We have *no* results for the k-server problem in *general metric spaces.* It would be interesting to study the cat and mouse game under a wider class of strategies in the case when *the cat is not blind*; this would extend the interesting work of *Baeza-Yates et al.* [35]. It is believed that a somewhat different random graph approach will solve the *k-server conjecture* (where $k \geq 3$) for general metric spaces in a randomized environment as well. Finally, we would like to point out that there are several challenging open problems for k-server probiem (e.g., see [98,99]).

4.3 The Distributed k-Server Problem

In the previous sections we have seen the standard setting of the k-server problem where the communication cost was free, that is, there was a centralized (*global control*) algorithm that got the requests for service with no cost and transferred the motions instructions to the servers.

A more realistic distributed (*local control*) setting of the k-server problem is that when the information messages to the servers are costly. The problem arises in computer network of n processors when k identical mobile servers have to be scheduled between the processors of the network. The objective is to develop on-line algorithms that optimize not *only* the total distance the servers travel but also the communication cost incurred for the transmission of control incomplete information about the requests. This problem is also related to distributed file allocation problem and to other problems of data management [26,28,29].

In some special cases (e.g., for the *uniform* metric spaces) deriving distributed algorithms from the standard ones is straight forward by choosing a *leader* that runs the global-control algorithms and ignores requests on covered points. Generally, the transmission of any *deterministic* competitive global-control k-server strategy for any metric space into a competitive distributed algorithm is too expensive.

Bartal and ***Rosen*** [42] have developed a ***general translator*** to make k-server algorithms distributed and designed *poly(k)*-competitive distributed algorithms for the ***lines***, ***trees*** and the ***rings***. They also proposed a distributed k-server algorithm which achieves a competitive ratio of $\Omega(max\{k, \frac{1}{D}\cdot\frac{\log n}{\log\log n}\})$ against *adaptive* adversaries for arbitrary network topologies with n nodes, where D is the ratio between the cost of moving a server and of transmitting a message across the same distance. The same authors considered a distributed version of the randomized *harmonic* k-server algorithm, which has the *best* currently proved competitive ratio of $O(C_H\cdot(1 + \frac{1}{D}\cdot max\{k, \mu\}\cdot(\log\Delta)\cdot\log n))$, where C_H is the competitive ratio of the classical *harmonic* algorithm, Δ denotes the diameter of the network topology and $\mu = max\{\lceil \log n\rceil, \lceil \log\Delta\rceil\}$ which indicates the size of a *unit-length* message. Here, it would be interesting to mention that most of the results for the k-server problems on resistive and non-resistive graphs can easily be transformed into the distributed environment.

The larger the island of knowledge,
the longer the shoreline of wonder.
R. W. Scockman

Chapter 5

Combinatorial On-line Algorithms

Heuristic has concerned with language-dynamics,

while logic has concerned with language-static.

Imre Lakatos

The aim of heuristics, or heuretics, or "ars inveniendi" is to

study the methods and rules of discovery and invention.

George Polya

There is tremendous amount of literature on off-line optimization problems and algorithms. This chapter deals exclusively with the *combinatorial problems* in *on-line* manner. Particularly, we study the *on-line graph coloring* and *matching* problems as well as their algorithms. Moreover, we give a very brief presentation of *on-line string matching* and *on-line flow problem* in a network.

5. 1 On-line Graph Coloring

5. 1.1 Problem Statement and Related Terminology

The *problem of coloring* a graph is that of assigning a color to vertices such that no two adjacent nodes (*bins*) receive the same color. A *valid coloring* of a graph G = (V, E) is a partitioning of the nodes into color classes such that the vertices of the same color are non-adjacent. Let $\chi(G)$ be the *chromatic number* of a graph G; that is, the minimum number of colors used in any valid coloring of G.

A graph (*off-line*) coloring algorithm receives an input graph G and determines a valid assignment of colors to nodes. It is well known that the problem of finding a valid coloring graph which uses the minimum number of colors is *NP-hard* [156].

We proceed with some definitions and notations which will be used in the section.

An *on-line graph is* a structure $G^{\prec} = (V, E, \prec)$, which is also called an *on-line presentation* of a graph, where V is finite or countable infinite, and $\prec$ is a linear ordering of V. Let $V_i = \{\upsilon_1, \ldots, \upsilon_i\}$ denote the first i vertices of V in the linear order $\prec$ and the set $G_i^{\prec} = (V_i, E_i, \prec)$, where E_i is the set of edges in V_i, for $1 \le i \le n = |V|$.

An algorithm for coloring the vertices of an on-line $G^{\prec}$ is said to be *on-line* graph if the color of a vertex υ_i is determined solely by $G_i^{\prec}$. Intuitively, in the on-line version of the graph coloring problem, the graph is presented one vertex at a time when a vertex is presented and only the adjacent edges to all already presented vertices are also revealed. An on-line algorithm has to irrevocably assign a color to a vertex before proceeding to the next vertex. The goal of on-line algorithm is to minimize the number of colors used in coloring of the graph.

A simple but important example of an on-line graph coloring algorithm is the *First-Fit (FF)* algorithm.

Algorithm *FF (G)*

Assign to each υ_i of G with the lowest possible color which is not already numbered to any vertex $\upsilon \in V_{i-1}$ adjacent to υ_i.

Figure 5.1: On-line Graph Coloring Algorithm ***FF(G).***

We use competitive analysis to measure an on-line coloring algorithm $\mathcal{A}$. Let $\chi_{\mathcal{A}}(G)$ denote the chromatic number that $\mathcal{A}$ uses to color G. The *performance* (or *competitive*) *ratio* of an on-line graph coloring algorithm $\mathcal{A}$, denoted by $\rho_{\mathcal{A}}(G)$, is defined as $\rho_{\mathcal{A}}(G) = \max_{G \in C} \frac{\chi_{A}(G)}{\chi(G)}$, where G is ranging over all input graphs for a class of graphs C.

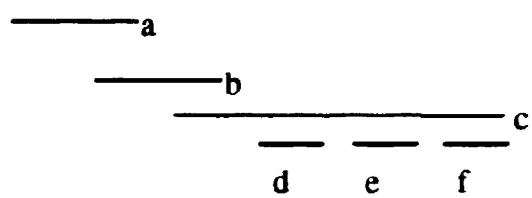
On-line graph coloring has applications to *parallel process assignment* and *register (storage) allocation* problems [69,170,278]. Recently, *Lovasz et al.* used the upper bounds of on-line coloring algorithms to examine the relative power of determinism, randomization and non-determinism to search problems in the *Boolean decision tree model* [248,188].

In the next subsection we consider on-line coloring on some restricted classes of graphs.

5. 1.2 On-line Interval Graph Coloring

We consider the *interval coloring problem* as an introductory example of on-line graph coloring.

A graph G is said to be an *interval graph* if it is the intersection graph of a family of intervals along the real line; for example,



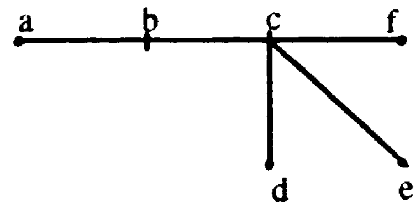


Figure 5.2: i) Interval representation ii) Interval graph.

In on-line setting of the problem, each request is an interval on the *real line* and each action assigns a color to the current request, with no two overlapping intervals receiving the same color. The cost of a request sequence is the number of colors used. Set $w(G) = \max_{I \in G} w(I)$ the *clique number* of the interval graph G; that is, the maximum width assigned to any interval ranging over all input intervals of G.

Kierstead and Trotter [217] give an on-line algorithm for the interval graph, which is a modified *FF(G)* coloring algorithm.

***Algorithm** On-lineColor (G, w)*
begin
As each interval $I \in G$ arrives it is assigned a positive integer w(I) called the *width* of interval I and a color;
If I does not intersect any previous interval of width 1, **then**
w(I) := 1;
Assign any color to I, among those received for its width, that has not been assigned to any previous interval that intersects I;
else
w(I) := the set of the least j > 1 such that I does not intersect more than two previous intervals of width j;
Assign three colors for each interval of width j;
endif
end.

Figure 5.3: On-line Interval Coloring.

Kierstead and Trotter [217] showed that the above on-line algorithm colors any on-line interval graph $G^{\prec}$ with at most $3 \cdot w(G^{\prec}) - 2$ colors. Arguing by induction on w, one shows that $G^{\prec}$ can be partitioned on-line (just be greedy) into a maximal graph $G^{*\prec}$ with clique size w - 1 and an induced subgraph $H^{\prec}$ of $G^{\prec}$ with maximum degree 2. Thus, G can be colored on-line using $3 \cdot (w - 1) - 2 + 3$ colors. Moreover, it can be shown by means of an adversary argument that no on-line algorithm can do better. Therefore, *Kierstead-Trotter*'s on-line coloring algorithm achieves an *optimal* performance ratio of 3 on interval graphs.

Finally, we would like to point out that the interval coloring problem can be seen as a scheduling one, in which each interval represents the time span of some task and the color represents the processor assigned to execute the task.

5.1.3 On-line Coloring on Special Graphs

The on-line coloring has been extensively studied for special graph classes. The *bipartite* graphs can be colored on-line using O(log n) colors [248]. The previous best lower bounds known were $\Omega(\log n)$ for *n-node trees* (since trees are also *chordal* graphs) and $O(\log^k n)$ for k-colorable graphs, where k is fixed [248].

Kierstead [214] has proved that *FF* algorithm has a constant performance ratio on *interval* graphs. *Gyarfas and Lehel* [164,165] have also shown that *FF* achieves a constant performance ratio on *split graphs*, *complements of bipartite graphs*, and *complements of chordal graphs*.

Recently, *Irani* [188] examined on-line coloring for the *inductive* graphs. A graph G is *d-inductive* if its vertices can be ordered (called an *inductive order*) in not necessarily such a *unique* way that each vertex is adjacent to at most d higher-numbered vertices. This inductive order of G gives an *inductive orientation* for the edges of the inductive graphs from the higher numbered vertices to the lower numbered ones.

Irani [188] (also *Karloff*, independently) showed that *FF* on-line algorithm uses O(d·*log* n) colors to color a d-inductive graph G with n = *n*(G) vertices. This yields that any on-line coloring algorithm for d-inductive graphs has a performance ratio of Ω(*log* n). The upper bound on the chromatic number of colors used yields an upper bound on the performance ratio for graphs, where d and the chromatic number χ are closely related. For example, planar graphs are *5-inductive* and *chordal graphs* are χ(G)-inductive, which implies that both of them have a performance ratio of O(*log* n).

When the on-line model is slightly altered by allowing the algorithm to see the next $l \geq 1$ vertices before assigning a color to the present vertex, we say that the on-line coloring algorithm has a *weak lookahead of size l*.

Irani [188] showed that even with a *weak lookahead* of size $\frac{n}{\log n}$, an on-line algorithm still requires Ω(d· *log* n) colors to color a d-inductive graph. For a *weak lookahead* of size $l > \frac{n}{\log n}$ we can do better, because we can *on-line* color a d-inductive graph in $\Theta(\min\{d \cdot \log n, \frac{d \cdot n}{l}\})$ colors.

We now use the new on-line model of *strong lookahaead*, which has practical and theoretical importance. An on-line coloring algorithm has a *strong lookahead of size l* if it has a *weak lookahead* of size l and at step $t + l$ at least t requests have been answered, where t is an integer ≥ 1. This on-line model is also called *on-line with a buffer of size l* and it is more powerful (as an algorithmic feature) than the *weak lookahead* model improving slightly *Irani*'s bounds.

Theorem 5.1. *If G is a d-inductive graph on n nodes, then G can be colored on-line with strong lookahead of size l using* $\Theta(\min\{d \cdot \log n, \frac{d \cdot n}{t-l}\})$ *colors for* $1 \le l \le t\text{-}1$ *and* $t \ge 2$ *an positive integer.*

Proof: The proof is similar to that of *Theorem 8* in [188].

If $d \cdot \log n < (d+1) \cdot \frac{n}{t-l}$, this ignore the *strong lookahead* and use *FF* algorithm to color a d-inductive graph. By *Theorem 6* [188], *FF* uses O(d·*log* n) colors.

If $d \cdot \log n \ge (d+1) \cdot \frac{n}{t-l}$, then divide the nodes into $\frac{n}{t-l}$ consecutive nodes of the inductively oriented graph. The algorithm can see the d-inductive subgraph induced by the nodes in each group before having assign a color to the first node in the group. Therefore, a d-inductive graph can be colored using at most d + 1 colors for every group. Totally, at most $(d+1) \cdot \frac{n}{t-l}$ colors are used. The above bound is asymptotically the *best* possible and it can be shown with a similar way as in the *Theorem 9* of [188]. □

We have seen that *FF* coloring algorithm does well on some special graph classes, but it does quite poorly in general. Again, we conclude that the point at which (*weak* or *strong*) *lookahead* becomes an advantage is quite high for on-line coloring as we have already seen for *paging* problem.

5.1.4 On-line Coloring on Hypergraphs

A *hypergraph H* is a collection of edge subsets $E_1, E_2, \ldots, E_s$ of a set of vertices $V = \{1, \ldots, n\}$. A *k-hypergraph* is a hypergraph where each edge set E_i contains exactly *k* vertices.

Let *m(k)* be the largest *s* such that each *k*-hypergraph with *s* edges can be 2-colored. *Erdös* [352] has shown that

$$2^{k-1} < m(k) < k^2 \cdot 2^{k+1}$$

These bounds are *not constructable* (i.e., *algorithmic*) and show that all k-hypergraphs with fewer than 2^{k-1} edges are 2-colorable, but if the number of edges is greater than $k^2 \cdot 2^{k+1}$, then there exists a k-hypergraph which has *no* proper 2-coloring.

Unfortunately, the general problem of 2-coloring hypergraphs is reducible to *set splitting problem* and thus, it is an *NP-complete* [156]. We instead find 2-coloring of hypergraphs restricted by size and degree.

We consider the problem of on-line coloring for k-hypergraphs. Let f(k) be the largest *s* such that all k-hypergraphs with *s* edges can be 2-colored in on-line setting. *Aslam and Dhagat* [14] have shown that an on-line coloring adversarial strategy exists, it is so called *two chip game*, which achieves the following bounds

$$2^{k-1} < f(k) < k(3+2\sqrt{2}) = k \cdot \phi^{2k},$$

where ϕ is the *golden ratio* $\dfrac{1+\sqrt{5}}{2}$.

In this case, the upper bound demonstrates an inherent weakness of on-line algorithms against any adaptive adversary. They stated an interesting *open problem*:

construct an on-line strategy to achieve a better upper bound for 2-coloring any k-hypergraph with respect to *two chip adversarial game* or any other strategy.

This problem can be easily solved using a simple modification of the randomized *adversarial algorithm* [169]. Therefore, there exists an on-line adversarial algorithm for any on-line 2-coloring algorithm A and every $s \geq k^2 \cdot 2^{k+1}$, produces a k-hypergraph with s edges which A fails to 2-color. This algorithm runs on-line in $\Omega(\frac{n \cdot s}{(\log n)^3})$ time complexity and is O(1)- competitive against any *adaptive* on-line adversary.

5.1.5 On-line Coloring on General Graphs

There has been a lot of work on on-line graph coloring. For example, *Lovász et al.* [248] give an on-line coloring algorithm for *general graphs* that achieves a performance ratio of $O(n/\log^* n)$[1], which slightly improves the worst possible performance ratio of $o(n)$, where n is the number of vertices.

Halldórsson and Szegedy [169] show that for every *deterministic* on-line coloring algorithm there is a k-colorable graph with $k \cdot 2^{k-1}$ vertices on which the algorithm uses $2^k - 1$ colors. This implies a $\Omega(\frac{n}{(\log n)^2})$ lower bound for the performance ratio of any on-line algorithm on general graphs. In the randomized case, they show that the above results hold within a factor of k. This randomized on-line coloring yields a lower bound of $\Omega(\frac{n}{(\log n)^3})$ performance ratio. Additionally, they show two optimal lower bounds on the $\theta(\frac{n}{l})$ approximation of both deterministic and randomized on-line coloring with lookahead of size $l = \Omega(\log^3 n)$.

---

[1] We remind that $\log^* n = \min\{i;\ \log^{(i)} n \leq 2\}$, where $\log^{(i)} n = \log(\log^{(i-1)} n)$ for each $i \in \mathbf{Z}^+$.

Definition 5.1. A *maximal partial greed s-coloring* of a graph G is the assignment of the nodes into a fixed number of greedily color G with s colors, leaving out vertices that cannot be colored. The set R of the uncolored vertices is called the *residual set*, while the set of vertices that have been assigned the same color by a maximal greed s-coloring is defined as a *greedy color class*.

Maximal partial coloring can be achieved sequentially by a natural *heuristic*: find a node of maximum degree, recursively color its neighborhood, and iterate this procedure on the remaining graph. This is essentially the method of *Wigderson* [342] and can easily be found via the *FF* algorithm which assigns a vertex to the first compatible color class (if one exists).

In order to describe the algorithm let the *greedy color classes* be $C_1, C_2, \ldots, C_s$. Consider a color class C_i and denote the vertices in this color class to be $\upsilon_1, \ldots, \upsilon_l$, where $\upsilon_1 \prec \upsilon_2 \prec \cdots \prec \upsilon_l$. We associate the first vertex in C_i to which it is adjacent with every vertex in R. This partitions R into $B_1, \ldots, B_l$ *blocks*. We define a function $S(n,\chi) = min\{\lceil 2^{\chi} n^{(\chi-2)/(\chi-1)} (log n)^{1/(\chi-1)} \rceil, n\}$ that determines the number of color classes that we use.

