UNIVERSITY OF CALIFORNIA, SAN DIEGO

On the Design and Worst-Case Analysis of Certain Interactive and Approximation Algorithms

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

Jia Mao

Committee in charge:

Professor Ronald L. Graham, Chair
Professor Samuel R. Buss
Professor Fan Chung Graham
Professor Alon Orlitsky
Professor George Varghese

2007

The dissertation of Jia Mao is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2007

To my parents

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First I thank the Lord for His unfathomable love and grace for me. To Him be all the glory.

My sincere gratitude also goes to many friends, colleagues and family members. Without their support, this dissertation would still be far from complete.

Specially, I would like to thank my advisor Ron Graham for his continuous encouragement and invaluable guidance throughout my graduate study; my mentors/collaborators Fan Chung, Andrew Chi-Chih Yao and George Varghese for helping me in so many ways; my other collaborator Steven Butler with whom it has been a great pleasure working; my family for being there for me in good times and bad; and Fu Liu for her faithful friendship.

Chapter 2 and 4 contain material appearing in "Finding favorites" from *Electronic Colloquium on Computational Complexity(ECCC) (078) 2003* and "Oblivious and Adaptive Strategies for the Majority and Plurality Problems" from the *11th International Computing and Combinatorics Conference (COCOON) 2005: 329-338*, and material to appear in *Algorithmica*. Chapter 5 contains material submitted for publication. Chapter 6 and 8 contain material appearing in "Parallelism vs. Memory Allocation in Pipelined Router Forwarding Engines" from *Theory of Computing Systems (ToCS), ISSN 1432-4350 (Print) 1433-0490 (Online) 2006*. Chapter 6 and 7 contain material submitted for publication. I would like to thank my co-authors Fan Chung, Ron Graham, George Varghese, Andrew Chi-Chih Yao, and Steven Butler.

VITA

2002	B.S. with Honors, California Institute of Technology
2004	M.S., University of California, San Diego
2007	Ph.D., University of California, San Diego

RELATED PAPERS and PREPRINTS

"Oblivious Strategies in the Majority and Plurality Problems". (with F. Chung, R. Graham, and A. Yao). *Algorithmica, to appear, 2006.*

"How to Play the Majority Game with Liars". (with S. Butler and R. Graham). *Discrete Applied Mathematics, under review, 2006.*

"Parallelism vs. Memory Allocation in Pipelined Router Forwarding Engines". (with F. Chung, R. Graham, and G. Varghese). *Theory of Computing Systems (ToCS), ISSN 1432-4350 (Print) 1433-0490 (Online), 2006.*

"Finding Favorites". (with F. Chung, R. Graham, and A. Yao). *Electronic Colloquium on Computational Complexity (ECCC)(078), 2003.*

"Oblivious Strategies in the Majority and Plurality Problems". (with F. Chung, R. Graham, and A. Yao). *Computing and Combinatorics (COCOON) 2005, pp. 329-338.*

"How to Play the Majority Game with Liars". (with S. Butler and R. Graham). *Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications (CS), oral presentation, 2006.*

"Parallel Resource Allocation of Splittable Items with Cardinality Constraints". (with R. Graham). *Preprint, 2006.*

FIELDS OF STUDY

Computer Science

Studies in Algorithms and Complexity

Professor Ronald L. Graham

ABSTRACT OF THE DISSERTATION

On the Design and Worst-Case Analysis of Certain Interactive and Approximation Algorithms

by

Jia Mao
Doctor of Philosophy in Computer Science
University of California, San Diego, 2007
Professor Ronald L. Graham, Chair

With the speed of current technological changes, computation models are evolving to become more interactive and dynamic. These computation models often differ from traditional ones in that not every piece of the information needed for decision making is available a priori. Efficient algorithm design to solve these problems poses new challenges.

In this work we present and study some interactive and dynamic computations and design efficient algorithmic schemes to solve them. Our approach for performance evaluation falls within the framework of *worst-case* analysis. The worst-case scenarios are analyzed through the incorporation of imaginary adversaries or adversarial input sequences. *Worst-case* analysis provides safe performance guarantees even when we have little or no prior knowledge about the input sequences. Another natural yet powerful tool we utilize is an auxiliary graph which evolves as the computation progresses. It helps us to visualize the computation step by step, and more importantly, offers us powerful mathematical tools from the well-developed area of graph theory.

We first address a particular computation problem of interactive nature, best

known as the Majority/Plurality game. This interactive game has appeared in several different contexts since the 1980s such as system diagnosis and group testing. We design and analyze optimal strategies to minimize the amount of communication needed in different settings against an imaginary adversary. We also consider error-tolerance features to make our strategies robust even in the presence of communication errors.

We then introduce a new variant of the classical bin packing problem that allows arbitrary splitting of the items with the restriction on the number of different types in each bin. This problem is specifically motivated by a practical problem of allocating memories to parallel processors in high-speed routers. It is also natural to other similar resource allocation applications. Even the simplest case of this problem can be shown to be NP-hard. We design efficient approximation algorithms in the offline, online, and dynamic settings. We also use an interesting ε-improvement technique to show improved approximation ratios.

1

Introduction

The rapid developments of computing device and network technologies have changed the world around us. One aspect of this change is that traditional computation models have evolved to be increasingly distributive and interactive. Inevitably, these new computation models are of great importance and have gained more and more attention in the research community in recent years. However, our understanding for these models is still quite inadequate and challenging new problems are emerging on a regular basis. It is common to observe that some of these problems are interactive in nature and others face the reality that not every piece of the information needed for decision making is available a priori.

Consequently, algorithm design for these problems is in need of new insight and new techniques. In this work we present and study some interactive and dynamic computations and design efficient algorithmic schemes to solve them. A powerful tool in our analysis is graphical representation which evolves as the computation progresses, naturally adapting to an interactive or dynamic computation environment. Graphs not only help us visualize the computations step by step, but more importantly, offer us powerful mathematical tools from the well-developed area of graph theory.

To evaluate the performance of the algorithms, a *worst-case* approach is adopted through the incorporation of imaginary adversaries or adversarial input sequences. This is in contrast to another commonly used approach, *average-case* analysis, where a distribution of input sequences is hypothesized and the expected total cost or performance is evaluated. *Worst-case* analysis attempts to finesse the issue of little or no prior knowledge about what the input sequences are likely by taking a pessimistic approach.

1.1 Interactive Computations

A basic theme for interactive computing involves two or more parties and a sequence of queries/answers where each query or answer can depend on previous ones. Here by *interactive computing*, what we mean is different from the operating system research point of view, where *interactive computing* usually corresponds to timesharing and refers to the case that a user can communicate and respond to the computer's responses in a way that batch processing does not allow [62]. Unlike the generic computation models, our understanding for interactive computing is far from adequate. However, with the convergence of communication, computation, and large shared information sources, the need for a solid theoretical foundation for interactive computing is imperative.

In the first part of this dissertation, we focus on a particular type of interactive game, called the Majority/Plurality game. It is an information-theoretic identification problem that first appeared in the 1980s [51]. We encountered this problem again in a practical context where a good sensor needs to be identified from a set of sensors in which some are non-operational or corrupted and it is desired to minimize the amount of intercommunication used in doing so [64] [23]. There are many interesting variants of the original problem [1]. We will survey related

literature, discuss these variants and devise effective strategies to solve them.

1.2 Online and Dynamic Computations

Online computations make a sequence of decisions under uncertainty [17]. One of the most powerful methods of analyzing such problems is *competitive analysis*, which is a type of *worst-case* analysis. The *competitive ratio* of an algorithm is the worst-case ratio of its performance to the performance of the best offline algorithm.

In the second part of this dissertation, motivated by a practical problem of allocating memories to parallel processors in the context of designing fast IP lookup schemes [21], we propose a new variant of the classical bin packing problem - kBPS . This variant can be shown to be NP-hard even in the simplest case. We will design efficient approximation algorithms for this problem in the offline, online, and dynamic settings. A dynamic setting extends an ordinary online setting in that real-time resource requests can be allocation or deallocation requests. In the dynamic setting, we capture the tradeoff between repacking cost and resource utilization by defining a *compaction ratio* parameter and devise efficient approximation algorithm with bounded compaction ratio.

1.3 Game-theoretic Notions

Ben-David, Borodin, Karp, G. Tardos, and Wigderson [11] introduced a general framework called *request-answer* games to study online algorithms. In a request-answer game, an imaginary adversary makes a sequence of requests, which need to be answered (served) one at a time by the online algorithm.

For interactive games such as the majority game, a similar general framework

can be proposed, the *query-answer* game. In a query-answer game, the algorithm poses a sequence of queries, and an imaginary adversary has to answer these queries in one or several batches. When only one batch is allowed, this game is in an *oblivious* setting. When the query is answered one at a time, this game is in an *adaptive* setting.

1.4 Thesis Outline

In Chapter 2, we introduce the *majority* problem and its game-theoretic notion. A graphical representation to keep track of the game configuration is described. Optimal winning strategies are then defined with respect to minimum communication needed. In Chapter 3, expander graphs are brought in to devise oblivious strategies for the *majority* game. This is joint work with Fan Chung, Ron Graham and Andrew C. Yao [23] [24].

In Chapter 4, a natural variant of the *majority* game is introduced, namely, the *plurality* game. We present upper bounds for optimal winning strategy in the adaptive setting. We then summarize other known results and discuss the power of randomization for the strategies. This is joint work with Ron Graham [24] [25].

In Chapter 5, we consider *error-resilient* strategies for the *majority* game. This feature is vital in the presence of possible communication errors. We construct clever combinatorial gadgets to design optimal winning strategies in both the adaptive and oblivious settings. This is joint work with Steven Butler and Ron Graham [18].

The 2BPS problem is introduced in Chapter 6. The NP-hardness result is recalled [21]. Then a few approximation algorithms are presented and analyzed. A simple online approximation algorithm for kBPS is also described. This is joint

work with Fan Chung, Ron Graham and George Varghese [22] [50].

In Chapter 7, we present an improved approximation algorithm $\mathcal{INC}$ for 2BPS with its analysis using an ε-improvement technique. For the general kBPS problem, an improved approximation algorithm $\mathcal{INC}_k$ is described and analyzed. This is joint work with Ron Graham [**?**].

In Chapter 8, the dynamic setting for 2BPS is introduced. To design efficient algorithms in this setting, a parameter called the *compaction ratio* is defined to capture the tradeoff between repacking and resource utilization. In particular, an algorithm $\mathcal{DYN}$ is given with bounded compaction ratio. This is joint work with Fan Chung, Ron Graham and George Varghese [22].

We summarize and conclude in Chapter 9.

2

The Majority Problem

2.1 Background

The earliest variant of the *Majority* problem was proposed by Moore in the context of fault-tolerant system design in 1982 [51]. The goal was to find the majority vote among n processors with a minimum number of paired comparisons. A number of different variants were subsequently proposed and analyzed. This problem reappeared to us after about twenty years in a military application where communication needs to be minimized to locate one sensor that has not been corrupted among a group of sensors [23].

This problem fits nicely into a general game-theoretic framework - the *query-anwer* games. The *Majority* game involves two players: **Q**, the Questioner, and **A**, the Adversary. **A** holds a set of n elements, each of which is labelled in one of $k \in Z^+$ possible labels $\phi = \{l_1, l_2, \ldots, l_k\}$. **Q** wants to identify one element of the majority label (or in the case of a tie, claim that there is none) using only pairwise equal/unequal label comparisons of elements, i.e., queries of the form "Is $\phi(a) = \phi(b)$?". **A** can answer each such query with the hope of extending the

game as long as possible. At the end, **Q** provides his final identification and then **A** reveals the actual labelling of the elements, which must be consistent with all the answers given. **Q** wins the game if **Q**'s identification is correct with respect to the actual labelling, otherwise **A** wins. We say **Q** has a *winning strategy* of length q if he/she can always win the game in at most q questions, regardless of what **A** does. Our goal is to construct such strategies with the smallest possible q.

In general, two types of settings can be considered, corresponding to two types of strategies for **Q**:

Definition 2.1. *In the* **adaptive** *setting,* **A** *answers queries one at a time and* **Q***'s next query can depend on the answers given to all previous queries.*

Definition 2.2. *In the* **oblivious** *setting,* **Q** *has to specify all queries in one batch for* **A** *to answer.*

Clearly, in the oblivious case, **A** has more opportunity to be evasive. Indeed, our bounds for the minimum length of **Q**'s winning strategy are much weaker in the oblivious setting than in the adaptive setting.

2.2 Optimal Winning Strategies

2.2.1 Current best bounds

We are interested in constructing winning strategies for **Q** with minimum length. The variants that have been studied in the literature so far can be classified based on:

(i). k: the number of permissible labels;

(ii). Level of interactivity between **Q** and **A**: adaptive or oblivious;

(iii). A majority label is known to exist or not.

The following notations for the minimum length of **Q**'s winning strategies will be used throughout this dissertation:

Table 2.1: Notations for minimum length of **Q**'s winning strategies for the Majority Game

Majority game of n elements		
$MA_k(n)$	Adaptive setting	k possible different labels
$MA_*(n)$	Adaptive setting	k unknown, can be arbitrary
$MO_k(n)$	Oblivious setting	k possible different labels
$MO_*(n)$	Oblivious setting	k unknown, can be arbitrary

Current best bounds for the minimum length of winning strategies are listed in Table 2.2, in which the ones in bold are our contributions.

Table 2.2: Current Best Bounds for the Majority Game

Majority Game	known existence	
	yes	no
MA_2	$n - \mu_2(n)$	
MO_2	$\mathbf{2\lfloor n/2 \rfloor - 2}$	$2\ \lfloor n/2 \rfloor - 1$
MA_*	$n - \mu_2(n)$	$\lceil 3n/2 \rceil - 2$
MO_*	$\mathbf{(1 + o(1))21n}$ *upper*[1]	$(\frac{\mathbf{1}}{\mathbf{4}} - \mathbf{o(1)})\mathbf{n^2}$ *lower*
MA_k	$n - \mu_2(n)$	linear
MO_k	linear	quadratic

[1]The coefficient 21 can be further reduced to 19.5 if only existence of such a strategy is desired [25]

In the adaptive case, Saks and Werman [57] were the first to prove a tight bound of the minimum length of a winning strategy for **Q** to be $n - \mu_2(n)$ when $k = 2$, where $\mu_2(n)$ is the number of 1's in n's binary expansion. Different proofs for the same result were subsequently given in [4] and [67]. When k is unknown, a tight bound of $\lceil 3n/2 \rceil - 2$ was given in [36]. The average case of the same setting was analyzed in [5]. Similar bounds were proven for randomized algorithms [44].

In the oblivious case, when k is unknown, the optimal winning strategy for **Q** is much harder to design or analyze. If the existence of a majority label is not known a priori, **Q** needs at least a quadratic number $(\frac{1}{4} - o(1))n^2$ many questions. However, if a majority label is known to exist, by using a special type of graphs, called Ramanujan graphs, we show that there is a constructive strategy for **Q** that uses no more than $(1 + o(1))21n$ queries. The constant 21 can be further improved to 19.5 if only existence of such a strategy is desired.

When k is unknown for the Majority game, it is also interesting to observe how much computation the extra piece of information of just knowing the existence of a majority can save us in both the adaptive and the oblivious cases.

For fixed k, no explicit bounds for $MA_k(n)$ or $MO_k(n)$ have been presented in past literature. However, they can be readily deduced from other bounds for the Majority and Plurality game. We roughly list their asymptotic behavior in Table 2.2.

2.2.2 Auxiliary Graphs

We will use an auxiliary graph H to represent the current state of the game at each time step. The n elements will be the vertices of this undirected (multi)graph. Any equal/unequal label comparison query corresponds to the selection of two nodes, and the answer given by **A** corresponds to a colored edge drawn between

them. We let color *blue* represent an *equal* answer, and *red* an *unequal* answer. So as the game progresses the vertex set of H remains fixed, but the edge set is growing because every query and answer adds a colored edge to this graph. Note that there will be no loops in this graph, i.e., any cycle is of length at least 2. See Figure 2.1 for an example.

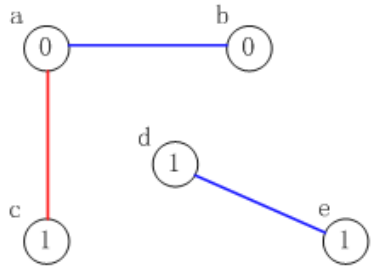

Figure 2.1: An auxiliary graph on a set of $n = 5$ elements $\{a, b, c, d, e\}$ each with a binary label ($k = 2$). The *blue* edges denote the "equal" answer for label comparisons and the *red* edges denote the "unequal" answers. Note that the actual labelling is not revealed to **Q** until the end of the game.

The simplest case for the majority game is when $k = 2$, i.e., the elements have binary labels. The following definitions and properties apply to the majority game with binary labels only. Some of them may be extended to general fixed k easily.

Definition 2.3. *At any time step, we say an edge coloring of H is valid if there exists a partition of the vertex set $V(H)$ into (V_1, V_2) such that:*

i. $V_1 \cap V_2 = \emptyset$,

ii. $V_1 \cup V_2 = V(H)$,

iii. *For fixed $i \in \{1, 2\}$, $\forall u, v \in V_i$, the edge (u, v) is either non-existent, or has color blue*

iv. $\forall u \in V_1, v \in V_2$, *the edge (u, v) is either non-existent or has color red.*

We list some properties of the auxiliary graph H below, all of which are easy to

verify.

Fact 2.1. *The number of queries asked so far is at least $(n - m)$, where m is the number of connected components of H.*

Fact 2.2. *At any time step, for any connected component D in G and any valid edge coloring of G, the corresponding edge coloring of D must also be valid.*

Definition 2.4. *At any time step, we define a discrepancy value for every connected component D in G as follows:*

$$\delta(D) = \big||V_1(D)| - |V_2(D)|\big|$$

For any vertex $v \in V(D)$, define

$$\delta(v) = \begin{cases} |V_1(D)| - |V_2(D)|, & \text{if } v \in V_1(D), \\ |V_2(D)| - |V_1(D)|, & \text{if } v \in V_2(D). \end{cases}$$

Definition 2.5. *A connected component of G is called odd if it contains an odd number of vertices, and even otherwise.*

Fact 2.3. *The δ value of an odd component is always odd; the δ value of an even component is always even.*

Fact 2.4. *Whenever a query connects two previously separated components with discrepancies δ_1 and δ_2, depending on the answer given by $\mathbf{A}$, the newly formed component has either $\delta = \delta_1 + \delta_2$ or $\delta = |\delta_1 - \delta_2|$.*

Fact 2.5. *When all components are either isolated vertices or paths with all connecting edges being red, we have the property that any odd component has $\delta = 1$ and any even component has $\delta = 0$.*

2.2.3 Upper bounds for MA_2 and MA_*

At any time in the game, the auxiliary graph H has a certain number of connected components. Let us call a connected component C_r of H *mixed* if it has

at least one red edge. Otherwise, we say that C_r is *pure*. Note that with this definition, an isolated point is a pure component. The size of a component $|C_r|$ is simply the number of vertices in it.

A simple binary strategy for **Q** can give us the upper bounds for MA_2 in general and MA_* when the existence of majority label is known a priori.

Since H initially consists of n pure components of size 1, then in general, any connected component C_r of H will have $|C_r| = 2^t$ for some $t \geq 0$.

Q's Strategy

- Loop until all pure components have distinct sizes
  - Ask the query "Is $\phi(s_i) = \phi(s_j)$?" where s_i and s_j belong to two different pure components having the same size.

Let G be the final graph when the process stops, i.e., when all pure components $C_1, \ldots, C_m$ of G have distinct sizes, say $|C_i| = 2^{t_i}$ with $t_1 > t_2 > \ldots > t_m \geq 0$. If G does not have any pure components, **Q** can declare that there is no majority label because all mixed components have a perfect balance of the labels. If $k \geq 1$, i.e., G has at least one pure component. we claim that any vertex $s_i \in C_1$ must have a majority label.

Finally, if G has m components altogether (pure and mixed), then n can be represented as a sum of m powers of 2. If $n = \sum_{j=1}^{p} 2^{a_j}$ is such a representation with the minimum possible value of p, then we must have all the a_j distinct, since otherwise we can replace $2^{a_j} + 2^{a_j}$ by 2^{a_j+1}. Therefore, $p = \mu_2(n)$, the number of 1's in the binary expansion of n. Since by the definition of **Q**'s strategy, all the components of G are trees, therefore **Q** asks at most $n - \mu_2(n)$ queries.

Proofs for the lower bound $n - \mu_2(n)$ are much more involved. Saks and Werman [57] gave the first such proof. Different proofs for the same result were sub-

sequently given in [4] and [67]. We include a sketch of Wiener's proof [67] in Appendix A for the readers' benefit.

There is an interesting number-theoretic formulation of the majority game with binary labels, which is utilized in our earlier experimental implementations as well as in Wiener's proof. This formulation still models the problem as a game played between two players **Q** and **A**. At any given point during the game, the configuration of the game can be uniquely and fully described by a multiset $M = (m_1, m_2, ..., m_t)$ of nonnegative integers. The starting multiset consists of n 1's. In every round, **Q** picks two members m_i, m_j from the multiset, and **A** replaces them by either their sum or their absolute difference to his will. The game ends when the largest member of M is greater than the sum of the others or when all members of M are zero. **Q** wants to minimize the number of rounds, namely to maximize $|M|$ and **A** wants the opposite. It should be clear that the two formulations are equivalent and that each entry in the multiset M corresponds to the δ value of a particular connected component at that time.

Remark: It is also worth noting that Fischer and Salzberg [36] proposed an optimal strategy for MA_* with a tight bound of $(\lceil 3n/2 \rceil - 2)$. Compared with the $n - \mu_2(n)$, it is interesting to observe how much computation the extra piece of information of just knowing the existence of a majority can save us.

2.2.4 Bounds for MO_2

In the oblivious setting, we have the following result

Theorem 2.1. *For $n \geq 3$,*

$$MO_2(n) = 2\lfloor n/2 \rfloor - 2$$

assuming a majority label exists.

Proof. It is easy to see (by induction on $|C_i|$) that the Adversary can always assign binary labels to the vertices of a component C_i so that $\delta(C_i) = 1$ if C_i is odd, and $\delta(C_i) = 0$ (or $\delta(C_i) = 2$) if C_i is even.

