Randomized Optimization by Simplex-Type Methods

Dissertation
zur Erlangung des Doktorgrades

vorgelegt am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

von
Bernd Gärtner

verteidigt am 20. Dezember 1995

Betreuer:	Prof. Dr. Emo Welzl Institut für Informatik, Freie Universität Berlin
Zweitgutachter:	Prof. Dr. Friedhelm Meyer auf der Heide Heinz Nixdorf Institut, Universität-GH Paderborn

Contents

Preface and Acknowledgment

This thesis is about randomized optimization, and even though it is doubtful whether the work itself approaches any reasonable optimization criteria, I can definitely say that a lot of randomization was involved in the process of writing it. In fact, people have been the most valuable source of randomness. To begin with, the subject of this thesis is almost completely random. When I started back in October 1991, I had just finished my Diplomarbeit with Emo Welzl. This was about VC-dimension [19], and the only thing clear at that time was that I had to find something else to work on. As it happened, Micha Sharir and Emo had just developed a simple and fast randomized algorithm to solve linear programming and a whole class of other optimization problems (including e.g. the minimum spanning ball problem) in a unified abstract framework [45]. Independently (another accident!), Gil Kalai – working on the polytope diameter problem where he had obtained a striking new result [28] – came up with a randomized variant of the simplex algorithm, capable of solving linear programs in an expected subexponential [1] number of steps [27]. Since all prior bounds were exponential, this was a major step forward on the longstanding open question about the worst-case performance of the simplex algorithm. Kalai's simplex variant turned out to be closely related to the Sharir-Welzl algorithm, and together with Jirka Matoušek, they were able to prove that this algorithm was actually subexponential as well, at least for linear programming [33].

This little 'at least' led to my involvement in this research. It started with Emo asking me – quite innocently – whether I had any idea how to make the subexponential analysis applicable to the minimum spanning ball problem as well. The difficulty was that although the *algorithm* worked, the *analysis* didn't go through because it was using a special feature of linear programming.

Within short time, I was convinced that the problem was solved, and I proposed a solution, which quickly turned out to be wrong (like many ideas and 'solutions' still to come). This failure made me dig even deeper into the problem, and during the winter, the minimum spanning ball problem became 'my' problem. By March 1992 I had collected all kinds of half-baked ideas but none of them seemed to be any good. I was stuck. During that time, Franz Aurenhammer was my office-mate (remarkably, he left Berlin only a month later), and I kept telling him the troubles I had with the 'miniball' problem. On one of these occasions, from the depths of his memory, Franz recalled a paper by Rajan about d-dimensional Delaunay triangulations (this surely was no accident) where he remembered

[1] A function is called subexponential if its logarithm is sublinear.

having seen something mentioned about the miniball problem [42]. When I looked up the paper, I found that Rajan had noted a close connection between the miniball problem and the problem of finding the point in a polytope closest to the origin. I immediately had an idea about how to solve the latter problem![2] Not only that: it turned out that even before, I had, without noticing it, already collected all the ideas to make the subexponential analysis applicable to the whole class of abstract problems originally considered by Micha and Emo – except for the concrete miniball problem on which I had focused too hard to see beyond it. Only in the process of preparing a talk about all this for our notorious 'noon seminar' did I notice that even more general problems could be handled. This led to the concept of *abstract optimization problems* (AOPs) and a subexponential algorithm for solving them. Without Franz innocently leading me on the right track, I would not have achieved this.

Still, I was not quite happy. After having spent so many months on the problem, I found the solution too obvious and the algorithm – containing a recursive 'restart scheme' based on an idea in Kalai's paper – still too complicated. I spent another couple of weeks trying to simplify it, without any success. Meanwhile, Emo was trying to push forward publication of the new subexponential bounds and suggested that I should submit the results to the forthcoming *33rd Symposium on Foundations of Computer Science* (FOCS). I didn't want to submit it until I could simplify the algorithm, but I kept on trying in vain. Three days before the submission deadline, I happened to talk to Johannes Blömer. He was about to submit a paper to FOCS for his second time, and he told me that the first time he had been very unsure about the quality of his paper – just like me – but had (successfully) submitted it anyway [7]. He urged me to do the same, his main argument being: it can't hurt. This was quite convincing; within a day and a night I wrote down the current status of my research, and we got our submissions delivered together and on time, by express mail. In the end, both our papers got accepted [8, 18]; for Johannes, this was the beginning of the final phase of his thesis. For me it was a first and very encouraging start that I owe to Johannes.

After I returned from FOCS – which was an exciting event for me, partially because it was my first visit ever to the US – I tried to build on the results I had for AOPs. In particular, I was looking for lower bounds. After all, the AOP framework was so weak that it seemed possible to establish a superpolynomial lower bound on the runtime of any algorithm for solving them. Although I spent weeks and weeks, I must admit that my insights into this problem are still practically nonexistent. Another issue I tackled was trying to improve the subexponential upper bound. The exponent was of the order $\sqrt{d}$. Maybe one could get $\sqrt[3]{d}$, or even $\log^2 d$? There was reason to believe that such bounds were not impossible, at least for linear programming and other concrete problems. Well, I couldn't prove any.

On the one hand, I wanted to give up on this line of research, and on the other, I had no idea how to continue with my thesis if I did. For several months, I kept muddling

[2]Later I noticed that this was basically the idea already pursued by Wolfe [48].

along and made no considerable progress. Unfortunately, Emo was on sabbatical leave and therefore did not notice what was going on (and I didn't tell him, either).

At this point – when my depression reached a local maximum in the middle of 1993 – Günter Ziegler came to the rescue. He only happened to intervene because Jirka Matoušek was visiting us in Berlin at that time, and Günter took the opportunity to tell us both about one of his favorite open problems. It was about the behavior of the RANDOM-EDGE-SIMPLEX algorithm on the so-called Klee-Minty cube. Fortunately, he mentioned this application only in a side remark but otherwise presented the problem as a randomized flipping game on binary numbers that he would like to have analyzed. In this formulation, its pure combinatorial flavor immediately attracted me. Moreover, Günter cleverly adorned his presentation with stories about people having either claimed wrong solutions or tried in vain. Although I saw no immediate connection to my previous research, I instantly started working on the problem, partly because I was just lucky to have found something new and interesting to think about. Günter had appeared just at the right time, with the right problem.

History was beginning to repeat itself. Just like in the miniball case almost two years ago, the problem looked doable at first sight, but very soon turned out to be much harder than I had originally thought. This time the fall found me at my desk, filling one notepad after another. It was late December when the only visible result of my investigations was a vague idea how to analyze another flipping game that I thought might have something to do with the original one.

Emo had returned from his sabbatical and had gone back to the old habit of letting his graduate students report on their work in regular intervals. When I was due, I told him about my idea and what I thought it might give in the end. Almost instantly, Emo came up with a way of formalizing my vague thoughts, and within half an hour we set up a recurrence relation. By the next day, Emo had found a beautiful way of solving it, incidentally rediscovering Chebyshev's summation inequalities, Appendix (7.9), (7.10). Still, it was not clear what the bound would mean for the original game, but after this preliminary success, my mind was free enough to realize that the correspondence was in my notes already. I just had to look at things another way. Talking for half an hour with Emo had made all the difference.

Günter, who had noticed since December that things were gaining momentum, meanwhile had a couple of other results, and we put everything together in a technical report. Just to inform Jirka that we now had a solution to this problem that Günter had told us about months ago, I sent him a copy. Jirka not only liked the solution, he also found that the correspondence between the two flipping games was actually much more explicit (and much nicer) than I had realized. We gratefully incorporated his remarks into the paper, and in this version it finally ended up at FOCS '94 [21]. Like Johannes two years before, I had the end of my thesis in sight.

However, it took another year until it was finally written, during which two more accidents crucially determined the way it looks now. First, I discovered Chvátal's beautiful book on linear programming [10] and second, I visited Jirka in Prague. From Chvátal I

learned about the simplex method, and this had more impact on the style of this thesis than any other text I read. Of course, I always had in mind the geometric interpretation of the simplex algorithm as a path-following method on the boundary of a polyhedron, but never got around learning the actual algebra, mostly because I thought it was complicated and unnecessary. Just following Chvátal's initial example explaining how the method really works, was a revelation. Not only did I find it amazingly simple, I also discovered that arguing on the algebraic level is nothing to be afraid of! Geometry helps, but restricting oneself to geometric arguments can impede the view. Without this insight I would not have been able to write the chapter about convex optimization. Moreover, the whole approach of this thesis ('simplex is not everything, but everything is simplex') would have been different without Chvátal's book.

Finally, I still had the problem that my results on the AOPs did not naturally fit together with the material on the Klee-Minty cube (except probably under the very general 'everything is simplex' philosophy). Then I visited Jirka in Prague for a week. We talked about lower bounds for the RANDOM-EDGE-SIMPLEX algorithm in an abstract setting and what kinds of input might be useful in order to obtain such bounds. At first, this didn't seem to lead anywhere, but back home I decided in this context to reread Jirka's paper about lower bounds for an abstraction of the RANDOM-FACET-SIMPLEX algorithm [34]. Reading it then, I suddenly realized that the Klee-Minty cube was just a special case of the general examples Jirka had considered and that the techniques I had applied to it did actually work in this general setting as well. Although this insight has not so far led to new results, it provided exactly the missing link for embedding my result on the Klee-Minty cube very naturally into the chapter about lower bounds on abstract optimization.

All the people mentioned above have crucially influenced this thesis. Of course, there are other persons that contributed to it in one or the other way.[3] I would explicitly like to thank Günter Rote who showed an ongoing interest in any developments concerning the AOPs. In particular, he advertised them for me by giving a talk about the subject in China. Günter always used to look at AOPs in a different (and more general) way than I did; the conversations with him and the careful handwritten notes he made during preparation of his talk have been very inspiring. The only reason 'his' AOPs did not make it into this thesis is that I felt they would take me just a little to far out into the 'sea of abstraction' to be able to get back home safely.