Now, we describe *Vishwanathan's algorithm*:

Algorithm *Online-Color1*(n,χ)
begin
  if ($\chi \leq 2$), **then** *BipartiteColor*(G)
    {* Algorithm *BipartiteColor*(G) uses at most $4 \cdot log n$ colors [248] *}
  else
    Set $s := S(n, \chi)$;
    Chose a random integer r uniformly from $\{1, \ldots, s\}$;
    while (there are no more vertices) **do**
      if the number of vertices in the partition exceeds s, **then**
        get the next vertex υ in the partitioning class;
        if υ can be colored using the greedy set of colors $\{1, \ldots, s\}$, **then**
          color the vertex greedily.

```
            else {* υ is in the residual set R *}
                Determine which block B of the partition of the vertex falls into;
                Input the vertex to the copy of Online-color1 corresponding to B;
            endif
            Online-Color1 (s, χ-1);
        endif
    end {* while *}
  endif
end.
```

Figure 5.4: *Vishwanathan*'s Randomized On-line Coloring Algorithm.

Indeed, when $k = 2$ the problem is reduced to bipartite coloring which is fairly straightforward *sequentially*, in parallel, and in *on-line* fashion.

Theorem 5.2. *The number of colors used by algorithm Online-Color1(n,χ) on χ-colorable graphs is at most s(n, χ) (see* [337]).

Proof: Let $A(G, n, \chi)$ denote the number of colors the on-line algorithm uses in total to color a χ-colorable graph G on n vertices. Thus, we want to show that $A(G, n, \chi) \leq s(n, \chi)$, where $s(n, \chi) = \binom{2^{\chi}}{\chi} \cdot n^{(\chi-2)/(\chi-1)} (\log n)^{1/(\chi-1)}$ for $\chi \geq 2$, which can be proved by induction on χ. □

Therefore, *Vishwanatan's algorithm* has a performance ratio of $O(n / (\log n)^{1/2})$ against an *oblivious* adversary. This result shows that randomization helps in on-line graph coloring.

Very recently, *Halldórsson* [170] modified *Wigderson*'s *algorithm* [342] or the deterministic *Vishwanatan*'s off-line algorithm to improve the performance ratio to $O(n / \log n)$. The sequential (off-line) coloring algorithm finds a maximal partial coloring, partitions the remaining vertices around the smallest color class and recourses on the relatively few sub-problems. The recursion stops at bipartite graphs, otherwise if the chromatic number χ of G is too large with respect to the size of the graph, we settle on the trivial coloring of one color per vertex. Thus, the pseudo-code of the algorithm follows:

Algorithm *Offline-Color1(G, k);*
begin
  $\sigma(n, k) := (n / (k - 2))^{(k-2)/(k-1)}$;
  if ($k \leq \log n$), **then** *BipartiteColor*(G)
  else if ($k > \log n$), **then** assign each vertex a different color ;
else
  ResidueNodes := *MaximalPartialColor (G, σ (n, k));*
  Find the smallest greedy color class, and let $w_1, \ldots, w_p$ be its nodes;
  Partition the *ResidueNodes* into $R_1, \ldots, R_p$ such that nodes in R_i are adjacent to w_i ;
    for i = 1 **to** p
      Offline-Color1(R_i, *k - 1*);
    endif
end.

Figure 5.5· An Approximate *Off-line Coloring* Algorithm.

We can easily prove by induction on k and with a similar way as in *Theorem 5.2* that the number of colors used by the above *Offline-Color1*(G, k) algorithm on k-colorable graphs is at most $\frac{k-1}{(k-2)^{(k-2)/(k-1)}} \cdot n^{(k-2)/(k-1)}$. So, our algorithm achieves a performance ratio of $O(n^{(k-2)/(k-1)})$ which is maximized for $k \approx \log n$. Therefore, algorithm *Offline-Color1*(G, k) has a performance ratio of $O(n / \log n)$.

Haldórsson [162] constructed the *first parallel coloring algorithm* with the same non-trivial performance ratio of $O(n / \log n)$ and complexity time $O((\log^4 n) \cdot \log \log n)$ using the above *sequential* (*off-line*) coloring algorithm. He just substituted the sequential *MaximalPartialColor* (G, $\sigma(n, k)$) with the following *parallel* one:

Algorithm *MaximalPartialColor1(G, x);*
begin
Construct the following graph G′ on (n·x) vertices (*see* [170]):
Make x identical copies of G : $G^1, G^2, \ldots, G^x$;
$(\upsilon_i^j, \upsilon_s^t) \in E(G')$ *iff* $i = s$, or $j = t$ and $(\upsilon_i, \upsilon_s) \in E(G)$;
I := MIS(G′); { Maximal Independent Set of G′. *}*
$G_i := I \cap G^i$; *ResidueNodes* $:= G - \bigcup_{i=1}^{x} C_i$;
Re*turn* the greedy color classes $\{C_i\}$ and *ResidueNodes*;
end.

Figure 5.6: A Parallel Maximal Partial Coloring Algorithm.

Next, we convert *Offline*-Color*1* algorithm into an on-line algorithm. The algorithm assigns a color only to the formal variable υ (a vertex) in each invocation, while updating a static data structure called *coloring tree*, which is layered into chromatic levels (see [170] for more details).

Algorithm *Online-Color2(T, k, υ);*
{* Assign a color to the vertex υ. *}
(* T is a k-colorable tree. *}

begin
  if (k ≤ 2), **then**
    BipartiteColor(T, υ)
  elseif (*FF*(T, k, υ) is *not* sufficient), **then**
    choose some node υ_j adjacent to υ in the partitioning class;
    Online-Color2(R_j, k - 1, υ);
  endif
end.

Figure 5.7: Haldórsson's Randomized On-line Coloring Algorithm.

This approximate, randomized, on-line coloring algorithm has a performance ratio of O(n / *log* n), which can be proved with a similar proof as in the *off-line* case. In addition, *Haldórsson*'s formulation [170] showed how to apply the parallel coloring algorithm to obtain an *NC* approximation algorithm for the independent sets of size $\Omega(n^{1/(k-1)})$ in a k-clique free graph with an independence number greater than $\frac{n}{k}$. Unfortunately, the processor complexity of removing the k-cliques grows as fast as n^k.

We conclude this section summarizing the *upper bounds* of the performance ratios of the on-line graph coloring algorithms shown the literature. Thus, a challenging *open problem* is to improve any *non-optimal* bound in the following table:

On-line Graph Coloring Algorithms and their Performance Ratios

Performance ratio	Graph	Source
O(1)	*Split* graphs	[164, 165]
O(1)	Complement of *Chordal*	[164, 165]
O(1)	Complement of *Bipartite*	[164, 165]
3	Complement of Tree Bipartite	[164, 165]
3	*Interval*	[215]
$o(n)$	Any graph	[248]
$O(n / \log^* n)$	Any graph	[248]
$O(n / (\log n)^2)$	Any graph	[169]
$O(n / (\log n)^{1/2})$	Any graph	[337]
$O(n / \log n)$	Any graph	[170]
$O(\log n)$	*Bipartite*	[164, 165]
$\Omega(\log n)$	*Tree Bipartite*	[164, 165]
$O(\log n)$	d-*inductive*	[188]
$O(\log n)$	5-*inductive*	[188]
$O(\log n)$	*Chordal*	[188]

Table 5.1: Performance Ratios of On-line Coloring Algorithms for Graphs.

5.2 On-line Graph Matching

In this section, we present *on-line minimum and maximum matching problems* for both unweighted and weighted graphs. In particular, we apply the *dual bounding technique* to simply reanalyze the weighted matching algorithms and examine the general applicability of this technique.

5.2.1 Off-line Problem Statement and Algorithms

Matching and related problems have been studied extensively in the contexts of both *sequential* and *parallel* computation.

Given a graph G = (V, E), a *matching M* is a subset of the edges such that no two edges in *M* share vertices. The problem is similar to that of finding an *independent set* of edges. In the *minimum matching* (*min-matching*, for short) we wish to minimize $|M|$. In contrast, we maximize $|M|$ for the *maximum matching* (*max-matching*) of weighted graph.

We first need to define some standard terms and technical results, before studying the on-line setting of the problem.

- A *bipartite graph*[1] G = (U, V, E) has $E \subseteq U \times V$ is the set of nodes with $U \cap V = \emptyset$.
- A *Perfect matching* is a matching such that each vertex adjoins exactly one edge.

  In bipartite graphs, we must have $|U| = |V|$ in order for a perfect matching to be possibly exist.
- The *cost of matching* in a weighted graph is the sum of the weights of the edges in the matching.

---

[1] In the following, as commonly done, we refer to the set U as "*the boys*" and the set V as "*the girls*"

- An *one-sided assignment* (or *assignment*, for short) is a perfect matching in a bipartite graph. An abusing terminology considers that a bipartite graph G = (U, V, E) with vertex set $U \cup V$ and edge set $E \subseteq U \times V$ has $|U| \neq |V|$; we also call a matching of size *min* { $|U|$, $|V|$ } an *assignment*.
- The sum of the weights of the vertices assigned to a vertex $\upsilon \in V$ is referred to as the *load of vertex* υ. Clearly, if a perfect matching exists, the maximum load equals the maximum weight.
- A *metric graph* is a *complete bipartite* (or *complete*, in short) graph with symmetric edge weights satisfying the triangle inequality.

It is not difficult to prove that computing an optimal solution in the *off-line assignment* is *NP-complete* for arbitrary weights. (This is done by reduction to the *Knapsack problem* [237,238]). However, if the weights are all equal, then an optimal solution can be computed in polynomial time by reduction to *Maximum Flow Problem* [161].

We consider The following three *off-line* (i.e., standard) *matching algorithms*:

- *Offline-MIN* : An algorithm that derives *min- weight perfect matching* [326].
- *Offline-MAX1* : An algorithm that produces a *max-weight perfect matching*, assuming that the weights of edges are non-negative [100, 237].
- *Offline-MAX2* : The greedy heuristic for *max-weight matching* of graph G [20]:

Algorithm *Offline-Max2;*
begin
  $M := \emptyset$; $\Gamma := G$;
  while $E(\Gamma) \neq \emptyset$ **do**
  begin
    Choose a maximum weight edge $e = (u, \upsilon) \in E(\Gamma)$ *not* adjacent to any edge currently in the matching M.
    $\Gamma := \Gamma \setminus (u, \upsilon)$; {* $\Gamma \setminus (u, \upsilon)$ denotes the subgraph induced by the vertex set $V \setminus \{u, \upsilon\}$ *}

```
        M := M ∪ {e};
    end
    output M
end.
```

Figure 5.8: A Maximum-weight Off-line Matching Algorithm.

5.2.2 Duality Analysis of Weighted Matching Algorithms

The *dual bounding technique*[1] can be used to more easily reanalyze the weighted matching algorithms.

The *dual problem*[2] of finding an assignment in a weighted, bipartite graph with edge weights is to find a maximum-cost potential such that the weight of any edge is *at most* (for the *upper* bounds) or *at least* (for the *lower* bounds) the sum of the weights of the endpoints.

Therefore, in order to find the *upper* (resp., *lower*) bound in terms of a dual transformation, we modify the edge weights d(i, j) to d(i, j) - Π_i - Π_j, where Π_k denotes the weight of vertex k. This reduces the cost of the matching by *at most* (resp., *at least*) $\sum_k \Pi_k$, but leaves the cost non-negative.

In order to provide an example, we apply this technique to show the performance ratio of the last matching algorithm.

Theorem 5.3. *In any non-negatively weighted graph, the Offline-MAX2 matching algorithm has a competitive factor of 2.*

[1] The dual problem allows a larger class of solutions, and possibly tighter bounds, see [272, pp. 225].
[2] See subsection 4.1.2.

Proof: If the greedy algorithm adds an edge (i, j) to the matching, let $\Pi_i = \Pi_j = d(i, j)$; otherwise $\Pi_i = 0$.

If (i, j) is a matched edge, we have that $d(i, j) \le max\{\Pi_i, \Pi_j\} \le \Pi_i + \Pi_j$; otherwise, suppose *i* was matched first to k, then $d(i, j) \le d(i, k) = \Pi_i$.

The cost of the matching is $\sum_i \Pi_i$. Since $\sum_i \Pi_i$ is an upper bound of the maximum cost of matching, the competitiveness of the algorithm is 2. □

5.2.3 On-line Unweighted Matching Algorithms

We consider the on-line version of the problem of constructing a large matching in a bipartite graph. In the on-line setting, the *boys* (the vertices of U) appear either one-by-one or in groups, in some arbitrary order. As each *boy* answers, the algorithm is told the disclosure of its identity, its weight (only, in the weighted matching) and all the edges incident to it. The on-line algorithm must assign at most one *girl* from V to each *boy* (vertex) of U; of course, the algorithm is not permitted to choose two edges incident with the same *girl*.

We use the competitive analysis to measure the performance of on-line algorithms for the matching problems. Here, we would like to note that we consider the minimal competitive ratio to account for the case we deal with a maximization problem rather than a minimization one. In the deterministic case, the adversary constructs the graph and assigns the weight in advance; thus, it can construct the worst possible sequence. In the randomized environment we first assume an *oblivious* adversary.

Below, we consider both deterministic as well as randomized on-line matching algorithms for *unweighted* bipartite graphs and derive their competitive ratios for either case.

The first competitive ratio achievable by a deterministic algorithm for the bipartite matching problem is 1/2. In a bipartite graph one can easily force any deterministic algorithm to match only half of the *boys*, even though there exists a matching that covers all the *boys*. For example, let us consider the following simple deterministic algorithm:

Algorithm *Online-D-BM1*[1]*;*

Present a boy who is adjacent to two girls; whichever girl the algorithm chooses, present a second boy who is adjacent to chosen girl but *not* to the other one.

Figure 5.9: On-line Deterministic Bipartite Matching Algorithm.

We can also show that the above result applies even for randomized algorithms against an adaptive on-line adversary using the following more complicated algorithm, which is referred to as *ranking algorithm* [211].

Algorithm *Online-R-BM2;*

For the first n/2 girls, the adversary adds edges between the new vertex and any boy that has *not* been matched by either the adversary or the algorithm. The adversary adds the *random* one of these edges to the matching.

Figure 5.10: Ranking Algorithm.

If T(n) denotes the number of edges in the intersection of the adversary's and the algorithm's matching after the first n/2 girls have arrived, then $E(T(n)) = O(\log n)$. Clearly, the adversary matches every girl, and the on-line algorithm matches at most $n/2 + T(n) = n/2 + O(\log n)$ boys.

Karp et al. [211] presented the following randomized on-line algorithm for bipartite matching against an *oblivious* adversary.

---

[1] On-line Deterministic, Bipartite, Matching Algorithm.

Algorithm *Online-R-BM3;*
Choose a random order $g_1, g_2, \ldots, g_n$ of the n girls, and make the first boy adjacent to all the girls, the second boy to $g_1, g_2, \ldots, g_{n-1}$ and make the *i*th boy adjacent to $g_1, g_2, \ldots, g_{n-i+1}$, in general.

Figure 5.11: Karp's Randomized Bipartite On-line Matching Algorithm.

The above simple randomized algorithm achieves an asymptotically tight bound of $n \cdot (1 - 1/e) + o(n)$, where e is the base of natural logarithms.

This adversary strategy limits every randomized on-line algorithm to a competitive ratio of $1-\frac{1}{e}$ and illustrates what seems to be a rather *general phenomenon*: *randomization helps considerably against oblivious adversaries, but not against adaptive adversaries*. A good exercise for the interested reader is to show that this phenomenon also holds for the *ski rental* and the *update list* problems.

5.2.4 On-line Assignment Algorithms

The *assignment problem* (i.e., the problem of finding a bipartite matching of minimum weight) is one of the archetypal problems in algorithmic graph theory and in combinatorial optimization [100,272].

The natural on-line version of the assignment problem in a weighted bipartite graph G = (U, V, U x V) is defined as follows: the vertices of U appear in some order. When a vertex appears, the cost of all adjoining edges are revealed, and some such edge has to be added to the matching. We explore the on-line assignment problem in weighted bipartite graphs for both deterministic and randomized environments and derive exact competitive ratios for either case.

Khuller et al. [213] and *Kalyanasundaram et al.* [196,198] independently considered the following *strongly* competitive deterministic algorithm for the *min-matching assignment*:

Algorithm *Online-D-PERM1*;

Let M_i be the on-line matching computed by the algorithm after arrival of vertex $\upsilon_i \in V$ and P_i denote the matching (called *partial matching*) constructed by the algorithm for the first *i* service. Initially, M_0 and N_0 are empty.

Step 1: Upon arrival of $\upsilon_i \in V$ compute the off-line matching N_i in graph $G_{P_i} = (P_i, V, P_i \times V)$ (N_i is called the *minimum matchng weight* on P_i).

Without loss of generality (*w. l. o. g.*), we assume that the *exclusive - or* $N_i \oplus N_{i-1}$ consists of a simple *odd* length *augmenting path* from υ_i to a vertex $u_i \in U$ (e.g., see [213], *lemma 2.1* for more explanations).

Step 2: The vertex u_i will be free in M_{i-1}; match υ_i to u_i to obtain M_i.

Figure 5.12: A Deterministic Permutation Algorithm for *Min*-matching Problem.

The competitive analysis of the algorithm, using the dual bounding technique, is a little more complicated.

Theorem 5.4. *Algorithm Online-D-PERM1 is at least $(2 \cdot n-1)$-competitive on any 2n-node, metric, bipartite graph..*

Proof: The odd edges of the path (if any) form a subset of the current *min-weight* assignment with weight no more than the current potential.

Using the *dual bounding technique*, we find that the augmenting path $N_i \oplus N_{i-1}$ consists ol one edge. Thus, the weight is bounded by $2 \cdot i-1$ the weight of the current potential, since the weight of the potential is only increased during the course of the algorithm, after $i \leq n$ vertices are presented. So, using the fact that $N_{i-1} \leq N_i, \forall i \leq n$, we get that $P_i \leq (2 \cdot (i-1) - 1) N_{i-1} + 2 N_i \leq (2i-1) N_i$, $\forall i \leq n$.

Therefore, this on-line greedy algorithm is at least $(2 \cdot n - 1)$-competitive and needs $O(n^2)$ time complexity to find a minimum partial bipartite matching at each decision step. □

Kalyanasundaram and *Pruhs* [198] proposed another deterministic on-line greedy minimum matching (so-called *Nearest neighbor*) algorithm which achieves a performance ratio of $(2^n - 1)$ and needs $O(n)$ time at each decision step, in any 2n-node metric space. If both of his algorithms are combined, we can easily get a simple deterministic greedy algorithm for solving the on-line minimum matching problem when the points are constrained to lie on *Euclidean space*.