First, suppose n is odd. Then H must have an odd number t of odd components. However, if $t \geq 3$ then the Adversary could force t values of $\delta(C_i)$ to be 1, in which case **Q** cannot conclude. Hence we must have $t = 1$ when n is odd. We now claim that H can have at most two components. For suppose to the contrary that H has $m \geq 3$ components $C_i, 1 \leq i \leq m$. Then the Adversary can assign binary labels so that $\delta(C_i) = 1$ for each odd C_i, and $\delta(C_i) = 2$ for each even C_i. Again in this case **Q** cannot conclude. Therefore $MO_2(n) \geq n - 2$ if n is odd.

A very similar argument applies to the case that n is even to show that $m \leq 3$. Thus, we can conclude that $MO_2(n) \geq n - 3$ when n is even.

To establish the upper bounds, **Q** chooses components as follows:
If n is odd, then $H = \{C_1, C_2\}$ with $|C_1| = n - 1, |C_2| = 1$;
If n is even, then $H = \{C_1, C_2, C_3\}$ with $|C_1| = n - 2, |C_2| = |C_3| = 1$.

Since $\delta(C) = 1$ if $|C| = 1$ and $\delta(C)$ is even if $|C|$ is even, then in each case we can always identify an element of the majority label once the Adversary answers all the queries. This proves the theorem. □

The proof for the bound of MO_2 when the existence of a majority label is not known a priori is entirely analogous. When the labels are no longer binary, the problem becomes more complicated.

2.2.5 Lower bound for MO_* when existence is not known

Theorem 2.2. *When a majority label is not known to exist a priori,*

$$MO_*(n) \geq (\frac{1}{4} - o(1))n^2$$

Proof. This proof uses similar argument as in Theorem 4.1. Consider any auxiliary graph G with n vertices and at most $\frac{n^2}{4}$ edges. Therefore there must exist a vertex v with $deg(v) \leq n/2$. Denote the neighborhood of v by $N(v)$. Let H_1 be a set of exactly $\lfloor n/2 \rfloor$ vertices such that $N(v) \subseteq H_1$ and $v \notin H_1$. Let $H_0 = G \setminus (H_1 \cup \{v\})$. Therefore, H_0 and H_1 have exactly the same number of vertices when n is odd, or off by 1 when n is even.

Now assign label 0 to all vertices in H_0 and label 1 to all vertices in H_1. For all queries involving v, let the answers be *no* so that these edges are colored *red*, i.e., v cannot be assigned label 1. **A** can either assign label 0 to v or assign another label, say, 3 to v, and **Q** cannot deduce a correct answer with this piece of information missing. Hence the bound is proved. □

Remark 2.1. *When existence of a majority label is known a priori, we will show that in fact no more than a linear number of queries are needed for* **Q** *in the next chapter. Here we provide some intuitive explanation for such a surprising result. Intuitively, if more than half of the vertices in G share the same label, any edge between these vertices have to be colored blue. When G is* reasonably *well connected, we should expect to observe a* large blue *component embedded in G. And we are done if such a component is unique.*

2.3 Acknowledgement

This chapter contains material appearing in "Finding favorites" from *Electronic Colloquium on Computational Complexity(ECCC) (078) 2003* and "Oblivious and Adaptive Strategies for the Majority and Plurality Problems" from the *11th International Computing and Combinatorics Conference (COCOON) 2005: 329-338*, and material to appear in *Algorithmica*. I would like to thank my co-authors Fan Chung, Ron Graham, and Andrew Chi-Chih Yao.

3

Expander Graphs to Rescue

3.1 Expander Graphs

Expander graphs were first introduced in the 1970s and have turned out to be a very useful tool for many theoretical and practical areas. Their applications span the areas of communication networks, error correcting codes, computational complexity and number theory. Expander graphs possess seemingly conflicting properties simultaneously: high connectivity and sparseness.

Given an undirected finite (multi)graph $G = (V, E)$, with vertex set V such that $|V| = n$ and edge set E, there are a few notions to quantify the expansion properties:
(i). The edge expansion $h(G)$ of G is defined as

$$h(G) = \min_{1 \leq |S| \leq \frac{n}{2}} \frac{|\partial(S)|}{|S|}$$

where S is any nonempty set of at most $n/2$ vertices and $\partial(S)$ is the edge boundary of S, i.e., the set of edges with exactly one endpoint in S.

(ii). The α-vertex expansion $g_\alpha(G)$ of G is defined as

$$g_\alpha(G) = \min_{1 \le |S| \le \alpha n} \frac{|\Gamma(S)|}{|S|}$$

where S is any nonempty set of at most αn vertices and $\Gamma(S)$ is the vertex boundary of S, i.e., the set of vertices that are neighbors of S.

Every finite connected graph G has some positive expansion parameters. It is not obvious whether we can keep the expansion parameters bounded away from zero when G gets large. Complete graphs have the best possible expansion, but to achieve sparseness, we are interested in constant degree graphs. Expander graphs can then be defined accordingly. For example, a family $\mathcal{G} = \{G_1, G_2, ...\}$ of d-regular graphs is an edge expander family if there is a constant $c > 0$ such that $h(G) \ge c$ for $\forall G \in \mathcal{G}$. Similarly, $\mathcal{G}$ is a vertex expander family if there is a constant $c > 1$ such that $g_{1/2}(G) \ge c$ for $\forall G \in \mathcal{G}$. These expansion parameters are interrelated and every vertex expander family is also an edge expander family.

The spectrum of a graph G is the set of eigenvalues of its adjacency matrix. In many contexts, the normalized adjacency matrix is used instead and all results are analogous. Let G be a d-regular (multi)graph with adjacency matrix A. Because A is symmetric, it has n real-valued eigenvalues. The largest eigenvalue of A is $\lambda_0 = d$ with eigenvector $u = (1, \ldots, 1)$. The second largest eigenvalue is given by

$$\lambda = \max_{i \ne 0} \{|\lambda_i|\}$$

There is interesting connection between the expansion of G and its spectrum. In particular, the larger the spectral gap $(d - \lambda)$ (i.e., small λ), the better expansion G has. The following theorem holds due to Cheeger & Buser in the continuous case and to Tanner, Alon & Milman in the discrete case [49].

Theorem 3.1.

$$\frac{d - \lambda}{2} \le h(G) \le \sqrt{2d(d - \lambda)}$$

Large spectral gap indicates good expansion. However, there is limit on how large the spectral gap can be [49] due to Alon and Boppana.

Theorem 3.2.

$$\lambda \geq 2\sqrt{d-1} - o_n(1) \tag{3.1}$$

With high probability, a random d-regular graph has good expansion. Therefore, expander graphs exist in abundance. They are all around us but we cannot easily find or even recognize them. When the spectral limit is achieved, the d-regular expander graphs have the best expansion in the spectral sense and are called *Ramanujan* graphs. Explicit constructions have been found for some Ramanujan graphs [48].

In the following section, we are going to use Ramanujan graphs $X^{p,q}$, which are defined for any primes p and q congruent to 1 modulo 4.

$X^{p,q}$ has the following properties:
(i) $X^{p,q}$ has $n = \frac{1}{2}q(q^2-1)$ vertices;
(ii) $X^{p,q}$ is regular of degree $p+1$;
(iii) The adjacency matrix of $X^{p,q}$ has the large eigenvalue $\lambda_0 = p+1$ and all other eigenvalues λ_i satisfying $|\lambda_i| \leq 2\sqrt{p}$.

Expander graphs behave very much like random graphs. In fact, λ can be used to measure how "random" a graph is. The relationships are often referred to as expander mixing lemmas or discrepancy inequalities. We will use the following discrepancy inequality (see [3] and [20]) for a d-regular graph H of n vertices with eigenvalues satisfying $\lambda \leq \delta$.

For any subset $X, Y \subseteq V(H)$, we have

$$| \, e(X,Y) - \frac{d}{n}|X| \, |Y| \, | \leq \frac{\delta}{n}\sqrt{|X|(n-|X|)|Y|(n-|Y|)} \tag{3.2}$$

where $e(X,Y)$ denotes the number of edges between X and Y.

3.2 Expanders and Optimal Oblivious Strategy

Consider the case where the number of possible labels k is unrestricted. In principle, this is a more challenging situation for **Q**. At least the upper bounds we have in this case are weaker than those for $k = 2$ labels. A linear upper bound can be shown assuming the existence of a majority label. Without such assumption, a quadratic lower bound is already proven in the previous chapter.

Theorem 3.3. *For all n,*

$$MO_*(n) \leq (1 + o(1))21n,$$

assuming a majority label exists.

Proof. If the existence of a majority label is known a priori, let us denote the majority label by $l \in \phi$. The set of the remaining possible labels is still unknown. In the game configuration graph H, the vertices correspond to the elements, the queries correspond to edges, and the answers are represented by edge colors. The vertex set of H is $V(H) = \{v_1, \ldots, v_n\}$. An edge $\{v_i, v_j\}$ in H corresponds to the query $Q(v_i, v_j) :=$ "Is $\phi(v_i) = \phi(v_j)$?". The edge is colored *blue* if they are equal, and *red* if they are not equal.

Here by a *valid assignment* ϕ on $V(H)$ we mean that:
(i) $\phi(v_i) = \phi(v_j) \Rightarrow \{v_i, v_j\}$ is blue,
(ii) $\phi(v_i) \neq \phi(v_j) \Rightarrow \{v_i, v_j\}$ is red,
(iii) $|\phi^{-1}(l)| > n/2$.

An oblivious strategy for **Q** is simply a layout of edges in H such that **Q** can correctly identify an element of majority label regardless of the edge coloring **A** specifies. Intuitively, H should be reasonably well connected so that the elements of majority label always form a large embedded blue component. This notion of good connectivity leads us naturally to expander graphs [3].

Applying (3.2) to $X^{p,q}$, we obtain for all $X, Y \subseteq V(X^{p,q})$,

$$| e(X,Y) - \frac{p+1}{n}|X|\ |Y|\ | \leq \frac{2\sqrt{p}}{n}\sqrt{|X|(n-|X|)|Y|(n-|Y|)} \tag{3.3}$$

where $n = \frac{1}{2}q(q^2-1) = |V(X^{p,q})|$.

The oblivious strategy for $\mathbf{Q}$ is first to construct a Ramanujan graph $X^{p,q}$ on the vertex set $V(H) = \{v_1, \ldots, v_n\}$. Let ϕ be a valid assignment of $V(H)$ and consider the subgraph M of $X^{p,q}$ induced by $\phi^{-1}(l)$ (the majority-labelled vertices of $X^{p,q}$ under the mapping ϕ).

Claim: If $p \geq 38$, then M has a connected component C with size at least $c'n$, where $c' > 1/3$.

Proof: We will use (3.3) with $X = C$, the largest connected component of M, and $Y = \phi^{-1}(c) \setminus X$. Write $|\phi^{-1}(c)| = \alpha n$ and $|C| = \beta n$. Since $e(X,Y) = 0$ for this choice, then by (3.3) we have

$$\begin{aligned}
(p+1)^2|X|\ |Y| &\leq 4p(n-|X|)(n-|Y|), \\
(p+1)^2\beta(\alpha-\beta) &\leq 4p(1-\beta)(1-\alpha+\beta), \\
\beta(\alpha-\beta) &\leq \frac{4(1-\alpha)p}{(p-1)^2},
\end{aligned}$$

There are two possibilities:

$$\beta \geq \frac{1}{2}\left(\alpha + \sqrt{\alpha^2 - \frac{16(1-\alpha)p}{(p-1)^2}}\ \right) \text{ or } \beta \leq \frac{1}{2}\left(\alpha - \sqrt{\alpha^2 - \frac{16(1-\alpha)p}{(p-1)^2}}\ \right)$$

Subcase (a).

$$\begin{aligned}
\beta &\geq \frac{1}{2}\left(\alpha + \sqrt{\alpha^2 - \frac{16(1-\alpha)p}{(p-1)^2}}\ \right) \\
&> \frac{1}{4}\left(1 + \sqrt{1 - \frac{32p}{(p-1)^2}}\ \right) \qquad \text{since } \alpha \geq 1/2 \\
&> 1/3 \qquad \text{since } p \geq 38
\end{aligned}$$

as desired.

Subcase (b).

$$\begin{aligned} \beta &\leq \frac{1}{2}\left(\alpha - \sqrt{\alpha^2 - \frac{16(1-\alpha)p}{(p-1)^2}}\ \right) \\ &\leq \frac{8(1-\alpha)p}{\alpha(p-1)^2} \end{aligned}$$

Thus, we can choose a subset F of some of the connected components whose union $\cup F$ has size $xn = |\cup F|$ satisfying

$$\frac{\alpha}{2} - \frac{4(1-\alpha)p}{\alpha(p-1)^2} \leq x < \frac{\alpha}{2} + \frac{4(1-\alpha)p}{\alpha(p-1)^2} \tag{3.4}$$

Now we apply the discrepancy inequality (3.3) again by choosing $X = \cup F$ and $Y = \phi^{-1}(l) \setminus X$. We have

$$\begin{aligned} (p+1)^2 x(\alpha - x) &\leq 4p(1-x)(1-\alpha+x) \\ x(\alpha - x) &\leq \frac{4(1-\alpha)p}{(p-1)^2}. \end{aligned}$$

However, it is easily checked that because of (3.4) this is not possible for $\alpha \geq 1/2$ and $p \geq 34$. Hence, subcase (b) cannot occur. This proves the claim.

Now we are ready to prove Theorem 7.1. We will show that when $p \geq 38$, an element of the majority label can always be identified after all the queries are answered. Suppose we have an arbitrary blue/red coloring of the edges of $H = X^{p,q}$ with $p \geq 38$, and ϕ is a valid assignment on $V(H) = V(X^{p,q})$. Consider the connected components formed by the blue edges of $X^{p,q}$. By the Claim there is at least one blue component of size at least $\frac{1}{3}n$ since $p \geq 38$. Call any such blue component *large*.

If there is only one large component then we are done, i.e., every element in it must be of the majority label. Since $p \geq 38$, there cannot be three large blue components. So the only remaining case is that we have exactly two large blue components, say S_1 and S_2. Again, if either $S_1 \subseteq \phi^{-1}(l)$ or $S_2 \subseteq \phi^{-1}(l)$ is forced, then we are done. So we can assume there is a valid assignment ϕ_1 with $S_1 \subseteq \phi_1^{-1}(l)$, $S_2 \subseteq \phi_1^{-1}(\neg l)$, and a valid assignment ϕ_2 with $S_2 \subseteq \phi_2^{-1}(l), S_1 \subseteq \phi_2^{-1}(\neg l)$ (where $\neg l$ denotes any label in the label set except l).

Let us write $S_i' = \phi_i^{-1}(l) \setminus S_i, i = 1, 2$. Clearly we must have $A := S_1' \cap S_2' \neq \emptyset$. Also note that $|A| \leq n - |S_1| - |S_2| < \frac{16p}{(p-1)^2} n$.

Define $B_1 = S_1' \setminus A, B_2 = S_2' \setminus A$. Observe that there can be no edge between A and $S_1 \cup S_2 \cup B_1 \cup B_2$. Now we are going to use (3.3) again, this time choosing $X = A, Y = S_1 \cup S_2 \cup B_1 \cup B_2$. Note that

$$n > |Y| = |\phi_1^{-1}(l)| - |A| + |\phi_2^{-1}(l)| - |A| > n - 2|A|.$$

Since $e(X, Y) = 0$, we have by (3.3),

$$\begin{aligned} (p+1)^2 |X|\ |Y| &\leq 4p(n - |X|)(n - |Y|), \\ (p+1)^2 |A|(n - 2|A|) &\leq 4p(n - |A|)2|A|. \end{aligned}$$

However, this implies

$$\begin{aligned} (p+1)^2(n - 2|A|) &\leq 8p(n - |A|), \\ \text{i.e.,} \quad n((p+1)^2 - 8p) &\leq 2|A|((p+1)^2 - 4p) \\ &\leq 2|A|(p-1)^2 \\ &< 32pn \\ (p+1)^2 - 8p &< 32p \end{aligned}$$

which is impossible for $p \geq 38$.

Setting $p = 41$ (so that $X^{p,q} = X^{41,q}$ is regular of degree $p + 1 = 42$), we see that $X^{41,q}$ has $(1 + o(1))21n$ edges. This shows that Theorem 7.1 holds when $n = \frac{1}{2}q(q^2 - 1)$ for a prime $q \equiv 1 (\text{mod } 4)$.

If $\frac{1}{2}q_i(q_i^2 - 1) < n < \frac{1}{2}q_{i+1}(q_{i+1}^2 - 1) = n'$ where q_i and q_{i+1} are consecutive primes of the form $1(\text{mod } 4)$, we can simply augment our initial set $V(H)$ to a slightly larger set $V'(H)$ of size n' by adding $n' - n = \delta(n)$ additional elements of the majority label. Standard results from number theory show that $\delta(n) = o(n^{3/5})$, for example. Since the Ramanujan graph query strategy of $\mathbf{Q}$ actually identifies $\Omega(n')$ elements of the majority label l from $V'(H)$ (for fixed p) then it certainly identifies an element of the majority label of our original set $V(H)$.

This proves Theorem 7.1 for all n. □

Remark 3.1. *Instead of using Ramanujan graphs, we can consider random graphs $G(n, p)$ on n vertices and the probability of each pair to be chosen as an edge is p. Because random graphs with best achievable spectral gap are known to exist, the same proof using discrepancy inequalities still works. For the claims to hold, the degree for such a random graph only has to be greater than or equal to 39. So, we can at least reduce the constant* 21 *to* 19.5 *if we do not require explicit constructions.*

3.3 Acknowledgement

This chapter contains material appearing in "Finding favorites" from *Electronic Colloquium on Computational Complexity(ECCC) (078) 2003* and "Oblivious and Adaptive Strategies for the Majority and Plurality Problems" from the *11th International Computing and Combinatorics Conference (COCOON) 2005: 329-338*, and material to appear in *Algorithmica*. I would like to thank my co-authors Fan Chung, Ron Graham, and Andrew Chi-Chih Yao.

4

From Majority to Plurality

4.1 A Natural Extension of Majority

The Plurality game is a natural generalization of the Majority game where **Q** wants to identify one element of the *plurality* label (most frequently occurring) or show that there is no dominant label, still using only pairwise equal/unequal label comparisons of elements. When there are only $k = 2$ possible labels, the *Plurality problem* degenerates to the *Majority problem* of binary labels, and hence there are tight bounds for both adaptive and oblivious strategies (see Table 2.2).

In this chapter, we use an equivalent notion for the majority/plurality, which has been used extensively in the literature. This is the colored-ball setting, where we are given a set of n balls, each of which is colored in one of $k \in Z^+$ possible colors $\phi = \{c_1, c_2, ..., c_k\}$. We can choose any two balls a and b and ask questions of the form: "Do a and b have the same color?". Our goal is to identify a ball of the majority color (plurality color, respectively) or determine there is no majority color (plurality color, respectively), using a minimum possible number of questions.

4.2 Optimal Winning Strategies

Let k denote the number of permissible labels. Similar to Table 2.1, the following notations will be used throughout this dissertation:

Table 4.1: Notations for minimum length of **Q**'s winning strategies for the Plurality Game

Plurality game of n elements		
$PA_k(n)$	Adaptive setting	k possible different labels
$PA_*(n)$	Adaptive setting	k unknown, can be arbitrary
$PO_k(n)$	Oblivious setting	k possible different labels
$PO_*(n)$	Oblivious setting	k unknown, can be arbitrary

The current best bounds for minimum length of **Q**'s winning strategies for the plurality game are listed in Table 4.2, in which the ones in bold are our contributions.

Table 4.2: Current Best Bounds for the Plurality Game

Plurality Game ($k \geq 3$)	upper	lower
PA_k	$(\mathbf{k - \frac{1}{k} - 1})\mathbf{n}$ *upper*[1]	$(k-1)(n-k)/2$ *lower*
PO_k	$(\mathbf{1 - \frac{1}{k} + \epsilon})\binom{\mathbf{n}}{\mathbf{2}}$ *upper*[2]	$(\frac{\mathbf{1}}{\mathbf{6}} - \mathbf{o(1)})\mathbf{n^2}$ *lower*
PA_*	trivial	
PO_*	$\binom{n}{2}$	

[1]For $k \geq 9$, recently improved to $((0.775k + 3.6)n + O(k^2))$ in [6]
[2]Non-constructive
[3]Bounds for $PO_k(n)$ remain quadratic even when existence is known

We have designed strategies that give the current best upper bounds for PA_k and PA_* in general. Using probabilistic arguments, we have also been able to prove the current best lower bound for PO_*. In the literature, much attention has also been given to designing adaptive strategies for special cases of fixed or unknown k (see [2] [6] [32] [44] [63]), deterministic or randomized. Recently, a few papers have focused on average-case analysis for the plurality game in the adaptive setting [7] [8].

Lower bound

In general, it seems clear that the *k-color Plurality problem* should take more queries than the corresponding *Majority problem*. But exactly how much more difficult it is compared with the *Majority problem* was not so clear to us at the beginning. Similar arguments using concentration inequalities in random graphs seemed possible for achieving a linear upper bound. Here we will prove the contrary by establishing a quadratic lower bound, even for the case when $k = 3$. Also note that this lower bound is quadratic even when the existence of a plurality color is known a priori. Intuitively, this is because the existence of a majority color gives us much more information than the existence of a plurality color.

Theorem 4.1. *For the Plurality problem with $k = 3$ colors, the number of queries needed for any oblivious strategy satisfies*

$$PO_3(n) > \left(\frac{1}{6} - o(1)\right) n^2$$

Proof. Consider any query graph G with n vertices and at most $\frac{n^2}{6} - \frac{3}{2}n$ edges. Therefore there must exist a vertex v with $deg(v) \leq n/3 - 3$. Denote the neighborhood of v by $N(v)$ (which consists of all vertices adjacent to v in G), and the remaining graph by $H = G \setminus (N(v) \cup \{v\})$. Hence, H has at least $2n/3+3$ vertices.

Now split H into three parts H_1, H_2, and X where $|H_1| = |H_2|$ and $|X| \leq 1$. Assign color 1 to all vertices in H_1, color 2 to all vertices in H_2, color 3 to all vertices in $N(v)$ and X, and color 1 or 2 to v. Note that based on either one of the two possible color assignments, all query answers are forced.

Since color 3 cannot possibly be the dominant color, we see that whether color 1 or color 2 is the dominant color solely depends on the color of v, which the Questioner cannot deduce from the query answers.