Finally, the most decisive thanks go to my advisor Emo Welzl, for never loosing faith in me. I knew whatever I did would in one way or the other make sense to him. Even in the depressing times of zero progress, I didn't need to explain myself – we had a quiet agreement that things would go on sooner or later. I was free to think about whatever I wanted to, and I was free to do it in my style and at my own pace. Only at the very end did Emo urge me to get it over with. As I know now, this was necessary – and hopefully sufficient.

[3]Nina Amenta etwa hat das Vorwort sprachlich überarbeitet, mit Ausnahme dieser Fußnote.

Chapter 0

Introduction

Suppose this was the abstract of a journal paper rather than the introduction to a dissertation. Then it would probably end with some cryptic AMS subject classifications and a few key words. One of them surely would be *randomized complexity of linear programming*. Further suppose that you happen to stumble on this keyword while searching a database for references that have to do with your own work. Assuming that you are a random person, chances are 75% that the paper is not of interest to you.

The reason is that both the terms *randomization* and *complexity* in connection with linear programming (LP) independently allow for two rather different interpretations.

0.1 Complexity

Two major questions have dominated the theory of linear programming for a long time.

(i) Are linear programs solvable in polynomial time? In other words, does linear programming belong to the complexity class **P**?

(ii) How many steps does the simplex method take to solve a linear program?

While the first question has been answered in the affirmative, only partial answers have been found to the second one.

The Turing machine model. It was in 1980 when the russian mathematician Khachyian developed his polynomial *ellipsoid method* for linear programming, thereby showing that LP – formulated as the problem of deciding feasibility – is in **P** [30].[1] Although the ellipsoid method is only an approximation algorithm that converges to a feasible solution, if there is one, Khachyian was able to prove that a polynomial number of iterations (polynomial in the bitlengths of the inputs) suffices to find a solution or certify that none exists. Thus, he constructed a polynomial-time *Turing machine* to solve the problem, and under this measure of complexity, the issue was settled from the point of view of classical complexity theory.

[1] See [25] for an introduction into the ellipsoid method.

The real RAM model. Parallel to this development, people were investigating the complexity of the simplex method. This method has been introduced by Dantzig in the early fifties [14], and it still presents the most efficient method to solve LPs in practice. Unlike the ellipsoid method, simplex is a discrete method. It traces a sequence of vertices on the boundary of the feasible region, and its actions are determined by the combinatorial structure of the LP but not by the bitlengths of its actual coordinates. Therefore, the simplex method never takes more steps than the number of vertices, even if the coordinates have arbitrary encoding lengths, and in the unit-cost model or *random access machine* (RAM) model, this is exactly what one assumes. Motivated by the fact that on a 'real' computer numbers are stored and manipulated within a constant amount of space and time, the real RAM model allows arithmetic operations on any real numbers to be performed in constant time per operation. Only this convenient model allows statements like 'A system of n equations in n variables can be solved in time $O(n^3)$ by Gaussian elimination.' To disable 'dirty tricks' of speeding up computations by encoding huge amounts of information into a single number, the RAM model is usually enhanced with the 'fairness assumption' that unit cost applies only to numbers that are not much larger than the input numbers (what 'not much larger' means, depends on how fair you want to be, of course).

To summarize, in the RAM model, the ellipsoid method is not even a finite algorithm, while one gets *combinatorial* upper bounds on the runtime of the simplex method, and exactly such combinatorial bounds (i.e bounds in the RAM model) will be considered in this thesis. Unfortunately, these bounds on the simplex method are not known to be polynomial, and the same is true for any other combinatorial bounds that exist for LP. We say that LP is not known to be *strongly polynomial*.

0.2 Randomization

The situation concerning the analysis of the simplex algorithm is even worse than suggested above. First of all, talking about 'the' simplex method does not really make sense because it becomes an actual algorithm only via a *pivot rule*, and under many pivot rules (among them the one originally proposed by Dantzig), the simplex method needs an exponential number of steps in the worst case. This was first shown by Klee and Minty [31], thereby destroying any hope that the simplex method might turn out to be polynomial in the end, at least under Dantzig's pivot rule. Later this negative result was extended to many other commonly used pivot rules. Two remedies are apparent and this is where the randomization comes in.

(i) Analyze the *average* performance of the simplex method, i.e. its expected behavior on problems chosen according to some natural probability distribution. A good bound in this model would explain the efficiency of the method in practice.

(ii) Analyze *randomized* methods, i.e. methods which base their decisions on internal coin flips. All the exponential worst case examples rely on the fact that a malicious adversary knows the strategy of the algorithm in advance and therefore can come

up with just the input for which the strategy is bad. Randomized strategies cannot be fooled in this easy way, if the measure of complexity is the *maximum expected* number of steps, expectation over the internal coin flips performed by the algorithm.

Average performance. In his pioneering work, Borgwardt has shown that the average behavior of the simplex method is indeed polynomial, for a certain variant called the *shadow vertex* algorithm, and under a certain probability distribution imposed on the input [9]. This breakthrough was followed by a sequence of other results improving on the polynomials, weakening the probabilistic model, or considering other pivot rules. It seems that nowadays the average performance of the simplex method is fairly well understood.

Randomized performance. In suggesting remedy (ii) above (which – as you might guess by now – is the one we treat in this thesis), we have not explicitly mentioned the simplex method but randomized methods in general. This is no accident. In fact, randomized algorithms for solving LP in the RAM model have been proposed that are not simplex, although they have 'converged' to the simplex method over the years. For this, the RAM model needs to be enhanced with the assumption that a random number from the set $\{1, \ldots, k\}$ can be obtained in constant time, for any integer k, where 'random' means that each element is chosen with the same probability $1/k$.

Interestingly, this development started with a deterministic algorithm. Megiddo showed that LPs with fixed number of variables (i.e. fixed dimension) can be solved in time linear in the number n of constraints, which is optimal[36]. However, the dependence on the dimension d was doubly exponential. This was improved in a series of papers, by Dyer [15], Dyer and Frieze [16], Clarkson [11], [12] and Seidel [43], the latter establishing a very simple randomized algorithm with expected runtime $O(d!n)$. (We come back to the remarkable algorithm by Clarkson [12] in the Discussion concluding Chapter 5). It was not noticed that Seidel's algorithm was already close in spirit to a dual Random-Facet-Simplex algorithm, see Chapter 2, but Seidel deserves credit for the subsequent developments, including the results of this thesis.

Then two things happened independently. First, Sharir and Welzl enhanced Seidel's incremental algorithm with a method of reusing in each step much of the information obtained in previous steps. This led to a more efficient algorithm, with runtime $O(d^2 2^d n)$. Moreover, this algorithm was formulated in the abstract framework of *LP-type problems* that made it possible to solve problems more general than linear programming [45].

At the same time, Kalai really applied randomization to the simplex method, and only his result lets remedy (ii) actually appear as a remedy. Kalai was able to prove that the Random-Facet-Simplex algorithm takes an expected *subexponential* number of steps on any linear program [27]. In other words, this variant defeats all the worst case counterexamples that have been developed for the deterministic variants. (Still, it is not a polynomial variant, of course.) Only at this stage it explicitly became clear that Sharir and Welzl's algorithm was actually a dual Random-Facet-Simplex algorithm in disguise, their information reusage scheme corresponding to a dual pivot step. (How

explicit the correspondence really is has only recently been shown by Goldwasser who gives a nice survey about the developments we address here [23].)

Motivated by Kalai's result, Sharir and Welzl, together with Matoušek, subsequently proved a subexponential bound on the expected number of iterations their algorithm needs, under a certain condition even in the abstract setting of LP-type problems. However, LP was the only concrete problem that could actually be solved within these time bounds [33].

0.3 Overview and Statement of Results

The main new results of this thesis are the following.

- **Chapter 4, Convex Optimization.** We introduce a class of convex programming problems (including the *polytope distance* problem and the *minimum spanning ball* problem) and develop a concrete *simplex-type* method for solving all the problems in this class. In the framework of LP-type problems, this leads to a unified, linear time algorithm for all problems in this class, when the dimension is fixed. Previously, only specific solutions to some problems were known, like Wolfe's simplex algorithm for quadratic programming [41, section 7.3].

- **Chapter 5, Upper bounds for Abstract Optimization**. We derive *randomized, subexponential* upper bounds for the polytope distance and the minimum spanning ball problem (and more general, for any quadratic programming problem in our class of convex programs). This follows from a subexponential algorithm we develop for *abstract optimization problems* which is an abstract class of problems still substantially more general than LP-type problems. We thus establish subexponential bounds both for concrete geometric problems as well as for general abstract problems (in particular, for all LP-type problems), where prior to this they were only known to apply to LP and some LP-type problems.

- **Chapter 6, Lower Bounds for Abstract Optimization.** We prove a *deterministic, exponential* lower bound for abstract optimization problems, showing that randomization is a crucial tool in solving them.

  We derive a nearly tight lower bound on the performance of the Random-Edge-Simplex algorithm on specific LPs, thereby answering a long-standing open question about the behavior of this algorithm. The result naturally extends to the framework of LP-type problems and the abstract version of Random-Edge-Simplex in this framework.

All the material in this thesis is consequently motivated and derived from the simplex method that we carefully review in Chapter 1.

Turning history upside down, we obtain Sharir and Welzl's algorithm for LP-type problems as an abstraction of the Random-Facet-Simplex algorithm. This means, we pursue an inverse approach in two respects. First, we proceed 'backwards in time' and second,

we derive the runtime bounds of [33] for LP by applying the *primal* Random-Facet-Simplex algorithm to the *dual* LP. In this approach, the primitive operations – whose nature and realization remain somewhat obscure in [33] – are just well-understood pivot steps in the simplex method. Chapter 2 introduces algorithms Random-Facet-Simplex and Random-Edge-Simplex, and prepares the ground for the subsequent 'lifting' of these algorithms to the abstract level.