Algorithm *Online-EMM;*
Let U be a set of points and $V = \{\upsilon_1, \upsilon_2, \ldots, \upsilon_n\}$ be a set of points on *Euclidean space*.
begin
  M := ∅; {* the matching M is *initially empty* *}
  Input (U);
  for i = 1 **to** n **do**
    At the arrival of $\upsilon_i \in V$, add the shortest path between υ_i and the unmatched points in U to matching M.
  endfor
  return (M);
end.

Figure 5.13: An On-line Minimum Matching Algorithm with n points Euclidean Space.

This *Online-EMM* algorithm has a tight competitive ratio of $(2^n - 1)$, because of the metric space and the worst case data structure described in *Theorem 2.6* of [196].

Now, we present a simple, deterministic on-line assignment algorithm:

Algorithm *Online-D-AS1;*
Upon arrival of a vertex $u \in U$ assign it to a neighbor with the current minimum load (ties are broken arbitrarily)

Figure 5.14: A Deterministic On-line Assignment Algorithm.

Theorem 5.5. *Online-D-AS1 achieves a competitive ratio of* $\lceil \log n \rceil + 1$.

Azar et al. [33] showed that the competitive ratio of any on-line bipartite assignment algorithm is at least $\lceil \log (n + 1) \rceil$.

We combine the above deterministic algorithm with the randomized *Online-D-BM3* to get the following randomized, assignment on-line algorithm.

Algorithm *Online-R-PERM2;*

Choose a random permutation Π_i of the vertices in V, $\forall\ 1 \le i \le n$.
Upon arrival of vertex $\upsilon_i \in V$, let denote $j \ge 0$ the minimum *load* among u_i's neighbors of V. Assign vertex u_i to the highest priority according to Π_{j+1}.

Figure 5.15: A Randomized Permutation Algorithm.

Again, Azar et al. have shown that the expected competitive ratio of the above algorithm is at most $k = 1 + \ln(n)$, where $n = |U| = |V|$. They also proved that the competitive ratio of any randomized on-line assignment algorithm is at least $k - 1 = \ln(n)$.

5.2.5 On-line Maximum Matching

Kalyanasundaran and Pruhs [198] consider the on-line algorithm of maximum weight bipartite matching problem. They require the bipartite graph being complete with the positive weights and satisfying the triangle inequality.

Algorithm *On'ine-D-MAX3;*

Upon arrival of a *boy* (i.e., a vertex $u \in U$), add the *max-weight* edge that adjoins the presented *boy*, but is *not* adjacent to any edge already in the matching.

Figure 5.16: A Deterministic On-line *Max*-matching Algorithm for Metric, Bipartite Graphs.

We apply again the *dual bounding technique* to find the optimal competitive ratio of the maximum weighted matching algorithm.

Theorem 5.6. *In any metric, bipartite graph, the Online-D-MAX3 matching algorithm achieves an optimal competitive ratio of 3.*

Proof: If Algorithm *Online-D-MAX3* adds an edge (i, j) to the matching, let $\Pi_i = 2\cdot d(i, j)$ and $\Pi_j = d(i, j)$.

If (i, j) is a matched edge, we clearly have that $d(i, j) \leq \Pi_i + \Pi_j$; otherwise, suppose j was presented and matched to k. If *i* was *not* yet matched at the point, then $d(i, j) \leq d(j, k) = \Pi_j$; otherwise, suppose *i* was already matched to h.

When h was presented, k was *not* yet matched, so $d(k, h) \leq d(i, h)$. Thus, we have $d(i, j) \leq d(i, h) + d(h, k) + d(k, j) \leq 2\cdot d(i, h) + d(k, j) = \Pi_i + \Pi_j$. Therefore, the weight of the matching is $\sum_i \Pi_i / 3$. Since $\sum_i \Pi_i$ is an upper bound on the maximum weight of a matching, the performance ratio of the considered algorithm is 3. □

All previous work provides analysis *only* for *metric, bipartite* graphs with restricted positive weights. In contrast, *Bernstein and Rajagopalan* [55] propose a variant on-line maximum matching algorithm which is 4-competitive on general graph with arbitrary weights.

In order to describe this algorithm we first need some *definitions* and *conventions*. Given a graph G = G(V, E) with the edge set $E \subseteq \binom{V}{2}$. An instance of the *on-line matching problem*, consists of G plus some ordering $\prec$ on V; if a vertex *i* arrived earlier than j we say $i \prec j$.. We refer to the vertex that just arrived as υ. Let Y be the set of vertices that have not yet arrived plus υ and let X be the set of vertices that have not yet matched but have arrived in the past. Let denote by (X, Y) = B the bipartite graph on the vertices X and Y with weight as have been given to us. We let $m(\beta)$ be the weight of maximum matching M(B) on B.

Next, we define a potential $\beta(y) \stackrel{def}{=} m(\beta) - m(B - \{y\})$ which is associated of that vertex and let also define a *global potential function*

$$\Phi \stackrel{def}{=} m(\beta) + 2 \cdot \{\text{weight of the edges that have matches so far}\},$$

which is exactly the maximum weight of the matching and measures the efficiency of the current service.

We state the on-line maximum matching algorithm using u to denote the vertex that υ is matched to (if one exists), in some such matching M.

Algorithm *Online-D-WMM;*
Examine only two options:
• *MATCH* option: Match υ to u.
• *NONMATCH* option: Add υ to X.
Pick the option that minimize Φ.

Figure 5.17: An On-line Maximum Weighted Matching on General Graphs.

We use conductive analysis and the dual bounding technique to find the competitive ratio of this algorithm.

Theorem 5.7. *Algorithm Online-D-WMM has a (minimal) competitive ratio of 4 on general graphs with arbitrary weights.*

Proof: We use the dual bounding technique in a similar way as in *Theorem 5.3.* Consider any edge (i, j) in the graph. We get that the global potential function has to be at least

$\frac{1}{2}d_{ij}$ Φ, when *i* and j arrive. Thus, $\Phi_{final} \geq \sum_{(i,j)\in M} d_{ij}/2 = \frac{1}{2}|M|$ for any matching M in the graph. Now, since Φ_{final} = 2·{the weight of the algorithm's matching}, the algorithm has a worst case performance of at lest 1/4 (i.e., a competitive ratio of at least 4). □

Bernstein and Rajagopalan [55] proved that any deterministic, on-line (*maximum*) weighted matching algorithm has a competitive ratio of at least 3. We can easily extend the proof of *Theorem 3.2* of [196] to get the same result on general graphs with arbitrary weights.

The same article [55] presents the following *deterministic on-line max-matching* algorithm which achieves an optimal competitive ratio of at least $\frac{3}{2}$ for *unweighted* graphs.

Algorithm *Online-UMM;*
1. Compute T, the new B that would result if NOMATCH was chosen.
2. If $m(B) \geq m(T)$, then MATCH υ and u (if u exists), and DISCARD υ otherwise.
3. Otherwise, NOMATCH: set B := T.

Figure 5.18: An On-line Maximum Unweighted Matching on General Graphs.

An interesting open question is whether we can bridge the gap between the lower and upper competitive bounds of an on-line *max*-weight matching algorithm on general graphs. We know that randomization has been shown to be very helpful in designing on-line algorithms with better competitive ratios. Clearly, we can see that any randomized algorithm cannot achieve a better competitive ratio than $\frac{5}{4}$ in the *unweighted* case by using an extension of the competitive analysis through *Yao's lemma* [344].

Another interesting open problem comes up: *Can we design randomized algorithms against oblivious or lazy adversaries (even harder) with better competitive ratios ?*

5.3 Specific Combinatorial On-line Problems

In this section we briefly discuss two specific problems, the *String Matching* and the *Network Flow* problems in on-line setting.

5.3.1 On-line String Matching

The classical *(off-line) string matching problem* detects occurrence of a particular substring (called a *partition*) in another string (i.e., the *text*).

In on-line setting, the *on-line string matching* tests of each prefix of the input string is *superprimitive* (i.e., it is covered only by itself) as soon as that the prefix is revealed.

Breslauer [67] recently proposed an on-line algorithm which works under the general alphabet assumption where the only access to the input string is by comparisons of pairs of symbols. This algorithm is simpler and more effective than the (off-line) algorithm of *Apostolico et al.* [353] and uses the pattern processing steps of the *Knuth-Morris-Pratt* string matching algorithm [100] *only* once. *Breslauer's algorithm* scans the input string $S_{[1..n]}$ one symbol at a time and uses linear auxiliary space. The new algorithm takes O(n) time and at most 2·n comparisons of input symbols.

It is *not* our intention to describe this algorithm here, but we would like to point out that there are some interesting open problems if we consider variant on-line models for the on-line string matching.

5.3.2 On-line Network Flow

Very recently, ***Phillips and Westbrook*** [280] used the method of competitive analysis to study the ***on-line load balancing*** problem and describe an efficient scheduler that uses only a small number of reassignments to reduce its competitive ratio.

They then applied this problem to compute the ***maximum flow*** in a network. In addition, they used an on-line game, ***the kill game*** [76], on a bipartite graph $G = (U, V, E)$ as a fundamental step in improving the network flow algorithm. They proposed a simple, efficient and deterministic on-line algorithm for ***network maximum flow***, which runs in $O(m \cdot n \cdot log_{m/n}\, n + n^2 \cdot log\, 2 + \varepsilon \cdot n)$ for any constant ε, where $|V| = |U| = n$ and $|E| = m$.

5.3.3 On-line Scheduling

Classical (or ***clairvoyant***) ***scheduling theory*** of tasks (i.e., the characteristics of the tasks are known a ***priori***) is a basic problem in computer science and has been studied extensively [159,238]. This problem is often inherently ***on-line*** in nature and in many areas of operating systems (e.g., ***time-sharing operation systems*** [323]); one needs algorithms to schedule a sequence of tasks where each task has to be processed before the future of the sequence is determined. Most research on on-line scheduling concerns the problem of minimizing the length ***makespan*** of schedule [41,158,202,311].

There has been some ***recent work*** on ***non-clairvoyant scheduling*** [8,29,39,259] using the competitive analytic approach. Some of these problems are the following:

- *On-line* task *scheduling* on a *single* machine where tasks have fixed start and end times [343].

- *On-line scheduling* in *real-time systems* [116,227,228].

- *On-line scheduling* on *parallel machines* with different network topologies of n processors [44,135,311].

We summarize the upper and lower competitive bounds for scheduling on a parallel machine with a specific network topology under the *assumption* that *only* running times are given dynamically and that there are *no dependencies* among tasks (in the *case of dependencies*, see [134]).

Network topology	**Upper bound**	**Lower bound**
Two-dimensional *mesh*	$O(\sqrt{\log\log n})$	$\Omega(\sqrt{\log\log n})$
PRAM	2-1/n	2-1/n
Hypercube	2-1/n	2-1/n
One-dimensional *mesh*	2.5	2-1/n
d-dimensional *mesh*	$O(2^d \cdot d \cdot \log d \cdot \sqrt{\log\log n}) + 2^d \cdot (d \cdot \log d)^d$	$\Omega(\sqrt{\log\log n})$

Table 5.2: On-line Scheduling Algorithms on Parallel Machines and their Competitiv Bounds.

A problem closely related to on-line scheduling is *on-line load balancing* [15,30,31,280], where an algorithm has to assign a sit of tasks to processors and the objective is to minimize the maximum processor load.

We list below some different *on-line problems* which have searched very recently:

- *On-line bin-packing* [73];
- *On-line knapsack problem* [358]; and
- *On-line routing* for *virtual circuits* [15,16,23].

Generally, there are some fundamental questions which remain *open* in on-line scheduling:

- How can *on-line models* be extended to serve *practical* scheduling even better?
- Can we design randomized, competitive, scheduling algorithms and show that *randomization* is a powerful tool for *on-line scheduling*?

The great tragedy of science is the slaying
of a beautiful hypothesis by an ugly fact.

T. H. Huxley

Chapter 6

On-line Algorithms in Computational Geometry

Science is nothing more than a searching.

Albert Einstein

This chapter is concerned with the *incremental* and *on-line* applications in *Computational Geometry*. Particularly, we consider the *on-line navigation problem* in an unknown geometric environment and the *on-line* (*visual or geometric*) *routing* problems for *planar* graphs under the model of *fixed graph scenario*.

6.1 Introduction

Computational Geometry studies the design and analysis of algorithms for solving geometric problems. It is a recent field of *Theoretical Computer Science*, that has developed rapidly since it first appeared in *M. I. Shamos' thesis* [314] in 1978. The field has already reached a high level of research sophistication and it was important to develop more practical algorithms avoiding the use of complicated data structures in order to design efficient geometric algorithms.

Randomized incremental algorithms introduced to the field by *Clarkson* [91] in 1985 and have been successfully applied to a variety of geometric problems [266,267,331]. These algorithms are simpler or asymptotically more efficient in practice rather than those previously known. Randomization helps in design and analysis of such incremental constructions and gives a general way to "*divide and conquer*" geometric problems, which can be used in the *parallel* as well as in the *sequential* computation.

Clarkson and Shor [91] have given a general framework in which geometrical problems are stated in terms of objects, regions, and conflicts between objects and regions. The algorithms *incrementally* (i.e., in the sense that the points are introduced one at the time) construct the set of regions defined by a current subset of the input objects which are not in conflict with these subsets and are maintained in an additional data structure which is called the *conflict graph*. Domains for such incremental geometric problems have included:

- *Convex hulls* [125];
- *Delaunay trees* [61];
- *Delaunay triangulation* of a set of points in any dimension [62,63,74,167];
- *Voronoi diagram* in any dimension [61,62].
- *Visibility graphs* [168];

There are also some algorithms which do not impose the restriction that all the points have to be known in advance and maintained in an auxiliary data structure (i.e., the *conflict graph*) and thus, are more "*on-line*". Some of such on-line algorithms have been for the following problems:

- *Convex polygons* [266,281,331];
- *Convex hulls* of a set of points [266,331];
- *Packing and Covering* geometric objects [102,192,229,236,297];
- *Steiner trees* [7,334,341];
- *Closest-pair problem* [304];
- *Robot navigation* [110,112,199,273].

Furthermore, some generalized techniques have been developed, in order to dynamize large classes of geometric algorithms (summarized in [266,331]).

6.2 On-line Navigation in an Unknown Environment

In this section, we study the *on-line navigation problem*, where an on-line algorithm is trying to reach a specific target point in some unknown geometric environment. The *goal* of an on-line algorithm is to optimize the amount of *searching* (i.e., minimize its competitive ratio of the on-line strategy) before the target point is found. This on-line problem has connections with the *k-server problem*, when the environment is a *layered graph* [112,140,195,200,288].

6.2.1 Problem Motivation and Related Results

A natural problem in robot motion planning is the searching for a specific recognizable object in a geometric environment *with* or *without* obstacles in it.

Particularly, this problem can be divided into two categories:

- *Motion path planning* through a *static* and *known* geometric environment in which the robot has a complete information (e.g., a map) of the environment in advance [303,305,345]; and

- *Navigation in an unknown scene* in which an autonomous robot has to efficiently traverse its way through a new environment [195,199].

The design and evaluation of algorithms for such navigation is a classical and interesting algorithmic problem of motion planning for which a few results exist [36,109,110,112,220,222,230,251]. However, this problem deserves more theoretical research.

We consider the problem of a point robot (*automaton*) which has to travel in an unknown *simple* class of polygons from any point *s* (*starting point*) to another point *g* (*goal*). It is interesting for many real life situations to consider the second category of the problem for finding a path *dynamically* (i.e., in *on-line* fashion) based only on the local visual information that the mobile robot (i.e., a robot with an on-board vision system) gathers through. During the last five years, the interest of on-line algorithms of motion planning has grown [60,70,106,109,183,184].

Lumelsky and Stepanov [251] earlier studied a similar problem when a robot with a *tactile sensor* moves in an unknown environment of non-convex obstacles and the robot can perceive an obstacle only when it hits it. Then it searches the obstacle's contour for a leaving point with minimum distance to the goal and updates from there. Recently, this algorithm has been extended for solving the *three-dimensional* path planning problem in an unknown environment containing obstacles of arbitrary shape, under the assumption that an *exploration algorithm* is available to the robot [250].

Blum et al. [60] have constructed a $(6 \cdot k + 4)$-vertex scene with *only one* obstacle, for an integer $k \geq 2$ such that every deterministic on-line algorithm (even if it perceives the currently visible part of the scene) needs more than $3 \cdot (k - 2)$ steps up to reaching the target.

Papadimitriou and Yanakakis [273] were the first to consider *competitive algorithms* and *analysis* for scenes of disjoint *isothetic* rectangles (i.e., unit-size squares) with sides parallel to the axes. They were able to find an asymptotically 3/2-competitive algorithm and prove that there was *no* on-line algorithm that had a bounded competitive factor for scenes with arbitrarily *thin* rectangles (i.e., rectangles of unbounded *aspect ratio*: the ratio of the longer side to its shortest side).

Later on, *Chan and Lam* [70] constructed an on-line algorithm for the robot to determine an obstacle-free path to its *goal* point dynamically, that is, with no information about the obstacles in advance. They showed that if the aspect ratios of the obstacles were bounded by some constant aspect ratio r of every *rectangular* obstacle in the scene, then an asymptotically $(1+\frac{r}{2})$-competitive on-line algorithm could be designed for navigating in an unknown environment. Recently, *Mei* and *Igarashi* [263] proposed an efficient $(1+\frac{3}{5}\cdot r)$-competitive strategy for robot navigation in an unknown environment containing rectangular and rectilinear obstacles. This on-line algorithm gives a better competitive ratio of $\frac{8}{5}$ than the ratio $\frac{5}{3}$ obtained by the mixed heuristic presented in [273] for the special case of *square* obstacles (i.e., when $r = 1$).