This proves the lower bound to the number of queries needed for the oblivious strategy is $\left(\frac{1}{6} - o(1)\right) n^2$. □

This quadratic lower bound also applies to all $k \geq 3$ colors for the *Plurality problem* using oblivious strategies, since we don't need to use any additional colors beyond 3 for this argument.

Notice that when existence of a plurality color is not known a priori, the better quadratic bound $(\frac{1}{4} - o(1))n^2$ in Theorem 2.2 also applies here.

Upper bound

A trivial upper bound is the maximum number of possible queries we can ask, which is $\binom{n}{2}$. In this section we will show that for fixed k colors and n sufficiently large, essentially $(1 - 1/k)\binom{n}{2}$ queries suffice for oblivious strategies, using probabilistic arguments. Again we use the equivalent element/label setting as specified in Theorem 7.1 to avoid confusion between edge coloring and vertex coloring.

Let us consider the usual random graph $G(n, p)$ where n is the number of vertices and p is the probability for any particular edge to be included in the graph. We also use the standard notion "almost surely" to denote "probability going to 1 as

$n \to \infty$". Our upper bound is based on the following lemma.

Lemma 4.1. *If $p \geq 1 - 1/k + \epsilon$ for some $\epsilon > 0$, a random graph $G \in G(n,p)$ almost surely has the property that for any subset S of vertices of size at least n/k, the graph $G(n,p)[S]$ induced by S is connected.*

Proof. Consider $G \in G(n, 1 - 1/k + \epsilon)$ where $\epsilon > 0$. The expected vertex degree is $np = (1 - 1/k + \epsilon)n$. With concentration inequalities (see Theorem 4 of [26]), almost surely the degree for any vertex is lower bounded by $(1-1/k+\epsilon/2)n$. Given any subset of vertices S with size $|S| \geq n/k$ and any vertex v in S, by degree concentration v must be adjacent to at least $\frac{\epsilon}{2}n$ other vertices in S. Therefore there is no connected component of $G(n,p)[S]$ with order smaller than $\frac{\epsilon}{2}n$.

Now consider $G \in G(n,p)$ with $p > 0$ fixed. Then almost surely for any two disjoint vertex sets T and U in G such that $|T|, |U| = m = \Omega(\log n)$, there is an edge joining a vertex in T to a vertex in U. This is because the probability that this fails to hold is upper bounded by

$$n^{2m}(1-p)^{m^2} = e^{m(2\log n + m\log(1-p))}$$

which goes to 0 when $n \to \infty$ (because $m = \Omega(\log n)$).

Therefore G almost surely has the property that for any S with size at least n/k, the components in $G(n,p)[S]$ are so large (i.e., of size $\frac{\epsilon}{2}n$) that they all have to be connected. This proves the lemma. □

An oblivious strategy for **Q** can then be specified by a random graph G with this property. Regardless of how **A** colors the query edges, for each potential dominant label (i.e., it has occurred for at least n/k times), **A** cannot avoid forming an induced blue component of every vertex with that label and the calculation for **Q** to figure out the correct answer is the following:

Q's calculation

- Remove all the red edges from H to obtain H' and then remove all isolated vertices from H';
- Identify all connected components in H' with size at least n/k;
- If one such component has size strictly bigger than any other component, all elements in it must have the dominant label; Otherwise, there is no dominant label.

Hence the following upper bound holds for **Q**'s oblivious strategies. Note that the previous probabilistic arguments need n to be sufficiently large. In particular, the threshold value is a function of ϵ, say $n_0(\epsilon)$.

Theorem 4.2. *For every* $\epsilon > 0$

$$PO_k(n) < (1 - 1/k + \epsilon)\binom{n}{2}$$

provided $n > n_0(\epsilon)$.

4.2.1 Adaptive strategies for the Plurality problem

Aigner et al. [2] showed linear bounds for adaptive strategies for the *Plurality problem* with $k = 3$ colors. In this section, we first note a linear upper bound for general k in this case, and then strengthen it using a generalized argument.

Theorem 4.3. *For the Plurality problem with k colors where* $k \in Z^+$*, the minimum number of queries needed for any adaptive strategy satisfies*

$$PA_k(n) \leq (k-1)n - \frac{k(k-1)}{2}$$

Proof. There are k possible colors for the given n balls. We will use k buckets, each for a different possible color. All buckets are empty initially. The first ball s_1 is put in the first bucket b_1. The second ball is compared against a ball from b_1; if they have the same color, it is put in b_1, otherwise, it is put in a new bucket b_2. Similarly,

the i^{th} ball has to be compared against a ball from every non-empty bucket (at most $(i-1) \leq k-1$ many of them). Therefore the number of comparisons is no more than

$$1 + 2 + ... + (k-1) + (k-1)(n-k) = (k-1)n - \frac{k(k-1)}{2}$$

. □

In [2], it was proved that $PA_3(n) \leq \frac{5}{3}n - 2$. We will now give a generalized proof for all $k \geq 3$ in this setting.

Theorem 4.4. *For the Plurality problem with k colors where $k \in Z^+$, the minimum number of queries needed for any adaptive strategy*

$$PA_k(n) \leq (k - \frac{1}{k} - 1)n - 2$$

Proof. Let us denote the comparison of ball a against b by $(a : b)$, and define a *color class* to be a set of balls having the same color. There are two phases in this game. Given n balls $\{s_1, s_2, ..., s_n\}$, in Phase I the Questioner handles one ball at a time (except for the first query) and keeps a state vector v_i after ball s_i is handled. Each v_i is simply the list of color class cardinalities, in non-increasing order, $(a_{i1}, a_{i2}, ..., a_{ik})$ where $a_{i1} \geq a_{i2} ... \geq a_{ik}$. The Questioner also keeps a representative ball for each of the largest $(k-1)$ color classes (if they are nonempty) for comparisons and updates this list accordingly whenever there is a change in the state vector.

Claim: At every state, the Questioner has a strategy such that the total number t_i of comparisons up to v_i (inclusive) satisfies

$$t_i \leq (k-1)a_{i1} + (k-2)\sum_{j=2}^{k-1} a_{ij} + (k-1)a_{ik} - 2$$

Proof. We proceed by induction. After the first comparison, $v_2 = (1, 1, 0, ...)$ or $(2, 0, ...)$, so $t_2 = 1 \leq (k-1) + (k-2) - 2 \leq 2(k-1) - 2$ because $k \geq 3$.

For $2 \leq i \leq n$, let $v_i = (a_{i1}, a_{i2}, ..., a_{ik})$ be the state vector and $\{A_{i1}, A_{i2}, ..., A_{i(k-1)}\}$ be the set of corresponding representative balls (some may be null if the color class has cardinality 0). Now, ball s_{i+1} is to be handled as follows:

1. If $a_{i(k-1)} \neq a_{ik}$, we will compare s_{i+1} with the representative balls in the following order:

   $$(s_{i+1} : A_{i2}), (s_{i+1} : A_{i3}), ..., (s_{i+1} : A_{i(k-1)}), (s_{i+1} : A_{i1})$$

   with a total number of no more than $(k-1)$ comparisons. Note whenever the Adversary answers *Yes*, we know to which color class s_{i+1} belongs, and hence, we can skip the remaining comparisons.

2. Otherwise, compare s_{i+1} with the representative balls in the following order:

   $$(s_{i+1} : A_{i1}), (s_{i+1} : A_{i2}), ..., (s_{i+1} : A_{i(k-2)})$$

   with a total number of no more than $(k-2)$ comparisons. If all these $(k-2)$ answers are *No*, we can increment $a_{i(k-1)}$ and set representative ball $A_{i(k-1)} := s_{i+1}$.

After identifying to which color class s_{i+1} belongs, only one of the numbers in v_i gets incremented by 1 and possibly moved forward, to maintain the non-increasing order in v_{i+1}. Using the above strategy, we can ensure that no more than $(k-2)$ comparisons have been used in this round unless a_{i1} or a_{ik} gets incremented, in which case, their positions in the list do not change. Therefore, by the inductive hypothesis, we have

$$t_{i+1} \leq (k-1)a_{(i+1)1} + (k-2)\sum_{j=2}^{k-1} a_{(i+1)j} + (k-1)a_{(i+1)k} - 2$$

This proves the claim.

At state v_i, let r_i be the number of the remaining balls that have not been involved in any queries. Phase I ends when one of the following happens:

(A) $a_{i1} = a_{i2} = ... = a_{ik}$

(B) $r_i = a_{i1} - a_{i2} - 1$

(C) $r_i = a_{i1} - a_{i2}$

Note that one of (A), (B), (C) will eventually occur.) To prove the theorem, we use induction where the cases for $n \leq 3$ are easy to verify. More comparisons may be needed in Phase II depending on in which case Phase I ends. If Phase I ends in case (A), we use the induction hypothesis; in case (B), no more comparisons are needed because A_{i1} is a Plurality ball; in case (C), we need no more than r_i more comparisons to identify A_{i1} or A_{i2} as a Plurality ball. In all cases, we can show (using the claim) with arguments similar to those in [2] that

$$PA_k(n) \leq (k-1)n - n/k - 2 = (k - \frac{1}{k} - 1)n - 2$$

This proves the theorem. □

4.3 Acknowledgement

This chapter contains material appearing in "Oblivious and Adaptive Strategies for the Majority and Plurality Problems" from the *11th International Computing and Combinatorics Conference (COCOON) 2005: 329-338*, and material to appear in *Algorithmica*. I would like to thank my co-authors Fan Chung, Ron Graham, and Andrew Chi-Chih Yao.

5

Majority Game with Liars

5.1 Error-tolerance and Rényi-Ulam's Liar Game

In the current literature, **A** is always a *malevolent but truthful* adversary in the sense that his/her answers must be *consistent* with all previously given answers. However, an *error-tolerant* feature is desired when the answers to the queries in the application may be faulty due to communication errors, for example. In this paper we address this issue by putting the *Majority* game in a broader context of fault-tolerant communication, namely, searching games with errors. One such famous game is the *Rényi-Ulam liar game*. For a comprehensive overview of this topic, we refer the readers to a recent survey [52]. The Rényi-Ulam liar game is closely related to fault-tolerant communication and error-correcting codes. The basic form of this game is as follows.

There are two players: a *chooser* (also called Carol) and a *partitioner* (also called Paul). A game is defined by three nonnegative integers N, k and q that are known to both players. Carol is assumed to have selected a secret number x from the set $\{1, ..., N\}$. Paul's goal is to find out what this number is by asking Carol questions

of the form "Is x in S?", where S is any subset of $\{1, ..., N\}$. Carol is required to answer either "yes" or "no". However she is allowed to lie up to k times. We say that Paul wins the (N, k, q) game if and only if he can always identify Carol's secret number after at most q questions, regardless of Carol's strategy.

The many variants of the Rényi-Ulam game have been extensively studied over the past 50 years, and the term "Rényi-Ulam games" are used to denote all searching games with errors [52]. There is some notable similarity between the Majority/Plurality game and the Rényi-Ulam game. However, we do not yet know whether it is possible to extend the techniques used for analyzing the Rényi-Ulam game to gain more understanding for the Majority/Plurality game in the presence of faulty answers. We take our first step in the following section to analyze the binary-label Majority game with one lie.

5.2 Majority Game with Liars

We consider bounded error tolerance for the *Majority* game, where **A** is allowed to *lie up to a fixed number t times.* More precisely, this means that after **Q** provides his final identification, **A** has the freedom to flip up to t answers of the previously asked queries and reveal a labelling that is consistent with this modified set of answers. Now that **A** can lie how much will this handicap **Q**? What is the new minimal q^* and what strategy should **Q** adopt to achieve this bound?

In this chapter we will begin to answer some of these questions for the *Majority* game with binary labels. We will give upper and lower bounds for q^* by producing strategies for **Q** and **A** in various versions of the game. We summarize our results in the table below, where t is the number of lies and n is the number of objects. We only consider the case of binary labels (i.e., $k = 2$).

Table 5.1: Current Best Bounds for the Majority Game with Liar (up to t lies, binary labels)

	n odd	n even
adaptive, $t = 1$	$(n+1)$ *upper*	$(n+2)$ *upper*
	n *lower*	$(n+1)$ *lower*
adaptive, $t > 1$	$\left(\frac{t+1}{2}n + 6t^2 + 2t + 3\log n\right)$ *upper*	
	$\lceil \frac{t+3}{4}n - \frac{t+1}{4} \rceil$ *lower*	$\left(\frac{t+1}{2}n\right)$ *lower*
oblivious, $t \geq 1$	$\lceil (t + \frac{1}{2})n \rceil$	

We note that different application contexts may imply different interpretations of these bounds. For example, the upper bound for the adaptive case with $t > 1$ holds for $t = o(n^{1/2})$. When t is large compared to n then the oblivious bound will give a better result. We can also consider the case when t is upper bounded by a fixed percentage of n. In this case, the upper bound for the adaptive case with $t > 1$ is asymptotically better than the oblivious bound and the general bound given in Theorem 5.1 when $\alpha < 1/12$.

Also note that there is a difference between the case when n is odd and n is even. This is due to the fact that when n is odd there *must* be a majority element which gives additional information **Q** can use in forming a strategy.

5.3 Adaptive Strategies

All previously discussed settings of the Majority game depict **A** a *malevolent but truthful* adversary in the sense that his/her answers must be *consistent* with all

previously given answers. In this section, we consider the most limited case of error tolerance for the *Majority* game, where **A** is allowed to *lie once*. More precisely, this means that after **Q** provides his final identification, **A** has the freedom to flip zero or one answer of the previously asked queries and reveal a labelling that is consistent with this modified set of answers. Does **A** gain by lying? If yes, how much power does **A** gain by lying?

If **A** is allowed to lie for a fixed number t times, **Q** has a simple winning strategy of length at most $(2t+1)(n-\mu_2(n))$ in the *Majority* game of binary labels. This bound already tells that only lying for no more than a fixed number of times does not give **A** too much power since the bound is still linear. When **A** is truthful (i.e., no lie is allowed), the coloring of G is always *valid*. When **A** is allowed to lie, this is no longer the case. In fact, *validity* checking can help us identify the lies. The following upper bound is based on *validity checking* through cycles.

There is a simple linear upper bound for any fixed number of lies allowed by **A**.

Theorem 5.1. *In the adaptive Majority game on n elements with binary labels and at most t lies,*

$$q^* \leq (t+1)(n-\mu_2(n)) + t.$$

Proof. Let **Q** ask the same queries as in the *Majority* game with no lies allowed, only that each query is repeated until $(t+1)$ answers agree before going to the next query. Because **A** is not allowed to lie more than t times, the total number of queries **Q** needs to ask is $(t+1)\big(n-\mu_2(n)\big)$ plus at most t. □

This simple linear bound already tells us that lying for no more than a fixed number of times does not give **A** too much power. In fact, we will show that the coefficient of n can be substantially strengthened if **Q** uses a smarter way of *validity checking*.

5.3.1 Majority Game with at most $t = 1$ lie

Upper bound

Theorem 5.1 gives a $2(n - \mu_2(n)) + 1$ upper bound for **Q**'s adaptive winning strategy when **A** is allowed to lie once during the game. The *oblivious* strategy presented in Section 5.4 supplies a better upper bound $\lceil \frac{3}{2}n \rceil$. This bound can be further improved by *validity checking* of the coloring in the auxiliary graph.

Recall Definition 2.3, the definition of a *valid* coloring of the auxiliary graph G. When **A** is truthful (i.e., no lie is allowed) the coloring of G is always *valid*. When **A** is allowed to lie, this may no longer be the case. In fact, *validity* checking can help us identify the lies. First we make the following observation concerning possible cycles in this auxiliary graph G.

Observation 5.1. *No valid coloring of a cycle can have an odd number of red edges.*

Proof. This observation follows by noting that the number of red edges corresponds to the number of times that the path crosses between V_1 and V_2, which for a closed cycle must be even. From this observation we will say that a cycle is invalid if it contains an odd number of red edges. It is easy to see that a cycle in the graph is invalid if and only if it contains an odd number of lies. □

Theorem 5.2. *In the adaptive Majority game on n elements with binary labels and at most 1 lie*

$$q^* \leq \begin{cases} n+1 & n \text{ odd,} \\ n+2 & n \text{ even.} \end{cases}$$

Proof. **Q**'s strategy consists of two stages.

Stage 1:

- **Q** asks queries so that the *active* auxiliary graph consists of only paths and isolated vertices.
- Initially G consists of n *active* isolated vertices. **Q**'s next query is always connecting two (previously separated) *active* connected components C_1 and C_2 where $\delta(C_1) = \delta(C_2)$ UNTIL **Q** gets a *red* edge or there are no components left to join.
- If the newly formed path has a *red* edge in the middle, **Q** queries the two end nodes of the path to form a cycle. If this new query is also colored *red*, **Q** can be sure that there is no lie in this cycle, and it can be marked *inactive* since it has an equal number of each label and **Q** continues in Stage 1. Otherwise, go to Stage 2.
- If all *active* components are of different δ value, let C be the component of largest size. **Q** asks a query to connect C's two end nodes to form a cycle. If this new query is colored *red*, go to Stage 2. Otherwise, **Q** can conclude that any node in C is of the majority label.

Stage 2:

- There is only one active component containing one *red* edge (the cycle). Let this component be C and this *red* edge be $e(u, v)$. Denote all other *active* components by $G_a \setminus C$. As noted earlier we can ignore any inactive components since they contain an equal number of each label.
- **Q** asks one query per each active component besides C to connect it to u. We now have one active component left which consists of a cycle with a single red edge with a tree connected to one vertex of the red edge. To get to this stage **Q** will have asked exactly n queries.

- Since the lie is in the cycle the edges of the tree are honest answers, in particular we can determine how many vertices of C need to have the same label as u in order for u to have a majority label. Along the cycle C, starting at u, count (away from v) exactly that many steps and stop at vertex w. **Q** then queries u and w. If the new query is colored *blue*, **Q** can conclude that u is of majority label.

- If the new query is colored *red* and n is odd (i.e., no possibility of a tie), then it must be that v is of the majority label. When n is even, one more query in C is needed to deal with the possibility of a *tie*.

In the worst case, this strategy will use no more than $(n+1)$ queries for odd n and no more than $(n+2)$ queries for even n.

Lower Bound

A has at least as much power as he/she does in the *Majority* game with no lies, so $n-\mu_2(n)$ serves as a lower bound for the *Majority* game with one lie allowed. In this section, we will show that this bound can be further strengthened to n when n is odd and $(n+1)$ when n is even. Thus the upper and lower bounds differ exactly by 1 in both cases.

Theorem 5.3. *In the adaptive Majority game on n elements with binary labels and at most 1 lie*

$$q^* \geq \begin{cases} n & n \text{ odd,} \\ n+1 & n \text{ even.} \end{cases}$$

Proof. The case for n odd will follow from Theorem 5.5 in Section 5.3.2 with $t=1$. Here we only consider the case for n is even, when this theorem improves the bound given in Theorem 5.5.

The strategy for **A** will be to answer the first $n-1$ questions consistently in such a way that the δ value for an even component is 0 and for an odd component is 1 (that this is always possible is an easy exercise left to the reader). For the nth question, if the resulting query would result in the graph consisting of disjoint even cycles then **A** lies, otherwise **A** answers as before. After the nth question, **A** fixes a possible labelling consistent with the given answers and is truthful afterwards.

It suffices to show that after the nth question that **Q** cannot distinguish between the existence of a majority and a tie.

Suppose **A** did not have to lie at the nth step. Since all of the δ values are either 0 (for even components) or 1 (for odd components, of which there must be an even number), **A** can always produce a labelling that is consistent with the given answers and is a tie between the two labels. We now show that **A** can reverse some portion of this labelling to produce a majority element.

If the graph after the nth question did not consist solely of cycles then there is a vertex of degree ≤ 1 for which **A** can reverse the label (since **A** has a lie available) and thus produce a labelling with a majority element. Similarly, if the graph after the nth question solely consists of cycles, some of them odd, then by reversing the labelling on an odd cycle **A** can produce a majority element.

Suppose now that **A** lied at the nth step, i.e., the graph after the nth step consists of even cycles only. From **Q**'s point of view, if **A** had lied at the nth step then any possible valid labelling consistent with all previously given answers contains a tie; on the other hand if **A** had lied at the step corresponding to an adjacent edge with the nth step in the *invalid* cycle, then a possible valid labelling now has a majority element. □

5.3.2 Majority Game with at most $t \geq 2$ lies

Upper bound

We now turn to finding an upper bound for the majority game with at most $t \geq 2$ lies.

Theorem 5.4. *In the adaptive Majority game on n elements with binary labels and at most $t \geq 2$ lies,*

$$q^* \leq \frac{t+1}{2}n + 6t^2 + 2t + 1 + 2\log n = \left(\frac{t+1}{2} + o(1)\right)n.$$

Proof. We produce a two round strategy which will establish the bound. The first round will be "oblivious" in that we will always ask the same set of questions and not adapt our strategy (this round will use $(t+1)n/2$ questions). We then use the answers from the first round to find and correct all lies in $o(n)$ steps establishing the result.

Let t be fixed and first consider the case $n > 2t$ with n even.

Stage 1:

Q forms an n-cycle with the n vertices and asks $\lfloor t/2 \rfloor$ questions on each edge of the cycle. **Q** then makes

$$t + 1 - 2\left\lfloor \frac{t}{2} \right\rfloor = \begin{cases} 1 & \text{if } n \text{ even,} \\ 2 & \text{if } n \text{ odd,} \end{cases}$$

queries between opposite vertices of the cycles (we will refer to these queries as spokes). An example is shown in Figure 5.3.2.

Claim: If **A** has lied we can find an invalid cycle.

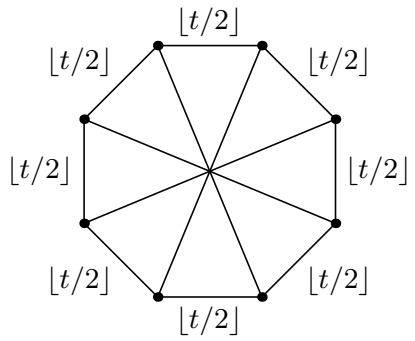

Figure 5.1: "Oblivious" first round queries.

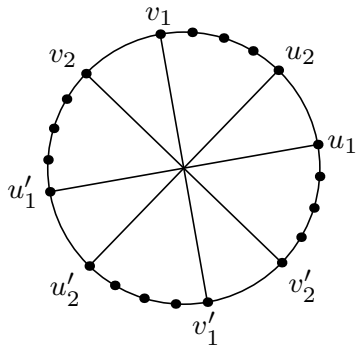

Figure 5.2: Looking for lies in the spokes.