In Chapter 3, we review the concept of LP-type problems and derive the $O(d^2 2^d n)$ bound for LP in the abstract framework. We actually start with a different concept, namely *LP-type systems* which we prove to be essentially equivalent to LP-type problems. LP-type systems naturally arise in our simplex-type approach that builds on the *bases* of an LP, while LP-type problems rather come up in the original – dual – approach that builds on *subproblems*. In fact, *subproblem solvability* is a crucial feature of both. We conclude the chapter by proving that the framework of *abstract polytopes* with objective function as introduced by Adler and Saigal [3] to abstract from properties of linear programming, is closely related to LP-type systems, although less general. However, the spirit of both frameworks turns out to be the same, it boils down to a feature called *locality* in [33].

Chapter 4 presents concrete non-linear LP-type problems and proves that they allow a fast 'pivoting routine' similar to the one performed by the simplex algorithm on LP. Therefore, the resulting solution method is called 'simplex-type'. After having developed such pivoting routines, the problems under consideration are solvable in linear time (in fixed dimension) by the algorithm for LP-type problems that we have derived in Chapter 3. Among the problems that the approach can handle are the polytope distance problem and the minimum spanning ball problem, where we need to prove the latter to have a reformulation as a problem of minimizing a convex function subject to linear constraints; at first sight, such a reformulation is not apparent from the problem.

In Chapter 5 we prove the subexponential bounds for abstract optimization problems (AOP). These problems are obtained by further relaxing the defining properties of LP-type problems. In particular, we will drop the locality condition addressed above and only keep subproblem solvability. As it turns out, the latter is the crucial requirement for the subexponential results. Locality becomes obsolete under the *improvement oracle* that we consider for AOP. This oracle is an abstraction of the pivot step in case of LP resp. the primitive operations in case of LP-type problems. The results on AOPs apply to the geometric problems of Chapter 4 and lead to subexponential bounds there.

Chapter 6 slowly takes us back, all the way to linear programming. We start by showing that randomization provably helps in solving AOPs. More specifically, we prove that no deterministic algorithm can beat a trivial runtime bound, while exactly such a bound has been developed for a randomized algorithm in Chapter 5.

We proceed by reviewing a result of Matoušek, proving that the kind of subexponential analysis performed in Chapter 5 is essentially best possible for the algorithms under consideration, already for LP-type problems [34]. In other words, to obtain better bounds, the algorithm would have to be altered.

Finally, we analyze the behavior of the abstract RANDOM-EDGE-SIMPLEX algorithm on Matoušek's examples. We find that a quadratic upper and nearly quadratic lower bound holds. (This does not mean that RANDOM-EDGE-SIMPLEX is polynomial or even better than RANDOM-FACET-SIMPLEX on other problems – Matoušek's example are of special nature). As the most interesting application of our methods we obtain a lower bound for the actual RANDOM-EDGE-SIMPLEX algorithm on an actual LP, the so-called *Klee-Minty cube*. This is exactly the LP that serves as the basic building block for all known worst-case examples proving exponential behavior of deterministic simplex variants. Therefore, this LP is a natural test problem for randomized variants; the particular question how RANDOM-EDGE-SIMPLEX behaves on it was open since a wrong answer had been suggested in 1982.

Chapter 7 is an appendix containing facts and bounds referenced elsewhere in the text. This is meant to keep the thesis mostly self-contained. If you find some piece of terminology in the text that you don't understand, it might appear in the Appendix.

One final remark: this thesis does not contain any 'big-Oh's, unless they occur in bounds that are standard or that we do not prove (like $O(n^3)$ denoting the time to solve a system of linear equations by Gaussian elimination). All the bounds that we derive contain explicit constants. Just to stress their order of magnitude, 'big-Oh' versions are occasionally added. Only in some places does this rigor require substantial technical effort. Most of the time, the explicit constants are just a consequence of a clean proof.

Chapter 1

Linear Programming and the Simplex Method

This thesis is about 'simplex-type methods' for several kinds of optimization problems, and it is only natural to start with the simplex method itself. This chapter contains an informal but detailed introduction. Informal means that you will see no proofs – we mainly proceed by example. Detailed means that we are going to describe the standard simplex method using tableaus, as well as the revised simplex method; we discuss what happens in case of unboundedness, infeasibility or degeneracy, and we are talking about pivot rules. The material of this chapter serves as

- a solid foundation for our subsequent results on linear programming,
- a motivation of further, more general algorithms and problem classes, and finally
- a good peg on which to hang our view of the simplex method and its role for this thesis (which in many aspects is quite different from its role in practice).

On the technical side, our presentation stays close to Chvátal's, and most proofs and details that are left out here can be found in his book [10].

1.1 Linear Programming

Linear Programming (LP) in a quite general form is the problem of maximizing a linear function in d variables subject to n linear inequalities. If, in addition, we require all variables to be nonnegative, we have an LP in *standard form*[1] which can be written as follows.

$$\begin{array}{lll} \text{(LP)} \quad \text{maximize} & \sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \leq b_i & (i = 1, \ldots, n), \\ & x_j \geq 0 & (j = 1, \ldots, d), \end{array} \tag{1.1}$$

[1] This is Chvátal's standard form [10]. There are other standard forms, e.g. Murty [39] has equality constraints.

where the c_j, b_i and a_{ij} are real numbers. By defining

$$\begin{aligned} x &:= (x_1, \ldots, x_d)^T, \\ c &:= (c_1, \ldots, c_d)^T, \\ b &:= (b_1, \ldots, b_n)^T, \\ A &:= \begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nd} \end{pmatrix} \end{aligned}$$

this can be written in more compact form as

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & c^T x \\ & \text{subject to} & Ax \leq b, \\ & & x \geq 0, \end{array} \tag{1.2}$$

where the relations $\leq$ and $\geq$ hold for vectors of the same length if and only if they hold componentwise.

The vector c is called the *cost vector* of the LP, and the linear function $z : x \mapsto c^T x$ is called the *objective function*. The vector b is referred to as the *right-hand side* of the LP. The inequalities $\sum_{j=1}^{d} a_{ij} x_j \leq b_i$, for $i = 1, \ldots, n$ and $x_j \geq 0$, for $j = 1, \ldots, d$ are the *constraints* of the linear program. (Due to their special nature, the constraints $x_j \geq 0$ are sometimes called *nonnegativity constraints* or *restrictions*).

The LP is called *feasible* if there exists a nonnegative vector $\tilde{x}$ satisfying $A\tilde{x} \leq b$ (such an $\tilde{x}$ is called a *feasible solution*); otherwise the program is called *infeasible*. If there are feasible solutions with arbitrarily large objective function value, the LP is called *unbounded*; otherwise it is *bounded*. A linear program which is both feasible and bounded has a unique maximum value $c^T \tilde{x}$ attained at a (not necessarily unique) optimal feasible solution $\tilde{x}$. Solving the LP means finding such an optimal solution $\tilde{x}$ (if it exists). To avoid trivialities we assume that the cost vector and all rows of A are nonzero.

1.2 The Standard Simplex Method

1.2.1 Tableaus

When confronted with an LP in standard form, the simplex algorithm starts off by introducing *slack variables* $x_{d+1}, \ldots, x_{d+n}$ to transform the inequality system $Ax \leq b$ into an equivalent system of equalities and additional nonnegativity constraints on the slack variables. The slack variable x_{d+i} closes the gap between the left-hand side and right-hand side of the i-th constraint, i.e.

$$x_{d+i} := b_i - \sum_{j=1}^{d} a_{ij} x_j,$$

for all $i = 1, \ldots, n$. The i-th constraint is then equivalent to

$$x_{d+i} \geq 0,$$

and the linear program can be written as

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & \sum_{j=1}^{d} c_j x_j \\ & \text{subject to} & x_{d+i} = b_i - \sum_{j=1}^{d} a_{ij} x_j \quad (i = 1, \ldots, n), \\ & & x_j \geq 0 \qquad\qquad\qquad\quad (j = 1, \ldots, d+n), \end{array} \tag{1.3}$$

or in a more compact form as

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & \underline{c}^T x \\ & \text{subject to} & \underline{A} x = b, \\ & & x \geq 0, \end{array} \tag{1.4}$$

where $\underline{A}$ is the $n \times (d+n)$-matrix

$$\underline{A} := (A|E), \tag{1.5}$$

$\underline{c}$ is the $(d+n)$-vector

$$\underline{c} := \begin{pmatrix} c \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{1.6}$$

and x is the $(d+n)$-vector

$$x = \begin{pmatrix} x_O \\ x_S \end{pmatrix},$$

where x_O is the vector of **o**riginal variables, x_S the vector of **s**lack variables.

Together with the objective function, the n equations for the x_{d+i} in (1.3) contain all the information about the LP. Following tradition, we will represent this information in *tableau* form where the objective function – denoted by z – is written last and separated from the other equations by a solid line. (The restrictions $x_j \geq 0$ do not show up in the tableau but represent implicit knowledge.) In this way we obtain the *initial tableau* for the LP.

$$\begin{array}{rcrrcrr} x_{d+1} & = & b_1 & -a_{11}x_1 & - & \cdots & -a_{1d}x_d \\ & & & \vdots & & & \\ x_{d+n} & = & b_n & -a_{n1}x_1 & - & \cdots & -a_{nd}x_d \\ \hline z & = & & c_1 x_1 & + & \cdots & +c_d x_d \end{array} \tag{1.7}$$

The compact form here is

$$\begin{array}{rcrr} x_S & = & b & - \quad Ax_O \\ \hline z & = & & c^T x_O \end{array} \tag{1.8}$$

An example illustrates the process of getting the initial tableau from an LP in standard form.