Similar on-line problems have been studied for *searching*, *exploring* and *mapping* using *visual* information [110,195,197,199,273]. In particular, *Blum et al.* [60] formulated the *room problem*, in which the robot has to move from a corner to the corner of a square room, provided that the obstacles are rectangles or convex polygons. They presented an on-line algorithm with a tight lower bound of $\Omega(\sqrt{n})$ on the competitive ratio of the Euclidean distance n traveled by the robot to the shortest obstacle avoiding path. Recently, *Blum* [360] generalized the above result by developing an optimal deterministic $\Omega(\sqrt{\frac{n}{i}})$-competitive on the robot's ith trip for all $i \leq n$. *Karloff et al.* [206] proposed a randomized O(1)-competitive algorithm for the *room* problem.

Klein [220] studied another navigation problem in a simple polygon so-called a *street*. A simple planar polygon (P, s, g) with two distinguished vertices, s and g, is a *street* if and only if the two boundary oriented chains L (*left*) and R (*right*) from s to g are *mutually weakly visible* (i.e., each point of L can be seen from at least one point of R and vice versa). He described an on-line strategy for finding a short path from s to g in a street, which achieved a competitive factor of $(1 + \frac{3}{2} \cdot \pi)$ (< 5.72) in the *Euclidean metric* L_2. Moreover, this strategy has a lower bound of $\sqrt{2}$ (> 1.41) on the competitive factor for searching in a street.

Recently, *Kleinberg* [222] has considered a simple on-line algorithm for this problem improving the competitive ratio to $2 \cdot \sqrt{2}$ (< 2.83). He also proved that his strategy has an optimal $\sqrt{2}$-competitiveness for searching in *rectilinear streets*.

Additionally, *Dalta and Icking* [106] defined a new, *strictly* larger class of simple polygons, called *Generalized streets* (*G-streets*, for short) and presented an on-line strategy which achieves an optimal 9-competitive ratio (resp., $\sqrt{82}$ -competitive) in L_1 (resp., L_2) metric for searching in an unknown *rectilinear G-street*. We can easily extend the results and develop an on-line strategy which achieves a competitive ratio of 18 in L_1 metric for searching in unknown rectilinear *twice-G-streets (2-G-streets)*; that is, *a rectilinear simple planar polygon*, every boundary point of which is *mutually weakly visible* from a point on a *horizontal or vertical* line segment connecting the two boundary oriented chains L and R from the points s and g. The interesting open question remains if there exists a more general natural class of simple polygons that can be searched competitively.

In the next section, we present a greedy on-line algorithm which achieves a competitive ratio of $\sqrt{3}$ (< 1.733), improving the best upper bound known is the literature for the visual searching in a street. Moreover, we show that $ln5 = 1.6094$ in the *best* randomized competitive upper bound for any on-line algorithm for visual searching a street. The last result shows that randomization is strictly more powerful for this problem.

6.2.2 On-line Visual Searching in Unknown Streets

Let P be a *street* with a *starting* point s, and a *goal* point g on the boundary Bd(P). For simplicity, we assume that *no* three vertices of P are collinear and the polygonal chains L and R are ordered in direction from s to g. We state some *definitions* and *visibility properties of streets*, before we describe the on-line strategy itself.

The *visibility polygon* $Vis_P(p)$ of the polygon P from $p \in P$ is the set $\{y \in P$; y is visible from p$\}$. The *extended visibility polygon* $EV_P(p)$ of P at a point $p \in P$ consists of all the boundary points of P that have seen so far.

We define a *bay* (or *cave* [222])[1] B to be a connected chain of $Bd(P)$ such that the robot has seen the endpoints of the chain but *no* other points of it. A *pharos* (*sightpoint* or *cavemouth*) of a bay B is the closest reflex vertex of the $Bl(P)$ that robot sees from some point of its path. Clearly, we have the *left pharos* υ_l (or *right pharos* υ_r) of the *left bay* B_L (resp., *right bay* B_R) of the street P. Let $d(.,.)$ denote the L_2 length of the shortest path between two points in P. The shortest (s, g)-path $\mathcal{T}$ from s to g is a chain of the segments joined at reflex vertices of P.

We have the following easy facts:

Lemma 6.1. [222]

(i) If g is contained in a bay B (left bay B_L or right bay B_R) and $x \in \mathcal{T}$, then the (x,g)-path of $\mathcal{T}$ touches either pharos (resp., left υ_l or right υ_r) of B.

(ii) Let $p \in Bd(L)$ (or $Bd(R)$) and let $\Psi \in L$ (resp. $\Psi \in R$) be the (s, p)-boundary chain of P. If the robot moves from s to p in P, it will have seen every point on Ψ.

(iii) *All left (right) pharos of P lie to the left (right) of all right (left) pharos of P.*

---

[1] Here we would like to mention that we adopt *Kleinberg's* terminology [222], although *different* notations were used when our on-line strategy has been first developed indepentently.

We remind that a *monotone path* from a point p_1 to another point p_2 of the shortest *(s, g)*-path $\mathcal{T}$ is one where the *x*- and *y-coordinates* of the points on the path never decrease along the direction of the straight-line path. Next, our on-line strategy is stated *iterative*ly.

Procedure Street-SPS; {* *Shortest Path Strategy* *}

{* This strategy finds a short path from *s* to *g* in a street *not* known in advance. *}

const *s*: Point-of-*P*; { * the starting point * }

g: Point-of-P; { * the goal (target) point * }

var *p* : Point-in-*P*; { * current position * }

q : Point-in-*P*; { * here an event occurs; q is called an *event point* * }

υ_l , υ_r : Point-of-*P*; { * the most advanced points on *L* and *R*, respectively, that the robot has so far identified * }

begin { * *Street-PSP* * }

p := *s*;

Determine EV_P (p) and υ_l, υ_r ; { * if both of υ_l and υ_r exist * }

while (*g* is not visible from p) **do**

If the reflex vertex υ_l (or υ_r) is *not* defined,

then { * *Case 2*: there are *no* right (resp., left) *pharos* * }

p := υ_r ; (resp., p := υ_l ;)

else if p, υ_l , υ_r are collinear,

then { * *Case 3* * }

p := the closer of (υ_l , υ_r);

else (* *Case 4*: Both of υ_l and υ_r are visible in EV_P(p) *)

Choose a direction of motion such that υ_l lies to its left and υ_r lies to its right. Walk straight to this direction until at some point q of robot's path we have that $\overline{\upsilon_l q}$ (or $\overline{\upsilon_r q}$) is parallel to the x-axis.

Move on the direction $y = -x$ (resp., $y = x$); it depends if υ_l (resp., υ_r) has been seen first. The robot continues to move on the diagonal direction *monotonically*, updating the extented visibility and the points υ_l, υ_r, until it hits the boundary of P or arrives at some diagonal point p' in which one of the following events happens:

(E1). The robot has the same $u = max(x_l, y_r)$ x- or y-coordinate with υ_l or υ_r, respectively:

(E2). One of the chains L_X or R_X becomes completely visible, where L_X (resp., R_X) is the portion of boundary chain between υ_l (resp., υ_r) and the endpoint of X lying on the negative x- (resp., y-) axis.

Set p := p' ;

end; {* if *}

Determine new EV_P(p), υ_l *and/or* υ_r in $EV_P(p)$;

Update $EV_P(p)$, υ_l and/or υ_r ;

end; {* *while* *}

Walk straight towards g; {* *Case 1* *}

end. {* ***Street-SPS*** *}

Figure 6.1: A $\sqrt{3}$-competitive On-line Strategy for a Street.

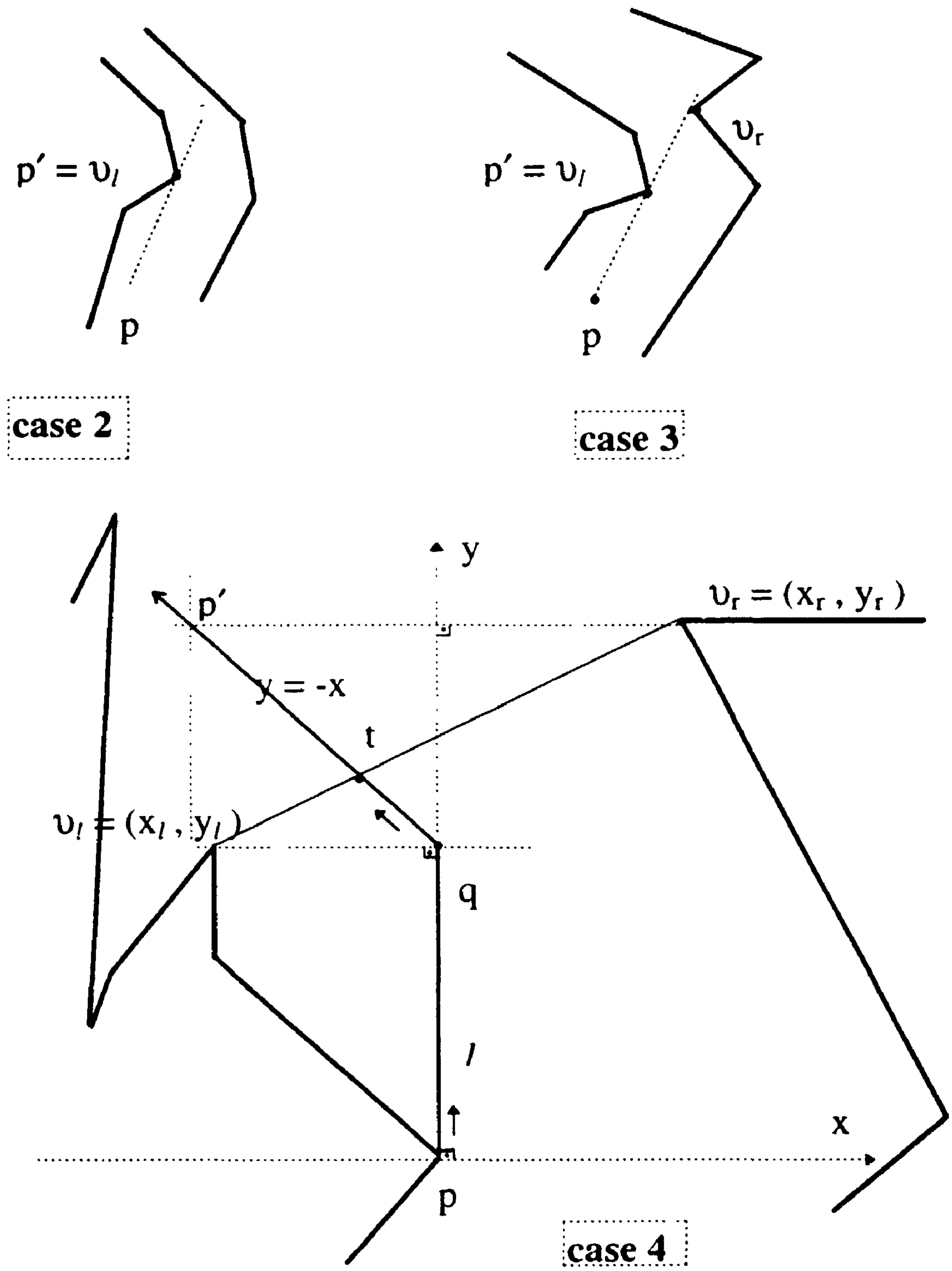


Figure 6.2: Robot Movement Cases.

The following theorem provides the main inductive result.

Theorem 6.1. *For any street P, the on-line algorithm* **Street-SPS** *creates a (s,g)-path that does not exceed $\sqrt{3}$ times the length d(s, g) of the shortest (s, g)-path in P (i.e., this deterministic strategy achieves a competitive ratio of $\sqrt{3}$).*

Proof: If only one of the *cases 1-3* applies (see *figure 6.2*), we easily get that the robot follows the path from p to p′ that is monotone with respect to the chosen coordinate system. That is, the robot has traveled no more than $\sqrt{2}\cdot$d(p, p′) in L_2 metric.

Now, suppose that *Case 4* holds. Let l denote the distance traveled by the robot (see *Figure 6.2*) before $\overline{\upsilon_l q}$ or $\overline{\upsilon_r q}$ to be parallel to the x-axis and let consider the case in which *event* (E1) occurs first. Assume that the right *pharos* $\upsilon_r = (x_r, y_r)$ has the same *y-coordinate* as the robot. Also, let be $\upsilon_l = (x_l, y_l)$ and $x_l \le y_r$. Then the robot travels $l + \sqrt{2}\cdot y_r$, while we have d(p, p′) $\ge \sqrt{(l+x_r)^2 + x_r^2}$. Thus, the worst-case competitive ratio of our strategy has to be bounded by

$$\sup_{l, x_r, x_r \in \Re^+} \left(\frac{l+\sqrt{2}\cdot x_r}{\sqrt{(l+x_r)^2+x_r^2}}\right) \le \sqrt{3} \quad \textbf{(6.1.1)},$$

where $\Re^+$ denotes the set of non-negative real numbers.

Next, suppose that the event (E1) occurs, and $\upsilon_l = (-x_l, -y_l)$ has the same x-coordinate as the robot. If we have $x_l > y_r$, then the robot travels $l + \sqrt{2}\cdot x_l$ and we get that d(p, p′) $\ge \sqrt{l^2 + x_l^2}$. Therefore, the worst-case competitive factor of our strategy has to be bounded above by

$$\max_{l, x_l \in \Re^+} \left(\frac{l+\sqrt{2}\cdot x_l}{\sqrt{l^2+x_l^2}}\right) = \max_{w \in \Re^+} \left(\frac{\sqrt{2}\cdot w+1}{\sqrt{w^2+1}}\right) \quad \textbf{(6.1.2)},$$

where $w = \frac{x_l}{l}$ and $l \neq 0$. A simple analysis shows that the maximum is reached at $w = \sqrt{2}$ at which the maximum value is $\sqrt{(\sqrt{2})^2+1} = \sqrt{3}$.

Corollary 6.1. *If the street P is rectilinear, the on-line strategy* ***Street-SPS*** *has an optimal competitive ratio of* $\sqrt{2}$.

Proof: Since *Case 4* cannot occur when we apply the greedy algorithm ***Street-SPS*** in a rectilinear street.

Corollary 6.2. *The space complexity of the on-line algorithm* ***Street-SPS*** *(i.e., the memory size needed by the robot) does not depend on the street but only on the maximum complexity of the visibility polygons encountered.*

Kleinberg [222] mentioned (*without proof!*) that his simple on-line strategy is $(\frac{1+\sqrt{5}}{2})$-competitive. This argument is *not* true. Even our strategy *cannot* achieve the above competitive ratio, because $\sqrt{2}$ is the minimum value that maximizes both formulas (6.1.1) and (6.1.2).

Theorem 6.2. *There is no better randomized ln5-competitive strategy (ln5 < 1.6095) against oblivious adversaries for visual searching an unknown street.*

Proof: The result follows from a randomized technique similar to those in [110] for on-line motion planning.

If a street has *four* vertices (see *Figure 6.3 (a)*), there is a strategy with competitive ratio of $\sqrt{2}$ in metric L_2.

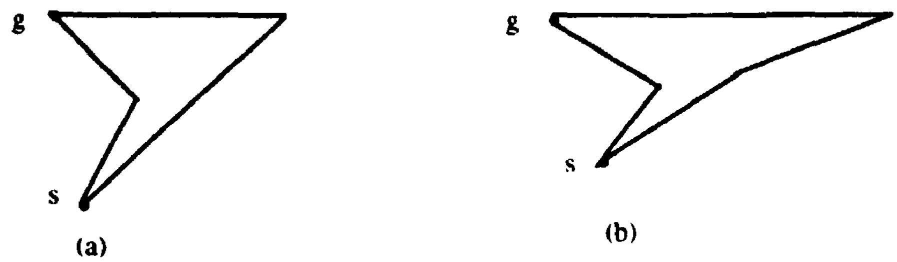


Figure 6.3: Visual Search in *Streets* with *four* and *five* Vertices.

If the street has *five* vertices (see *Figure 6.3 (b)*), there exists a randomized H_5-competitive algorithm (i.e., the *harmonic* number $H_5 \approx ln5 < 1.6095$) against an *oblivious* adversary. Therefore, there is *no* randomized strategy which achieves a better competitiveness than H_5 against an oblivious adversary for any street with more than five vertices.

6.3 On-line Geometric Routing for Planar Graphs

We consider the *on-line* (*geometric* or *visual*) *routing problems* on an initially unknown weighted *plana.* graph under the *fixed graph scenario* [195] and present deterministic competitive algorithms obtain the route. This problem is mostly an on-line graph problem, than one that has been done within a framework of computational geometry.

6.3.1 On-line Traveling Salesperson Problem

Routing problems [46,126,272] involve the periodic collection and delivery of goods and services which are of great practical importance. The practical goal of finding a route of such problems is the cost minimization and service improvement. Abstractions of these problems can be modeled easily and naturally with graphs. Unfortunately, many of these interesting standard (*off-line*) routing problems, including for instance the well known k- *traveling salesperson problem* [237,238,272] (i.e., k-*TSP*, for short, where k ≥ 1) are *NP-complete* in the sense of *Cook* [97] and *Karp* [209].

Pruhs et al. [195] considered the *on-line 1-TSP* (also called *Visual 1-TSP*) for a *planar* weighted graph G = (V, E) under the *fixed graph scenario* (*FGS*, for short; where the on-line algorithm is aware of every edge incident to a visited vertex, but |V| = n is *not* known in advance). This on-line *FGS* is variant to that so-called *point-by-point scenario* [112,273], where the points are revealed one at a time. The *goal* of the searcher's robot is to visit each vertex of G incurring as little cost as possible.

Pruhs et al. [195] presented the following *modified* on-line algorithm for the visual *1-TSP* under the *FGS*. We note that the distances of the vertices require *only* to be non-negative w.l.o.g. and need *not* satisfy the *triangle inequality* in the planar embedding.

Algorithm ***Visual-1-TSP*** *(x, y: Vertices; G : Graph)*;

{* Note that x is the starting vertex of G *}

1. Compute *ONG (G)*;

   {* *ONG(G)* is a planar graph that contains the MST of Visibility graph of G; this step takes $O(n^2 \cdot log n)$ tim complexity *}.

2. $\forall y \in$ *ONG(G)*, apply *Modified-Shortcut (x, y: Vertices; ONG(G): Graph)*; {* in $O(n^2 \cdot log n)$ time *}.