To see this we proceed by considering cases.

- *There is a lie in the spokes.* Since we have assumed that $n > 2t$ then not all of the spokes can be lies. In particular, visiting the spokes in turn there will be two consecutive spokes (say, spokes $u_1u'_1$ and $u_2u'_2$) so that the first is not a lie and the second is a lie, then continuing we will find another set of spokes (say spokes $v_1v'_1$ and $v_2v'_2$) so that the first is a lie and the second is not a lie. (See Figure 5.3.2; note it might happen that $u_1u'_1$ ($u_2u'_2$) and $v_2v'_2$ ($v_1v'_1$) are the same spoke.)

  In order for all cycles of the form $u_1u_2u'_2u'_1$ to be valid then either all queries between u_1 and u_2 were lies or all queries between u'_1 and u'_2 were lies. Similarly in order for all cycles of the form $v_1v_2v'_2v'_1$ to be valid then either all queries between v_1 and v_2 were lies or all queries between v'_1 and v'_2 were lies.

In the case that t is even we would need at least $2(t/2)+1$ lies to validate the cycles which is impossible. In the case that t is odd we would need at least $2\lfloor t/2 \rfloor + 1 = t$ lies, and in particular, there could be at most one lie used in the spokes. However recall that for t odd each spoke is asked twice and in this case there must be a two-cycle of the form u_2u_2' which is invalid and so we can still find an invalid cycle.

- *No lies in the spokes but some set of opposite intervals on the cycles are not fully saturated with lies.* In this case we can find some cycle of the form $u_1u_2u_2'u_1'$ which is invalid.

- *No lies in the spokes and lies only occur in opposite intervals which are fully saturated.* We first point out that it must be the case that only one pair of opposite intervals are fully saturated (i.e., not enough lies to do this with more than one pair). In this case we can find a cycle of length $n/2$ which is invalid, namely for any spoke u_1u_1', we form an $n/2$ cycle by going over the spoke and then use edges along the original cycle to close our new cycle.

This concludes the proof of the claim.

Stage 2:

We now examine all of the two-cycles and four-cycles of our answered queries. If we find an invalid cycle we can correct it using at most $6t$ additional queries (i.e., make $2t$ queries on each of 3 edges, and take majority answer on each edge; if no lie is found among the three edges then the fourth edge was the lie). Since there are t lies available we need no more than $6t^2$ queries so that all two-cycles and four-cycles are valid.

We now check to see if there is an invalid cycle of length $n/2$ as described in the claim above. If there is then there is only at most one lie available for **A** to

use and so we can use a divide and conquer technique on the cycle of length $n/2$ as follows. By joining two opposing pairs of vertices we query until we get two answers which agree, this effectively splits the cycle in half. We then test which half has an invalid coloring. The lie must be in that half and we continue the process. In particular in at most $2\log n + 1$ steps we can find the error in the cycle of length $n/2$. Translating back this means we can find the opposing intervals which are fully saturated with lies and then correct them.

Q is now finished because he can remove all lies given by **A** and relate all elements together. In particular, **Q** has used at most $(t+1)n/2 + 6t^2 + 2\log n + 1$ queries to accomplish this.

For the case n odd we set aside a single element and run the procedure and then at the end connect the element back into the graph by making at most $2t+1$ queries relating the odd element out with some arbitrary element.

Finally for the case $n \leq 2t$ we can simply build a tree where we keep asking questions on each edge until we get $t+1$ responses which agree. In particular, we would need at most $2t^2 + t$ queries in such a case.

Putting this all together gives the desired result. □

Lower bound

When we only have binary labels, a majority label does not necessarily exist when n is even because there may be a tie. If up to t lies are allowed, we make the following observation:

Observation 5.2. *If the coloring is valid (i.e., no lies are detected), then* **Q** *will not be able to determine the correct relationship for an element which is involved in no more than* t *queries.*

Proof. This observation follows by noting that since the coloring is valid and **A** is allowed to change the color of up to t edges, then **A** can change all the queries involved with a vertex of low degree (i.e., no more than t) and still produce an admissible labelling. □

A lower bound for general fixed t when n is even follows:

Theorem 5.5. *In the adaptive majority game on n elements with binary labels and at most $t \geq 2$ lies,*

$$q^* \geq \begin{cases} \dfrac{t+1}{2}n & n \text{ even,} \\ \left\lceil \dfrac{t+3}{4}n - \dfrac{t+1}{4} \right\rceil & n \text{ odd.} \end{cases}$$

Proof. When n is even, **A**'s strategy is to label half of the elements 0 and the other half 1 and answer all of the questions truthfully. If **Q** asks fewer than $\lceil (\frac{t+1}{2})n \rceil$ many queries, by degree considerations there is a vertex with degree at most t. Based on Observation 5.2, **Q** cannot distinguish the case as to whether all of the edges connected to that vertex were lies or were all honest answers. In particular **Q** cannot determine the vertex labelling for any vertex with degree at most t and since **A** started in an exact balance, **Q** will not be able to determine the true status of the labelling.

For n odd, **A** will try to employ a similar strategy. But now that n cannot be evenly split, the strategy becomes more involved in trying to keep the labelling in balance and still forcing **Q** to make many queries.

As **A** answers the questions he will keep track of two graphs. The first is the auxiliary graph which we have been using before. The second is an underlying "connectedness" graph. The two graphs will always have the same vertex labelling, but in the connectedness graph we only include an edge if it connects two previously

disconnected components. Let W be a connected subset of the vertices of the connectedness graph, W_0 be the vertices of W labelled 0 and W_1 be the vertices of W labelled 1. As before $\delta(W) = \big||W_0| - |W_1|\big|$. We also will consider $D(W_0)$ and $D(W_1)$ which are the sum of the degrees of vertices of W labelled 0 and labelled 1, respectively.

A's Strategy

- **A** answers so that all even components have $\delta = 0$ and all odd components have $\delta = 1$. For any query involving only one component **A** answers consistently with the already given answers.
- For a query connecting an even component with an odd component, both answers are allowable (i.e., can be made consistent by perhaps a switching of the labelling on the even component). In such a case, **A** will answer so that if W is the newly formed component then $D(W_1) \geq D(W_0)$.
- For a query connecting two odd components or two even components, **A** will give an answer so that $\delta = 0$ for the newly formed component, then if $D(W_1) < D(W_0)$ will reverse the labelling on the newly formed component.

To verify the second statement we note that if the query involves at least one vertex with a label of 1 then it easily holds by **A** answering truthfully. So now suppose both vertices are labelled 0. If X is the even component and $D(X_0) = D(X_1)$ then reverse the labelling on the even component and answer truthfully. On the other hand if $D(X_1) \geq D(X_0) + 2$ then it is easy to check that **A** can answer truthfully without needing to switch the labelling. This covers all possible conditions since $D(X_1) \neq D(X_0) + 1$ (i.e., the sum of degrees would be odd which is impossible).

Next note that if there are three components for which $\delta = 1$ then it is impossible for **Q** to win. So if **Q** is in a position to win there must be exactly one odd component. We now will compute the minimum number of queries that **Q** needs to ask in order to be in a winning position.

If there is any vertex of degree at most t in an even component then it is easy to verify that **Q** cannot win (i.e., **A** can, if needed, switch the labelling of the component containing the vertex of degree at most t and/or change all of the queries involving that vertex to lies). Therefore if there are m vertices in the odd component, the sum of the degrees in the even components will be at least $(n-m)(t+1)$.

Now let us consider the odd component (denoted W). By the strategy employed by **A**, we know that $D(W_1) \geq D(W_0)$. On the other hand $D(W_1) + D(W_0) = 2m-2$. Combining these statements we have $D(W_1) \geq m-1$. Now let us consider the vertices labelled 0. It is easy to verify that if there are two vertices labelled 0 with degree at most t then **Q** cannot win. Therefore the sum of the degrees in the odd component of the auxiliary graph must be at least $(t+1)(m-1)/2+1+(m-1)$.

Thus the minimum number of questions that **Q** must ask (given that $n-m$ of the vertices are in even components) is:

$$\frac{1}{2}\big[(n-m)(t+1)+(t+1)\frac{m-1}{2}+1+(m-1)\big] = \frac{1}{2}\big[(t+1)n-\frac{t+1}{2}+m\left(\frac{1-t}{2}\right)\big].$$

Since this is minimized when $m = n$ (i.e., no even components), it follows that the fewest possible number of questions that **Q** needs is $(t+3)n/4-(t+1)/4$, giving the result. □

5.4 Oblivious Strategies

In the oblivious setting, **Q** has to specify all the edges in the auxiliary graph G before **A** colors any of them. This implies that **Q** has to accomplish *detection* and *location* of lies simultaneously. We have another important observation.

Observation 5.3. *In the Majority game of binary labels with at most t lies, if an edge e is part of 2t cycles that pairwise edge-intersect only at e (though they might*

share many vertices in common), then a lie is located at e if and only if at least $(t+1)$ of these cycles are invalid.

Proof. This observation follows by noting that if an edge corresponds to a truthful answer then there can be at most t of the $2t$ cycles intersecting at e which can be invalid (i.e., each invalid cycle would have to involve at least one lie and there is no other overlap). On the other hand if the edge corresponded to a lie then there could be at most $t-1$ of the $2t$ cycles intersecting at e which can be valid, or equivalently, at least $t+1$ invalid cycles (i.e., each valid cycle would have to involve at least one additional lie and there is no other overlap). □

Theorem 5.6. *In the oblivious Majority game on n elements with binary labels and at most $t \geq 1$ lies,*

$$q^* = \lceil (t + \frac{1}{2})n \rceil.$$

Proof. We first establish the upper bound. Observation 5.3 implies that if we can construct a query graph for $\mathbf{Q}$ such that for any particular edge we can find $2t$ cycles that are pairwise joined only at that edge, we can correct all possible lies with no more queries needed.

We handle the base cases first. For $n = 2$, we use $(2t+1)$ edges for the same query. For $n = 3$, the query graph is a triangle with one query asked for t times and the other two queries each asked for $(t+1)$ times.

For even $n \geq 4$, we construct a multigraph as shown in Figure 5.4 where all edges in the outer cycle are multiedges (repeated t times) and single edges (or spokes) connect each pair of opposite vertices. The total number of edges is therefore $(t + \frac{1}{2})n$. For odd $n \geq 3$, first construct a graph as in the $n+1$ case and then contract a set of edges on the outer cycle, an example is shown in Figure 5.4. In this case it can be checked that there are $\lceil (t + \frac{1}{2})n \rceil$ edges in the graph.

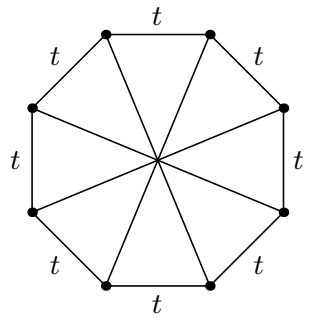

Figure 5.3: Oblivious graph, n even.

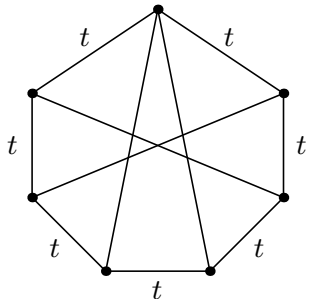

Figure 5.4: Oblivious graph, n odd.

For each spoke, we can find t cycles using one half of the outer cycle and another t using the other half. For each side edge e, first we can find $(t-1)$ small cycles by joining it with the other $(t-1)$ multiedges with the same endpoints, then we use edges in the outer cycle to obtain another $(t-1)$ cycles. We need two more cycles and these are constructed using the spokes and the remaining unused edges of the outer cycle as shown in Figure 5.4. Because each edge lies in at least $2t$ cycles pairwise joined only at that edge, all lies can be located and hence corrected, establishing the upper bound.

For the lower bound, we can use a similar argument of degree concentration as was used in the proof of Theorem 5.5. If **Q** asks fewer than $\lceil (t+\frac{1}{2})n \rceil$ then since **Q**'s strategy is oblivious, **A** can examine the entire auxiliary graph and find a vertex v with degree at most $2t$. **A** can split the remaining vertices into two sets U and V as equally as possible (i.e., $\big||U|-|V|\big| \leq 1$) with labels 0 and 1 respectively. For

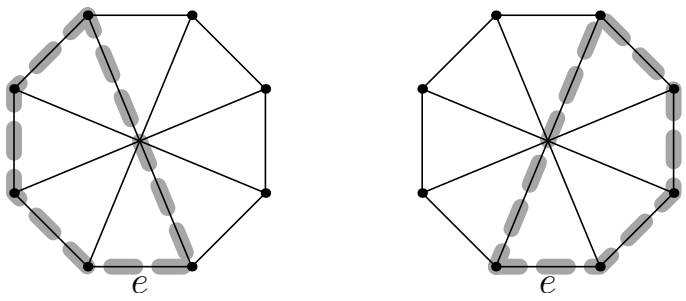

Figure 5.5: The remaining two-cycles for a side edge e.

queries not involving v, **A** answers truthfully. For queries involving v, **A** answers half of the queries as if v is labelled 0 and the other half as if v is labelled 1. This is possible because **A** is allowed to lie up to t times. Now **Q** cannot distinguish which half are lies and hence cannot determine the label for v which is essential because the other vertices are in an (almost) exact balance. This establishes the lower bound and concludes the proof. □

5.4.1 Discussion

Motivated by the practical need of an *error-tolerant* feature, we have concentrated on optimizing the questioner's strategy in the presence of lies (or errors) for binary labels in the *Majority* game. We point out that when the number of lies is upper bounded by a constant t, **Q** can still win the game with a linear number of questions. Upper and lower bounds on the length of **Q**'s optimal strategy were derived in both the adaptive setting and the oblivious setting.

Consideration of fault-tolerance may also be useful for the many other variants of the *Majority* game, such as when the number of different labels is more than two. A natural generalization of the Majority game is the *Plurality* game where **Q** wants to identify one element of the *plurality* label (most frequently occurring), still using only pairwise equal/unequal label comparisons of elements. Much attention has been given to designing adaptive strategies (deterministic or randomized) for fixed

or unknown k (see [2] [6] [32] [44] [63]). We remark here that the same reasoning of Theorem 5.1 applies to existing bounds for all variants of the *Majority* game (including the *Plurality* game) if the maximum number of lies allowed t is fixed. The new upper bounds will only be at most worse by a multiplicative constant $(t+1)$ and an additive constant t.

In this chapter, we gave a complete picture for the *oblivious* setting in the *Majority* game with a constant bounded number of lies. In the *adaptive* setting, however, there are still various gaps between the upper and lower bounds obtained. Proving better lower bounds for error-tolerant strategies in these games is crucial for evaluating our currently best query strategies. Closing these gaps would be an interesting direction to pursue. In the meantime, other types of error-tolerance may also be considered, such as bounded error fraction or random errors.

5.5 Acknowledgement

This chapter contains material submitted for publication. I would like to thank my co-authors Steven Butler and Ron Graham.

6

The 2BPS Problem

6.1 Motivation and Background

kBPS : Suppose we are given a fixed $k \in \mathbb{Z}^+$, an unlimited number of unit-size bins and a list of weights $W = \{w_1, w_2, w_3, \ldots, w_n\}$ ($w_i \in \mathbb{R}^+$) of n different types. Given that we can partition each w_i into any $p_i \in \mathbb{Z}^+$ pieces, we are asked to pack these weights into a minimum number of bins such that no bin contains weight parts of more than k different types.

Bin packing is one of the fundamental combinatorial optimization problems. The most elementary form asks to pack a given list of non-splittable weights into a minimum number of unit-size bins. It has many variants and applications. Although it is NP-hard, it has a PTAS[1]. We refer the readers to two recent surveys for a comprehensive overview [27] [28].

The above mentioned kBPS differs from traditional bin packing in two aspects.

[1]A Polynomial-Time Approximation Scheme is an algorithm which takes an instance of an optimization problem of size n and a parameter $\epsilon > 0$ and produces a solution of an optimization problems that is within ϵ factor of being optimal. The running time of a PTAS has to be polynomial in n but can depend arbitrarily on ϵ.

One is the k-cardinality constraint for each bin. The other is that the weights can be split arbitrarily before the actual packing. Cardinality constrained bin packing kBP first appeared in a 1975 paper by Krause et al. [45] [46]. kBP is the same as kBPS except that items are not allowed to be split. It was studied as a model for a machine scheduling problem with multiple processors and a fixed-size memory bank. More recent results with improved approximation ratios were given for both the offline and online settings [9] [41]. Although kBP is NP-hard in general, several known approximation algorithms achieve optimal packing for 2BP such as the First-Fit-Decreasing (FFD) algorithm.

Another related problem is CCBP , also for non-splittable items. CCBP problems arise in allocating resources of different types to a set of users. Consider a set of bins each having capacity v and c compartments. Given n items of unit size of M different types, the goal is to pack these items (non-splittable) into a minimum number of bins such that items of different types are packed in different compartments. PTAS have been designed for CCBP for both the offline and online settings [59] [60].

kBPS first arose in the context when memories need to be allocated efficiently to processors in pipelined router forwarding engines for faster IP lookup schemes [21]. Even the simplest variant 2BPS (i.e., $k = 2$) is NP-hard. This is in surprising contrast with kBP where optimal algorithms exist for $k = 2$. Because of the inherent flexibility to slice items arbitrarily in the kBPS problem, any positive item size is permissable. This is a vital feature for the manifold optimization problems in real applications such as resource allocation and job scheduling where each item can be sliced and then fit into the resource bins.

Since even the most elementary form of bin packing problem is NP-complete, it is not very surprising that the decisional version of kBPS even for $k = 2$ is NP-complete [21].

Besides the usual offline setting, two other more practical settings can also be considered. In the *online* setting, the items arrive one-by-one. A new item needs to be packed on the spot without knowledge about future items and previously packed items are not allowed to be moved or removed. In the *dynamic* setting, items also arrive one-by-one with the additional flexibility that they can also disappear at a later time.

NP-completeness of the decisional version of 2BPS is shown in [21] by Chung, Graham, and Varghese. The transformation is from the 3-PARTITION stated as follows.

3-PARTITION

Instance: A set A of $3m$ elements, a bound $B \in \mathbb{Z}^+$, and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = mB$.

Question: Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$ (note that each A_i must therefore contain exactly 3 elements from A)?

Garey and Johnson [38] showed the 3-PARTITION problem is NP-complete by using transformation from the problem of 3-dimensional matching. In fact, the 3-PARTITION problem has been shown to be NP-complete in the strong sense (see [38]).

Theorem 6.1. *[21] The decisional version of the* 2BPS *problem is NP-complete.*

Proof. See Appendix B.

6.2 Simple Greedy Algorithm $\mathcal{A}$

In the offline setting, a simple greedy algorithm $\mathcal{A}$ can be given, which resembles the classical *First-Fit* algorithm for bin packing. Based on Algorithm $\mathcal{A}$, we give another more sophisticated approximation algorithm $\mathcal{B}$ in the next section. Algorithm $\mathcal{B}$ has approximation ratio at least 3/2 but is optimal in the special case when the total weight is greater than or equal to the number of weight types n.

In our terminology, a *new bin* refers to an empty bin; a *live bin* refers to a partially filled bin with only one type, due to the 2-cardinality constraint. Given a list of weights, Algorithm $\mathcal{A}$ packs the weights in order and places the maximum possible part of current weight into a *live bin* if possible or else into a *new bin*. Algorithm $\mathcal{A}$ achieves an approximation ratio 3/2.

Algorithm $\mathcal{A}$ [21]

Given a list of weights $W = (w_1, w_2, \ldots, w_n)$,

- For each $i = 1, 2, \ldots, n$, we pack greedily as follows:
  - Place the maximum possible part of w_i into a *live bin* if possible;
  - Otherwise, put it into one or more *new bins*.

6.2.1 The packing graphs

For any legitimate[2] packing P produced for an instance of 2BPS we can define a graph $G(P)$ as follows: [21]

i. n vertices correspond to the n weights.

ii. m edges correspond to the m bins used.

iii. If a bin contains only one type of weight, its corresponding edge is a *loop*;

[2]Here *legitimate* means that the cardinality constraint is respected.

otherwise, we have an (ordinary) *edge*.

iv. If a bin is only partially filled, its corresponding edge or loop is *weak*, denoted by a dotted line or cycle.

v. We also distinguish between *cycles* and *loops* in that a *cycle* must have at least two vertices.

For instance, given a list of three weights $W = (\frac{1}{2}, \frac{2}{3}, \frac{1}{4})$, two valid packings and their corresponding graphs are illustrated in Figure 6.1.

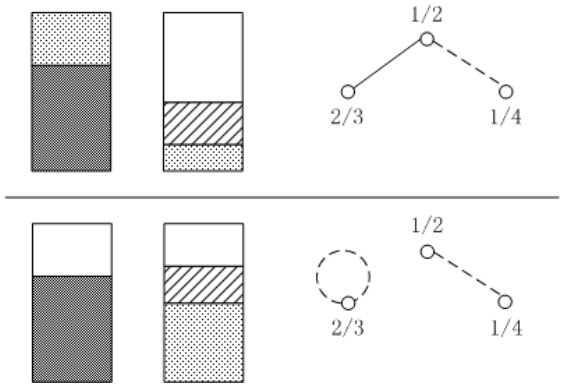

Figure 6.1: Two valid packings with corresponding graphs for a list of three weights $(2/3, 1/2, 1/4)$

After we have processed all n weights using Algorithm $\mathcal{A}$, the resulting bin packing is legitimate and satisfies the following properties:

i. Each connected component is a path with possibly some loops.

ii. There is altogether at most one weak loop (i.e., one *live bin*) which possibly appears at the end of the last connected component formed during the algorithm.

iii. Each connected component except for the last one has at most one weak edge which can only appear at the end of the component.

These properties are intuitive and easy to verify. Let $|\mathcal{OPT}|$ denote the number of bins used in an optimal packing, it has been shown that:

Theorem 6.2. *[21] Algorithm $\mathcal{A}$ always generates a bin packing which has size within a factor of 3/2 of the optimum asymptotically.*

Proof. The argument is very similar to Theorem 6.8. We omit it here.

It is worth noting that if the associated graph of the resulted packing does not have any weak loop, Algorithm $\mathcal{A}$ is exactly at most $\frac{3}{2}$ from optimal. It is also clear that Algorithm $\mathcal{A}$ runs in time $O(n)$, where n is the number of types.