Example 1.1 Consider the problem

$$\begin{array}{lrcrcl} \text{maximize} & x_1 + x_2 & & & & \\ \text{subject to} & -x_1 & + & x_2 & \leq & 1, \\ & x_1 & & & \leq & 3, \\ & & & x_2 & \leq & 2, \\ & x_1, x_2 \geq 0. & & & & \end{array} \tag{1.9}$$

After introducing slack variables x_3, x_4, x_5, the LP in equality form is

$$\begin{array}{lrcrcrcl} \text{maximize} & x_1 + x_2 & & & & & & \\ \text{subject to} & x_3 & = & 1 & + & x_1 & - & x_2, \\ & x_4 & = & 3 & - & x_1, & & \\ & x_5 & = & 2 & & & - & x_2, \\ & x_1, \ldots, x_5 \geq 0. & & & & & & \end{array} \tag{1.10}$$

From this we obtain the initial tableau

$$\begin{array}{rcrcrcl} x_3 & = & 1 & + & x_1 & - & x_2 \\ x_4 & = & 3 & - & x_1 & & \\ x_5 & = & 2 & & & - & x_2 \\ \hline z & = & & & x_1 & + & x_2 \end{array} \tag{1.11}$$

Abstracting from the initial tableau (1.7), a general tableau for the LP is any system $\mathcal{T}$ of $n+1$ linear equations in the variables $x_1, \ldots, x_{d+n}$ and z, with the properties that

(i) $\mathcal{T}$ expresses n left-hand side variables x_B and z in terms of the remaining d right-hand side variables x_N, i.e. there is an n-vector β, a d-vector γ, an $n \times d$-matrix Λ and a real number z_0 such that $\mathcal{T}$ (written in compact form) is the system

$$\begin{array}{rcrcl} x_B & = & \beta & - & \Lambda x_N \\ \hline z & = & z_0 & + & \gamma^T x_N \end{array} \tag{1.12}$$

(ii) Any solution of (1.12) is a solution of (1.8) and vice versa.

By property (ii), any tableau contains the *same* information about the LP but represented in a *different* way. All that the simplex algorithm is about is constructing a sequence of tableaus by gradually rewriting them, finally leading to a tableau in which the information is represented in such a way that the desired optimal solution can be read off directly. We will immediately show how this works in our example.

1.2.2 Pivoting

Here is the initial tableau (1.11) to Example 1.1 again.

$$\begin{array}{lcrcrcr} x_3 & = & 1 & + & x_1 & - & x_2 \\ x_4 & = & 3 & - & x_1 & & \\ x_5 & = & 2 & & & - & x_2 \\ \hline z & = & & & x_1 & + & x_2 \end{array}$$

By setting the right-hand side variables x_1, x_2 to zero, we find that the left-hand side variables x_3, x_4, x_5 assume nonnegative values $x_3 = 1, x_4 = 3, x_5 = 2$. This means, the vector $x = (0, 0, 1, 3, 2)$ is a feasible solution of (1.10) (and the vector $x' = (0, 0)$ is a feasible solution of (1.9)). The objective function value $z = 0$ associated with this feasible solution is computed from the last row of the tableau. In general, any feasible solution that can be obtained by setting the right-hand side variables of a tableau to zero is called a *basic feasible solution* (BFS). In this case we also refer to the tableau as a feasible tableau. The left-hand side variables of a feasible tableau are called *basic* and are said to constitute a *basis*, the right-hand side ones are *nonbasic*. The goal of the simplex algorithm is now either to construct a new feasible tableau with a corresponding BFS of higher z-value, or to prove that there exists no feasible solution at all with higher z-value. In the latter case the BFS obtained from the tableau is reported as an optimal solution to the LP; in the former case, the process is repeated, starting from the new tableau.

In the above tableau we observe that increasing the value of x_1 (i.e. making x_1 positive) will increase the z-value. The same is true for x_2, and this is due to the fact that both variables have positive coefficients in the z-row of the tableau. Let us arbitrarily choose x_2. By how much can we increase x_2? If we want to maintain feasibility, we have to be careful not to let any of the basic variables go below zero. This means, the equations determining the values of the basic variables may limit x_2's increment. Consider the first equation

$$x_3 = 1 + x_1 - x_2. \tag{1.13}$$

Together with the implicit constraint $x_3 \geq 0$, this equation lets us increase x_2 up to the value $x_2 = 1$ (the other nonbasic variable x_1 keeps its zero value). The second equation

$$x_4 = 3 - x_1$$

does not limit the increment of x_2 at all, and the third equation

$$x_5 = 2 - x_2$$

allows for an increase up to the value $x_2 = 2$ before x_5 gets negative. The most stringent restriction therefore is $x_3 \geq 0$, imposed by (1.13), and we will increase x_2 just as much as we can, so we get $x_2 = 1$ and x_3=0. From the remaining tableau equations, the values of the other variables are obtained as

$$\begin{aligned} x_4 &= 3 - x_1 = 3, \\ x_5 &= 2 - x_2 = 1. \end{aligned}$$

To establish this as a BFS, we would like to have a tableau with the new zero variable x_3 replacing x_2 as a nonbasic variable. This is easy – the equation (1.13) which determined the new value of x_2 relates both variables. This equation can be rewritten as

$$x_2 = 1 + x_1 - x_3,$$

and substituting the right-hand side for x_2 into the remaining equations gives the new tableau

$$\begin{array}{rcrcrcr} x_2 & = & 1 & + & x_1 & - & x_3 \\ x_4 & = & 3 & - & x_1 & & \\ x_5 & = & 1 & - & x_1 & + & x_3 \\ \hline z & = & 1 & + & 2x_1 & - & x_3 \end{array}$$

with corresponding BFS $x = (0, 1, 0, 3, 1)$ and objective function value $z = 1$. This process of rewriting a tableau into another one is called a *pivot step*, and it is clear by construction that both systems have the same set of solutions. The effect of a pivot step is that a nonbasic variable (in this case x_2) *enters* the basis, while a basic one (in this case x_3) *leaves* it. Let us call x_2 the *entering variable* and x_3 the *leaving variable*.

In the new tableau, we can still increase x_1 and obtain a larger z-value. x_3 cannot be increased since this would lead to smaller z-value. The first equation puts no restriction on the increment, from the second one we get $x_1 \leq 3$ and from the third one $x_1 \leq 1$. So the third one is the most stringent, will be rewritten and substituted into the remaining equations as above. This means, x_1 enters the basis, x_5 leaves it, and the tableau we obtain is

$$\begin{array}{rcrcrcr} x_2 & = & 2 & & & - & x_5 \\ x_4 & = & 2 & - & x_3 & + & x_5 \\ x_1 & = & 1 & + & x_3 & - & x_5 \\ \hline z & = & 3 & + & x_3 & - & 2x_5 \end{array}$$

with BFS $x = (1, 2, 0, 2, 0)$ and $z = 3$. Performing one more pivot step (this time with x_3 the entering and x_4 the leaving variable), we arrive at the tableau

$$\begin{array}{rcrcrcr} x_2 & = & 2 & & & - & x_5 \\ x_3 & = & 2 & - & x_4 & + & x_5 \\ x_1 & = & 3 & - & x_4 & & \\ \hline z & = & 5 & - & x_4 & - & x_5 \end{array} \tag{1.14}$$

with BFS $x = (3, 2, 2, 0, 0)$ and $z = 5$. In this tableau, no nonbasic variable can increase without making the objective function value smaller, so we are stuck. Luckily, this means that we have already found an optimal solution. Why? Consider any *feasible* solution $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_5)$ for (1.10), with objective function value z_0. This is a solution to (1.11) and therefore a solution to (1.14). Thus,

$$z_0 = 5 - \tilde{x}_4 - \tilde{x}_5$$

must hold, and together with the implicit restrictions $x_4, x_5 \geq 0$ this implies $z_0 \leq 5$. The tableau even delivers a proof that the BFS we have computed is the unique optimal solution to the problem: $z = 5$ implies $x_4 = x_5 = 0$, and this determines the values of the other variables. Ambiguities occur only if some of the nonbasic variables have zero coefficients in the z-row of the final tableau. Unless a specific optimal solution is required, the simplex algorithm in this case just reports the optimal BFS it has at hand.

1.2.3 Geometric Interpretation

Consider the standard form LP (1.1). For each constraint

$$\begin{aligned} \sum_{j=1}^{d} a_{ij} x_j &\leq b_i \quad \text{or} \\ x_j &\geq 0, \end{aligned}$$

the points $\tilde{x} \in \mathbb{R}^d$ satisfying the constraint form a closed *halfspace* in $\mathbb{R}^d$. The points for which equality holds form the boundary of this halfspace, the *constraint hyperplane*.

The set of feasible solutions of the LP is therefore an intersection of halfspaces, which is by definition a (possibly empty) *polyhedron* P. The facets of P are induced by (not necessarily all) constraint hyperplanes. The nonnegativity constraints $x_j \geq 0$ restrict P to lie inside the positive orthant of $\mathbb{R}^d$. The following correspondence between basic feasible solutions of the LP and vertices of P justifies the geometric interpretation of the simplex method as an algorithm that traverses a sequence of vertices of P until an optimal vertex is found.

Fact 1.2 *Consider a standard form LP with feasible polyhedron P. The point $\tilde{x}' = (\tilde{x}_1, \ldots, \tilde{x}_d)$ is a vertex of P if and only if the vector $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_{d+n})$ with*

$$\tilde{x}_{d+i} := b_i - \sum_{j=1}^{d} a_{ij} \tilde{x}_j, \;\; i = 1, \ldots, n$$

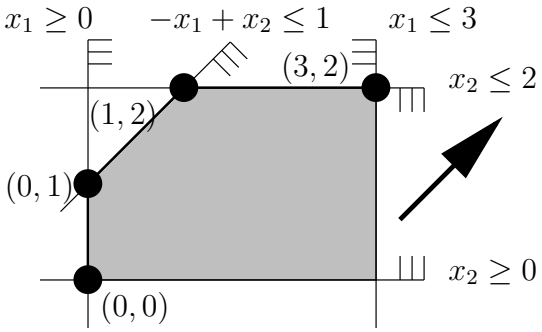
is a basic feasible solution of the LP.