Procedure Modified-Shortcut (x, y: Vertices; G: Graph);

{* Traveling from x, the searcher visits y for the *first* time *}

begin

for each *boundary edge* vw

  if the visited for first time vertex y belongs to an object (obstacle), **then**
  the searcher (robot) circumnavigates the perimeter of the object.

  end; {* if *}

if Block (υw) = ∅, **then**

add a *jump edge* yw at the end of *Incident (y)* and *Incident (w)*;.

endfor;

for each edge yz ∈ Incident (y) **do**

if z is a *boundary vertex* and yz is a *shortcut*, **then** traverse the edge yz;.

Shortcut *(y, z, G)*;

elseif z is a *boundary vertex* and yz is a *jump edge*, **then**

traverse the shortest known path from y to z;

Shortcut *(y, z, G)*;

endif

endfor

return to x along the shortest known path;

end. {* of *procedure** }

Figure 6.4: A Modified On-line Algorithm for the Visual *1-TSP*.

Theorem 6.3. The on-line algorithm[1] *Visual 1-TSP* is 17-competitive.

Proof: *Theorem 4.7* [195].

The total complexity time of of the above on-line heuristic is $O(n^2 \cdot \log n)$; that is, the same time required by the standard (off-line) *all-pair shortest path algorithms* for *sparse* graphs [272]. This result shows that the ability of an adversary to map from a distance is the reason that competitive algorithms cannot be obtained for *mapping problems* [110,199] under a *point-by-point scenario* (*PBPS*).

Furthermore, this heuristic easily solves an interesting *open* question (see [109], *Conjecture 1*) that there exists a constant competitive algorithm for exploring rectilinear

[1] Note that this on-line strategy uses terminology from [195], which is not repeated here.

polygons with any number of rectilinear obstacles in it. Clearly, this problem is solvable under *FGS* even for general simple polygons and obstacles, but it remains open under the *PBPS* for rectilinear polygons and obstacles.

The computation of a tour in on-line setting under the *FGS* has some relations to broadcasting in a network with unknown topology. It would be challenging and interesting to generalize the result for on-line *TSP* on a general weighted graph under *FGS*.

6.3.2 On-line Geometric k-*CPP* for Planar Graphs

We extend the visual *1-TSP* to *k-CCP* (i.e., *k-Chinese Postman Problem*, k>1) and other on-line routing problems in plane.

Definition 6.1. Let G = (V, E) be an undirected multigraph with a positive cost function defined on $E \subseteq V \times V$. A *k*-route (or *k*-circuit) is a set of *k*-cycles that start from a fixed vertex υ_s (*post office*) and collectively cover every edge in the planar graph. When $k = 1$, *k-CPP* is degenerated as *1-CPP*.

Lemma 6.2. *There is an $O(n^2 \cdot \log n)$ approximation algorithm for the visual 1-CPP which achieves a competitive ratio of 17.*

Proof: By *Theorem 6.3*.

Lemma 6.3. *Assume that the degree of a fixed node υ_s in a planar graph G is 2k-1 (or 2k; k=1,2,3,...). Then the optimal tour of the visual k-CPP can be immediately formed from that of visual 1-CPP.*

Proof: The odd node (the case of an even node is similar) υ_s must be matched for visual *1-CPP*. After forming the optimal tour for visual *1-CPP*, υ_s becomes an even node the degree of which is at least 2k. Starting from υ_s a postman has to go through it at least 2k times in order to traverse each edge at least once in G. The 2k times correspond to k-cycle tours for which at least one edge differs from others. Considering adding no new edge to G when the tour of visual *1-CPP* is decomposed into that of *k-CPP*, which is *optimal*.

The on-line *k-CPP cannot* be obtained by *Lemma 6.3* when the degree of fixed node υ_s is $2 \cdot n$ less than $2 \cdot k$ after applying a non-negative weighted on-line matching algorithm. In other words, assuming that n postmen in tours can be formed at υ_s using matching algorithm, there are k-n postmen to be arranged. We want to construct k-n non-decreasing tours p_i (where i = n+1,..., k) through node υ_s subject to the on-line version of *definition 6.1*. These tours obtained in such way are called *artificial tours*.

Definition 6.2. A *spanning tree (ST)* with root node υ_s is defined as an *arborecent spanning tree (AST) T* if the distance is the shortest one between υ_s and each node belonging to *T*.

Lemma 6.4. *Each artificial tour contains at most one edge which does not belong to a arborecent spanning tree T.*

Proof: Assume that an artificial tour p contains two edges $e_1, e_2 \notin T$. We consider the e_1 directly (or indirectly) connected with e_2, we easily reach a *contradiction*.

The above *Lemma 6.4* gives the construction of those k-n artificial tours, denoted by *P(E)*, based on *AST T* using the following procedure:

Algorithm $A_n^{(k-n)}$

1. Select ea h e ∈ E.
2. Compute each smallest *Euler tour* p(e) containing e from vertex υ_s.

Figure 6.5: An Algorithm to find the Artificial Tours of a Planar Graph.

Clearly, $|E|$ -artificial tours can be found in O(|V|²) time using *Lemma 6.4*. We can also order these tours *P(E)* in O(|V|²·*log*|V|²) time. Therefore, the tours *P(E)* can be found in at most O(|V|³) time, since the arborecent *spanning tree T* can be computed in O(|V|³) time.

Next, we assume that the optimum solution for visual *1-CPP* in a planar G is obtained from node υ_s the degree of which is changed into 2·n after matching. Thereore, there are (k-n) artificial tours obtained from *P(E)*, denoted by $P_n^{(k-n)}$. Now, we have the following (*semi)* on-line algorithm for the geometric *k-CPP*:

Algorithm $V_n^{(k-n)}$

1. Compute visual *1-CPP*, denoted by V_n^1; {* This step takes O(|V|²·*log*|V|) time complexity *}.
2. Find *ASP T* with root υ_s; {* in O(|V|³) time *}.
3. Find *P*(E), based on *T*; {* in O(|V|³) time *}.
4. Design k post tours from V_n^1 and $P_n^{(k-n)} \in$ *P*(E); {* in O(|V|) time *}.

Figure 6.6: An On-line Visual k-*CPP* Algorithm

Theorem 6.4. *The Algorithm* $V_n^{(k-n)}$ *for on-line k-CPP is an* $O(|V|^3)$ *approximation on-line algorithm which achieves a competitive ratio of 18.*

Proof: *Step* 1 designs n postmen's tours, denoted by $OPT_l(I)$, where I is an instance for the visual *k-CPP*. *Step* 2 constructs (k-n) 1-post tours denoted by $P_n^{(k-n)} \in P(E)$. We have that the optimum solution $OPT_k(I)$ of *k-CPP* obtained by the above algorithm $V_n^{(k-n)}$ satisfies the following formulas:

$$|OPT_k(I)| \geq |P_n^{(k-n)}(I)| \text{ and } 17 \cdot |OPT_k(I)| \geq |OPT_1(I)|.$$

Therefore, $|V_n^{(k-n)}(I)| = |OPT_1(I) + P_n^{(k-n)}(I)| \leq 18 \cdot |OPT_k(I)|$.

Similarly, it is easy to obtain greedy competitive algorithms which yield approximate solutions for other *routing problems*, for example k-*DCPP* (*directed k-CPP*), *k-TSP*, *k-SCP* (*k-stacker-cranes problem*) which are all *NP-complete* for k > 1, since they contain the *Hamiltonian Path Problem* as a particular case [209].

Recently, *Ausiello et al.* [16] have considered a variant *on-line* version of the *routing problem* for *planar* graphs in which each request can be served only after a certain *release time*. This *on-line scheduling* problem has many applications from *robotics* to several *transportation* problems. They have proposed a 5/2-competitive exponential routing algorithm and a 3-competitive (resp., 7/3-competitive) strategy for an Euclidean space (resp., line). They also proved that no on-line routing algorithm (either deterministic or randomized) could achieve a competitive factor lower than 2 in Euclidean plane. This lower bound is not necessarily valid for every metric space.

The main open question is to bridge the gap between lower and upper bounds for al these problems mentioned in this section. Moreover, anothe' challenging open research problem is to extend these results for general graphs or different metric spaces and network topologies using more than one server (robot) as well..

The real danger is not that robots will begin to think like humans, but that humans will begin to think like robots.

S. J. Harris

Chapter 7

Conclusions and Future Directions

There is nothing new except what has been forgotten.
Marie Antoinette

How extremely stupid not to have thought of that!
T. H. Huxley

There remain many on-line applications which have not yet been explored. Our research by no means covers all the areas where the theory of on-line algorithms applies. Aside from designing algorithms in many areas where on-line problems arise, more general issues in this field are not searched. Moreover, the field of incremental algorithms has many open problems of both theoretical and practical interest.

We conclude with some remarks and suggestions for new directions in the study of on-line algorithms. Finally, we summarize our results and identify some important new areas worth considering for future research.

7.1 A Critique of Competitive Analysis

Competitive analysis of on-line algorithms is defined as the worst-case ratio between its cost and that of a hypothetical optimal off-line algorithm. In other words, it is a theoretical framework used to determine the disadvantage of an on-line algorithm, which has incomplete information about the future (e.g., think of stock market investment). Thus, on-line algorithms often perform much worse than the off-line strategies in many situations inherently on-line in nature. On the other hand, it may seem unfair to allow the off-line algorithm to select (without cost) the best initial configuration, whereas the on-line algorithm is assumed to start with the worst one.

Since competitiveness is a worst-case analysis, it may fail to reflect the "typical" behavior of any algorithm. For example, *NP-completeness* is an analogous situation where a problem is *hard* in the worst case, but not necessarily in the typical case. A variant approach is to combine the competitive and average-case analyses by looking at on-line algorithms, which achieve small competitiveness and also perform efficiently against typical request sequences.

A criticism that competitive analysis measures how well an algorithm performs in the case of an adversarial future has the following two implications:

- It results in large theoretical lower bounds which are not practical, and
- If an algorithm has an optimal competitive ratio, this does not give any information about the running time of the algorithm when the future is pathological.

There may be alternative measures to competitive analysis that are relevant to the usefulness of an algorithm, although the measure used to evaluate algorithms influences the kind of the developed algorithms, paradoxically. Perhaps on-line problems are a means of exploring this issue.

7.1.2 New On-line Models

Ben-David and Borodin [350] have suggested a new measure of an on-line algorithm. They presented the following example to describe a shortcoming of competitive analysis. Let us consider the *problem of buying an insurance policy:* paying an annual premium of p to insure a car against theft is a *non-competitive strategy*! Let the cost of repairing or replacing a car be c. An algorithm has to decide every year whether to buy auto insurance or not. The (c/p)-competitive algorithm that never buys insurance is optimal. This is a contradiction to our intuition that insurance is not good if a claim is never presented to the insurance company.

This problem which is difficult to be solved by the traditional competitive analysis has an optimal solution using the *Max / Max measure* [350], which compares the worst case amortized behavior of an algorithm with that of the off-line one. On the other hand, minimizing *Max / Max* ration forces us to buy insurance evry year as long as $p < c$. Although this measure has the additional benefit that on-line algorithms can be directly compared, there are many interesting unsolved problems.

It is unlikely to have a case where there exists an algorithm that performs better than all the others on every input. Thus, the following interesting question remains open: *Is there a better and general way to compare on-line algorithms efficiently without comparing each one to the optimal off-line algorithm?*

7.3 Future Work

7.3.1 Lookahead

The *weak lookahead* is a theoretical on-line model against an oblivious adversary, while *strong lookahead* is a practical one as well, which improves the competitive ratio of some on-line algorithms (i.e., paging problem).

Paradoxically, *no* (*finite*[1]) lookahead is sufficient for any improvement of competitive performance for the decision-making tasks (e.g., k-server problem). The objective here is to provide a more realistic and reasonably general on-line framework that can suggest how to design efficient algorithms which are competitively better.

[1] The potential benefit of the *finite lookahead* with respect to the new *Max/Max ratio* becomes an important issue.

Generally, there may exist a "principle of optimality" for a whole class of on-line problems so that it can competitively determine the current state of the optimal off-line problem when provided with the k-subsequent future requests. It is also plausible that every on-line problem with lookahead (or even *finite* lookahead) may efficiently identify the optimal on-line strategy (in the form of a dynamic program) using some approach.

7.3.2 On-line Learning Versus Off-line Learning

An interesting application of on-line theory of algorithms should be on *learning theory* in the field of *Artificial Intelligence* (*AI*). Here, just as in the well studied *on-line model*, only the set of possible queries is known, while in the *off-line model* the sequence of queries is known to the learner in advance. We would like a student in *on-line model* to learn an unknown concept from a sequence of "guess and test" trials and to make as few mistakes as possible.

It would be interesting to give a combinatorial characterization of the number of instances in the off-line model and design a competitive (maybe a *permutation*) algorithm which bounds the number of mistakes of on-line learning versus off-line learning.

7.3.3 Central Open Problems

There are several fundamental topics in the theory of on-line algorithms and many challenging problems, that remain unsolved. The following general considerations are of the most interest to theoretical computer science, while some specific open questions have already been discussed in each chapter:

- Find new complexity models for *on-line* and *incremental computation*. Specifically, we are interested in practical on-line models to analyze typical request sequences even better. We also ask whether there exists a better performance measure than the competitive analytic approach without *risk or uncertainty* for the decision making on-line problems.

- Improve the lower or upper bounds on competitiveness of on-line algorithms and bridge all the (large) gaps left between the already proven ones. For example, find lower bounds on *loose competitiveness* [347] for *LRU*, *FIFO* and *MARK* on-line strategies. Also, improve or optimize the competitive ratios for weighted caching, k-server and other combinatorial on-line problems.
- Extend the theory of on-line algorithms to packing and covering geometric objects (i.e., on-line tiling).
- Consider new on-line algorithms and applications in parallel and distributed environment.
- It is of great interest whether *randomization* can help to improve the competitive performance of the algorithms for on-line problems, generally.

Finally, a very important direction for future research is to derive a general complexity theory for the tradeoff between running time and competitiveness.

7.4 Thesis Summary

In this thesis we studied the design and analysis of on-line algorithms for several *combinatorial optimization problems: paging, weighted caching*, the *k-server problem, graph coloring* and *weighted matching*.

We first presented some notations and results about the on-line computation as well as the on-line complexity bounds, including those for *NP-complete* problems, when the computational resources were restricted.

We then applied the method of competitive analysis to study the list update and paging problems considering simple related results under variant on-line models.

Next, we extended the theory of random walks and that of electrical networks to the k-server and its related problems for non-resistive spaces against an adaptive adversary.

We continued the study with the on-line coloring algorithms for particular graphs giving a slightly tighter competitive performance ratio for the coloring d-inductive graphs under the framework of the strong lookahead.

We examined the on-line algorithms for minimum or maximum weighted matching as well as for the on-line assignment problem using the dual bounding technique to simply reanalyze them.

Lastly, we applied the theory of on-line algorithms for specific distributed and geometric computational problems. Particularly, we presented an on-line navigation strategy in an unknown simple polygonal environment of streets, which achieves the best (nearly optimal) competitive ratio known in the literature.

In closing, we would like to believe that new theory and beautiful mathematics will grow up as the world of on-line algorithms matures. Furthermore, we hope that more research and cross-fertilization in the areas of *dynamic* and *on-line algorithms* will bridge the gap between practical and theoretical algorithms.

A pessimist is an optimist who tried to put the theory into practice.
Anonymous

This is not the end. It is not even the beginning of the end.
But it is, perhaps, the end of the beginning.
Winston Churchill

A Bibliography of On-line Algorithms

A bibliography proves the author's competence by showing the mountain of dross he has to win one nugget of truth.

L. J. Peter, hierarchiologist

This bibliography is an extensive coverage of references which are reflected in the title and have been in the literature by this time. It should be pointed out that most of these references are covered in our research. Additional references for particular topics (e.g., Dynamic data structures and algorithms) are known to exist and provide a wider coverage than the list offered by this bibliography.

[1] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *Proc. of The ACM-SIGMOD*, 1989.

[2] A. Aggarwal and A. K. Chandra. Virtual memory algorithms. *Proc. 20th Annual ACM Symp. on Theory of Computing*, pp. 173-185, 1988.

[3] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[4] S. Albers and H. Koga. New on-line algorithms for the page replication problem. *L. N. in Comp. Sc., Vol. 824. Algorithm Theory - SWAT' 94*, pp. 24-36, 1994.

[5] S. Albers. The influence of lookahead in competitive paging algorithms. *L. N. in Computer Science, Vol. 726. Algorithms - ESA' 93*, pp. 1-12, 1993.

[6] N. Alon. Generating pseudo-random permutation and maximum flow algorithms. *Information Processing Letters*, pp. 201-204, 1990.

[7] N. Alon and Y. Azar. On-line steiner trees in the Euclidean plane. In *Proc. 8th Annual ACM Symp. on Computational Geometry*, pp. 337-343, 1992.

[8] N. Alon, G. Kalai, M. Ricklin, and L. Stockmeyer. Lower bounds on the competitive ratio for mobile user tracking and distributed job scheduling. In *Theoretical Computer Science* 130, 175-201, 1994. Also in *Proc. 33rd IEEE FOCS*, pp. 334-343, 1992.

[9] N. Alon, R. M. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the k-server problem. (Extended abstract). *On-line Algorithms, DIMACS Series in Discrete Math. and Theor. Comp. Sc.*, 1-10, 1992.

[10] N. Alon and J. H. Spencer. *The Probability Method* . Wiley, New York, 1991.

[11] J. Aronson, M. Dyer, A. Frieze, and S. Suen. On the greedy heuristic for matching. In *Proc. of 5th ACM-SIAM on Symp. on Discrete Algorithms,* pp. 141-148, 1994.

[12] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in nonblocking network. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pp. 149-158, 1990.

[13] B. Alpern, R. Hoover, B.K. Rosen, P.F. Sweeney, and F.K. Zadeck. Incremental evaluation of computational circuits. In *Proc. of the 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 32-42, 1990.