6.2.2 Better Approximation Ratio for 2BPS for *large* weights

Given that all bins have uniform size 1, when the average weight has size less than 1, Algorithms $\mathcal{A}$ and $\mathcal{B}$ are equivalent. In Subsection 6.2.2 only, we will confine ourselves to the case where every w_i is of size no more than 1. Therefore all subsequent results in this subsection that hold for Algorithm $\mathcal{A}$ hold also for Algorithm $\mathcal{B}$. We will now proceed with Algorithm $\mathcal{A}$ without further mentioning of Algorithm $\mathcal{B}$.

We will show that Algorithm $\mathcal{A}$ has a much better performance guarantee when all weights are *large*. Given any list of weights, Algorithm $\mathcal{A}$ produces a packing with its associated graph having r paths, each of which has at most one weak edge, and perhaps one weak loop in the entire graph. The number of bins used is therefore $(n - r)$ in which at most (r+1) are partially filled, and so at least $(n - 2r - 1)$ bins are filled. This gives us:

$$|\mathcal{A}| = n - r$$

$$|\mathcal{OPT}| \geq w \geq n - 2r - 1$$

Hence

$$\frac{|\mathcal{A}|}{|\mathcal{OPT}|} \leq \frac{n-r}{n-2r-1}$$

Now letting $r = \alpha n$ for some $\alpha \in [0,1]$, the ratio above becomes

$$\frac{|\mathcal{A}|}{|\mathcal{OPT}|} \leq \frac{1-\alpha}{1-2\alpha} = \frac{1}{2}(1 + \frac{1}{1-2\alpha})$$

asymptotically.

When the weights are all relatively large, say $1 \geq w_i > 1/2$, the above ratio can be strengthened to

$$\frac{|\mathcal{A}|}{|\mathcal{OPT}|} \leq \frac{1-\alpha}{1-3\alpha/2} = \frac{2}{3}(1 + \frac{1}{2-3\alpha})$$

We can show that Algorithm $\mathcal{A}$ has a much better performance guarantee in this situation. First we note the following lemma:

Lemma 6.1. *Given a list of weights $W = (w_1, w_2, \ldots, w_n)$ where $w_i + w_j > 1$ for any $i \neq j$ and each $w_i \leq 1$, we have $|\mathcal{OPT}| \geq 2n/3$.*

Proof: Among all n weights, we call a weight of Type A if it is all put in one bin, a weight of Type B if it gets split and put into more than 1 bin. Given an optimal algorithm, let k denote the number of Type A weights in the packing and $(n-k)$ the number of Type B weights in the packing. Therefore the total number of parts are at least $k + 2(n-k)$. We have $|\mathcal{OPT}| \geq \frac{2n-k}{2} = n - k/2$ and also $|\mathcal{OPT}| \geq k$. These together give us $|\mathcal{OPT}| \geq 2n/3$. □

Now we are ready to prove the approximation ratio in this case.

Theorem 6.3. *Given a list of weights $W = (w_1, w_2, \ldots, w_n)$ where each $1 \geq w_i > 1/2$, Algorithm $\mathcal{A}$ is 7/6-optimal.*

Proof: When each $w_i > 1/2$, Lemma 6.3 ensures that

$$|\mathcal{OPT}| \geq w \geq 2n/3$$

When $\alpha \leq 2/9$, we have

$$\frac{|\mathcal{A}|}{|\mathcal{OPT}|} \leq \frac{2}{3}(1 + \frac{1}{2-3\alpha}) = 7/6$$

When $\alpha > 2/9$, we have

$$\frac{|\mathcal{A}|}{|\mathcal{OPT}|} \leq \frac{n(1-\alpha)}{2n/3} < 7/6$$

□

This 7/6 bound can be shown to be tight given a list of weights $\{1-2\epsilon, ..., 1-2\epsilon, \frac{1}{2}+\epsilon, ..., \frac{1}{2}+\epsilon\}$ where we have n weights of size $(1-2\epsilon)$ followed by $2n$ weights of size $(1/2+\epsilon)$. An optimal packing would be to pack each one large weight and two small weights fully into two bins which takes a total of $2n$ bins and the bins are all full. Algorithm $\mathcal{A}$ however, will pack the large weights first which takes about n bins, then each three small weights can be put into two bins which takes about $4n/3$ bins. This gives us a 7/6 ratio as desired.

Remark 6.1. *Notice that the weaker condition in Lemma 6.3 (i.e., $w_i + w_j > 1$ for any $i \neq j$) would only ensure a 5/4-approximation ratio for Algorithm $\mathcal{A}$.*

In fact, we can generalize this bound as follows:

Theorem 6.4. *Given a list of weights $W = (w_1, w_2, \ldots, w_n)$ where each $1 \geq w_i > \frac{a-1}{a}$ for integer $a > 0$, Algorithm $\mathcal{A}$ is $\left(1 + \frac{1}{a(a+1)}\right)$ optimal.*

This generalization is based on the following lemma:

Lemma 6.2. *Given a list of weights $W = (w_1, w_2, \ldots, w_n)$ for all $1 \geq w_i > \frac{a-1}{a}$, we have $|\mathcal{OPT}| \geq \frac{a}{a+1}n$.*

Proof: Let us consider an optimal packing, in which each weight is either packed entirely in one bin or is broken into several pieces in several bins. In the latter

case, we can write $w_i = p_i(1) + p_i(2) + ... + p_i(l)$ to represent that w_i is broken into l pieces in this optimal packing.

Define the *value* for a weight piece p to be

$$v(p) = \frac{1}{a}\lceil ap \rceil$$

and notice the following properties:

Fact 6.1. $p \leq v(p) < p + \frac{1}{a}$

Fact 6.2. $\sum_j v(p_i(j)) \geq 1$

Fact 6.3. $v(p) \leq 1$ *for one weight piece* p

Fact 6.4. *If two pieces from two different weights* p_i *and* p'_j *are in one bin, then* $v(p_i) + v(p'_j) \leq \frac{a+1}{a}$.

The proofs are straightforward and omitted here. By Fact 6.1, the sum of $v(p)$ for all weight pieces p in this optimal packing is lower bounded by n. By Fact 6.2 and 6.3, it is also upper bounded by $\left(\frac{a+1}{a} \times |\mathcal{OPT}|\right)$. Combining the two bounds gives us the desired bound $|\mathcal{OPT}| \geq \frac{a}{a+1}n$. □

With this lemma, using similar techniques as in the proof of Theorem 6.3, we can prove Theorem 6.4 which shows that Algorithm $\mathcal{A}$ performs well on large weights in general. Figure 6.2 illustrates this property.

Remark 6.2. *Again this bound can be shown to be tight given a list of weights* $\{1 - a\epsilon, ..., 1 - a\epsilon, \frac{a-1}{a} + \epsilon, ..., \frac{a-1}{a} + \epsilon\}$ *where we have an weights of size* $(1 - a\epsilon)$ *followed by* $(a \times n)$ *weights of size* $\left(\frac{a-1}{a} + \epsilon\right)$. *An optimal packing would be to pack each one large weight and a small weights fully into a bins which takes a total of an bins and the bins are all full. Algorithm* $\mathcal{A}$ *however, will pack the large weights first which takes about* n *bins, then each* $(a + 1)$ *small weights can be put into a bins which takes about* $(a^2 n/(a + 1))$ *bins. This gives us the desired ratio.*

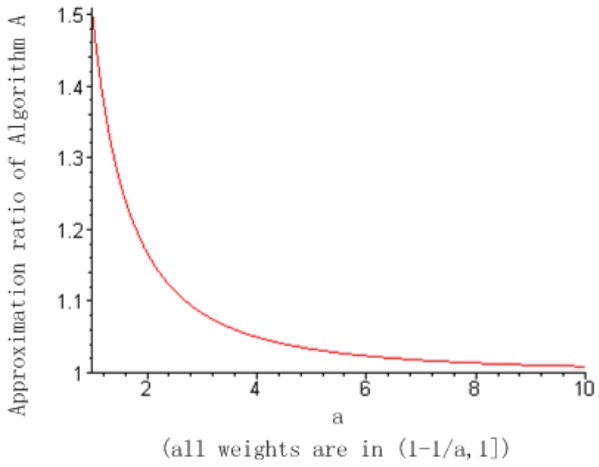

Figure 6.2: Algorithm $\mathcal{A}$ achieves a close-to-optimal approximation ratio as all weights become large.

6.3 Improved Algorithm $\mathcal{B}$

6.3.1 More about the packing graphs

Here we examine several basic properties of the associated graphs of bin packings for a given list of weights W. These properties provide the foundation for the reduction steps in the improved approximation algorithm to be discussed later.

Definition 6.1. *An associated graph G is said to be* **stable** *if and only if all of the following conditions hold:*
(i). G has no cycle;
(ii). Each connected component has at most one weak arc;
(iii). G has at most one weak loop in total.

Given that G has no cycle, each connected component is a tree with some possible loops. A crucial observation is that if there is a weak edge in this connected component, we can move it freely within this component. During the moving process, we might split the original component into two, but the total number of

bins will never increase.

We describe a few atomic repacking operations here as follows. None of them requires more bins than originally given in the packing. None of them creates cycles or strong loops in the associated graph.

Operation 1. *If a strong edge $e_1 = (i, j)$ and a weak edge $e_2 = (j, k)$ are adjacent (sharing one type j) in a component, we can repack weights so that e_1 becomes weak and e_2 becomes strong, or split the component into two with one having a weak loop e_1 and the other having an edge e_2 which can be strong or weak.*

Operation 2. *If two weak edges $e_1 = (i, j)$ and $e_2 = (j, k)$ are adjacent in one component, we can repack weights so that e_2 becomes strong and e_1 stays weak, or split the component into two with one having a weak loop e_1 and the other having an edge e_2 which can be strong or weak.*

Operation 3. *If a weak loop $e_1 = (i, i)$ and a weak edge $e_2 = (i, j)$ are adjacent in a component, we can repack weights so that we only have a single edge e_2 (i.e., we get rid of one bin e_1), or a strong edge e_2 and a weak loop e_1.*

Operation 4. *If two weak loops $e_1 = (i, i)$ and $e_2 = (j, j)$ are in two separate components C_1 and C_2, we can repack weights to merge them into one component C such that we only have a single edge e_2 (i.e., we get rid of one bin e_1), or a strong edge e_2 and a weak loop e_1.*

Lemma 6.3. *Suppose that P is a packing of a list of weights $W = (w_1, w_2, \ldots, w_n)$ into b bins, where no bin contains weights of more than two types. If the associated graph G_P has a connected component C which contains two weak edges and the rest of the graph is stable, we can find another packing P' which uses no more than b bins with its entire associated graph stable.*

Proof: Suppose a connected component of G_P contains two weak edges e_1 and e_2.

The two weak arcs cannot have the same vertices, since this would form a 2-cycle, contradicting our initial hypothesis. There must be a unique path with no loops (that is, a sequence edges so that two consecutive edges share a common vertex), say, with edges $e_1 = f_1, f_2, \ldots, f_t = e_2$. Here e_1 and e_2 are weak edges while all other f_i's are strong edges.

Select either weak edge to repack, say e_1, and proceed with Operation (1) one step at a time in order to bring the two weak edges closer together. Now we have two cases:

If we successfully carry this on until the two weak edges become adjacent, we then use Operation (2) to get rid of one weak edge or split the component into two. In the latter case, we need to check whether there are two weak loops in the entire graph. Operation (4) is needed if this is true. Now the graph is stable.

If during the moving process the component splits into two, one component only has at most one weak edge while the other component actually has one weak edge and one newly formed weak loop. For the latter component C', we need to check if there are two weak loops in the entire graph:

Case a. If this is true, Operation (4) is needed to first get rid of the extra weak loop. This will possibly result in another weak edge in this smaller component and now it contains two weak edges. Notice however now these two weak edges are closer compared with the original two weak edges in C. We carry out Lemma 6.3 recursively in this case.

Case b. Otherwise, we select the weak edge in C' and move it towards the weak loop the same way as 6.3 recursively. Use Operation (2) or Operation (3) when the two partially filled bins become adjacent in the graph.

This process will stop in a finite number of steps and the graph will become

stable. □

Lemma 6.4. *Suppose that P is a packing of a list of weights $W = (w_1, w_2, \ldots, w_n)$ into b bins, where no bin contains weights of more than two types. If the associated graph G_P contains a strong loop in one connected component X and a weak edge in another connected component Y and is stable, we can find another packing P' which uses no more than b bins with its associated graph with one fewer strong loops than packing P and also stable.*

Proof: Suppose in X there is a loop that is strong (associated with a filled bin, say e_1, in one type j) and there is a weak edge $\{k, l\}$ (associated with a partially filled bin, say e_2) in another component Y. We reconfigure the two bins as follows:

Suppose e_2 contains parts of weights w'_k and w''_l. We partition the weight of type j in e_1 into two parts w'_j (of size the same as w'_k) and w''_j of size $(1 - w'_k)$ and switch the parts w'_k and w'_j.

Check if there is more than one weak edge in the newly formed connected component that contains j, k and l. If it does, use the steps as described in Lemma 6.3 until the graph is stable. The resulting packing has its associated graph containing one fewer strong loop. □

6.3.2 Algorithm $\mathcal{B}$

We now consider a modified version of the simple approximation algorithm $\mathcal{A}$ given above. We will show that the modified algorithm $\mathcal{B}$ gives an optimal solution when the total weight is greater than or equal to the number of types. In general, the modified algorithm gives an approximation solution within a factor of 3/2 of the optimum asymptotically.

Theorem 6.5. *In $O(n)$ time, Algorithm $\mathcal{B}$ generates a bin packing that is optimal*

if the total weight is at least as large as the number of types. In general, the bin packing using Algorithm $\mathcal{B}$ has size within a factor of 3/2 of the optimum asymptotically.

Before we introduce the improved algorithm $\mathcal{B}$, let us first consider an intermediate form of it, say Algorithm $\mathcal{A}$', which takes the output packing of Algorithm $\mathcal{A}$ as input and processes it using the following steps:

Algorithm $\mathcal{A}$'

Given a list of weights $W = (w_1, w_2, \ldots, w_n)$,

- Use Algorithm $\mathcal{A}$ to generate a valid packing P.
- While there exists a connected component X containing a strong loop and another component Y containing a weak edge,
  - Use the steps as described in the proof of Lemma 6.4 to merge these two components into one.

The resulting bin packing using Algorithm $\mathcal{A}$' has an associated graph G with no cycle and each connected component having at most one weak edge. In addition, if there is a strong loop, then all other components have no weak edges.

Suppose the total weight $w = \sum_i w_i$ is greater than or equal to n, the number of types. From the reduction steps in the algorithm, G can have at most $n - 1$ edges and there is at most one weak loop. Since the total weight is at least n, there is at least one loop that is strong. Thus there is no weak edge outside of the connected component C that contains the loop. In C, there is at most one weak edge. So altogether, there is at most one weak edge. This implies that the number of bins is exactly $\lceil w \rceil$ which is optimum. When $w < n$, we can still use Theorem 6.2 to show the resulting packing is within a factor of 3/2 of the optimum.

We have proved the following:

Theorem 6.6. *Algorithm $\mathcal{A}$' generates a bin packing that is optimal if the total weight is at least as large as the number of types. In general, the bin packing using Algorithm $\mathcal{A}$' has size within a factor of $3/2$ of the optimum asymptotically.*

Now let us consider the complexity of Algorithm $\mathcal{A}$', we note the following:

1). Algorithm $\mathcal{A}$ produces a *stable* packing P in $O(n)$ time.

2). During the execution of the *while* loop, the number of *strong* loops is strictly decreasing. Every time we get rid of one strong loop, at most a linear number of atomic operations are involved, each taking constant time.

Therefore, Algorithm $\mathcal{A}$' runs in time $O(n^2)$ at most, where n is the number of types. In fact, if we are a little more careful about the order of the atomic operations applied, we can achieve a linear time algorithm. To describe it, we need the following lemma:

Lemma 6.5. *Suppose that P is a packing of a list of weights $W = (w_1, w_2, \ldots, w_n)$ into b bins, where no bin contains weights of more than two types. If the associated graph G_P is a forest where some or all of the components have $k \geq 2$ weak edges but are otherwise stable, we can find another packing P' which uses no more than b bins with its associated graph stable in linear time in n.*

Proof: Let X be a component which has $k \geq 2$ weak edges. Pick an arbitrary vertex v in X to be the root. If we traverse X using DFS, there is a natural order $O = \{v_1, v_2, ..., v_n\}$ of the vertices defined by the last visiting time. In other words, the time when v_1 is last visited is earlier than the time when v_2 is last visited and so on.

For $i = 1, ..., n$, we consider all the edges that are adjacent to v_i, say $\{e_1, ..., e_s\}$ in which e_1 is the only edge that is closer to the root. If two or more of the other

$(s-1)$ edges are weak, we can use Operation (2) to merge them in a pairwise manner. This process could result in just one weak edge, say e_j, in which case we use Operation (2) or (1) to push the weak edge towards the root, or it may split this component X into smaller trees but the total number of weak edges will never increase. Whenever there are two or more weak loops in the entire graph, we use Operation (4) to get rid of the extra weak loops immediately.

Any component X that has $k \geq 2$ weak edges needs to be processed in this way. If it splits during the process, any newly formed component that has $k \geq 2$ weak edges will also be processed. The total number of atomic operations needed until no more such component exists is linear in n.

This proves our lemma. □

Such a repacking also does not create cycles or more strong loops. Now we have our linear time algorithm B:

Algorithm $\mathcal{B}$

Given a list of weights $W = (w_1, w_2, \ldots, w_n)$,

- Use Algorithm $\mathcal{A}$ to generate a valid packing P.
- While there exists a component X containing a strong loop and another component Y containing a weak edge,
  - Use only the first step as described in the proof of Lemma 6.4 to merge these two components into one, without taking care of the possible multiple edges in any one connected component.
  - Use the steps as described in the proof of Lemma 6.5 to get rid of the extra weak edges in each component.

Algorithm $\mathcal{B}$ produces packings as good as Algorithm $\mathcal{A}$' with the improvement that its running time is linear instead of quadratic. Therefore we have the following

theorem:

Theorem 6.7. *In $O(n)$ time, Algorithm $\mathcal{B}$ generates a bin packing that is optimal if the total weight is at least as large as the number of types. In general, the bin packing using Algorithm $\mathcal{B}$ has size within a factor of $3/2$ of the optimum asymptotically.*

Here we will give an example which shows that Algorithm $\mathcal{A}$ and $\mathcal{B}$ can generate bin packings with the number of bins off by a factor $(3/2 + o(1))$ of the optimum. Therefore the approximation ratios shown in Theorem 6.2 and 6.7 are tight.

Suppose that k is an integer. We are given a list W of weights where the first $2(k+1)$ weights are of size $k/(k+1)$ and then the next $2(k+1)$ weights are of size $1/(k+1)$.

Using Algorithm $\mathcal{A}$ or $\mathcal{B}$, we will end up with a packing which uses the first $2k$ bins to pack the first $2(k+1)$ weights fully without any waste. Then the next group of bins each contain two weights of size $1/(k+1)$. Altogether, $3k+1$ bins are used. Nevertheless, the optimum packing consists of $2(k+1)$ bins each contain one weight of size $k/(k+1)$ and one weight of size $1/(k+1)$. Thus we have the ratio

$$\frac{|\mathcal{A}| \text{ or } |\mathcal{B}|}{|\mathcal{OPT}|} = \frac{3k+1}{2(k+1)} = \frac{3}{2} - \frac{1}{k+1}$$

which is arbitrarily close to $3/2$ when k is large.

6.4 General kBPS

6.4.1 $(2 - 1/k)$-approximation Algorithm $\mathcal{A}_k$

The approximation algorithms $\mathcal{A}$ and $\mathcal{B}$ presented for the 2BPS problem are simple and efficient. What is the situation if we consider the general kBPS problem? This was left open in [21] [22]. Here we present positive answers to this question.

First we notice that given any fixed integer $k \geq 2$, there is a simple k-approximation algorithm for kBPS . This can be easily reasoned using the k-cardinality restriction, and the simple algorithm is just to never share any bin. We also note that this ratio also holds for online scenarios using the same algorithm. The obvious drawback is that approximation ratio increases as k increases. A constant bound on the approximation ratio is more desirable.

Algorithm $\mathcal{A}_k$ is a natural generalization of $\mathcal{A}$ for any fixed $k \in \mathcal{Z}^+$. Note that now a *live bin* refers to a partially filled bin with fewer than k types. Algorithm $\mathcal{A}_k$ works in both the offline and online settings and achieves an approximation ratio of $(2 - 1/k)$.

Algorithm $\mathcal{A}_k$

Given a list of weights $W = (w_1, w_2, \ldots, w_n)$,
- For each $i = 1, 2, \ldots, n$, we pack greedily as follows:
  - Place the maximum possible part of w_i into a *live bin* if possible;
  - Otherwise, put it into one or more *new bins*.

Theorem 6.8. *For any fixed $k \in \mathcal{Z}^+$, Algorithm $\mathcal{A}_k$ achieves an asymptotic approximation ratio of $(2 - 1/k)$.*

Proof. Given any list of weights $W = (w_1, w_2, \ldots, w_n)$ where each $w_i > 0$, let $|\mathcal{A}_k|$ denote the number of bins used in the packing generated by Algorithm $\mathcal{A}_k$ and $|\mathcal{OPT}|$ the number of bins used in an optimal packing. Define $w = \sum_{i=1..n} w_i$ and $w^* = \sum_{i=1..n} \lceil w_i \rceil$.

Claim 6.1. $|\mathcal{OPT}| \geq \max\{w, \frac{w^*}{k}\}$

Proof. Each bin has unit size, so the weight sum w lower bounds $|\mathcal{OPT}|$. Due to the same reason, each w_i has to be decomposed into a least $\lceil w_i \rceil$ pieces. The k-cardinality constraint then gives us another lower bound $\frac{w^*}{k}$ for $|\mathcal{OPT}|$. □

Claim 6.2. $|\mathcal{A}_k| \leq \frac{w^*+(k-1)w+(k-1)}{k}$

Proof. Consider the final packing generated by Algorithm $\mathcal{A}_k$. Define a graph G as follows. Every vertex represents a bin used in the packing. If two bins contains weight pieces from the same item w_i, we connect the two bins by an edge. It is clear that G is composed of a set of paths and maybe some isolated vertices.