Two consecutive tableaus constructed by the simplex method have $d-1$ nonbasic variables in common. Their BFS thus share $d-1$ zero variables. Equivalently, the corresponding vertices lie on $d-1$ common constraint hyperplanes, and this means that they are adjacent in P. The feasible solutions obtained in the process of continuously increasing the value of a nonbasic variable until it becomes basic correspond to the points on the edge of P connecting the two vertices. Establishing these facts formally requires at least some basic polyhedra theory like it is found e.g. in Ziegler's book [50]. A proof of Fact 1.2 is given in a slightly more general form by Chvátal [10, §18].

Here we are content with checking the correlations in case of Example 1.1. The LP consists of five constraints over two variables, therefore the feasible region is a polygon in $\mathbb{R}^2$. Every constraint hyperplane defines a facet, so we get a polygon with five edges and

five vertices. In the previous subsection we were going through a sequence of four tableaus until we discovered an optimal BFS. The picture below shows how this corresponds to a sequence of adjacent vertices. The optimization direction c is drawn as a fat arrow. Since the objective function value gets higher in every iteration, the path of vertices is monotone in direction c.

BFS	vertex	$z = x_1 + x_2$
$(0, 0, 1, 3, 4)$	$(0, 0)$	0
$(0, 1, 0, 3, 1)$	$(0, 1)$	1
$(1, 2, 0, 2, 0)$	$(1, 2)$	3
$(3, 2, 2, 0, 0)$	$(3, 2)$	5



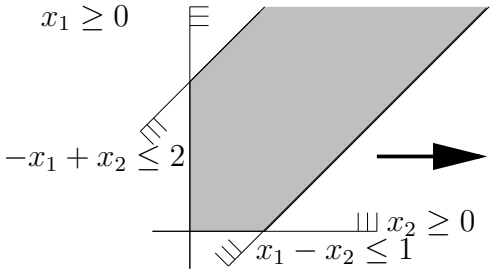
1.3 Exception Handling

So far our outline of the simplex method went pretty smooth. This is in part due to the fact that we have only seen one very small and trivial example of the way it works. On the other hand, the method *is* simple, and we will just incorporate some 'exception handling' and do a little fine tuning, again basically by example.

1.3.1 Unboundedness

During a pivot step, we make the value of a nonbasic variable just large enough to get the value of a basic variable down to zero. This, however, might never happen. Consider the example

$$\begin{array}{lrcl} \text{maximize} & x_1 & & \\ \text{subject to} & x_1 - x_2 & \leq & 1, \\ & -x_1 + x_2 & \leq & 2, \\ & x_1, x_2 \geq 0. & & \end{array}$$



with initial tableau

$$\begin{array}{rcrcrcr} x_3 & = & 1 & - & x_1 & + & x_2 \\ x_4 & = & 2 & + & x_1 & - & x_2 \\ \hline z & = & & & x_1 & & \end{array}$$

After one pivot step with x_1 entering the basis we get the tableau

$$\begin{array}{rcrcrcr} x_1 & = & 1 & + & x_2 & - & x_3 \\ x_4 & = & 3 & & & - & x_3 \\ \hline z & = & 1 & + & x_2 & - & x_3 \end{array}$$

If we now try to bring x_2 into the basis by increasing its value, we notice that none of the tableau equations puts a limit on the increment. We can make x_2 and z arbitrarily

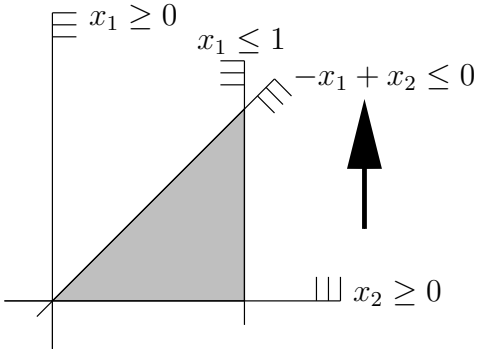
large – the problem is unbounded. By letting x_2 go to infinity we get a feasible halfline – starting from the current BFS – as a witness for the unboundedness. In our case this is the set of feasible solutions

$$\{(1,0,0,3) + x_2(1,1,0,0) \mid x_2 \geq 0\}.$$

Such a halfline will typically be the output of the algorithm in the unbounded case. Thus, unboundedness can quite naturally be handled with the existing machinery. In the geometric interpretation it just means that the feasible polyhedron P is unbounded in the optimization direction.

1.3.2 Degeneracy

While we can make some nonbasic variable arbitrarily large in the unbounded case, just the other extreme happens in the degenerate case: some tableau equation limits the increment to zero so that no progress in z is possible. Consider the LP



$$\begin{array}{lrcl} \text{maximize} & x_2 & & \\ \text{subject to} & -x_1 + x_2 & \leq & 0, \\ & x_1 & \leq & 2, \\ & x_1, x_2 \geq 0, & & \end{array} \tag{1.15}$$

with initial feasible tableau

$$\begin{array}{lcrcl} x_3 & = & & x_1 - & x_2 \\ x_4 & = & 2 - & x_1 & \\ \hline z & = & & & x_2 \end{array}$$

The only candidate for entering the basis is x_2, but the first tableau equation shows that its value cannot be increased without making x_3 negative. This may happen whenever in a BFS some basic variables assume zero value, and such a situation is called *degenerate*. Unfortunately, the impossibility of making progress in this case does not imply optimality, so we have to perform a 'zero progress' pivot step. In our example, bringing x_2 into the basis results in another degenerate tableau with the same BFS.

$$\begin{array}{lcrcl} x_2 & = & & x_1 - & x_3 \\ x_4 & = & 2 - & x_1 & \\ \hline z & = & & x_1 - & x_3 \end{array}$$

Nevertheless, the situation has improved. The nonbasic variable x_1 can be increased now, and by entering it into the basis (replacing x_4) we already obtain the final tableau

$$\begin{array}{lcrcrcl} x_1 & = & 2 & & & - & x_4 \\ x_2 & = & 2 & - & x_3 & - & x_4 \\ \hline z & = & 2 & - & x_3 & - & x_4 \end{array}$$

with optimal BFS $x = (x_1, \ldots, x_4) = (2, 2, 0, 0)$.

In this example, after one degenerate pivot we were able to make progress again. In general, there might be longer runs of degenerate pivots. Even worse, it may happen that a tableau repeats itself during a sequence of degenerate pivots, so the algorithm can go through an infinite sequence of tableaus without ever making progress. This phenomenon is known as *cycling*, and an example can be found in [10, pp. 30–32]. If the algorithm does not terminate, it must cycle. This follows from the fact that there are only finitely many different tableaus.

Fact 1.3 *The LP (1.1) has at most* $\binom{n+d}{n}$ *tableaus.*

To prove this, we show that any tableau $\mathcal{T}$ is already determined by its basis variables. Write $\mathcal{T}$ as

$$\begin{array}{rcrcr} x_B & = & \beta & - & \Lambda x_N \\ \hline z & = & z_0 & + & \gamma^T x_N, \end{array}$$

and assume there is another tableau $\mathcal{T}'$ with the same basic and nonbasic variables, i.e. $\mathcal{T}'$ is the system

$$\begin{array}{rcrcr} x_B & = & \beta' & - & \Lambda' x_N \\ \hline z & = & z_0' & + & \gamma'^T x_N, \end{array}$$

By the tableau properties, both systems have the same set of solutions. Therefore

$$\begin{aligned} (\beta - \beta') - (\Lambda - \Lambda')x_N &= 0 \text{ and} \\ (z_0 - z_0') + (\gamma^T - \gamma'^T)x_N &= 0 \end{aligned}$$

must hold for all d-vectors x_N, and this implies $\beta = \beta', \Lambda = \Lambda', \gamma = \gamma'$ and $z_0 = z_0'$. Hence $\mathcal{T} = \mathcal{T}'$.

There are two standard ways to avoid cycling:

- Bland's *smallest subscript rule*: If there is more than one candidate x_k for entering the basis (or more than one candidate for leaving the basis, which is another manifestation of degeneracy), choose the one with smallest subscript k.
- Avoid degeneracies altogether by symbolic perturbation.

By Bland's rule, there *is* always a way of escaping from a sequence of degenerate pivots. For this, however, one has to give up the freedom of choosing the entering variable. For us it will be crucial *not* to restrict the choice of the entering variable, so we will abandon Bland's rule and instead resort to the method of symbolic perturbation, although this requires more computational effort. The method – also known as the *lexicographic* method – perturbs the right-hand side vector b of the LP by adding powers of a symbolic constant ε (assumed to be infinitesimally small). The LP then becomes

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{d} a_{ij} x_j \le b_i + \varepsilon^i & (i = 1, \ldots, n), \\ & x_j \ge 0 & (j = 1, \ldots, d), \end{array} \tag{1.16}$$

and if the original LP (1.1) is feasible, so is (1.16). A solution to (1.1) can be obtained from a solution to (1.16) by ignoring the contribution of ε, i.e. by setting ε to zero. Moreover, any valid tableau for (1.16) reduces to a valid tableau for (1.1) when the terms involving powers of ε are disregarded.

In case of (1.15), the initial tableau of the perturbed problem is

$$\begin{array}{lcrcrcr} x_3 & = & & \varepsilon & + & x_1 & - & x_2 \\ x_4 & = & 2+\varepsilon^2 & & - & x_1 & & \\ \hline z & = & & & & & & x_2 \end{array}$$

Pivoting with x_2 entering the basis gives the tableau

$$\begin{array}{lcrcrcr} x_2 & = & & \varepsilon & + & x_1 & - & x_3 \\ x_4 & = & 2+\varepsilon^2 & & - & x_1 & & \\ \hline z & = & & \varepsilon & + & x_1 & - & x_3 \end{array} \tag{1.17}$$

This is no longer a degenerate pivot, since x_2 (and z) increased by ε. Finally, bringing x_1 into the basis gives the tableau

$$\begin{array}{lcrcrcr} x_1 & = & 2+\varepsilon^2 & & & - & x_4 \\ x_2 & = & 2+\varepsilon+\varepsilon^2 & - & x_3 & - & x_4 \\ \hline z & = & 2+\varepsilon+\varepsilon^2 & - & x_3 & - & x_4 \end{array} \tag{1.18}$$

with optimal BFS $x = (2+\varepsilon^2, 2+\varepsilon+\varepsilon^2, 0, 0)$. The optimal BFS for (1.15) is recovered from this by ignoring the additive terms in ε. In general, the following holds, which proves nondegeneracy of the perturbed problem.