[14] J.A. Aslam and A. Dhagat. On-line algorithms for 2-coloring hypergraphs via chip games. *Theor. Comput. Scie. 112*, pp. 355-369, 1993.

[15] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line machine Scheduling with applications to load balancing and virtual circuits routing. In *Proc. 25th ACM Symp. on Theory of Computing*, pp. 623-631, 1993.

[16] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Serving requests with on-line routing. *L. N. in Comp. Scie., Vol. 824, Algorithm Theory-SWAT' 94*, pp. 37-48, 1994.

[17] G. Ausiello and G.F. Italiano. On-line algorithms for polynomially solvable satisfiability problems. In *J. of logic Programming 10*, pp. 69-90, 1991.

[18] G. Ausiello, G.F. Italiano, A. Marchetti-Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. In *Proc. of the 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 12-21, 1990.

[19] A.V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[20] D. Avis. A survey of heuristics for the weighted matching problem. In *Networks 13*, pp. 475-493, 1983.

[21] F. Aurenhammer. Voronoi Diagrams: a survey of a fundamental geometric structure. *ACM Comp. Surveys 23*, pp. 345-405, 1991.

[22] *B. Awerbuch.* Complexity of network synchronization. *In J. ACM*, pp. 804-823, 1985.

[23] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *Proc. 25th Annual ACM Symp. on Theory of Computing*, pp. 321-327, 1993.

[24] B. Awerbuch, Y. Azar, and S. Plotkin. Throughout competitive on-line routing. In *Proc. 34th IEEE Annual Symp. on Foundation of Computer Science*, pp. 32-40, 1993.

[25] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. In *Proc. 25th ACM Symp. on Theory of Computing*, pp. 164-173, 1993.

[26] B. Awerbuch and D. Peleg. Concurrent on-line tracking of mobile users. In *Proc. SIGCOMM. Zurich, Sept. 1991.*

[27] B. Awerbuch and M. Sacks. A dining philosophers algorithm with polynomial response time. In *Proc. of 30th ACM Symp. on Foundations of Computer Science*, pp. 65-74, 1992.

[28] B. Awerbuch, Y. Bartal, and A. Fiat. Heat and Dump: Competitive distributed paging. In *Proc. 34th Symp. on the Foundation of Computing Science*, pp. 22-31, 1993.

[29] B. Awerbuch and D. Peleg. Concurrent on-line tracking of mobile users. In *Proc. SIGCOMM. Zurich*, Sept. 1991.

[30] Y. Azar, A. Broder, and A.R. Karlin. On-line load balancing. In P*roc. 33rd Symp. of Foundations of Computer Science*, pp. 218-225, 1992. Also in *Theoretical Computer Science 130*, pp. 73-84, 1994.

[31] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. In *Proc. Workshop on Algorithms and Data Structures*, pp. 119-130, 1993.

[32] Y. Azar, A.Z. Broder, and M.S. Manasse. On Choice of On-line Algorithms. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pp. 432-440, 1992.

[33] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pp. 203-210, 1992.

[34] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching in a plane. *Information and Computation, Vol. 106*, pp. 234-252, 1993.

[35] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching with uncertainty. *Technical Report, University of Waterloo*, October 1987.

[36] E. Bar-Eli, P. Berman, A. Fiat, and Peiyuan Yan. On-line navigation in a room. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pp. 237-249, 1992. Also in *J. of Algorithms 17*, pp. 319-341, 1994.

[37] A. Bar-Noy, F.K. Hwang, I. Kessler, and S. Kutten. A new competitive algorithms for group testing. *Discrete Applied Mathematics, Vol. 52*, pp. 29-38, 1994.

[38] A. Bar-Noy and B. Schieber. The Canadian travelers problem. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, pp. 261-270, 1991.

[39] Y. Bartal, H. Karloff, and Y. Rabani. A Better lower bound for on-line scheduling. In *Information Processing Letters 50*, pp. 113-116, 1994.

[40] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proc. 24th ACM Symp. on Theory of Computing*, pp. 39-50, 1992.

[41] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th ACM Symp. on Theory of Computing, pp. 51-58*, 1992.

[42] Y. Bartal and A. Rosen. The distributed k-server problem: a competitive distributed translator for k-server algorithms. In *Proc. 33rd Annual Symp. on Foundation of Computer Science*, pp. 344-353, 1992.

[43] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitive of on-line task real-time task scheduling. In *J. Real-Time Systems 4*, pp. 124-144, 1992.

[44] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha. On-line scheduling in the presence of overload. In *Proc. IEEE Foundations of Computer Science Conf.*, pp. 101-110, 1991.

[45] L.A. Belady. A Study of replacement for virtual storage computers. IBM Systems Journal 5, pp. 78-101, 1966.

[46] M. Bellmore and S. Hong. Transformation of the multisalesmen problem to the standard traveling salesmen problem. In *J. Assoc. Comput. Mach. 21*, pp. 500-504, 1974.

[47] S. Ben-David, A. Borodin, R. M. Karp, G. Tárdos, and A. Wigderson. On the power of randomization in on-line algorithms. In *Proc. ACM Symp. on Theory of Computing*, pp. 379-388, 1990. Also in *Algorithmica 11*, pp. 2-14, 1994.

[48] J.L. Bentley, K.L. Clarkson, and D.B. Levine. Fast Linear expected-time algorithms for computing maxima and convex hulls. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pp. 179-187,1990.

[49] J.L. Bentley, D.D. Sleator, R.E. Tarjan, and V. Wei. A Locally adaptive data compression scheme. *Comm. ACM 29*, pp. 320-330, 1986.

[50] J.L. Bentley and C. McGeogh. Worst-case of self-organizing sequential search heuristics. In *Proc. 20th Allerton Conference on Communication, Control, and Computing*, pp. 452-461, 1983.

[51] J.L. Bentley and C.C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. In *Communications of the ACM, Vol. 28*, pp. 404-411, 1985.

[52] A.M. Berman, M.C. Paull, and B.G. Ryder. Proving relative lower bounds for incremental algorithms. In *Acta Informatica 27*, pp. 665-683, 1990.

[53] M. Burke and B.G. Ryder. A critical analysis of incremental iterative data flow analysis algorithms. In *IEEE Trans. on Software Engineering 16*, 1990.

[54] M. Bern, D. H. Greene, A. Raghunathan, and M. Sudan. On-line algorithms for locating checkpoints. *Algorithmica 11*, pp. 33-52, 1994.

[55] E. Bernstein and S. Rajagapalan. The roommate problem: On-line matching and graphs. *UCB // CSD-93-757, Comput. Sc. Division, University California, Berkeley, CA*, 1993.

[56] P. Berman, H. Karloff, and G. Tárdos. A competitive 3-server algorithm. In *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 280-290, 1990.

[57] J. R. Bitner. Heuristics that dynamically organize data-structures for representing sorted lists. *SIAM Journal on Computing 8*, pp. 82-110, 1979.

[58] D.L. Black and D.D. Sleator. Competitive algorithms for replication and migration problems. *Technical Report CMU-CS-89-201, Carnegie Mellon University*, 1988.

[59] M. Blum and D. Kozin. On the power of the Compass. In *Proc. 19th IEEE Symp. on Foundations of Computer Science*, pp. 132-142, 1978.

[60] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. In *Proc. 23rd ACM Symp. Theory of Computing*, pp. 494-504, 1991.

[61] J.D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling on on-line algorithms in computational geometry. *Discrete Comput. Geom. 8*, pp. 51-71, 1992.

[62] J.D. Boissonnat, O. Devillers, and M. Teillaud. A semi-dynamic construction of higher order Voronoi diagrams and its randomized analysis. In *2nd Canadian Confer. on Computational Geometry*, pp. 278-281, Ottawa, 1990.

[63] J.D. Boissonnat and M. Teillaud. A hierarchical representation of objects: the Delaunay tree. In *2nd ACM Symp. on Computational Geometry*, pp. 260-268, 1986.

[64] B. Bollobás. *Extremal Graph Theory*. Academic Press, New York, 1978.

[65] A. Borodin, S. Irani, P. Raghavan, and B. Scieber. Competitive paging with locality of reference. In *Proc. 23rd ACM Symp. on Theory of Computing*, pp. 249-259, 1991.

[66] A. Borodin, N. Linial, and M. Saks. An optimal on-line algorithm for metrical task systems. In *Proc. 19th ACM Symp. on Theory of Computing*, pp. 373-382, 1987. Also in *J. of ACM 39*, pp. 745-763, 1992.

[67] D. Breslauer. An on-line string superprimitive test. *Technical Report CUCS-022-92, Columbia University*, 1992.

[68] A. Calderbank, E. Coffman, and L. Flatto. Sequencing problems in two-server systems. In *Math. Operations Research 10*, pp. 585-598, 1985.

[69] C.G. Chaitin. Register allocation and spilling via graph coloring. In *Proc. of Sigplan Symp. on Computer Construction, Sigplan, Note 17*, pp. 98-105, June 1982.

[70] K. -F Chan and T.W. Lam. An on-line algorithm for navigating in an unknown environment. In *Inter. J. of Computational Geometry and Algorithms, Vol. 3, No. 3*, pp. 227-244, 1993.

[71] A.K. Chandra, P. Raghavan, W.L. Ruzzo, R. Smolensky, and P. Tiwari. The electrical resistance of a gap captures its commute and cover times. In *Proc. of Computing*, pp. 574-586, 1989.

[72] K. Chandy and J. Misra. The drinking philosophers problem. *ACM TOPLAS 6*, pp. 632-646, 1984.

[73] E.-C. Chang, W. Wang, and M.S. Kankanhalli. Multidimensional on-line bin-packing : An algorithm and its average-case analysis. In *Inform. Processing Letters 48*, pp. 121-125, 1993.

[74] B. Chazelle. Efficient polygon triangulation. In *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 220-230, 1990.

[75] D.Z. Chen. On the all pairs Euclidean short path problem. Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, U.S.A.. Manuscript, 1994.

[76] J. Cheriyan and T. Hagerup. A randomized maximum flow algorithm. In *Proc. IEEE Symp. on Foundation of Computer Science*, pp. 118-123, 1989.

[77] J. Cheriyan T. Hagerup, and K. Mehlhorn. Can a maximum flow be computed in o(mn) time? *In Int. Colloq. on Automata, Languages, and Programming (ICALP)*, pp. 235-248, 1990.

[78] M. Chernoff. A measure of asymptotic efficiency for tests based on the sum of observations. *Annual of Mathematical Statistics 23*, pp. 493-509, 1952.

[79] G. A. Cheston. On-line connectivity algorithms. *Networks 14*, pp. 83-94, 1984.

[80] G.A. Cheston and D.G. Corneil. Graph property update algorithms and their application to distance matrices. *Inform. 20*, pp. 178-201, 1982.

[81] W. Chin an S. Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry 6*, pp. 9-31, 1991.

[82] M. Chrobak and L.L. Larmore. HARMONIC is 3-competitive for 2-servers. *Theoret. Comput. Sci. 98*, pp. 339-346, 1994.

[83] M. Chrobak and L.L. Larmore. Generosity helps on an 11-competitive algorithm for tree server. *J. of Algorithms 16*, pp. 234-263, 1994.

[84] M. Chrobak and L.L. Larmore. On fast algorithms for two servers. In *J. of Algorithms 12*, pp. 607-614, 1991.

[85] M. Chrobak, L. Larmore, N. Reingold, and J. Westbrook. Page migration algorithms using work functions. In *L. N. in Comp. Scie., Vol. 650. Algorithms and Computation. ISAAC' 92*, pp. 406-416, 1992.

[86] M. Chrobak and L.L Larmore. An optimal on-line algorithm for the k-servers on trees. In *SIAM J. on Computing 20*, pp. 144-148, 1991.

[87] M Chrobak, H.J. Karloff, and T. Payane. New results on server problems. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pp. 291-300, 1990.

[88] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *SIAM J. on Discrete Mathematics, Vol. 4*, pp. 172-181, 1991.

[89] M. Chrobak and L.L. Larmore. The server problem and on-line games. In *DIMACS, Vol. 7*, pp. 111-131, 1992.

[90] F.K. Chung, R. Graham, and M.E. Saks. A dynamic location problem for graphs. *Combinatorica 9*, pp. 11-131, 1989.

[91] K.L. Clarkson and P.W. Shor. Applications of random sampling in computational Geometry: II. *Discrete Comput. Geometry 4*, pp. 387-421, 1989.

[92] E.G. F.Jr. Coffman and P.J. Denning. *Operating Systems Theory*. Prentice-Hall, Englewood Cliffs, 1973.

[93] R.F. Cohen and R. Tamassia. Combine and conquer : A General technique for dynamic algorithms. *L. N. in Comput. Scie., Vol. 726, Algorithms-ESA' 93*, pp. 109-118, 1993.

[94] R. Cole and A. Raghunathan. On-line algorithms for finger searching. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pp. 480-489, 1991.

[95] A. Condon. Computational models of games. *An ACM Distinguished Dissertation*,1989.

[96] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Addison-Wesley, Reading, 1967.

[97] S.A. Cook. The complexity of theorem proving procedures. In 3rd *Symp. on Theory of Computing 5*. pp. 151-158, 1971.

[98] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs, and applications to on-line algorithms. In *Proc. 22nd ACM Symp. on Theory of Computing*, pp. 369-378, 1990.

[99] D. Coppersmith, P. Doyle, P. Rghavan, and M. Snir. Random walks on weighted graphs, and applications to on-line algorithms. In *J. of ACM, Vol. 40*, pp. 431-453, 1993.

[100] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. McGraw-Hill, New York, 1990.

[101] J. Csirik and D.S. Johnson. Bounded space on-line bin-packing: Best is better than first. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, pp. 309, 1991.

[102] C. Culberson and R.A. Reckow. Covering polygons is hard. *J. Algorithms 17*, pp. 2-44, 1994.

[103] F. d'Amore and V. Liberatore. The list update problem and the retrieval of sets. In *Theoretical Computer Science 130*, pp. 101-123, 1994.

[104] F. d'Amore and A. Marchetti. The weighted list update problem and the lazy adversary. *Theoret. Comp. Science 108*, pp. 371-384, 1993.

[105] F. d'Amore and A. Marchetti-Spaccamela, and U. Nanni. Competitive algorithms for the weighted list update problem. In *L. N. in Comput. Scie.,Vol. 519. Algorithms and Data Structures*, pp. 240-248, 1991.

[106] A. Datta and C. Icking. Competitive searching in a generalized street. *10th ACM Computational Geometry*, pp. 175-182, 1994.

[107] H.M. Deitel. *An Introduction to Operating Systems*. Addison-Wesley, Reading, 1990.

[108] X. Deng and C.H. Papadimitriou. Competitive distributed decision-making algorithms, software architecture. By J. Van Leeuwen (Ed.). Inform. Processing 92, Vol. I, pp. 350-356, 1992.

[109] X. Deng, T. Kameda, and C.M. Papadimitriou. *How to learn an unknown environment I: The rectilinear case.* Technical Report CS-93-04, Department of Computer Science, York University, 1993.

[110] X. Deng, T. Kameda, and C.M. Papadimitriou. How to learn an unknown environment. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pp. 298-303, 1991.

[111] X. Deng and S. Mahajan. Infinite games, randomization, compatibility and applications to on-line problems. In *Proc. of the 23rd Symp. on the Theory of Computing*, pp. 289-298, 1991.

[112] X. Deng and C.H. Papadimitriou. Exploring an unknown graph. In *Proc. 31st Annual Symp. on Foundations of Computer Science*, pp. 355-361, 1990.

[113] X. Deng and E. Koutsoupias. Competitive implementation of parallel Programs. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 455-461, 1994.

[114] P.J. Denning. Working sets past and present. *IEEE Trans. on Software Engineering, SE-6*: 64-84, 1980.

[115] C. Derman. *Finite State Markov Decision Processes.* Academic Press, New York, 1970.

[116] M. Dertouzos and A. Mok. Multiprocessor scheduling in a hard real-time constraints. In *Proc. 7th IEEE Conf. on Computing Systems*, 1978.

[117] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. In *L. N. in Comput. Scie., Vol. 519, WADS' 91*, 1991.

[118] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proc. of the 30th IEEE Symp. on the Foundations of Computer Science*, pp. 436-441, 1989.

[119] G. Di Battista and R. Tamassia. On-line graph algorithms with spqr-trees. In *Proc. of the 17th Intern. Colloquium on Automata, Languages and Programming*, 1990.

[120] P.F. Dietz and R. Raman. Persistence, amortization and randomization. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, pp. 78-88, 1991. Also *Technical Report 353 (revised). University of Rochester, Dept. of Computer Science*, 1991.

[121] P.G. Doyle and J.L. Snell. Random walks and electric networks. *The Mathematical Association of America, Vol. 22*, 1984.

[122] D.-Z. Du and H. Park. On competitive testing. *SIAM J. Computing 23*, pp. 1019-1025, 1994.

[123] D.-Z. Du, G.-L. Xue, S.-Z. Sun and S.-W. Cheng. Modifications of competitive group testing. *SIAM J. Computing 23*, pp. 82-96, 1994.

[124] J. Du, J.Y.-T. Leung, and G.H. Young. Minimizing mean flow time with release time constraints. Technical Report, Computer Science Program, University of Texas at Dallas, 1988.

[125] H. Edelsbrunner. *Algorithms on Combinatorial Geometry*. Springer, Berlin, 1987.

[126] J. Edmonds and E.L. Johnson. Matching, Euler tours and the Chinese postman. In *Math. Programming 5*, pp. 88-124, 1973.

[127] R. El-Yaniv, A. Fiat, R. Karp, and G. Turpin. Competitive analysis of financial games. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pp. 327-333, 1992.

[128] D. Eppstein, Z. Galil, G.F. Italiano, and S. Nissenzweig. Sparsification: a technique for speeding up dynamic graph algorithms. *Proc. 33rd Ann. Symp. on Foundations of Computer science*, pp. 60-69, 1992.

[129] D. Eppstein, Z. Galil, G.F. Italiano, and T. Spencer. Separator based sparsification for dynamic planar graph algorithms. *Proc. 25th Ann. Symp. on Theory of Computing*, pp. 208-217, 1993.