Consider a connected component A in G (i.e., a path or an isolated vertex), define w_A and w_A^* accordingly. Let A also denote the number of vertices in A. w_A^* upper bounds A. In the case that A is a path, we have possibly one partially filled bin, so $w \geq A - 1$. Therefore:

$$w_A^* + (k-1)w \geq A + (k-1)(A-1)$$
$$A \leq \frac{w^* + (k-1)w + (k-1)}{k}$$

Summing up this inequality for the connected components in G, we have the claim. □

Claim 6.3. $\frac{w^*+(k-1)w+(k-1)}{k} \leq (2 - \frac{1}{k}) \max\{w, \frac{w^*}{k}\} + \frac{k-1}{k}$

Proof. When $w \geq \frac{w^*}{k}$,

$$\frac{w^* + (k-1)w + (k-1)}{k} \leq (2 - \frac{1}{k})w + \frac{k-1}{k}$$

When $w < \frac{w^*}{k}$,

$$\frac{w^* + (k-1)w + (k-1)}{k} < (2 - \frac{1}{k})\frac{w^*}{k} + \frac{k-1}{k}$$

Hence the claim holds. □

From these three claims above, Theorem 6.8 follows immediately. □

Note that with the additive term $\frac{k-1}{k}$ in Claim 6.3, the approximation ratio $(2-1/k)$ holds only asymptotically and is not exact. This ratio cannot be improved for Algorithm $\mathcal{A}_k$ by considering the weight sequence of t weights each of size $(1-(k-1)\epsilon)$ followed by $((k-1)t)$ small weights each of size ϵ.

6.4.2 A lower bound for all online algorithms

In the online scenario, algorithms suffer from the lack of knowledge about future items to be packed. Intuitively, if k *small* items are packed into one bin, this bin is not used efficiently. The following theorem is based on this intuition.

Theorem 6.9. *No online algorithm $\tilde{\mathcal{O}}$ for* kBPS *can achieve approximation ratio better than $\left(1 + \frac{1}{k+\frac{1}{k-1}}\right)$, which is 4/3 for $k = 2$ and 9/7 for $k = 3$, etc.*

Proof. Consider the following list of items to be packed. Algorithm $\tilde{\mathcal{O}}$ is first given $(k-1)n$ tiny items each of size $\epsilon > 0$. It is clear that slicing such tiny items would only deteriorate the packing, so we assume that no slicing has occurred. We say a used bin is *dead* if it is not *live*. Let x denote the total number of bins used, y the number of live bins and z the number of dead bins in the packing. In this case, the dead bins are the ones each with exactly k items in them.

Now if $z \geq \frac{n}{k+\frac{1}{k-1}}$, we will then give Algorithm $\tilde{\mathcal{O}}$ a list of n big items each of size $(1-(k-1)\epsilon)$. The optimal algorithm will pack each big item with $(k-1)$

small items and use a total of n bins for the packing. Algorithm $\tilde{\mathcal{O}}$ however cannot utilize the z dead bins any more, which gives us the ratio:

$$\frac{|\tilde{\mathcal{O}}|}{|\mathcal{OPT}|} \geq \frac{y+z+(n-y)}{n} \geq 1 + \frac{1}{k+\frac{1}{k-1}}$$

Otherwise we have $z < \frac{n}{k+\frac{1}{k-1}}$. Consider the approximation ratio at this point. Currently the most efficient packing for the y live bins is to have $(k-1)$ small items in each, so

$$y(k-1) + zk = (k-1)n$$

$$xk - y = (k-1)n$$

Hence

$$y > \frac{(k-1)n - \frac{kn}{k+\frac{1}{k-1}}}{k-1} = n - \frac{kn}{k(k-1)+1} = \frac{(k-1)^2 n}{k(k-1)+1}$$

and

$$\begin{aligned} \frac{|\tilde{\mathcal{O}}|}{|\mathcal{OPT}|} &= \frac{xk}{(k-1)n} \\ &= 1 + \frac{y}{(k-1)n} \\ &> 1 + \frac{1}{k+\frac{1}{k-1}} \end{aligned}$$

□

6.5 Acknowledgement

This chapter contains material appearing in "Parallelism vs. Memory Allocation in Pipelined Router Forwarding Engines" from *Theory of Computing Systems (ToCS), ISSN 1432-4350 (Print) 1433-0490 (Online) 2006*, and material submitted for publication. I would like to thank my co-authors Fan Chung, Ron Graham, and George Varghese.

7

An ε-Improvement Approximation

7.1 Algorithm $\mathcal{INC}$

In the offline setting, simple greedy algorithm $\mathcal{A}$ already gives a $\frac{3}{2}$ approximation ratio. Can we obtain better approximation ratios with more sophisticated algorithms in the offline setting? The answer seems very likely to be affirmative. Given access to the entire input, we know the relative sizes of the weights to be packed, and can possibly avoid wasting space by putting two small weights into a bin for example.

In this section, we present a new algorithm $\mathcal{INC}$. The intuition is that small weights are the *troublemakers* and need to be taken care of first. This algorithm performs well on the worst-case example given for Algorithm $\mathcal{A}$ and $\mathcal{B}$.

Let $L = (w_1, w_2, \ldots, w_m)$ be a given list of weights. The running time of $\mathcal{INC}$ is no longer linear but remains polynomial in the form $O(m^2 \log m)$.

Algorithm $\mathcal{INC}$

- Sort L into non-decreasing order;
- Loop until all weights are packed:
    - Put the current minimum weight into a new bin;
    - If this bin is completely filled (so now it becomes dead),
        - Insert the possible left-over weight back into the sorted list with respect to the order;
        - Continue;
    - Else
        - Put the maximum possible part of the current largest weight into this live bin (so after this step this bin becomes dead);
        - Insert the possible left-over weight back into the sorted list with respect to the order;

7.2 ε-Improvement for 2BPS

The approximation ratio analysis is not straightforward. We circumvent the difficulty by what is known as the ε-improvement technique, introduced by Yao [68]. Hence we focus on a specific goal to show that $\mathcal{INC}$ is at least $(\frac{3}{2} - \varepsilon)$-optimal for some positive ε. The idea is to progressively zoom in to the worst-case input for $\mathcal{INC}$ and show that $\mathcal{INC}$ performs at least that well with some maximized ε. This will show that Algorithm $\mathcal{INC}$ performs strictly better than previously proposed algorithms $\mathcal{A}$ and $\mathcal{B}$.

Theorem 7.1. *There exists $\varepsilon > 0$ such that given any list of positive weights, Algorithm $\mathcal{INC}$ is $(\frac{3}{2} - \varepsilon)$ optimal.*

Proof: We will prove the theorem by contradiction. Given a list of weights $L = (w_1, w_2, \ldots, w_m)$, let us assume the $\mathcal{INC}$ algorithm is not $(\frac{3}{2} - \varepsilon)$-optimal for a

very small $\varepsilon > 0$ to be specified later, i.e.,

$$\frac{|\mathcal{INC}(L)|}{|\mathcal{OPT}(L)|} > \frac{3}{2} - \varepsilon \tag{7.1}$$

where $\mathcal{INC}(L)$ denotes the packing of L by the $\mathcal{INC}$ algorithm and $\mathcal{OPT}(L)$ denotes an optimal packing, and their cardinalities denote the number of bins used, respectively. L is then the potential worst-case input we like to examine closely.

In the following, we will denote various functions of ε by $\varepsilon_i = f_i(\varepsilon)$ where each $\varepsilon_i \to 0$ as $\varepsilon \to 0$.

Claim 7.1. *$\mathcal{INC}(L)$ cannot have a full bin with just one type of weight.*

Proof: If there is such a full bin B_i created at the ith step that only contains weight from w_j, then for $\forall u < i$, B_u must be full as well because all weights that are packed earlier than w_j have size at least 1. Also at the ith step, w_j is the smallest weight so all weights packed afterwards must also have size at least 1. Therefore every bin in $\mathcal{INC}(L)$, except possibly the last, is full, i.e., $\mathcal{INC}(L) = \mathcal{OPT}(L)$, which contradicts with assumption (7.1) if $\epsilon < 1/2$, for example. □

Therefore $\mathcal{INC}(L)$ consists of only full bins each with two types and partially filled bins (each with two types, except possibly for the last bin). Now suppose $\mathcal{INC}(L)$ has n full bins with two types and αn partially filled bins, i.e., $|\mathcal{INC}(L)| = (1 + \alpha)n$.

Claim 7.2.

$$\frac{1}{2} - \varepsilon_1 < \alpha < \frac{1}{2} + \varepsilon_2$$

where

$$\varepsilon_1 = \varepsilon \tag{7.2}$$

and

$$\varepsilon_2 = \frac{2\varepsilon}{1-2\varepsilon} \tag{7.3}$$

Proof: Consider the packing process of $\mathcal{INC}(L)$. When a weight piece is packed in bin B_i and it is the last piece of its type w_j, then we say that B_i *retires* type j. We observe that a full bin with two types retires at least the bottom type and a partially filled bin with two types retires both types. Therefore, we have the following inequalities regarding the number of types of weight m and the total weight w:

$$\begin{aligned} m &\geq (1+2\alpha)n, \\ w &\geq n \end{aligned}$$

Therefore we must have

$$|\mathcal{OPT}(L)| \geq \max\{(\frac{1}{2}+\alpha)n, n\} \tag{7.4}$$

On the other hand, Assumption (7.1) implies

$$|\mathcal{OPT}(L)| < \frac{|\mathcal{INC}(L)|}{\frac{3}{2}-\varepsilon} = \frac{(1+\alpha)n}{\frac{3}{2}-\varepsilon} \tag{7.5}$$

Case 1: If $\alpha \geq \frac{1}{2}$, (7.4) implies $|\mathcal{OPT}(L)| \geq (\frac{1}{2}+\alpha)n$. Combined with (7.1), we have

$$\begin{aligned} \frac{1+\alpha}{\frac{3}{2}-\varepsilon} &> \frac{1}{2}+\alpha \\ 1+\alpha &> \frac{3}{4}-\frac{\varepsilon}{2}+\left(\frac{3}{2}-\varepsilon\right)\alpha \\ \left(\frac{1}{2}-\varepsilon\right)\alpha &< \frac{1}{4}+\frac{\varepsilon}{2} \\ \alpha &< \frac{1+2\varepsilon}{2-4\varepsilon} \\ &= \frac{1}{2}+\frac{2\varepsilon}{1-2\varepsilon} \end{aligned}$$

Case 2: If $\alpha < \frac{1}{2}$, (7.4) and (7.1) imply that

$$\begin{aligned} \frac{1+\alpha}{\frac{3}{2}-\varepsilon} &> 1 \\ \alpha &> \frac{1}{2}-\varepsilon \end{aligned}$$

Therefore, we have

$$\frac{1}{2}-\varepsilon_1 < \alpha < \frac{1}{2}+\varepsilon_2 \tag{7.6}$$

which proves the claim. □

Consequently, from (7.1) and (7.6)

$$\begin{aligned} |\mathcal{OPT}(L)| &< \frac{|\mathcal{INC}(L)|}{\frac{3}{2}-\varepsilon} \\ &< \frac{1+\frac{1}{2}+\varepsilon_2}{\frac{3}{2}-\varepsilon} n \\ &= (1+\varepsilon_3)n \end{aligned} \tag{7.7}$$

where

$$\varepsilon_3 = \frac{2\varepsilon}{1-2\varepsilon} = \varepsilon_2 \tag{7.8}$$

Denote the total weight of the αn partially filled bins of $\mathcal{INC}(L)$ by w'. Then we have

$$w' < \varepsilon_3 n \tag{7.9}$$

Claim 7.3. *Given any $\delta \in (0,1)$, there are at least $(\frac{1}{2}-\varepsilon_4)n$ partially filled bins in $\mathcal{INC}(L)$ each having total weight less than δ where $\varepsilon_4 = \varepsilon_1 + \frac{\varepsilon_3}{\delta}$.*

Proof: Suppose $\mathcal{INC}(L)$ has βn partially filled bins each with total weight at least δ. Then

$$\begin{aligned} \delta\beta n &< \varepsilon_3 n \\ \beta &< \frac{\varepsilon_3}{\delta} \end{aligned}$$

Thus, the number of partially filled bins each with weight less than δ is

$$\begin{aligned}(\alpha-\beta)n &> (\alpha-\frac{\varepsilon_3}{\delta})n \\ &> (\frac{1}{2}-\varepsilon_1-\frac{\varepsilon_3}{\delta})n\end{aligned}$$

This proves our claim for

$$\varepsilon_4 = \varepsilon_1 + \frac{\varepsilon_3}{\delta} \tag{7.10}$$

□

Thus, there are many bins in $\mathcal{INC}(L)$ with very small total weight (bounded by δ). During the process in which Algorithm $\mathcal{INC}$ generates the packing for L, when does it start generating these highly *wasted* bins? We divide the process of $\mathcal{INC}(L)$ into two phases. Phase II begins as soon as a packed (dead) bin has total weight less than δ. Because $\mathcal{INC}$ packs the current largest weight with the current smallest weight, we can conclude that all remaining weights now must have size less than δ each and hence all subsequently packed bins will have weight less than 2δ each.

Definition 7.1. *We call a weight piece* tiny *if it has size less than* δ.

Claim 7.4. *After Phase I,* $\mathcal{INC}(L)$ *must have at least* $(1-\varepsilon_7)n$ *tiny weights where* $\varepsilon_7 = \varepsilon_4 + \varepsilon_5$.

Proof: With a similar argument to that in Claim 7.3, we know that $\mathcal{INC}(L)$ has at least $(\frac{1}{2}-\varepsilon_4)n$ bins each with weight less than δ, and has at least $(\frac{1}{2}-\varepsilon_5)n$ bins each with weight less than 2δ, where

$$\varepsilon_5 = \varepsilon_1 + \frac{\varepsilon_3}{2\delta} \tag{7.11}$$

Note that the $(\frac{1}{2}-\varepsilon_4)n$ bins will also be counted in the $(\frac{1}{2}-\varepsilon_5)n$ bins. A bin with total weight less than δ contributes two tiny weights. A bin with total weight at

least δ but less than 2δ contributes at least one tiny weight. So the total number of tiny weights in $\mathcal{INC}(L)$ is

$$\begin{aligned} &\geq 2\left(\frac{1}{2}-\varepsilon_4\right)n + [\left(\frac{1}{2}-\varepsilon_5\right) - \left(\frac{1}{2}-\varepsilon_4\right)]n \\ &= [1-(\varepsilon_4+\varepsilon_5)]n \end{aligned}$$

Hence the claim holds for

$$\varepsilon_7 = \varepsilon_4 + \varepsilon_5 \tag{7.12}$$

□

Claim 7.5. *There are at least* $(1-\varepsilon_7)n$ *tiny weights in* L *originally.*

Proof: At each step of $\mathcal{INC}(L)$, the smallest weight is *eaten* and at most one small weight is created and inserted back into the list. Since at the beginning of Phase II of $\mathcal{INC}(L)$, and by Claim 7.4 there are at least $(1-\varepsilon_7)n$ tiny weights, there must be at least that many tiny weights in L to begin with. □

Claim 7.6. *There are fewer than* $(1+\varepsilon_8)n$ *tiny weights in* L *originally where* $\varepsilon_8 = \varepsilon_3 + \frac{\varepsilon_3}{1-2\delta}$.

Proof: Suppose $\mathcal{OPT}(L)$ has γn bins with two tiny weights in each. Because of (7.4) and (7.7), the total weight packed into $\mathcal{OPT}(L)$ w must satisfy

$$\begin{aligned} n \leq w &< (1+\varepsilon_3-\gamma)n + 2\delta\gamma n \\ \gamma &< \frac{\varepsilon_3}{1-2\delta} \end{aligned}$$

Since each of these γn bins can hold at most two original tiny weights and the other bins can hold at most one original tiny weight, L must have

$$\begin{aligned} &\leq (1+\varepsilon_3-\gamma)n + 2\gamma n \\ &= (1+\varepsilon_3+\gamma)n \\ &< (1+\varepsilon_3+\frac{\varepsilon_3}{1-2\delta})n \end{aligned}$$

tiny weights, which proves our claim with

$$\varepsilon_8 = \varepsilon_3 + \frac{\varepsilon_3}{1-2\delta} \tag{7.13}$$

□

Definition 7.2. *We call a weight piece* large *if it has size in* $(1-\delta, 1+\delta)$.

Claim 7.7. *There are at least* $(1-\varepsilon_9)n$ *large weights in* L *originally where* $\varepsilon_9 = \varepsilon_7 + \varepsilon_8$.

Proof: Claim 7.6 shows that L has fewer than $(1+\varepsilon_8)n$ tiny weights originally. Claim 7.4 shows that at the beginning of Phase II in $\mathcal{INC}(L)$ there are at least $(1-\varepsilon_7)n$ tiny weights. During Phase I, the number of times we enter the loop of $\mathcal{INC}$ has to be at least n, because $\mathcal{INC}(L)$ has n full bins. Yet every step of $\mathcal{INC}(L)$ *eats* the currently smallest weight and creates at most one tiny weight. Note that no new weight will be created during Phase II.

Suppose x tiny weights are created during Phase I. At the beginning of Phase II, the number of tiny weights we have is at most $\leq (1+\varepsilon_8)n - n + x$, which has to be greater than or equal to $(1-\varepsilon_7)n$, and so

$$x \geq [1 - (\varepsilon_7 + \varepsilon_8)]n.$$

To create a new tiny weight while eating an existing one, the second weight being put into the bin has to have size in $(1-\delta, 1+\delta)$. Hence the claim holds with

$$\varepsilon_9 = \varepsilon_7 + \varepsilon_8 \tag{7.14}$$

□

Claim 7.8. $\mathcal{OPT}(L)$ *has to pack at least* $(1-\varepsilon_{10})n$ *original large weights entirely in one bin (i.e., no splitting) where* $\varepsilon_{10} = 2\varepsilon_3 + \varepsilon_7 + 2\varepsilon_9$.

Proof: By Claim 7.7, L has at least $(1-\varepsilon_9)n$ large weights; by Claim 7.4, L has at least $(1-\varepsilon_7)n$ tiny weights. If $\mathcal{OPT}(L)$ has split ξn original large weights each into two or more pieces, then the total number of weight pieces is at least

$$2\xi n + (1-\varepsilon_9-\xi)n + (1-\varepsilon_7)n$$

By (7.7), we also have

$$|\mathcal{OPT}(L)| < (1+\varepsilon_3)n$$

Hence

$$\begin{aligned} 2(1+\varepsilon_3)n &> 2\xi n + (1-\varepsilon_9-\xi)n + (1-\varepsilon_7)n \\ 2\varepsilon_3 &> \xi - \varepsilon_9 - \varepsilon_7 \\ \xi &< 2\varepsilon_3 + \varepsilon_7 + \varepsilon_9 \end{aligned}$$

Therefore at least $(1-\varepsilon_9-\xi)n$ original large weights are not split in $\mathcal{OPT}(L)$. The claim holds with

$$\varepsilon_{10} = 2\varepsilon_3 + \varepsilon_7 + 2\varepsilon_9 \tag{7.15}$$

□

Definition 7.3. *We call a weight piece a* nice large *weight if it has size in* $(1-\delta, 1)$.

Claim 7.9. *L must have at least* $(1-\varepsilon_{10})n$ nice large *weights.*

Proof: This is an immediate consequence of Claim 7.8. □

This mysterious list L has now unveiled its structure step by step. How does $\mathcal{INC}$ pack these many *nice large* weights of L? Note that as $\mathcal{INC}(L)$ eats the current tiny weight by putting in a nice large weight into the current bin, the surplus becomes the current smallest piece in the new list, and we repeat. So we have these chains of decreasing surplus pieces, with each chain ending with a complete nice large piece fully into a bin. An example of such a process is depicted in Fig. 7.2.

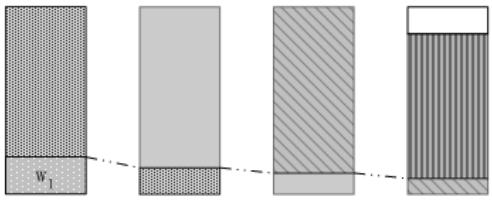

Figure 7.1: A chain of length 4.

□

Claim 7.10. *$\mathcal{INC}(L)$ has no more than $\varepsilon_{11}n$ chains where $\varepsilon_{11} = 2\varepsilon_3 + 2\varepsilon_5 + \varepsilon_{10}$.*

Proof: Suppose $\mathcal{INC}(L)$ has τn chains. In the proof for Claim 7.4, $\mathcal{INC}(L)$ has at least $(\frac{1}{2} - \varepsilon_5)n$ bins each with weight less than 2δ, each of which retires two types. By Claim 7.9, $\mathcal{INC}(L)$ must also have at least $(1 - \varepsilon_{10})n$ bins, each holding part or all of an original nice large weight. Each of these bins retires at least one type, except that every bin at the end of a chain retires two types. Therefore the total number of different types in $\mathcal{INC}(L)$ is

$$\geq (1 - \varepsilon_{10} + \tau)n + (1 - 2\varepsilon_5)n$$

Together with (7.7), we must have

$$\begin{aligned} (1 - \varepsilon_{10} + \tau) + (1 - 2\varepsilon_5) &< 2 + 2\varepsilon_3, \\ \tau &< 2\varepsilon_3 + 2\varepsilon_5 + \varepsilon_{10} \end{aligned}$$

This proves our claim with

$$\varepsilon_{11} = 2\varepsilon_3 + 2\varepsilon_5 + \varepsilon_{10} \tag{7.16}$$

□

Finally, we examine what $\mathcal{OPT}$ can do to deal with such a list L that has

(i). at least $(1-\varepsilon_7)n$ tiny weights (Claim 7.5);

(ii). at least $(1-\varepsilon_{10})n$ nice large weights that are not split in $\mathcal{OPT}(L)$ (Claim 7.8 and 7.9).

These many nice large weights must remain intact and each occupies one bin. We also know that tiny weights are *troublemakers*, in the sense that if two of them end up in one bin it is a big waste of resources. How many such tiny weights can be paired with the non-splittable nice large weights in $\mathcal{OPT}(L)$?