Fact 1.4 *In any BFS of 1.16, the values of the basic variables are nonzero polynomials in ε, of degree at most n. The tableau coefficients at the nonbasic variables are unaffected by the perturbation.*

To find the leaving variable, polynomials in ε have to be compared. This is done lexicographically, i.e.

$$\sum_{k=1}^{n} \lambda_k \varepsilon^k < \sum_{k=1}^{n} \lambda'_k \varepsilon^k$$

if and only if $(\lambda_1, \ldots, \lambda_n)$ is lexicographically smaller than $(\lambda'_1, \ldots, \lambda'_n)$. The justification for this is that one could actually assign a very small numerical value to ε (depending on the input numbers of the LP), such that comparing lexicographically is equivalent to comparing numerically, for all polynomials that turn up in the algorithm.

In the perturbed problem, progress is made in every pivot step. Cycling cannot occur and the algorithm terminates after at most $\binom{n+d}{n}$ pivots.[2] In the associated feasible polyhedron, degeneracies correspond to 'overcrowded vertices', which are vertices where more

[2] An upper bound is the number of *feasible* tableaus. By Mc Mullens's Upper Bound Theorem [35], this number is much smaller than $\binom{n+d}{n}$.

than d of the constraint hyperplanes meet. There are several ways to represent the same vertex as an intersection of exactly d hyperplanes, and a degenerate pivot switches between two such representations. The perturbation slightly moves the hyperplanes relative to each other in such a way that any degenerate vertex is split into a collection of nondegenerate ones very close together.

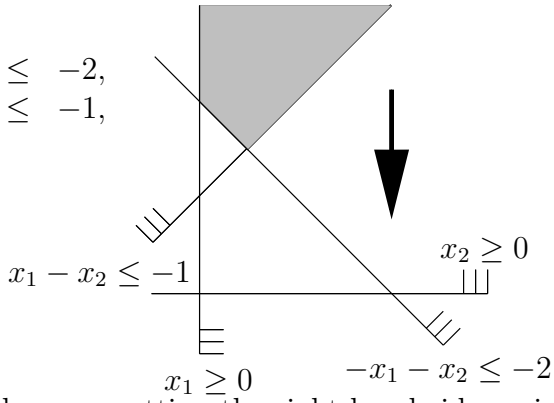
1.3.3 Infeasibility

To start off, the simplex method needs some feasible tableau. In all examples considered so far such a tableau was readily available since the initial tableau was feasible. We say that the problem has a *feasible origin*. This is equivalently expressed by the fact that the right-hand side vector b of the LP is nonnegative. If this is not the case, we first solve an auxiliary problem that either constructs a BFS to the original problem or proves that the original problem is infeasible. The auxiliary problem has an additional variable x_0 and is defined as

$$\begin{array}{ll} \text{minimize} & x_0 \\ \text{subject to} & \sum_{j=1}^{d} a_{ij}x_j - x_0 \le b_i \quad (i = 1, \ldots, n), \\ & x_j \ge 0 \qquad\qquad\qquad\quad (j = 0, \ldots, d). \end{array}$$

This problem is feasible (choose x_0 big enough), and it is clear that the original problem is feasible if and only if the optimum value of the auxiliary LP is zero. Let us do an example and consider the problem

$$\begin{array}{lrcl} \text{maximize} & -x_2 & & \\ \text{subject to} & -x_1 - x_2 & \le & -2, \\ & x_1 - x_2 & \le & -1, \\ & x_1, x_2 \ge 0. & & \end{array}$$



with initial tableau

$$\begin{array}{rcrcrcr} x_3 & = & -2 & + & x_1 & + & x_2 \\ x_4 & = & -1 & - & x_1 & + & x_2 \\ \hline z & = & & & & - & x_2 \end{array}$$

This problem has an infeasible origin, because setting the right-hand side variables to zero gives $x_3 = -2, x_4 = -1$. The auxiliary problem (written in maximization form to avoid confusion and with the objective function called w in the tableau) is

$$\begin{array}{lrcl} \text{maximize} & -x_0 & & \\ \text{subject to} & -x_1 - x_2 - x_0 & \le & -2, \\ & x_1 - x_2 - x_0 & \le & -1, \\ & x_0, x_1, x_2 \ge 0. & & \end{array}$$

with initial tableau

$$\begin{array}{rcrcrcrcr} x_3 & = & -2 & + & x_1 & + & x_2 & + & x_0 \\ x_4 & = & -1 & - & x_1 & + & x_2 & + & x_0 \\ \hline w & = & & & & & & - & x_0 \end{array}$$

The auxiliary problem has an infeasible initial tableau, too, but we can easily construct a feasible tableau by performing one pivot step. We start increasing the value of x_0, this time not with the goal of *maintaining* feasibility but with the goal of *reaching* feasibility. To get $x_3 \geq 0$, x_0 has to increase by at least 2, and this also makes x_4 positive. By setting $x_0 := 2$ we get $x_3 = 0$ and $x_4 = 1$. Solving the first tableau equation for x_0 and substituting from this into the remaining equations as usual gives a new *feasible* tableau with x_0 basic and x_3 nonbasic.

$$\begin{array}{rcrcrcrcr} x_0 & = & 2 & - & x_1 & - & x_2 & + & x_3 \\ x_4 & = & 1 & - & 2x_1 & & & + & x_3 \\ \hline w & = & -2 & + & x_1 & + & x_2 & - & x_3 \end{array}$$

The simplex method can now be used to solve the auxiliary problem. In our case, by choosing x_2 as the entering variable, we accomplish this in one step. The resulting tableau is

$$\begin{array}{rcrcrcrcr} x_2 & = & 2 & - & x_1 & + & x_3 & - & x_0 \\ x_4 & = & 1 & - & 2x_1 & + & x_3 & & \\ \hline w & = & & & & & & - & x_0 \end{array}$$

Since all coefficients of nonbasic variables in the w-row are nonpositive, this is an optimal tableau with BFS $x = (x_0, \ldots, x_4) = (0, 0, 2, 0, 1)$. The associated zero w-value asserts that the LP we originally wanted to solve is actually feasible, and we can even construct a feasible tableau for it from the final tableau of the auxiliary problem by ignoring x_0 and expressing the original objective function z in terms of the nonbasic variables; from the first tableau equation we get in our case $z = -x_2 = -2 + x_1 - x_3$, and this gives a valid feasible tableau

$$\begin{array}{rcrcrcrcr} x_2 & = & 2 & - & x_1 & + & x_3 \\ x_4 & = & 1 & - & 2x_1 & + & x_3 \\ \hline z & = & -2 & + & x_1 & - & x_3 \end{array}$$

with corresponding BFS $x = (x_1, \ldots, x_4) = (0, 2, 0, 1)$ for the original LP. For this to work, x_0 should be nonbasic in the final tableau of the auxiliary problem which is automatically the case if the problem is nondegenerate. (To guarantee this in the general situation, choose x_0 as the leaving variable whenever this is a possible choice.)

If the optimum value of the auxiliary problem is nonzero, we can conclude that the original LP is infeasible and simply report this fact.

1.4 The Revised Simplex Method

How much time does a single pivot step take? If we charge unit cost for comparisons and elementary arithmetic operations (additions, multiplications and divisions) we get

- $O(d)$ for finding the entering variable,
- $O(n)$ for finding the leaving variable,
- $O(nd)$ for rewriting the tableau.

The new BFS is already available after entering and leaving variable have been computed; only to be able to carry on with the next iteration we need to do an expensive tableau update. The so-called *revised* simplex method works on an implicit representation of the tableau and completely saves the update. At the same time, however, the process of finding the entering variable slows down to $O(nd)$ in the worst case, so what do we gain?

(i) The revised simplex method can be implemented in such a way that pivoting is much faster than $O(nd)$ on sparse problems typically encountered in practice. No such tuning is apparent for the *standard* simplex method described so far. From a practical point of view, this is the most important reason. For this thesis, it is the least important reason. We are aiming for bounds that hold for *all* inputs, so we can't profit from sparsity. We *will* save, but for a different reason.

(ii) Although the process of rewriting a tableau is simple, the relationship between consecutive tableaus and their connection to the original data remain somewhat obscure. In this respect, the revised simplex method is much more explicit.

(iii) The Random-Facet algorithm we introduce in the next chapter never actually searches for an entering variable during a pivot step; by that time previous computations have delivered a unique candidate for entering the basis, and only that candidate has to be checked. In this situation it pays off to shift the computational effort from the tableau updates to the search process which is saved anyway. A pivot step in the Random-Facet algorithm can then be performed in time independent of d (which is a saving, because we will see in Chapter 3, Subsection 3.2.3 that $d \geq n$ in a typical application of the simplex method; for $d < n$ we are better off by solving the dual problem).

To describe the revised simplex method, we recall that any tableau can be written in the form

$$\begin{array}{rcrcr} x_B & = & \beta & - & \Lambda x_N \\ \hline z & = & z_0 & + & \gamma^T x_N, \end{array} \tag{1.19}$$

and as we have already argued for Fact 1.3, the entries β, γ, Λ and z_0 are determined by the choice of the basic variables x_B. Formally, if G is some subset

$$G = \{j_1, \ldots, j_k\} \subseteq [d+n]$$

of variable subscripts, then

$$x_G := (x_{j_1}, \ldots x_{j_k})^T \tag{1.20}$$

is the vector of all the variables with subscripts in G. Any tableau is therefore uniquely specified by its basic subscript set B. We will make this specification explicit, i.e. we show how the entries β, γ, Λ and z_0 relate to B.