[130] D. Eppstein, G. Italiano, R. Tamassia, R.E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *Proc. 1st SODA, 1990.*

[131] V. Estivil-Castro and M. Sherk. Competitiveness and response time in on-line algorithms. In *Proc. of the Inter. Symp. on Algorithms*. pp. 284-293, 1991.

[132] S. Even and Y. Shiloach. An on-line edge deletion problem. In *J. of the ACM 28*, pp. 1-4, 1981.

[133] U. Faigle, W. Kern, and G. Turan. On the performance of on-line algorithms for partitioning problems. In *Acta Cybernetica 9*, pp. 107-119, 1989.

[134] A. Feldmann, J. Sgall, and S.-H. Teng. Dynamic scheduling on parallel machines with dependencies. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*. pp. 642-651, 1993.

[135] A. Feldmann, J. Sgall, and S-H. Teng. Dynamic scheduling on parallel machines. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pp. 111-120, 1991. Also in *Theoret. Comput. Scie. 130*, pp. 49-72, 1994.

[136] D. Ferrari. The improvement of program behavior. In *IEEE Computer, Vol. 9*, pp. 39-47, 1976.

[137] E. Feuerstein and A. Marchetti-Spaccamela. Memory paging for connectivity and path and path problems in graphs. In *L. N. in Comp. Scie., Vol. 650. Algorithms and Computation. ISAAC' 92*, pp. 416-425, 1992.

[138] A. Fiat and M. Rocklin. Competitive algorithms for the weighted server problem. In *Theoret. Comput. Scie. 130*, pp. 85-99, 1994.

[139] A. Fiat, Y. Rabane, and B. Schieber. A deterministic $O(k^3)$-competitive k-server algorithm for the circle. In *Algorithmica 11*, pp. 572-578, 1994.

[140] A. Fiat, D.P. Foster, H. Karloff, Y. Rabani, Y. Ravid, and S. Vishwanathan. Competitive algorithms for layered graph traversal. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pp. 288-297, 1991.

[141] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. In *J. Algorithms 12*, pp. 685-699, 1991.

[142] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pp. 454-463, 1990.

[143] M. Fiedler, C.R. Johnson, T.L. Markham, and M. Neumann. A trace inequality for M-matrices and the symmetrizability for a real matrix by a positive diagonal matrix. *Linear Algebra and its Appl. 71*, pp. 81-94, 1985.

[144] P.A. Franaszek and T.J. Wagner. Some distribution-free aspects of paging performance. In *J. of the ACM, Vol. 21*, pp. 31-39, 1974.

[145] G. Frederickson and D. Guan. Preemptive ensemble motion planning on a tree. In *SIAM J. on Computing 7*, pp. 1130-1152, 1992.

[146] G. Frederickson. Ambivalent data structures for dynamic 2-edge connectivity and k smallest spanning trees. In *Proc. 32nd FOCS, 1991.*

[147] G. Frederickson. Data structures for on-line updating of minimum spanning trees. *SIAM J. Computing 14*, pp. 781-798, 1985.

[148] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. In *SIAM J. Computing Vol. 7*, pp. 178-193, 1978.

[149] G.N. Frederickson and M.A. Srinivas. On-line updating for degree-constrained minimum spanning trees. In *Proc. of the 22nd Allerton Conference on Communication, Control, and Computing*, 1984.

[150] D. Gale and F.M. Stewart. Infinite games with perfect information. In W.H. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games, Vol. II. Annuals of Mathematics Studies, Vol. 28*, pp. 245-266. Princeton University Press, 1953.

[151] Z. Galil and G. Italiano. Fully dynamic algorithms for edge connectivity problems. In *Proc. 23rd STOC*, 1991.

[152] Z. Galil and G.F. Italiano. A note on set union with arbitrary deunion. In *Inform. Process. Letters 36*, pp. 331-335, 1991.

[153] G. Gambosi, A. Postiglione, and M. Talamo. New algorithms for on-line bin-packing. In *Proc. 1st Italian conf. on Algorithms*, pp. 44-59, 1990.

[154] J. Garay, I. Gopal, K. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proc. of 2nd Annual Israel Conference on Theory of Computing and Systems*, 1993.

[155] J. Garay and I.S. Gopal. Call preemption in communication networks. In *Proc. INFOCOM' 92, Vol. 44*, pp. 1043-1050, 1992.

[1 5] M. Garey and D. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.

[157] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal 45*, pp. 1563-1581, 1986.

[158] R.L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal of Applied Mathematics, Vol. 17*, pp. 263-269, 1969.

[159] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of Discrete Mathematics 5*, pp. 287-326, 1979.

[160] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. In *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 317-324, 1992.

[161] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM, Vol. 35*, pp. 921-940, 1988.

[162] M. Goldberg and T. Spencer. Constructing a maximal independent set in parallel. *SIAM J. Comput. 2*, pp. 322-328, 1989.

[163] E. Grove. The harmonic on-line k-server algorithm is competitive. In *Proc. 23rd ACM Symp. on Theory of Computing*, pp. 260-266, 1991.

[164] A. Gayarfa^ nd J. Lehel. On-line and first fit colorings of graphs. In *J. of Graph Theory, Vol. 12*, pp. 217-227, 1988.

[165] A. Gayarfas and J. Lehel. First-fit and on-line chromatic number of families of graphs. In A*rs Combinatoria 29C*, pp. 168-176, 1990.

[166] A. Gayarfas and J. Lehel. Effective on-line coloring of P_5-free graphs. In *Combinatoria 11*, pp. 181-184, 1991.

[167] L. J. Guidas, D.E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *ACM Trans. Graphics 4*, pp. 381-413, 1992.

[168] L.J. Guidas and J. Hershberger. Optimal shortest path queries in a simple polygon. In *Proc. 3rd ACM Symp. on Computational Geometry, Waterloo*, pp. 50-63, 1987.

[169] M.M. Halldórsson and M. Szegedy. Lower bounds for on-line graph coloring. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pp. 211-216, 1992. Also in *Theoret. Comput. Scie. 130*, pp. 163-174, 1994.

[170] M.M. Halldórsson. Parallel and on-line graph coloring algorithms. In *3rd Intern. Symp., ISSAAC' 92*, pp. 61-70, 1992.

[171] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[172] D. Hayfield, J. Gerald. Program restructuring for virtual memory. In *IBM J. Systems. and Tech., Vol. 10*, pp. 168-192, 1971.

[173] J.H. Hester and D.S. Hirschberg. Self-organizing linear search. In *ACM Comput. Surveys 17*, pp. 295-311, 1985.

[174] K.S. Hong and J.Y.-T. Leung. On-line scheduling of real-time tasks. In *IEEE Trans. Comput. 41*, pp. 1321-1331, 1992.

[175] E. Horowitz and S. Sahni. *Fundamental of Data Structures*. Computer Science Press, 1983.

[176] X.D. Hu, P.D. Chen and F.K. Hwang. A new competitive algorithm for the counterfeit coin problem. In *Inform. Processing Letters 51*, pp. 213-218, 1994.

[177] S. Huddleston, K. Melhorn. Robust balancing B-trees. In *Proc. 5th GI-Conference on Theoretical Computer Science, Lecture Notes in Computer Science 104, Springer-Verlag, New York*, pp. 234-244, 1981.

[178] S. Huddleston and K. Melhorn. A new data structure for representing sorted lists. In *Acta Inform. 17*, pp. 157-184, 1982.

[179] L.C.K. Hui and C.U. Martel. Analyzing deletions in competitive algorithms self-adjusting linear list algorithms. *L. N. in Comp. Sc., Vol. 834. Algorithms and Computation-ISAAC' 94*, pp. 433-441,1994.

[180] L.C.K. Hui and C.U. Martel. Randomized competitive algorithms for successful and unsuccessful search on self-adjusting lists. In *L. N. in Comp. Sc., Vol. 650. ISAAC' 93. Algorithms and Computation*, pp. 426-435, 1992.

[181] L. Hui and C. Martel. On efficient unsuccessful search. In *Proc. of the 3rd ACM-SIAM Symp. on Discrete Algorithms*, pp. 217-227,1992.

[182] T. Ibaraki and N. Kutoh. On-line computation of transitive closure of graphs. In *Information Processing Letters 16*, pp. 95-97, 1983.

[183] C. Icking, R. Klein and L. Ma. The optimal way for looking around a corner. In *IEEE-IEE Vehicle Navigation and Inform. Systems Conference*, pp. 547-550, 1993.

[184] Ch. Icking and R. Klein. The two guards problem. In *Proc. 7th ACM Symp. on Computational Geometry, North Conway,* 1991. Also in *Intern. J. of Computational Geometry and Applications 2,* pp. 257-285, 1992.

[185] G.F. Italiano. Amortized effici^ncy of a path retrieval data structure. In *Theoretical Computer Science* , pp. 273-281, 1986.

[186] G.M. Italiano. Finding paths and deleting edges in directed acyclic graphs. *Information Processing 28,* pp. 5-11, 1988.

[187] G.F. Italiano. *Dynamic Data Structure for Graphs.* Ph. D. thesis, Columbia University, 1991. Technical Report CUCS-019-91.

[188] S. Irani. Coloring inductive graphs on-line. In *Proc. 31st IEEE Symp. on Foundations of Computer Science,* pp. 470-479, 1990. Also in *Algorithmica 11,* pp. 53-72, 1994.

[189] S. Irani. Two results on the list update problem. In *TR-90-037, Computer Science Division, UCB, Berkeley, CA,* 1990. Also to appear in *Information Processing Letters.*

[190] S.S. Irani, A.R. Karlin, and S.J. Phillips. Strongly competitive paging with locality of reference. In *3rd ACM-SIAM Symp. on Discrete Algorithms,* pp. 228-236, 1992.

[191] S. Irani and R.. Rubinfeld. A competitive 2-server algorithm. In *Inform. Processing Letters 39,* pp. 85-91, 1991.

[192] J. Januszewski and M. Lassak. On-line covering the unit cube by cubes. In *Discrete Comput. Geometry 12,* pp. 433-438, 1994.

[193] D.S. Johnson an C.H. Papadimitriou. Computational complexity. In *Traveling Salesman Problem, edited by E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys,* pp. 37-85, 1985.

[194] S. Kahan. A model for data in motion. In *Proc. 23rd Annual ACM Symp. on Theory of Computing,* pp. 267-277, 1991.

[195] B. Kalyanasundaram and K.K. Pruhs. Constructing on-line tours from local information. In *Theoret. Comput. Scie. 130,* pp. 125-138, 1994.

[196] B. Kalyanasundaram and K.K. Pruhs. On-line weighted matching. In *J. of Algorithms 14,* pp. 478-488, 1993.

[197] B. Kalyanasundaram and K.K. Pruhs. A competitive analysis of nearest neighbor algorithms for searching unknown scenes. In *Proc. 9th Ann. Symp. on Theoretical Aspects of Computer Science*, pp. 147-157, 1992.

[198] B. Kalyanasundraram and K.K. Pruhs. On-line weighted matching. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, pp. 234-240, 1991.

[199] B. Kalyanasundaram and K. Pruhs. Visual searching and mapping. *On-line Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 157-162, 1992.

[200] M.-Y. Kao, J.H. Reif, and S.R. Tate. Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. In *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, pp. 441-447, 1993.

[201] M.-Y. Kao and S.R. Tate. A on-line matching with blocked input. In *Inform. Processing Letters 38*, pp. 113-116, 1991.

[202] D. Karger, S.J. Phillips, and E. Tong. A better algorithm for an ancient scheduling problem. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 132-140, 1994.

[203] A.R. Karlin, M.S. Manasse, L.A. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform Problems. In *Algorithmica 11*, pp. 542-571, 1994.

[204] A.R. Karlin, M.S. Manasse, Rudolph L., and D.D. Sleator. Competitive snoopy caching. In *Algorithmica 3*, pp. 79-119, 1988.

[205] A.R. Karlin, S.j. Phillips, and P. Raghavan. Markov paging. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pp. 208-217, 1992.

[206] H. Karloff, Y. Rabani, and Y. Ravid. Lower Bounds for randomized k-server and motion planning algorithms. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pp. 352-358, 1990. Also in *SIAM J. Computing 23*, pp. 293-312, 1994.

[207] R.M. Karp. On-line algorithms versus off-algorithms: How much is it worth to know the future? In: *J. Van Leeuwen, Ed., Information Processing 92. Prcoc. IFIP 12th World Computer Congress*, 1992.

[208] R.M. Karp. A characterization of the minimum cycle mean in a digraph. In *Discrete Mathematics, Vol. 23*, pp. 309-311, 1978.

[209] R.M. Karp. Reducibility among combinatorial problems. R.E Miller and J. W. Thatcher (ed.), *Complexity of Computer Computations* Plenum Press, New York, pp. 85-103, 1972.

[210] R. Karp and V. Ramachandran. A survey of parallel algorithms for shared memory machines. In *Handbook of Theoretical Computer Science, Vol. A. Alsevier*, 1990.

[211] R.M. Karp, U.V. Vazirani, and V.V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd ACM Symp. on Theory of Computing*, pp. 352-358, 1990.

[212] Y. Karuno, H. Nagamochi and T. Ibaraki. Vehicle scheduling on a tree with release times and handling times. In *Proc. 4th Intern. Symp. on Algorithms and Computation ISAAC, 93, L.N.C.S., Vol. 762*, pp. 486-495, 1993.

[213] S. Khuller, S. Mitchell, V. Vazirani. On-line algorithms for weighed matching and stable marriages. *Technical Report TR 90-1143, Department of Computer Science, Cornell University*, 1990. Also In *Theoretical Computer Science 127*, pp. 255-267, 1994.

[214] H. A. Kierstead. The linearity if first-fit coloring of interval graphs. In *SIAM J. of Discrete Mathematics, Vol. 1*, pp. 526-530, 1988.

[215] H.A. Kierstead, S.G. Penrice, and W.T. Trotter. On-line coloring and recursive graph theory. In *SIAM J. Discrete Mathematics Vol. 7*, pp. 72-89, 1994.

[216] H.A. Kierstead and W.A. Trotter. Lower bounds for on-line graph coloring. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 7*, pp. 85-92, 1992.

[217] H.A. Kierstead and W.A. Trotter. An extremal problem in recursive combinatorics. In *Congressus Numerantium 33*, pp. 143-153, 1981.

[218] T. Kilburn, D.B.G. Edwards, M.J. Lanigan, and F.H. Summer. One-level storage system. *IRE Trans. Elect. Computers 37*, pp. 223-235,1992.

[219] V. King, S. Rao, and R.E. Tarjan. A faster deterministic maximum flow algorithm. algorithm. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pp. 157-164, 1992. Also in *J. of Algorithms 17*, pp. 447-474,1994.

[220] R. Klein. Walking an unknown street bounded detour. In *Proc. 32nd Ann. on Foundations of Computer Science*, pp. 304-313, 1991. Also in *Computational Geometry: Theory and Applications 1*, pp. 325-351, 1992.

[221] J.M. Kleinberg. A lower bound for two-server balancing algorithms. In Inform. *Processing Letters 52*, pp. 39-43, 1994.

[222] J.M. Kleinberg. On-line search in a simple polygon. In *Proc. of the 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 8-15, 1994.

[223] L. Kleinrock. *Queuing Systems, Vol. I (Theory) and Vol. II (Computer Applications).* John Wiley & Sons, New York, 1990.

[224] D.J. Kleitman and D.B. West. Spanning trees with many leaves. In *SIAM J. of Discrete Mathematics, Vol. 4*, pp. 99-106, 1991.

[225] D.E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[226] H. Koga. Randomized on-line algorithms for the page replication problem. In *L. N. in Comp. Scie., Vol. 762, ISAAC' 93. Algorithms and Computation*, pp. 436-445, 1993.

[227] G. Koren and D. Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. In *Theoret. Comput. Scie. 128*, pp. 75-97, 1994.

[228] G. Koren and D. Shasha. D-over: An optimal on-line scheduling algorithm for over-loaded real-life systems. In *Proc. IEEE Real-Time Systems Symp.* pp. 290-292, 1992.

[229] W. Kuperberg. On-line covering a cube by a sequence of cubes. In *Discrete Comput. Geometry 12*, pp. 83-90, 1994.

[230] K.N. Kutulakos, C.R. Dyer, and V.J. Lumelsky. Vision-guided exploration: A step toward general motion planning in three dimension. Technical Report #1111, Dept. of Computer Science, University of Wisconsin, 1992.

[231] M.-K. Kwan (Guan). *Discrete Applied Math. 9*, pp. 41-46, 1984.

[232] T.W. Lai. Self-adjusting augmented search trees. In *L. N. in Comp. Scie., Vol. 650, Algorithms and Computation*, pp. 88-96, 1992.

[233] T.W. Lai and D. Wood. A new approach to competitive list-update algorithms. Technical Report #341 / 1993, Dept. of Computer Science, University of Western, Ontario.

[234] T.W. Lai and D. Wood. Adaptive heuristics for binary search trees and constant linkage cost. In *Proc. of the 2nd Annual ACM-SIAM Symp. on discrete Algorithms*, pp. 72-77, 1991.

[235] L.L. Larmore. On-line dynamic programming with applications to the predictions of RNA secondary structures. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pp. 503-512, 1990.

[236] M. Lassak and J. Zhang. An on-line potato-sack. In *Theoret. Discrete Comput. Geometry 6*, pp. 1-7,1991.

[237] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids.* New York, Holt, Rinehart & Winston, 1976.

[238] E.L. Lawler, J.K. Lenstra, A .H.G. Rinnooy Kan, and D.B. Shmoys. *Sequencing and Scheduling: Algorithms and Complexity.* 1990.

[239] E.D. Lazowska,J. Zahorjan, G.C. Graham, and K.C. Sevcik. *Quantitative System Performance.* Prentice-Hall, Englewood Cliffs, 1984.

[240] F.T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tárdos and S. Tragoudas. Fast approximation algorithms for multicommodity flow problem. In *Proc. 23rd ACM Symp. on the Theory of Computing*, pp. 101-111, 1991.

[241] F.T. Leighton. *Complexity Issues in VLSI.* MIT Press, Cambridge, MA, 1983.