Claim 7.11. *In $\mathcal{OPT}(L)$, no more than $\varepsilon_{12}n$ original tiny weights can be paired up each with an original nice large weight where $\varepsilon_{12} = \varepsilon_3 + (1-\delta)\varepsilon_{10} + \varepsilon_{11} + \delta$.*

Proof: We first take a look at the process of $\mathcal{INC}(L)$. We can classify all original tiny weights of L according to $\mathcal{INC}(L)$. Given that L has so many tiny weights, during Phase I, $\mathcal{INC}$ packs one such tiny weight at a time and possibly creates some along the way. These tiny weights will be called *Type A*. At some point in Phase I, $\mathcal{INC}$ starts packing tiny weights with nice large weights which will create chains. The tiny weights at the beginning of the chains will be called *Type B*. During Phase II, $\mathcal{INC}$ still has to pack two tiny weights into one bin. These tiny weights will be called *Type C*.

In $\mathcal{OPT}(L)$, let's now consider how many tiny weights of type B or C can be paired up with nice large weights. Construct a bipartite graph G_b as shown in Fig. 7.2. All original tiny weights of type B and C are listed on the left-hand side in non-decreasing order with respect to size. All original nice large weights are listed on the right hand side in non-increasing order with respect to size. An edge is drawn between a tiny weight and a nice large weight if they can be packed together entirely in one bin. According to the chains in $\mathcal{INC}(L)$, w_1 is connected only with the last nice large weight in the first chain and all others of equal or smaller size. Similarly, w_i is connected only with the last nice large weight in the

ith chain and all others of equal or smaller size. Since all original nice large weights are packed in chains in $\mathcal{INC}(L)$, no tiny weights of type C can be connected to any nice large weights in G_b. Therefore the size of a matching in G_b is upper bounded by the number of chains in $\mathcal{INC}(L)$, which is no more than $\varepsilon_{11}n$.

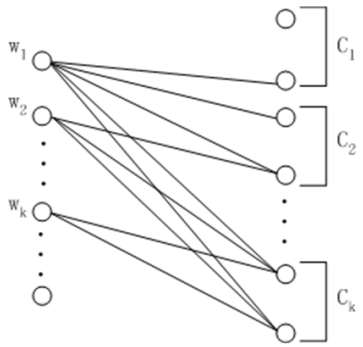

Figure 7.2: Matching graph of original tiny weights of type B and C and nice large weights, arranged in non-decreasing size on the left side and non-increasing size on the right side. The nice large weights are also grouped according to the chains in $\mathcal{INC}(L)$.

Note that type A tiny weights are packed in full bins in $\mathcal{INC}(L)$. Then the number of type A tiny weights is

$$< \quad (1+\varepsilon_3)n - (1-\varepsilon_{10})n(1-\delta)$$

Therefore even if all type A tiny weights can be paired up each with a nice large weight in $\mathcal{OPT}(L)$, our claim holds for

$$\begin{aligned} \varepsilon_{12} &= \varepsilon_{11} + (1+\varepsilon_3) - (1-\varepsilon_{10})(1-\delta) \\ &= \varepsilon_3 + (1-\delta)\varepsilon_{10} + \varepsilon_{11} + \delta \end{aligned} \tag{7.17}$$

□

Now examine $\mathcal{OPT}(L)$. By Claim 7.9 and (7.7), the number of bins with two tiny weights in each cannot be more than $(\varepsilon_{10}+\varepsilon_3)n$. By Claim 7.11, the number of bins each with one tiny weight paired with one nice large weight is no more than $\varepsilon_{12}n$. The rest of the bins can at most each contribute one original tiny weight. Therefore the total number of original tiny weights in $\mathcal{OPT}(L)$ is no more than

$$2(\varepsilon_{10}+\varepsilon_3)n+\varepsilon_{12}n+\frac{1-\varepsilon_{10}-\varepsilon_{12}}{2}n$$

According to Claim 7.5, this has to be greater than or equal to $(1-\varepsilon_7)n$, i.e.,

$$\begin{aligned} 4(\varepsilon_{10}+\varepsilon_3)+2\varepsilon_{12}n+1-\varepsilon_{10}-\varepsilon_{12} &\geq 2-2\varepsilon_7 \\ 4\varepsilon_3+3\varepsilon_{10}+2\varepsilon_7+\varepsilon_{12} &\geq 1 \end{aligned} \tag{7.18}$$

The ε_i's are functions of ε and δ. Fixing some $\varepsilon\in(0,1)$ and $\delta\in(0,1)$, if this inequality is false, we have a contradiction, which implies that (7.1) is false, i.e., Algorithm $\mathcal{INC}$ is at least $(\frac{3}{2}-\varepsilon)$-optimal for this value of ε.

Assemble all the ε_i's from (7.2), (7.3), (7.8), (7.10), (7.11), (7.12), (7.13), (7.14), (7.15), (7.16), and (7.17). Falsifying (7.18) while maximizing ε, the best ε we can obtain is at least 0.00232 when $\delta\simeq 0.292$ as shown in Fig. 7.2. Therefore Algorithm $\mathcal{INC}$ is at least $(\frac{3}{2}-0.00232)$-optimal. □

Remark 7.1. *The best worst-case example we have, however, can only ensure a* 6/5 *ratio. Consider a list* L' *of weights of increasing size* $(\frac{1}{2}-\epsilon_1,\frac{1}{2}-\epsilon_2,...,\frac{1}{2}-\epsilon_n;\frac{1}{2}+\epsilon_n,\frac{1}{2}+2\epsilon_n,...,\frac{1}{2}+\epsilon_1,\frac{1}{2}+2\epsilon_1)$, *where* $1>\epsilon_1>\epsilon_2>...>\epsilon_n>0$. *Algorithm* $\mathcal{INC}$ *will pack every three weights* $\{\frac{1}{2}-\epsilon_1,\frac{1}{2}+2\epsilon_1,\frac{1}{2}+\epsilon_1\}$ *into two bins.* $\mathcal{OPT}$ *however, can pair up each* $(\frac{1}{2}-\epsilon_i)$ *with* $(\frac{1}{2}+\epsilon_i)$ *and then pack every three* $(\frac{1}{2}+2\epsilon_i)$*'s into two bins. Therefore*

$$\frac{|\mathcal{INC}(L')|}{|\mathcal{OPT}(L')|}=\frac{2n}{n+\frac{2}{3}n}=\frac{6}{5}$$

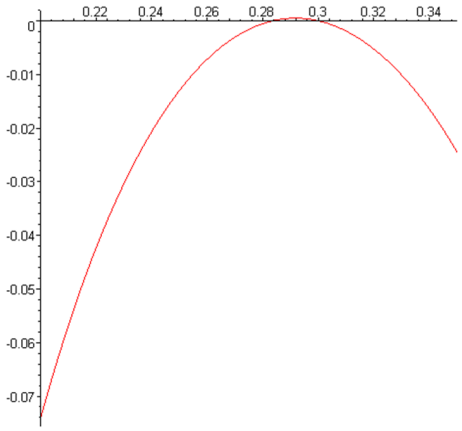

Figure 7.3: Horizontal axis $x = \delta$. Vertical axis $y = 1 - (4\varepsilon_3 + 3\varepsilon_{10} + 2\varepsilon_7 + \varepsilon_{12})$.

7.3 Better Approximation Ratio for *large* Weights

Similar to Algorithms $\mathcal{A}$ and $\mathcal{B}$, Algorithm $\mathcal{INC}$ also performs better for *large* weights. A crucial difference is that it does not need the restriction that upper bounds all weights by 1. This makes the performance guarantee of Algorithm $\mathcal{INC}$ given below much more attractive.

Theorem 7.2. *Given any list of weights $L = (w_1, w_2, \ldots, w_n)$ where each $w_i > 1/4$, Algorithm $\mathcal{INC}$ is at least 4/3-optimal.*

Proof: Classify the bins used in $\mathcal{INC}(L)$ into three categories. Let u denote the number of full bins, g the number of partially filled bins each with total weight at least $1/2$, v the number of partially filled bins each with total weight less than $1/2$. Because $w_i > 1/4$, we now have

$$\begin{aligned} |\mathcal{INC}(L)| &= u + g + v \\ |\mathcal{OPT}(L)| &\geq \max\{u + \frac{1}{2}g + \frac{1}{2}v, \frac{1}{2}u + g + v\} \end{aligned}$$

due to the constraints on total weight and cardinality for each bin.

Case 1: If $u > g + v$, we have

$$\begin{aligned}\frac{|\mathcal{INC}(L)|}{|\mathcal{OPT}(L)|} &= \frac{u+g+v}{u+\frac{1}{2}g+\frac{1}{2}v} \\ &= 1+\frac{\frac{1}{2}g+\frac{1}{2}v}{u+\frac{1}{2}g+\frac{1}{2}v} \\ &= 1+\frac{1}{1+\frac{u}{\frac{1}{2}(g+v)}} \\ &< 1+\frac{1}{1+2}=\frac{4}{3}\end{aligned}$$

Case 2: If $u \leq g + v$, we have

$$\begin{aligned}\frac{|\mathcal{INC}(L)|}{|\mathcal{OPT}(L)|} &= \frac{u+g+v}{\frac{1}{2}u+g+v} \\ &= 1+\frac{\frac{1}{2}u}{\frac{1}{2}u+g+v} \\ &= 1+\frac{1}{1+\frac{g+v}{\frac{1}{2}u}} \\ &< 1+\frac{1}{1+2}=\frac{4}{3}\end{aligned}$$

This proves our theorem. □

7.4 General kBPS

7.4.1 The $\mathcal{INC}_k$ algorithm

For the general kBPS problem, algorithm $\mathcal{A}_k$ has an asymptotic approximation ratio of $(2 - 1/k)$. Using a generalization of Algorithm $\mathcal{INC}$, we can improve the approximation ratio to 3/2 for $k = 3$ and to $(2 - 2/k)$ for any $k \geq 4$. These improvements are significant compared with the ε-improvement we presented in the previous section. However, similar argument fails to achieve improved ratio for $k = 2$, therefore suggesting the need for new techniques to demonstrate Algorithm $\mathcal{INC}$'s good performance for 2BPS .

Let $L = (w_1, w_2, \ldots, w_m)$ be a given list of weights. We give the following algorithm $\mathcal{INC}_k$ as a natural generalization of $\mathcal{INC}$. The running time of $\mathcal{INC}_k$ is still polynomial in the form $O(m^2 \log m)$.

Algorithm $\mathcal{INC}_k$

- Sort L into non-decreasing order;
- Loop until all weights are packed:
  - Start from the smallest, put at most $(k-1)$ current minimum weights into a new bin;
  - If this bin is completely filled (so now it becomes dead),
    - Insert the possible left-over weight back into the sorted list with respect to the order;
    - Continue;
  - Else
    - Put the maximum possible part of the current largest weight into this live bin (so after this step this bin becomes dead);
    - Insert the possible left-over weight back into the sorted list with respect to the order;

Algorithm $\mathcal{INC}_k$ is a natural generalization of $\mathcal{INC}$ for any fixed $k \in \mathcal{Z}^+$. Note that now a *live* bin refers to a partially filled bin with fewer than k types. Algorithm $\mathcal{INC}_k$ can be shown to achieve an approximation ratio 3/2 for $k = 3$ and no more than $(2 - 2/k)$ for any $k \geq 4$, which improves upon the $(2 - 1/k)$ approximation ratio of Algorithm $\mathcal{A}_k$.

7.4.2 $(2 - 2/k)$-approximation

Theorem 7.3. *Algorithm $\mathcal{INC}_k$ achieves an asymptotic approximation ratio of 3/2 for $k = 3$ and $(2 - 2/k)$ for any fixed $k \geq 4$.*

Proof. Given any list of weights $L = (w_1, w_2, \ldots, w_m)$ where each $w_i > 0$, let $\mathcal{INC}_k(L)$ denote the packing generated by Algorithm $\mathcal{INC}_k$ and $\mathcal{OPT}(L)$ an optimal packing. We also denote the corresponding number of bins used by $|\mathcal{INC}_k(L)|$ and $|\mathcal{OPT}(L)|$.

Suppose $\mathcal{INC}_k(L)$ consists of n full bins and αn partially filled bins for some integer $n \geq 0$ and some real number $\alpha \geq 0$.

Claim 7.12. *Asymptotically, all partially filled bins in $\mathcal{INC}_k(L)$ have k different types of weights in each.*

Proof. Algorithm $\mathcal{INC}_k$ is done working with a bin only when it becomes *dead*. If a dead bin is only partially filled, it has to contain k types of weights in it. □

Now we consider the following two cases.

Case 1: There is no bin in $\mathcal{INC}_k(L)$ with fewer than k types of weights in it.

If this is the case, all $(1+\alpha)n$ bins in $\mathcal{INC}_k(L)$ each holds k different types. Therefore each must *retire* at least $(k-1)$ types. And the total number of different types of weight

$$m \geq (k-1)(1+\alpha)n$$

Because of the cardinality constraint, we have

$$|\mathcal{OPT}(L)| \geq m/k \geq \frac{k-1}{k}(1+\alpha)n$$

which guarantees an approximation ratio

$$\begin{aligned} \frac{|\mathcal{INC}_k(L)|}{\mathcal{OPT}(L)|} &\leq \frac{k}{k-1} \\ &= 1 + \frac{1}{k-1} \end{aligned} \tag{7.19}$$

Case 2: There is at least one bin B_i in $\mathcal{INC}_k(L)$ with fewer than k types of weights in it.

If this is the case, we claim that

Claim 7.13. *Asymptotically, all αn partially filled bins in $\mathcal{INC}_k(L)$ must have total weight at least $\frac{1}{k-1}$ in each bin.*

Proof. Let B_i be the first such bin created in $\mathcal{INC}_k(L)$ with fewer than k types of weights in it. From Claim 7.12, B_i must be a full bin. Therefore the largest weight piece in B_i must have size at least $\frac{1}{k-1}$.

When B_i is created, all weights in B_i are the smallest in the list. With $\mathcal{INC}_k$, all bins created subsequently must have total weight $\geq \frac{1}{k-1}$ in each. For all bins created prior to B_i, it has to have packed part or whole of the largest weight at the time of creation. Therefore they also must have total weight $\geq \frac{1}{k-1}$ in each. □

From this claim, we can calculate the total weight in $\mathcal{INC}_k(L)$ is at least

$$n + \frac{1}{k-1}\alpha n$$

which guarantees an approximation ratio

$$\begin{aligned} \frac{|\mathcal{INC}_k(L)|}{\mathcal{OPT}(L)|} &\leq \frac{1+\alpha}{1+\frac{1}{k-1}\alpha} \\ &\leq 2 - 2/k \end{aligned} \tag{7.20}$$

Taking the maximum of (7.19) and (7.20), Theorem 7.3 follows immediately. □

Algorithm $\mathcal{INC}_k$ performs well for the general kBPS problem, improving the approximation ratio from 4/3 to 3/2 for $k = 3$ and from $(2-1/k)$ to $(2-2/k)$ for $k \geq 4$. These improvements are quite significant. Unfortunately, the same argument fails to give an improvement when $k = 2$ because $1 + \frac{1}{k-1} = 2$. We conjecture that the approximation ratio of $\mathcal{INC}$ for 2BPS can be improved significantly as well, but we may need new techniques for that purpose.

Remark 7.2. *The best worst-case example we have for $\mathcal{INC}_k$ can only ensure a $\left(\frac{k}{k-1} - \frac{1}{(k-1)^2}\right)$ ratio, which is 5/4 for $k = 3$ and 11/9 for $k = 4$, etc. Consider a list L' consisting of t weights of size ϵ for some tiny $\epsilon > 0$, t weights of size $(\frac{1}{k-1} - \epsilon)$ and $(k-2)t$ weights of size $\frac{1}{k-1}$. An optimal packing is to pack each set of one weight of size ϵ with one weight of size $(\frac{1}{k-1} - \epsilon)$ and $(k-2)$ weights of size $\frac{1}{k-1}$ fully into one bin. Therefore $|\mathcal{OPT}(L')| = t$. Algorithm $\mathcal{INC}_k$ however, will pack every two weights of size ϵ with one weight of size $\frac{1}{k-1}$. The rest of the weights can be assumed to pack perfectly. Therefore*

$$\begin{aligned} |\mathcal{INC}_k(L')| &\geq \frac{t}{k-1} + \frac{1}{k-1}\left(t + (k-2)t - \frac{t}{k-1}\right) \\ &= \frac{k}{k-1}t - \frac{1}{(k-1)^2}t \end{aligned}$$

which gives us the desired ratio. Notice that this ratio approximates $\left(1 + \frac{1}{k-1}\right)$ as k becomes large.

7.4.3 Discussion

The $\mathcal{INC}_k$ algorithm needs to keep a sorted list of weights at all times, thus requires an offline setting. Contrary to popular bin packing algorithms that process the list of weights in the order of non-increasing size such as FFD and BFD, the $\mathcal{INC}_k$ algorithm processes in the order of non-decreasing size. The cardinality constraint makes a vital difference in algorithm design.

Given that Algorithm $\mathcal{INC}_k$ provides strictly better performance guarantees with reasonable running time, it is desirable to extend it to the online settings as well. If some amount of look-ahead is possible in the online setting, we can apply $\mathcal{INC}_k$ as a heuristic to the weights we can expect in the look-ahead window. We believe that such a heuristic would at least be powerful in some real applications given a suitable amount of look-ahead.

7.5 Acknowledgement

This chapter contains material submitted for publication. I would like to thank my co-author Ron Graham.

8

Dynamic Approximation

8.1 Compaction vs. Packing

For optimization problems such as kBPS , offline approximation algorithms are static. In many real applications, however, allocation requests can come incrementally and have to be handled one at a time without knowledge about future requests. Online algorithms are designed specifically for this need. A classical example of online problems is caching and paging. Given a cache that can hold k pages, the sequence of requested pages are revealed one at a time. For every requested page, if it is in the cache, we simply return that page, otherwise, we have to evict one of the pages in the cache and replace it with the requested page. Requests must be handled one at a time without knowledge of future requests.

In the framework of competitive analysis, solutions computed by the online algorithms are compared with the best possible solution that could have been computed with unlimited computational power and complete advance knowledge of the whole input sequence.

We are in need of yet a more practical setting. Consider 2BPS and the application for memory allocation to pipelined processors in high-speed routers. In addition to the typical online setting where the allocation request comes in incrementally, in practice, we would also like to handle deallocation requests which correspond to the possibility that a previously required memory block may no longer be needed after some time. This setting is generic for other potential applications of kBPS and we call it the *dynamic* setting. To handle deallocation requests, we analyze the different cases when repacking may be allowed to a different extent depending on the specific application. Therefore another tradeoff between memory utilization and memory repacking cost needs to be considered. The question is how can we get good allocation algorithms that maintain overall efficiency in the *dynamic* setting.

We define the *compaction ratio* γ below as a parameter that captures this tradeoff. Depending on the allowable range of γ, the algorithm design criteria change. In particular, when γ is bounded by a constant from above, a dynamic algorithm $\mathcal{DYN}$ was given that achieves a 3/2 approximation ratio with $\gamma_C \leq 1$.

Definition 8.1. *The compaction ratio of any dynamic* kBPS *algorithm C is defined to be*

$$\begin{aligned} \gamma_C &= \max_t \tfrac{M}{W} \\ &= \tfrac{\textit{total repacked weight up to time } t}{\textit{total packed weight up to time } t} \end{aligned}$$

8.2 Algorithm $\mathcal{DYN}$

Depending on what range of γ is allowed, the algorithm design criteria can change. In particular, when γ is bounded by a constant from above, an online algorithm $\mathcal{DYN}$ is given that achieves a 3/2 approximation ratio with $\gamma_{\mathcal{DYN}} \leq 1$.

Case a: γ can be arbitrarily large.

If repacking or compaction is assumed of negligible cost and we have unlimited computing power, we can just solve the offline packing problem every time a new memory request comes in using the best approximation algorithm and perform repacking whenever needed. In practice, however, compaction almost certainly has a cost that is not negligible, especially in the memory allocation application to pipelined processors because it has to perform memory operations. Computation cost should also be taken into consideration since we have already seen that the offline allocation problem is NP-hard.

Case b: γ is bounded from above. In particular, $\gamma \leq c$ where $c > 0$ is a constant.

We now give an online allocation algorithms subject to the cardinality constraint with compaction ratio $\gamma \leq 1$.

Algorithm $\mathcal{DYN}$

- Upon any allocation request w_i, we say it is *large* if $w_i > 1/2$,
otherwise we say it is *small*.
    - A large allocation request is always put into a new bin.
    - A small request is put into a bin with only one small request if possible, otherwise it is put into a new bin.
- Upon any deallocation request,
    - If it results in two bins each with only one small request, move the *smaller* piece to double up the two requests.

Theorem 8.1. *Algorithm $\mathcal{DYN}$ always generates bin packing of size of size $|\mathcal{DYN}|$ which satisfies*

$$|\mathcal{DYN}| \leq (3/2)|\mathcal{OPT}| + 1$$

Proof. At any point of time, we can describe the packing produced by Algorithm $\mathcal{DYN}$ as t bins each with one large request, r bins each with two small requests,

with possibly one more bin with only one small request. Given these requests, how much better can the optimal packing strategy be?

First, let us suppose there is no bin with only one small request. Algorithm $\mathcal{DYN}$ uses $(t + r)$ bins.

Claim 4:

$$|\mathcal{OPT}| \geq \begin{cases} r + t/2 & \text{when } r \geq t/2 \\ 2/3(r + t) & \text{when } r < t/2 \end{cases}$$

Proof of Claim 4: The first inequality $|\mathcal{OPT}| \geq r + t/2$ always holds because of the cardinality constraint, since we have a total of $(2r + t)$ different types of weights. The exact bound could be achieved when each large weight can be paired up with a small weight into one bin and $t = 2r$.

When $r < t/2$, not all large weights can be paired up. The paired up ones would use $2r$ bins and the remaining $(t - 2r)$ large weights have to occupy at least $2/3(t - 2r)$ bins (proved in Claim 5). This adds up to a total of $2/3(r + t)$ bins and proves Claim 4.

Case a: $r \geq t/2$

$$\begin{aligned} \frac{|\mathcal{DYN}|}{|\mathcal{OPT}|} &\leq \frac{r+t}{r+t/2} \\ &= 1 + \frac{t/2}{r+t/2} \\ &\leq 1 + 1/2 \\ &= 3/2 \end{aligned}$$

Case b: $r < t/2$

$$\begin{aligned} \frac{|\mathcal{DYN}|}{|\mathcal{OPT}|} &\leq \frac{r+t}{2/3(r+t)} \\ &= 3/2 \end{aligned}$$

In the case where there is one bin having a single small request, the previous argument applies replacing $|\mathcal{DYN}|$ by $|\mathcal{DYN}| - 1$. It remains to prove Claim 5.