Consider the LP in compact equality form (1.4), which is

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & \underline{c}^T x \\ & \text{subject to} & \underline{A}x = b, \\ & & x \geq 0, \end{array} \tag{1.21}$$

with $\underline{A}$ and $\underline{c}$ defined according to (1.5) resp. (1.6).

Let $\underline{A}_j$ denote the j-th column of $\underline{A}$. For subscript set $G = \{j_1, \ldots, j_k\}$ let

$$\underline{A}_G := (\underline{A}_{j_1}, \ldots, \underline{A}_{j_k}) \tag{1.22}$$

collect the k columns corresponding to the variables with subscripts in G. Then the equations of (1.21) read as

$$\underline{A}_B x_B + \underline{A}_N x_N = b. \tag{1.23}$$

Since (1.19) has by definition of a tableau the same set of solutions as (1.23), the former is obtained by simply solving (1.23) for x_B, which gives

$$x_B = \underline{A}_B^{-1} b - \underline{A}_B^{-1} \underline{A}_N x_N, \tag{1.24}$$

and therefore

$$\begin{array}{rcl} \beta & = & \underline{A}_B^{-1} b, \\ \Lambda & = & \underline{A}_B^{-1} \underline{A}_N. \end{array} \tag{1.25}$$

By similar reasoning we compute γ and z_0. For $G = \{j_1, \ldots, j_k\}$ let

$$\underline{c}_G := (\underline{c}_{j_1}, \ldots, \underline{c}_{j_k})^T \tag{1.26}$$

collect the entries corresponding to variables with subscripts in G. Then the equation for z in (1.4) reads as

$$z = \underline{c}_B^T x_B + \underline{c}_N^T x_N. \tag{1.27}$$

Again by the tableau property, the last row of (1.19) is equivalent to (1.27), and the former is obtained by simply substituting from (1.24) into (1.27), which gives

$$z = \underline{c}_B^T \underline{A}_B^{-1} b + (\underline{c}_N^T - \underline{c}_B^T \underline{A}_B^{-1} \underline{A}_N) x_N, \tag{1.28}$$

and therefore

$$\begin{array}{rcl} z_0 & = & \underline{c}_B^T \underline{A}_B^{-1} b, \\ \gamma^T & = & \underline{c}_N^T - \underline{c}_B^T \underline{A}_B^{-1} \underline{A}_N. \end{array} \tag{1.29}$$

Rather than maintaining the tableau (1.19) explicitly, the revised simplex method maintains only the current BFS, the basic subscript set B as well as the *inverse* of the matrix $\underline{A}_B$. Note that before the first iteration, $\underline{A}_B = \underline{A}_B^{-1} = E$.

To obtain the new BFS, all necessary information is retrieved from the implicit tableau representation given by (1.24) and (1.27), as follows.

- To find the entering variable, evaluate γ^T according to (1.29), i.e. solve

  $$y^T \underline{A}_B = \underline{c}_B^T$$

  for y^T and set

  $$\gamma^T = \underline{c}_N^T - y^T \underline{A}_N.$$

  This takes $O(n^2)$ time to compute y^T, given that $\underline{A}_B^{-1}$ is available, and $O(n)$ time for every entry of γ^T. A variable x_{j_ℓ} may enter the basis if its corresponding γ-entry is positive. *Checking* whether a variable is eligible for entering the basis thus costs only $O(n^2)$ time while *searching* for an entering variable takes up to $O(n^2 + nd)$ time.

- The column λ of the tableau matrix Λ corresponding to the entering variable x_{j_ℓ} is computed according to (1.25), i.e. by solving

  $$\underline{A}_B \lambda = \underline{A}_{j_\ell} \tag{1.30}$$

  for λ. Again, since $\underline{A}_B^{-1}$ is available, this can be done in time $O(n^2)$.

The leaving variable and the new BFS are easily computed from this (and the old BFS) in time $O(n)$, just like the standard simplex method does it from the explicit tableau representation. One remark is in order. If we apply the method of symbolic perturbation to cope with degeneracies, this step gets more expensive. In order to obtain the new BFS, n polynomials of degree up to n in ε have to be compared and updated (consider the step from (1.17) to (1.18) in Subsection 1.3.2). This takes time $O(n^2)$ rather than $O(n)$ when only actual numbers are involved. Luckily, this does not introduce asymptotic overhead, since the previous steps already take $O(n^2)$ time.

The tableau update is replaced by an update of the set B and the matrix $\underline{A}_B^{-1}$. Let us first consider how $\underline{A}_B$ changes. After permuting the column order we can assume that the new matrix $\underline{A}_{B'}$ arises from $\underline{A}_B$ by replacing the k-th column, and the new column is exactly the column $\underline{A}_{j_\ell}$ referred to in (1.30). Consequently,

$$\underline{A}_{B'} = \underline{A}_B \begin{pmatrix} 1 & \cdots & \lambda_1 & \cdots & 0 \\ & \ddots & \vdots & & \\ & & \vdots & \ddots & \\ 0 & \cdots & \lambda_n & \cdots & 1 \end{pmatrix},$$

so $\underline{A}_B$ is updated by multiplying with a unit matrix whose k-th column is replaced by the column λ already computed in (1.30). This implies

$$\underline{A}_{B'}^{-1} = \begin{pmatrix} 1 & \cdots & -\frac{\lambda_1}{\lambda_k} & \cdots & 0 \\ & \ddots & \vdots & & \\ & & \frac{1}{\lambda_k} & & \\ & & \vdots & \ddots & \\ 0 & \cdots & -\frac{\lambda_n}{\lambda_k} & \cdots & 1 \end{pmatrix} \underline{A}_B^{-1},$$

and this multiplication can be performed in time $O(n^2)$.

Fact 1.5 *One pivot step in the revised simplex method takes time $O(n^2)$ if (a candidate for) the entering variable is known in advance, and time $O(n^2 + nd)$ otherwise.*

1.5 Pivot Rules

A *pivot rule* is any scheme for choosing the entering variable in case there is more than one candidate for it (which is the typical situation). The following would be a canonical (and time-saving) rule when applying the revised simplex method: compute the z-row of the current tableau (1.19) entry by entry, according to (1.29), until a positive coefficient is found; choose the associated variable as the entering variable. This rule is fast when a positive entry is found early in γ. On the other hand, it is *insensitive* to the problem. Another pivot rule might spend more time on a sophisticated choice of the entering variable, with the goal of decreasing the overall number of pivot steps. In this respect, the best possible (and most sensitive) rule would be to choose the entering variable that minimizes the number of subsequent pivot steps. Unfortunately, no efficient implementation of this rule is known. Typically, a pivot rule is a compromise between the goal of achieving a small number of pivot steps on the one hand and the effort necessary to achieve this goal on the other hand. Although most 'classical' pivot rules are local in the sense that only the current tableau influences the choice of the entering variable, practical implementations try to reuse information obtained in previous iterations. In principle, a pivot rule has access to the complete history of computation.

In this thesis we explicitly distinguish between *deterministic* and *randomized* pivot rules. The latter – which we are mainly interested in – may base their decisions on the outcome of coin flips. Randomized rules do not seem to play a major role in practice. (Randomization can be applied to deal with numerical problems [6]; however, this is not the effect we are interested in here). Among the reasons might be the following.

(i) Decisions based on coin flips are typically insensitive to the problem.

(ii) The often-used phrase of the 'problems encountered in practice' suggests that the simplex method is mostly confronted with 'average' problems, and in fact its runtime frequently lies within the bounds observed for random problems. In this setting, randomization is not very effective.[3] Moreover, (pseudo-)random number generation entails extra computations and is therefore not desirable in highly optimized codes.

The picture completely changes when the worst case behavior is studied. In this context, the insensitivity of coin flips can even be an advantage, and randomization is provably effective. More precisely,

[3] As an example, consider Quicksort [13, pp. 153–171]. On an average input sequence, pivoting with the first element already works fine. Therefore, pivoting with a random element would be overkill.

(i) In case of a deterministic pivot rule, the actions taken by the simplex method are completely determined by the input. Therefore, a malicious adversary might be able to construct inputs which force the method to go through a huge number of pivot steps, leading to exponential worst case behavior. This has actually happened to most known deterministic pivot rules (the first was Dantzig's largest coefficient rule, with Klee and Minty [31] serving as the malicious adversaries). Randomized pivot rules seem harder to fool, since the adversary does not know the decisions in advance. In particular, no superpolynomial lower bound on the *expected* number of pivot steps is known for any reasonable randomized rule.[4] It is important to note that the expectation here is *not* taken over an input distribution but over the coin flips performed by the algorithm.

(ii) The best known *upper* bound on the maximum number of pivot steps is achieved by a randomized algorithm, and this algorithm has subexponential expected worst case behavior (see chapter 5). No similar result is known for any deterministic rule.

In the following we list some classical rules, all of which are deterministic. The presentation of randomized rules is postponed to the next chapter. We have just mentioned them here to draw your attention to the fundamental differences between both types of rules. Later, we will build on exactly these differences.

Below, the term 'improving variable' refers to any nonbasic variable that improves the objective function value when it is entered into the basis.

LARGEST COEFFICIENT. Enter the improving variable with largest coefficient in the z-row. This is the rule originally proposed by Dantzig [14], and it maximizes the improvement in z *per unit increase* of the entering variable. As shown by Klee and Minty [31], this rule leads to exponential worst case behavior.

LARGEST INCREASE. Enter the improving variable which leads to the largest *absolute* improvement in z. This rule is computationally more expensive than the LARGEST COEFFICIENT rule but locally maximizes the progress. It was proposed by Klee and Minty [31], together with the question whether their result on the LARGEST COEFFICIENT rule also holds for this rule. By a modification of the Klee-Minty construction, Jeroslow [26] was able to show that LARGEST INCREASE does not behave better than largest coefficient in the worst case.