[242] A. Lempel, S. Even and I. Cederbaum. An algorithm for planarity testing of graphs. In *Proc. Int. Symp. on Theory of Graphs; P. Rosenstiehl Ed.*, pp. 215-232. Gordon and Breach, 1967.

[243] J.K. Lenstra, D.B. Shmoys, and E. Tàrdos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Progr. 46*, pp. 259-271, 1990.

[244] H. R. Lewis and L. Denenber. *Data Structures and Their Algorithms.* Harper Collins, 1991.

[245] P.A.W. Lewis and G.S. Shedler. Empirically derived models for sequences of page exceptions. *IBM Journal of Research and Development, Vol. 17*, pp. 86-100, 1973.

[246] R.J. Lipton and A. Tomkins. On-line interval scheduling. In *Proc. of 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 302-311, 1994.

[247] J.M. Lucas. On the competitiveness of splay trees: Relations to the union-find problem. *On-line Algorithms, DIMACS Series in Discrete Math. and Theor. Comp. Science*, pp. 95-124, 1992.

[248] L. Lovász, M. Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics, Vol. 75*, pp. 319-325, 1989.

[249] M.G. Luby, J. Naor, and A. Orda. Tight bounds for dynamic Storage allocation. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 724-738, 1994.

[250] V.J. Lumelsky, S. Mukhopadhyay, and K.Sun. Dynamic path planning in sensor-based terrain acquisition. In *IEEE Trans. Robotics Automation 6*, pp. 462-472, 1990.

[251] V.J. Lumelsky and A.A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. In *Algorithmica 2*, pp. 403-430, 1987.

[252] N. Lynch. Upper bounds for static resource allocation in a distributed system. In *Journal of Computer and System Sciences, Vol. 23*, 1981.

[253] A. Marchetti-Spaccamela, U. Nanni, and H. Rohnert. On-line graph incremental computation. In *L. N. Comput. Scie., Vol. 790.* In *Proc. of the 19th Intern. Workshop, WG' 93*, pp. 70-80, S-V, 1993.

[254] M.S. Manasse, McGeoch L.A., and D.D. Sleator. Competitive algorithms for server problems. *In J. Algorithms 11, pp. 208-230*, 1990.

[255] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th Annual ACM Symp. on Theory of Computing*, pp. 322-333, 1988.

[256] J. McCabe. On serial files with relocatable records. In *Oper. Res. 12*, pp. 609-618, 1965.

[257] K.M. McDonald and J.G. Peters. Smallest paths in simple rectilinear polygons. In *IEEE Trans. on CAD/ICAS, Vol. 11*, pp. 864-875.

[258] K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. EATCS, Springer-Verlag, 1984.

[259] T. Matsumoto. Competitive analysis of the round robin algorithm. In *L. N. in Comp. Sc., Vol. 650. ISAAC' 92. Algorithms and Computation*, pp. 71-77, 1992.

[260] L.A. McGeoch and D.D. Sleator. On-line algorithms. *DIMACS, Vol. 7, AMS, Providence, RI*, 1992.

[261] L.A. McGeoch and D.D. Sleator. A strongly competitive randomized paging algorithm. In *Algorithmica 6*, pp. 816-825, pp. 1991.

[262] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science 6*, pp. 1-12, 1959.

[263] A. Mei and Y. Igarashi. An efficient strategy for robot navigation in unknown environment. In *Inform. Processing Letters 52*, pp. 51-56, 1994.

[264] P.B. Miltersen, S. Subramanian, J.S. Vitter, and R. Tamassia. Complexity models for incremental computation. In *Theoret. Comput. Scie. 130*, pp. 203-236, 1994.

[265] R. Motwani, S. Phillips, and E. Tong. Non-clairvoyant scheduling. In *Proc. of 4th ACM-SIAM Symp. on Discrete Algorithms, 1993. Also in Theoret. Comput. Scie. 130*, pp. 17-47, 1994.

[266] K. Mulmuley. *Computational Geometry : Introduction through Randomized Algorithms*. Prentice Hall, 1994.

[267] K. Mulmuley. On obstruction in relation to a fixed viewpoint. In *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 592-597, 1989.

[268] J.Von Neumann. Zur Theorie der Gesellschaftsspiele. In *Math. Ann. 100*, pp. 295-320, 1928.

[269] J. Von Neumann and O. Morgensten. *Theory of Games and Economic Behavior*. 1944.

[270] J. O'Rourke. *Art Callery Theorems and Algorithms*. Oxford University Press, Oxford, 1987.

[271] M. Overmars, M. Smid, M. de Berg and M. Van Kreveld. Maintaining range trees in secondary memory, part I: partition problem. In *Acta Informatica 27*, pp. 436-445, 1992.

[272] C. Papadimitrou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, N.J., 1982.

[273] P. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *Int. Colloq. on Automata, Languages, and Programming (ICALP' 89)*, pp. 610-620, 1989.

[274] P. Papadimitriou. Games against nature. In *Proc. of the 24th IEEE Symp. on the Foundations of Computer Science*, pp. 446-450, 1983.

[275] S. Pawagi and I.V. Ramakrishnan. Parallel updates of graphs properties in logarithmic time. In *Intern. Conference on Parllel Processing*, pp. 186-193, 1985.

[276] A. Pedrotti. Analysis of a list-update strategy. In *Inform. Letters 52*, pp. 115-121, 1994.

[277] S. Plotkin, D. Shmoys, and E. Tárdos. Fast approximation algorithms for fractional packing and covering problems. In *Proc. 23rd IEEE Annual Symp. on Foundations of Computer Science*, pp. 495-504, 1991.

[278] D.A. Petterson. Reduced instruction set computers. Communications of the A.C.M., Vol. 28, No.1, January, 1985.

[279] D. Peleg and E. Upfal. The token distribution problem. In *SIAM J. Comput. 18*, pp. 229-243, 1989.

[280] S. Phillips and J. Westbrook. On-line load balancing and network flow. In *Proc. 25th ACM Symp. on Theory of Computing*, pp. 402-411, 1993.

[281] F.P. Preperata and M.I. Shamos. *Computational Geometry: an Introduction*. Springer, Berlin, 1985.

[282] K. R. Pruhs. Average-case scalable on-line algorithms for fault replacement. In *Inform. Processing Letters 52*, pp. 131-136, 1994.

[283] M. Rabin and D. Lehman. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proc. of 8th ACM Symp. POPL*, pp. 133-138, 1981.

[284] P. Raghavan. Statistical adversary for on-line algorithms. *On-line Algorithms, DIMACS Series in Discrete Math. and Theoret. Comput.* pp. 79-82, 1992.

[285] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. In *16th International Colloquium on Automata Languages, and Programming, Vol. 372 of L. N. in Comp. Sc.*, pp. 687-703, 1989. Revised version available as *IBM Research Report RC15840*. IBM Watson Research Center, Yorktown Heights, N.Y., June 1990.

[286] P. Raghavan and C.D. Thompson. Provably good routing in graphs: Regular arrays. In *Proc. of 17th ACM Symp. on Theory of Computing*, 1985.

[287] Ramalingam and T. Reps. On competitive on-line algorithms for the dynamic priority-ordering problem. *Inform. Processing Letters 51, pp. 155-161*, 1994.

[288] H. Ramesh On traversing layered graphs on-line. In *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, pp. 412-421, 1993.

[289] M. Rauch. Fully dynamic biconnectivity in graphs. In *Proc. 33rd Ann. Symp. on Foundations of Computer Science*, 1992.

[290] J.H. Reif. A topological approach to dynamic graph connectivity. In *Inform. Processing 25*, pp. 65-70, 1987.

[291] E. Reingold and R. Tarjan. On a greedy heuristic for complete matching. In *SIAM. J. of Computing 10*, pp. 676-681, 1981.

[292] N. Reingold, J. Westbrook, and D.D. Sleator. Randomized competitive algorithms for the list update problem. In *Algorithmica 11*, pp. [illegible]2, 1994.

[293] T. Reiss. An approach to incremental compilation. *ACM SIGPLAN Notices 19*, pp. 144-156, 1984.

[294] T. Reps Incremental evaluation for attribute grammars with unrestricted movement between tree modifications. *Acta Informatica 25*, pp. 155-178, 1988.

[295] T. Reps, T. Teitelbaum, and S. Demers. Incremental context-dependent analysis for language-bases editors. *ACM Trans. on Programming Languages and Systems 5*, pp. 449-477, 1983.

[296] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM, Vol. 19*, pp. 63-67, 1976.

[297] C.A. Rogers. *Packing and Covering*. Cambridge University Press, Cambridge, 1964.

[298] B.G. Ryder and M.C. Paull. Incremental data flow analysis. In *ACM Trans. Programming Languages Systems 10*, pp. 1-50, 1988.

[299] S. Sairam, J.S. Vitter, and R. Tamassia. A complexity-theoretic approach to incremental computation. In *L. N. in Comput. Scie, Vol. 665, STACS' 93*, 1993.

[300] R. Sedgewick. *Algorithms*. Addison-Wesley, Reading, MA, 1986.

[301] M. Shanahan. Incrementality and logic programming. In *Reason maintenance systems and their applications, ed. B. Smith et al., Halstead Press, New York, NY,* 1988.

[302] C. Scheurich and M. Dubois. Dynamic page migration in multiprocessors with distributed shared memory. *IEEE Trans. on Computer, Vol. 38*, pp. 1154-1163, 1989.

[303] J.T. Schwarz and M. Sharir. Algorithmic motion planning in robotics. *Handbook of Theoretical Computer Science. MIT Press*, pp. 391-430, 1991.

[304] C. Schwarz, M. Smid, and J. Snoeyink. An optimal algorithm for the on-line closest-pair problem. *Algorithmica 12*, pp. 18-29, 1994.

[305] J.T. Schwartz and C.-K. Yap. *Algorithms and Geometric Aspects of Robotics.* Lawrence Erlbaum Associates, Hillsdale, 1987.

[306] F. Shahrokhi and D. Matula. The maximum concurrent flow problem. In *J. Assoc. Comput. Mach.*, pp. 318-334, 1990.

[307] G.S. Shedler and C. Tung. Locality in page reference strings. In *SIAM J. on Computing 1*, pp. 218-241, 1972.

[308] M. Sherk. Self-adjusting k-ary search trees. In *Proc. of the Workshop on Algorithms and Data Structure*, pp. 381-392, 1989.

[309] M. Sherk. Self-adjusting k-ary search trees and self-adjusting balanced search trees. *Technical Report 234/ 1990, University of Toronto*, 1990.

[310] T. Shermer. Recent results in art galleries. *CMPT TR 90-10, School of Computing Science, Simon Fraser University*, Oct., 1990.

[311] D.B. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machines on-line. In *Proc. 32nd IEEE Symp. in Foundations of Computer Science*, pp. 131-140, 1991.

[312] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM, Vol. 28,* pp. 202-208, 1985.

[313] D.D Sleator and E.E. Tarjan. Self-adjusting binary search trees. In *J. of the ACM, Vol. 32,* pp. 652-686, 1985.

[314] *M.I. Shamos.* Computational Geometry. *Ph. D. thesis, Department of Computer Science, Yale University,* (USA), 1978.

[315] *W.E. Smith. Various optimizes for single-stage production. Naval Res. Logist. Quart., Vol. 3,* pp. 59-66, 1956.

[316] M. Solomon and J. Desroiers. Time window constrained routing and scheduling problems: a survey. *Trans. Science 22,* pp. 1-13, 1988.

[317] *J.R. Spirn. Program Behavior: Models and Measurements.* Elsevier Computer Science Library, Elsevier, Amsterdam, 1977.

[318] J. Spencer. Ten Lectures on the Probabilistic Method. Regioinal Conference Series in Applied Mathematics, Vol 64. SIAM Philadelphia, PA, 1994.

[319] P.M. Spira and A. Pan. On finding and updating shortest paths and spanning trees. In *Symp. on Switching and Automata Theory,* pp. 82-84, 1973.

[320] E. Styer and G. Peterson. Improved algorithms for distributed resource allocation. In *Proc. of 7th ACM Symp. on Principles of distributed Computing,* pp. 105-116, 1988.

[321] R. Tamassia. A dynamic data structure for planar graph embedding. In *Proc. of the 15th Intern. Colloquium on Automata, Languages and Programming, Vol. 317,* pp. 576-590, 1988.

[322] R. Tamassia and F.P. Preparata. Dynamic maintenance of planar digraphs. In *Algorithmica 5,* pp. 509-527, 1990.

[323] *A.S. Tanenbaum. Modern Operating Systems.* Prentice-Hall, Englewood Cliffs, 1988.

[324] A.S. Tanenbaum and R. van Renesse. Distributed operating systems. In *ACM Computing Surveys, Vol. 17,* pp. 419-470, 1985.

[325] R.E. Tarjan. Amortized computational complexity. In *SIAM J. Alg. Discrete Methods, Vol. 6*, pp. 306-318, 1985.

[326] R.E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, 1983.

[327] R.E. Tarjan. Depth-first search and linear graph algorithms. In *SIAM J. Computing*, 1972.

[328] R.E. Tarjan. A class of algorithms that require nonlinear time to maintaining disjoint sets. In *J. of Computer and System Sciences, Vol. 18*, pp. 110-227, 1979.

[329] R.E. Tarjan. Efficiency of a good but not linear set union algorithm. In *J. of the ACM, Vol. 22*, pp. 215-225, 1975.

[330] B.A. Teia. Lower bound for randomized list update algorithms. In *Information Processing Letters 47*, pp. 5-9, 1993.

[331] M. Teillaud. *Towards dynamic randomized algorithms in Computational Geometry*. Springer-Verlag, 1993.

[332] P. Tetali. Design of on-line algorithms using hitting times. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*. pp. 402-411, 1994.

[333] P. Tetali. Random walks and the effective resistance of networks. In *J. Theoretical Probability*, pp. 101-109, 1991.

[334] Y.T. Tsai and C.Y. Tang. The competitive of randomized algorithms for on-line steiner tree and on-line spanning tree problems. Inform. Processing Letters 48, pp. 177-182, 1993.

[335] Y.T. Tsai, C.Y. Tang, and Y.Y. Chen. Average performance of a greedy algorithm for the on-line minimum matching problem of Euclidean space. In *Inform. Processing Letters 51*, pp. 275-282, 1994.

[336] G. Turbin. Resent work on the server Problem. *Master's thesis, University of Toronto*, 1989.

[337] S. Vishwanathan. Randomized on-line graph coloring. In *Proc. of 31st IEEE Symp. on Foundations of Computer Science*, pp. 464-469, 1990. Also in *J. Algorithms 13*, pp. 657-669. 1992.

[338] J.S. Vitter and P. Krishmam. Optimal prefetching via data compression. In *32nd IEEE Symp. on Foundations of Computer Science*, pp. 121-130, 1991.

[339] J. WestbrooK. Randomized algorithms for multiprocessor page migration. *SIAM J. Computing 23*, pp. 951-965, 1994.

[340] J. Westbrook and R. E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica 7*, pp. 433-464, 1992.

[341] J. Westbrook and D.C.K. Yan. Greedy algorithms for on-line steiner tree and generalized steiner problems. In *4th WADS' 93. Algorithms and Data Structure*, pp. 622-633, 1993.

[342] A. Wig'erson. Improving the performance guarantee for approximate graph coloring. In *J. ACM 30*, pp. 729-735, 1983.

[343] G.J. Woeginger. On-line scheduling of jobs with fixed start and end times. In *Theoretical Computer Science 130*, pp. 5-16, 1994.

[344] A.C.-C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. of 18th IEEE Symp. on the Foundation of Computer Science*, pp. 222-227, 1977.

[345] C.-K. Yap. Algorithmic motion planning. In : *Algorithmic and Geometric Aspects of Robotics, Vol. 1, Hillsdale and London*, pp. 95-143, 1987.

[346] D.M. Yellin. Speeding up dynamic transitive closure for bounded degree graphs. In *Acta Informatica*, 1991.

[347] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica 11*, pp. 525-541, 1994.

[348] N. Young. On-line caching as cache size varies. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, pp. 241-250, 1991.

[349] Zhou, K. Schwan, and I.F. Akyildiz. Performance effects of information sharing in a distributed multiprocessor real-time scheduler. In *Proc. IEEE Real-Time Systems Symp.*, pp. 46-55, 1992.

[350] S. Ben-David and A. Borodin. A new measure for the study of on-line Algorithms. In *Algorithmica 11*, pp. 73-91, 1994.

[351] P. Raghavan. *Lectures Notes on Randomization Algorithms. Research Report RC 15340 (#68237). IBM Research Division. T.J. Watson research Centre,Yorktown Heights, NY 10598*, 1990.

[352] Erdös. On a combinatorial problem, I and II. In J. Spencer, ed., Paul Erdös: *The Art of Computing. MIT Press, Cambridge, MA*, 1973.

[353] A. Apostolico, M. Farah, and C.S. Iliopoulos. Optimal superprimitivity testing forstrings.In *Information Processing Letters 39*, pp. 17-20, 1991.

[354] D. Dowdy and Foster. Competitive models of the file assignment problem. In *Computing Surveys*, 14, pp. 287-313, 1982.

[355] M. Imase and B. M. Waxman. Dynamic steiner tree problem. In *SIAM J. on Discrete Mathematics, 4(3)*, pp. 369-384, 1991.

[356] C. Lund and N. Reingold, J. Westbrook and D. Yan. On-line distributed data management. *Manuscript submitted*. 1994.

[357] R. El-Yaniv. On-line algorithms and financial decision making. *Ph. D. Thesis, University of Toronto*, 1994.

[358] A. Marchetti - Spaccamela and C. Vercellis. Stochastic on-line knapsack problems. In *Math. Programming 68*, pp. 73-104, 1995.

[359] F. Luccio and A. Pedrotti. A parallel list update problem. *Information Processing Letters 52*, pp. 277-284, 1994.

[360] A. Blum and P. Chalasani. An on-line algorithm for improving performance in navigation. In *Proc. of 34th Symp. on Computer Science*, pp. 2-19, 1993.

Give them according to the deeds

Psalms 28: 4

END

05-06-96

FIN