Claim 5: Given m large weights, it is impossible to pack them into fewer than $\lceil \frac{2}{3}m \rceil$ bins.

Proof of Claim 5: m bins would be necessary if we do not split any weights. Suppose we break α of the m large weights each into two small pieces. We have a total of

$$2\alpha + (m - \alpha) = m + \alpha$$

pieces in which $(m - \alpha)$ pieces are still large and need $(m - \alpha)$ bins. Therefore we have

$$|\mathcal{OPT}| \geq \max\{(m + \alpha)/2, m - \alpha\}$$

The minimum of the right hand side occurs when $(m+\alpha)/2 = m-\alpha$ and is 2/3m. This proves our claim.

This completes the proof of Theorem 8.1. □

Theorem 8.2. *Algorithm $\mathcal{DYN}$ has compaction ratio $\gamma_{\mathcal{DYN}} \leq 1$.*

Proof. Call any memory piece that has not been swapped *clean* and otherwise *dirty*. Recall the compaction ratio is defined as $\gamma = M/W$. We claim that

$$W - M \geq \sum \text{size of all clean pieces}$$

This is because

- Before any deallocation request, our bin packing consists of only *clean* pieces which contribute to W.
- If any clean piece is moved due to a deallocation request, it becomes *dirty* and contributes to the numerator M.

- If a dirty piece is deallocated, there is no effect on $W - M$.
- In a shared bin of one clean piece and one dirty piece, the clean piece always has size no smaller than the dirty piece according to our algorithm. When the clean piece is deallocated, the weight it contributes to W is still there, so instead of throwing it away, we use its weight to *recharge* the dirty piece into a clean piece. This would also guarantee that no dirty piece will ever be moved again.

Note that $\sum$size of all clean pieces ≥ 0 at all times so $W - M \geq 0$, i.e., $\gamma = M/W \leq 1$.

We also observe that 1 is a tight bound for γ_C using the following memory request sequence. We begin with two allocation requests, each with weight $\alpha < 1/2$. We then allocate one more weight of size α, and follow this with a deallocation of one of the paired weights. This will cause the algorithm to move a weight. We repeat this process until it has been performed s times altogether. Thus, we will have allocated $s + 2$ weights and moved s weights, all of size α. This clearly gives a compaction ratio arbitrarily close to 1 as s goes to infinity. □

Case c: $\gamma = 0$, i.e., no compaction is allowed.

When the compaction cost is relatively high and we cannot afford to swap any memory bits, we first claim that any online algorithm $\mathcal{D}$ in this case cannot have an approximation ratio better than 2. This can be shown using the following memory request sequence.

For a large integer k, we are first given a list of k memory allocation requests each with size 1/3. Observing the allocation assignment made by the online algorithm $\mathcal{D}$, we are then given a list of $\lfloor \frac{k}{2} \rfloor$ deallocation requests which remove exactly one weight from every shared bin. At the end, the online algorithm will have each bin

holding one weight while the optimum packing uses only half as many bins. This would give us the lower bound of 2 on the approximation ratio.

A simple online algorithm that achieves approximation ratio 2 in this case is just to never share any bin. Given the cardinality constraint, the bin packing generated by this algorithm is always within a factor of 2 from the optimum.

8.3 Acknowledgement

This chapter contains material appearing in "Parallelism vs. Memory Allocation in Pipelined Router Forwarding Engines" from *Theory of Computing Systems (ToCS), ISSN 1432-4350 (Print) 1433-0490 (Online) 2006*. I would like to thank my co-authors Fan Chung, Ron Graham, and George Varghese.

9

Summary and Discussion

9.1 Summary

As traditional computation models evolve to become increasingly distributive and interactive, we need more tools and insight to study the limits of these models and how to achieve those limits through well-designed algorithms. A central characteristic of these computation models is the restriction that not every piece of the information needed for decision making is available a priori. In this dissertation, we study some of these interactive and dynamic computations and design efficient algorithms to solve them. To measure the performance of these algorithms, we use a *worst-case* analysis approach through the incorporation of imaginary adversaries or adversarial input sequences. This approach gives us better guarantees than the *average-case* analysis as we have little or no knowledge about what the input sequences will possibly be. These computational problems can be conveniently studied in general game-theoretic frameworks. In particular, the interactive computations can be studied as a *query-answer* game, while the online computations can be studied as a *request-answer* game.

In our analysis, we extensively use graphical representations to capture the configuration which evolves as the computation progresses, naturally adapting to an interactive or dynamic computation environment. Utilizing graphs not only helps us visualize the actual computations step by step, but also enables us to incorporate powerful mathematical tools to our advantage.

In the first part of this dissertation, we focus on a particular type of interactive game, called the Majority/Plurality game. Among the many variants of this game, we have proven several of the current best bounds on the minimum length of the questioner's winning strategies in the adaptive and oblivious settings. We have also considered error-tolerant strategies to combat possible communication errors or failures. With a graphical representation of the game configuration, our proofs are mostly combinatorial, sometimes utilizing powerful tools such as expander graphs, spectral graph theory, and probabilistic methods.

In the second part of this dissertation, we study a new variant of the classical bin packing problem - kBPS . It can be shown to be NP-hard even in the simplest case. With graphical representations of the packings generated, we devise efficient approximation algorithms for kBPS in the offline, online, and dynamic settings. An ε-improvement technique is used to prove one of the approximation ratio results.

9.2 Discussion

As we study the limits of these computations, many gaps remain between various upper and lower bounds. In particular, there is a fairly big gap on the bounds for MO_*. There is also an interesting gap to be closed for the approximation ratio of $\mathcal{INC}$ for 2BPS and $\mathcal{INC}_k$ for kBPS .

Consideration of fault-tolerance may also be useful for other variants of the

majority game such as the plurality game. Consideration of other types of error-tolerance may also be interesting, such as bounded error fraction or random errors. There are also numerous possibilities to adapt the different setting parameters of the majority/plurality games to come up with new models. For example, we may further generalize the notion of plurality so that the goal is to identify a representative element for every label that occurs above some percentage threshold. This has similar flavor with recent research in identifying "hot"(frequency above some threshold) items in data streams with applications in network congestion monitoring, data mining and web query log analysis [34] [35] [40].

The ε-improvement technique can only show that Algorithm $\mathcal{INC}$'s performance strictly better than Algorithms $\mathcal{A}$ and $\mathcal{B}$, although the best worst-case example we managed to find has a much better ratio. Similarly, the approximation ratio we proved for Algorithm $\mathcal{INC}_k$ and the best worst-case examples we gave also have a gap. Closing the gaps would be interesting directions to pursue. Algorithm $\mathcal{DYN}$ is designed for 2BPS only. Efficient algorithm design for general kBPS in the dynamic setting can also be of great interest.

APPENDIX A: Proof of Lower Bound of MO_2 [67]

First the monotonicity of $MO_2(n)$ can be shown, i.e.,

Lemma 9.1.

$$MO_2(n+1) \geq MO_2(n)$$

We now prove the bound for even $n = 2t$. With Lemma 9.1, the bound for odd n is then implied.

Rules of A's Strategy

- With the answers given so far, there always exists a labelling (consistent with all given answers) with exactly t 0-labelled elements and t 1-labelled elements.
- From sum and absolute difference, choose sum iff $P(|J_{M^+}|) < P(|J_{M^-}|)$

We will specify a strategy for **A** that forces **Q** to proceed all the way to the needed number of rounds. First rule of this strategy is that we should have t 0-labelled elements and t 1-labelled elements therefore the game ends in configuration $(0, 0, ..., 0)$.

Let $[k]$ be the shorthand notation for $\{1, 2, ..., k\}$, and let $m_I = \Sigma_{i \in I} m_i$ for $I \subseteq [k]$. Then the first rule of our strategy ensures that after each round there exists a set I of indices such that $m_I = m_{[|M|]\setminus I}$. Denote the set of such index sets by J_M for the current configuration M, so $J_M \neq \phi$.

The following lemma can also be proved.

Lemma 9.2. $|J_M| = |J_{M^+_{i,j}}| + |J_{M^-_{i,j}}|$ *for all* $i, j \in [|M|]$.

Now let $P(m)$ denote the largest power of 2 dividing m. It is not difficult to see that $P(a+b) \geq min(P(a), P(b))$ and that $P(\binom{2m}{m}) = \mu_2(m)$ (i.e., the number of 1's in the binary expansion of m).

The second rule of **A**'s strategy is to choose sum iff $P(|J_{M^+}|) < P(|J_{M^-}|)$. By Lemma 9.2, we have $P(|J_M|) = P(|J_{M^-}| + |J_{M^+}|) \geq min(P(|J_{M^-}|), P(|J_{M^+}|))$. Because of the second rule, $P(|J_M|)$ cannot increase during the course of the game. The beginning configuration has $M = (1, 1, ..., 1)$ with $|M| = 2t$ and $|J_M| = \binom{2t}{t}$; the terminal configuration has $M' = (0, 0, ..., 0)$ with $|M'| = k$ and $|J_M| = 2^k$. Then we have

$$k = P(2^k) = P(|J_{M'}|) \leq P(|J_M|) = P\left(\binom{2t}{t}\right) = \mu_2(t) = \mu_2(n)$$

Therefore the number of rounds is

$$n - k \geq n - \mu_2(n).$$

APPENDIX B: Proof of Theorem 6.1 [21]

3-PARTITION

Instance: A set A of $3m$ elements, a bound $B \in \mathbb{Z}^+$, and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = mB$.

Question: Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$ (note that each A_i must therefore contain exactly 3 elements from A)?

For a given instance of the 3-PARTITION problem as described above, we consider the following decisional version of 2BPS :

(*) We are given a list W of $3m$ weights

$$w_a = \frac{1}{2} + \frac{s(a)}{2B}$$

Determine if W can be packed into $2m$ bins such that no bin contains more than two types.

It suffices to show that the 3-PARTITION problem has an affirmative solution if and only if the above problem (*) has a solution.

First we consider the easy direction. Suppose the 3-PARTITION problem has a solution $A_1, A_2, \ldots, A_m$. For each i, we can pack the weights $w(a)$, for $a \in A_i$ into two bins since

$$\sum_{a \in A_i} w_a = \frac{3}{2} + \sum_{a \in A_i} \frac{s(a)}{2B} = 2$$

and w_i satisfies

$$\frac{3}{8} < w_i = \frac{1}{2} + \frac{s(a)}{2B} < \frac{3}{4}.$$

So, W can be packed into $2m$ bins when each bin has two types and thus problem (*) is solved.

Now suppose problem (*) has a solution with a packing into $2m$ bins. Clearly, each bin contains parts summing up to 1 since $\sum_a w_a = 2m$.

First, we observe that a weight type can not be partitioned into more than 2 parts. Suppose the contrary. There is a weight type, say w_1, that is partitioned into k parts which are contained in k bins where $k \geq 3$. One of the parts is less than $1/4$ since $w_1 < 3/4$. The bin that contains this small part can contain another part with weight at most $3/4$. Thus, this bin can not have parts summing up to 1, which is impossible.

Second, we claim that the number t of types of weights that are packed in two bins is exactly m. Suppose t is more than m. The total number of parts is more than $4m$. Since at most two parts can be packed into one bin, we need more than $2m$ bins, which is a contradiction. Now, suppose that t is fewer than m. Since there are at most $2t$ bins that can contain parts of two types, there are at least two bins that can contain at most one type. Those two bins can not have parts summing up to 1, which is again a contradiction.

Hence, there are exactly m weights S that are each partitioned into two parts. We write

$$S = \{a_{j_1}, a_{j_2}, \ldots, a_{j_m}\}$$

We consider A_i consisting of a_{j_i} and the types w_a that are contained in bins containing parts of $w_{a_{j_i}}$. Clearly A_i, $i = 1, \ldots, m$, is a partition of A. Furthermore, we have

$$\sum_{a \in A_i} w_a = \frac{3}{2} + \sum_{a \in A_i} \frac{s(a)}{2B} = 2$$

This implies that

$$\sum_{a \in A_i} s(a) = B$$

Thus, this gives a solution to the 3-PARTITION problem. This proves Theorem 6.1

Bibliography

[1] M. Aigner, "Variants of the majority problem", *Applied Discrete Mathematics,* **137***, (2004), 3-25.*

[2] M. Aigner, G. De Marco, M. Montangero, "The Plurality Problem with Three Colors and More", *Theoretical Computer Science, 337 (2005), 319-330.*

[3] N. Alon, "Eigenvalues and Expanders", *Combinatorica* **6** *(1986), 86-96.*

[4] L. Alonso, E. Reingold, R. Schott, "Determining the Majority", *Information Processing Letters* **47***, (1993), 253-255.*

[5] L. Alonso, E. Reingold, R. Schott, "Average-case Complexity of Determining the Majority", *SIAM J. Computing* **26***, (1997), 1-14.*

[6] L. Alonso, E. Reingold, "Determining Plurality", *manuscript, 2006*

[7] L. Alonso, E. Reingold, "Average-Case Analysis of Some Plurality Problems", *manuscript, 2006*

[8] L. Alonso, E. Reingold, "Average-Case Lower Bounds for the Plurality Problem", *manuscript, 2006*

[9] L. Babel, B. Chen, H. Kellerer, and V. Kotov, *Algorithms for on-line bin-packing problems with cardinality constraints*, Discrete Applied Mathematics 143 (2004), 238-251.

[10] A. Basu and G. Narlikar, "Fast Incremental Updates for Pipeline Forwarding Engines", *InfoCom 2003.*

[11] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson, "On the power of randomization in on-line algorithms", *STOC 1990, 379-386.*

[12] C. Berge, "Graphs and Hypergraphs", *North-Holland, Amsterdam, 1976.*

[13] E. R. Berlekamp, "Block Coding for the Binary Symmetric Channel with Noiseless, Delayless Feedback", *Error Correcting Codes (Proc. Sympos. math. Res. Center, Madison, Wis., 1968, 61-88.*

[14] P. M. Blecher, "On a Logical Problem", *Discrete Mathematics* **43**, *(1983), 107-110*

[15] Guy E. Blelloch, Phillip B. Gibbons and Yossi Matias "Accounting for Memory Bank Contention and Delay in High-Bandwidth Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems, volume 8, 1997.*

[16] B. Bollobás, "Random graphs", 2nd ed., *Cambridge Studies in Advanced Mathematics, 73. Cambridge University Press, Cambridge, 2001*

[17] A. Borodin and R. El-Yaniv, "Online Computation and Competitive Analysis", *Cambridge University Press, Cambridge, 1998*

[18] S. Butler, J. Mao and R. L. Graham, "How to Play the Majority Game with Liars", *Submitted, 2006*

[19] M. Charikar, V, Guruswami, and A. Wirth, "Clustering with Qualitative Information", *FOCS'03, Cambridge, 524-533*

[20] F. R. K. Chung, "Spectral Graph Theory", *CBMS Lecture Notes, AMS Publications.*

[21] F. R. K. Chung, R. L. Graham, and G. Varghese, "Parallelism vs. Memory Allocation in Pipelined Router Forwarding Engines", *SPAA 2004, 103-111*

[22] F. R. K. Chung, R. L. Graham, J. Mao, and G. Varghese, "Parallelism vs. Memory Allocation in Pipelined Router Forwarding Engines", *Theory of Computing Systems (TOCS), ISSN 1432-4350 (Print) 1433-0490 (Online), 2006*

[23] F. R. K. Chung, R. L. Graham, J. Mao, and A. C. Yao, "Finding favorites", *Electronic Colloquium on Computational Complexity(ECCC) (078): 2003*

[24] F. R. K. Chung, R. L. Graham, J. Mao, and A. C. Yao, "Oblivious and Adaptive Strategies for the Majority and Plurality Problems", *The 11th International Computing and Combinatorics Conference (COCOON) 2005: 329-338*

[25] F. R. K. Chung, R. L. Graham, J. Mao, and A. C. Yao, "Oblivious and Adaptive Strategies for the Majority and Plurality Problems", *Journal version to appear in Springer Algorithmica, 2006*

[26] F. R. K. Chung and L. Lu, "Concentration inequalities and martingale inequalities – a survey", *To appear in Internet Math, 43 pp.*

[27] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "Approximation Algorithms for Bin Packing: a Survey", *Approximation Algorithms for NP-Hard Problems*, D. Hochbaum (ed.), PWS Publishing, (1996), 46-93.

[28] E. G. Coffman, J. Csirik, and G. Woeginger, "Approximate Solutions of bin packing problems", *Handbook of Applied Optimization, Oxford University Press, 2002.*

[29] D. Culler, J, Singh, and A. Gupta. "Parallel Computer Architecture, A Hardware/Software Approach", *Morgan Kaufman , 1999.*

[30] M. Degermark, A. Brodnik, S. Carlsson, and Stephen Pink, Small forwarding tables for fast routing lookups, *Proc. SIGCOMM*, (1997), 3-14.

[31] D.-Z. Du and F. K. Hwang, "Combinatorial Group Testing and its Applications", *World Scientific Publishing Co., Inc., River Edge, NJ, 1993.*

[32] Z. Dvorak, V. Jelinek, D. Kral, J. Kyncl, and M. Saks, "Three optimal algorithms for balls of three colors", *STACS 2005*

[33] P. Erdős and A. Rényi, "On random graphs. I", *Publ. Math. Debrecen* **6** *(1959), 290-291.*

[34] C. Estan and G. Varghese, "New directions in traffic measurement and accounting", *Proc. of SIGCOMM Internet Measurement Workshop, ACM, 2001*

[35] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, J. Ullman, "Computing iceberg queries efficiently", *Proc. of 24th International Conf. on Very Large Data Bbases, 299-310, 1998*

[36] M. J. Fischer and S. L. Salzberg, "Finding a Majority among n Votes", *J. Algorithms 3: 375-379 (1982).*

[37] R. W. Floyd and D. E. Knuth, "The Bose-Nelson sorting problem", *A survey of combinatorial theory (Proc. Internat. Sympos., Colorado State Univ., Fort Collins, colo.,) 1971, pp. 163-172, North-Holland, Amsterdam, 1973.*

[38] M. R. Garey and D. S. Johnson, "Computer and Intractability, A Guide to the Theory of NP-completeness", *W. H. Freeman and Co., San Francisco, 1979.*

[39] M. R. Garey and D. S. Johnson, " Complexity results for multiprocessor scheduling under resource constraints", *SIAM J. Comput., (1975), 397-411.*

[40] R. M. Karp and S. Shenker, "A Simple Algorithm for Finding Frequent Elements in Streams and Bags", *ACM Transactions on Database Systems, Vol.28, No.1, 2003, 51-55*

[41] H. Kellerer, and U. Pferschy, *Cardinality constrained bin-packing problems*, Annals of Operations Research 92 (1999), 335-348.

[42] D. E. Knuth, *personal communication.*

[43] D. E. Knuth, "The Art of Computer Programming, Volume 3. Sorting and Searching", *Addison-Wesley Publishing Co., Reading, Mass., 1973.*

[44] D. Kral, J. Sgall, and T. Tichỳ, "Randomized strategies for the plurality problem", *manuscript, 2005*

[45] K. L. Krause, V. Y. Shen and H. D. Schwetman, "Analysis of several task-scheduing algorithms for a model of multiprogramming computer systems", *Journal of the ACM (22), 1975, 522-550.*

[46] K. L. Krause, V. Y. Shen and H. D. Schwetman, "Analysis of several task-scheduing algorithms for a model of multiprogramming computer systems, Errata", *Journal of the ACM (24), 1977, 527.*

[47] T.V. Lakshman and D. Staliadis, "High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching", *Proc. ACM SIGCOMM '98, 1998.*

[48] A. Lubotzky, R. Phillips and P. Sarnak, "Ramanujan graphs", *Combinatorica* **8** *(1988), 261-277.*

[49] N. Linial and A. Wigderson, "Expander Graphs and their Applications", *Lecture notes. The Hebrew University, Israel*

[50] J. Mao and R. L. Graham, "Parallel Resource Allocation of Splittable Items with Cardinality Constraints", *Submitted, 2006.*

[51] J. Moore, "Proposed problem 81-5", *J. Algorithms 2: 208-210 (1981).*

[52] A. Pelc, "Searching games with errors – fifty years of coping with liars", *Theoret. Comput. Sci., 270(1-2):71-109, 2002*

[53] A. Ranade, "How to emulate shared memory", *Journal of Computer and System Sciences, 42:307-326, 1991.*

[54] A. Rènyi, "On a problem in information theory", *Magyar Tud. Akad. Mat. Kutatò Int. Közl., 6:505-516, 1961.*

[55] J.M. Robson, "Storage allocation is NP-hard", *Information Processing Letters 11(3):119-125, 1980.*

[56] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", *IEEE Network, March/April 2001.*

[57] M. Saks and M. Werman, "On computing majority by comparisons", *Combinatorica* **11***(4) (1991), 383-387.*

[58] A. Saulsbury, F. Pong, and A. Nowatzyk, *Missing the Memory Wall: The Case for Processor/Memory Integration*, in Proceedings of the International Symposium on Computer Architechture, May 1996, pp. 90-101.

[59] H. Shachnai, and T. Tamir, *Polynomial time approximation schemes for class-constrained packing problems*, J. of Scheduling, vol. 4:6, 313-338, 2001.

[60] H. Shachnai, and T. Tamir, *Tight bounds for online class-constrained packing*, Theoretical Computer Science vol. 321, issue 1, 103-123, 2004.

[61] S. Sikka and G. Varghese, "Memory Efficient State Lookups with Fast Updates", *Proc. SIGCOMM 2000.*

[62] A. Silberschatz, G. Gagne, P. B. Galvin "Operating System Concepts", *Wiley Text Books, 6th edition (2002).*

[63] N. Srivastava, and A. D. Taylor, "Tight bounds on plurality", *Information Processing Letters 96 (2005) 93-95*

[64] A. Taylor and W. Zwicker, *personal communication.*

[65] S. M. Ulam, "Adventures of a mathematician", *Charles Scribner's Sons, New York, 1976.*

[66] L. Valiant, "A bridging model for parallel computation", *Communications of the ACM, 33(8), 1990.*

[67] G. Wiener, "Search for a majority element", *J. Statistical Planning and Inference 100 (2002), 313-318.*

[68] A. C. Yao, "New Algorithms for Bin Packing", *Journal of the ACM, Vol. 27, No. 2, 1980, 207-227.*