STEEPEST EDGE. Enter the improving variable which maximizes the z-improvement per *normalized* unit increase of the entering variable (this is the increase necessary to translate the current feasible solution by a vector of unit length). Geometrically, this corresponds to choosing the steepest upward edge starting at the current vertex of the feasible polyhedron.

[4] To invoke Quicksort again: when pivoting with the first element, an adversary can easily fool the algorithm by supplying an already sorted sequence (and this cannot even be considered malicious). The strategy of pivoting with a random element is immune to such attempts.

By yet another modification of the Klee-Minty construction, Goldfarb and Sit [22] can prove the exponential worst case behavior of this rule.

SMALLEST SUBSCRIPT. Enter the improving variable with smallest subscript. This is Bland's rule, and it is of theoretical interest because it avoids cycling, as mentioned in Subsection 1.3.2. In practice it is abandoned in favor of more sensitive rules. Avis and Chvátal have shown that Bland's rule leads to an exponential number of pivot steps on the original Klee-Minty example [5].

LEAST ENTERED. Zadeh [49] has observed that the Klee-Minty example and all the derived constructions that lead to exponential behavior of the above rules have a 'lexicographic' structure. This means that there are 'higher order' variables which only enter the basis after the lower-ordered ones have entered and left it sufficiently many times. To kill such examples he proposes to enter the improving variable which has been entered least often before. No input is known which forces this rule to perform badly, and to find one (or prove that the rule is polynomial in the worst case) is a challenging open problem.

Chapter 2

An Abstract View of the Simplex Method

The simplex method, as described in the previous chapter, follows the paradigm of *successive improvement*. Not only that: the simplex method has actually created this paradigm. To find an optimal solution s_{opt} to some problem, the method of successive improvement relies on the following three properties of the problem.

Successive improvement properties.

(i) Some initial solution s is known.

(ii) For any solution s some (reasonably fast) routine exists that either certifies $s = s_{opt}$ or proves that $s \neq s_{opt}$ by exhibiting another solution s' which is better than s (with respect to the optimality criterion).

(iii) There are only finitely many solutions s.

In this chapter we concentrate on the simplex method as a concrete successive improvement method. In particular, we shall adopt a more abstract view in the sense that the actual pivoting routine becomes a black box. Based on this more abstract view, we present two randomized pivot rules for the simplex method. The chapter is a preparation for the following ones where problems (more general than LP) are studied that are solvable by successive improvement, in one or the other form. These will be *concrete* problems related to LP as well as *abstract* problems which are defined just by the property that successive improvement applies. In any case, we give concrete algorithms, all of which – when applied to LP – boil down to the simplex method with special (randomized) pivot rules.

Let us first discuss concrete conditions under which the simplex method is really a successive improvement method.

2.1 Successive Improvement

The solutions s maintained by the simplex method are basic feasible solutions of the LP, and as we have stated in Fact 1.3, there are only finitely many, so successive improvement property (iii) holds. The goal of Subsections 1.3.1 (Unboundedness), 1.3.2 (Degeneracy) and 1.3.3 (Infeasibility) was on the one hand to show that properties 1 and 2 also hold under certain assumptions, and on the other hand to show that these assumptions can (more or less easily) be guaranteed. Let us make these assumptions explicit, once and for all.

Assumption 2.1

(i) Some initial BFS for the LP is known.

(ii) The LP is bounded (this ensures that the proof of $s \neq s_{opt}$ consists of a legal solution s', which must be a BFS).

(iii) The LP is nondegenerate (this ensures that the solution s' is better than s and therefore is *really a proof for $s \neq s_{opt}$).*

We have seen that unboundedness is easily discovered and dealt with during the computations at no extra cost, and if no initial BFS is known, computing one from scratch just introduces a factor of 2 in the overall runtime. Thus, the first two assumptions are no real restrictions.

The nondegeneracy assumption – in particular, the method of symbolic perturbation, Subsection 1.3.2, to resolve degeneracies – may seem inadequate from a practical point of view, since degeneracies (which *do* occur in practical problems) are harmless unless they lead to cycling, which is so rare that most likely no precautions are taken at all. In any case, maintaining and manipulating degree-n polynomials for this purpose is considered as too expensive.

In our theoretical setting, perturbation is acceptable, since it does not lead to asymptotic overhead in a pivot step of the revised simplex method, as we have described it. Still, if we were talking about termination only, we could easily handle degeneracies by switching to Bland's rule whenever they occur. However, from the abstract point of view of successive improvement, degeneracies are more problematic: even if we are guaranteed to escape from a sequence of zero-progress pivots sooner or later, it is not clear how long this takes! This again means that we just do not have a 'reasonably fast' routine to ensure successive improvement property (ii). Even if there was some small bound on the maximum number of successive degenerate pivots, this would solve the problem only for LP. In a more general setting (and we will develop such a setting), it is not clear how to deal with zero-progress pivots. Cycling might become unavoidable, 'killing' the successive improvement approach in the whole. If, on the other hand, some fast procedure *is* known to overcome a zero-progress phase, this procedure might itself be regarded as the pivoting routine, removing zero-progress pivots altogether. With the convention that a single pivot step always makes

progress, successive improvement safely works in any situation. For LP, the nondegeneracy assumption achieves this. It should be considered as a (necessary) tribute to the generality of the method, even if it looks like 'overkill' in the specific situation.

2.2 The Abstract View

We have already seen in Section 1.4 that any BFS associated with a tableau

$$\begin{array}{rcrcr} x_B & = & \beta & - & \Lambda x_N \\ \hline z & = & z_0 & + & \gamma^T x_N, \end{array} \tag{2.1}$$

is determined by the set B of basic subscripts, via (1.25) and (1.29), so B is an abstract representation of the BFS from which the actual coordinates can be retrieved if necessary. In most of the subsequent considerations this is not necessary (and not useful, either). We have seen in quite some detail *how* pivoting works, and we might at a few points refer to some of these details. On the whole, we have good reasons to prefer an abstract view and only remember *what* pivoting does.

(i) Our complexity measure will typically be the *number* of pivot steps. On the one hand, this number depends on the pivot rule used, and the rule itself might depend on actual tableau coordinates; on the other hand, unlike some of the rules from the previous subsection, the randomized rules we are investigating only need to tell whether a variable is improving or not. This can easily be represented on an abstract level.

(ii) As already mentioned above, we study not only linear programming problems. There are other related problems, where the basic idea of succesive improvements applies, but where the concrete realization of an improvement step either is quite different from an LP pivot, or is only given via an improving oracle. The abstract view allows us to treat all these problems in a unified framework.

This section develops the necessary terminology to handle the abstract view. As already indicated, the building blocks of this view are *sets* of (basic) subscripts.

2.2.1 Bases

Recall that two properties are needed to make sure that the equality system (2.1) is actually a tableau that induces a BFS. (i) (2.1) is equivalent to the initial tableau, i.e. it can be obtained from the former by solving for x_B, and (ii) (2.1) is feasible, i.e. the vector β satisfies $\beta \geq 0$. By Section 1.4, this translates to the following conditions on the set B of basic subscripts.

Observation 2.2 *Let $B \subseteq \{1, \ldots, d+n\}$ be an n-set. There is a feasible tableau (2.1) with set of basic subscripts B if and only if*

(i) $\underline{A}_B$ *is invertible,*

(ii) $\beta = \underline{A}_B^{-1} b \geq 0$.

Definition 2.3

(i) A set B *satisfying (i) and (ii) in Observation 2.2 is called a* basis *(and* B *serves as an abstract representation of the 'real' basis* x_B*).*

(ii) For a basis B*,* $x(B)$ *is the basic feasible solution associated with (2.1), defined by* $\tilde{x}_B = \beta, \tilde{x}_N = 0$.

(iii) $z(B) := z_0$ *is the objective function value at the BFS* $x(B)$ *associated with (2.1).*

(iv) Finally, $\mathcal{B}$ *denotes the set of all bases.*

The simplex method searches for a basis B with maximum value $z(B)$ among all bases. The following definition lets us refer to this optimal basis by generalizing $z(B)$ and $x(B)$ to arbitrary sets, not only bases.

Definition 2.4 *Let* $H := \{1, \ldots, d+n\}$ *denote the set of variable subscripts.*

(i) For a subset $G \subseteq H$ *we define its* value *as* $z(G) := \max\{z(B) \mid B \in \mathcal{B}, B \subseteq G\}$. *(*$z(G) = -\infty$ *whenever* G *does not contain a basis.)*

(ii) A basis *of* G *is any basis* $B(G) \subseteq G$ *defining the value of* G*, i.e.* $z(B(G)) = z(G)$. *(This basis is not necessarily unique –* $B(G)$ *stands for any possible choice. This won't lead to problems).*

(iii) Finally, $x(G)$ *is the BFS associated with* $B(G)$*, with value* $z(G)$*, i.e.* $x(G) := x(B(G))$.

2.2.2 Subproblems

$B(H)$ corresponds to a BFS $x(H)$ with largest z-value among all BFS. If the LP is bounded (which we assume), such a BFS is computed by the simplex method. In the nondegenerate situation, $x(H)$ is characterized by the property that it has no improving variable, and in this case $z(H) = z(B(H))$ is the optimum value of the LP.

For $G \subseteq H$, $B(G)$ corresponds to a BFS $x(G)$ with largest z-value among all BFS with basic variables in G. Equivalently, $x(G)$ is the best BFS subject to $x_j = 0$, for all $j \in H - G$. Therefore, one would expect $x(G)$ to be an optimal solution to the problem when its feasible region is restricted to the intersection of the facets $x_j = 0, j \in H - G$. The following lemma proves this, together with the fact that $x(G)$ is characterized by the property that it has no improving variable in G.

Lemma 2.5 *Consider a bounded and nondegenerate LP in (compact) equality form (1.4), some* $G \subseteq H$ *with basis* $B(G)$*, and some BFS* $\tilde{x}$*. The following statements are equivalent.*

(i) $\tilde{x} = x(G)$.

(ii) *No variable in* G *is improving for* $\tilde{x}$.

(iii) $\tilde{x}$ *is an optimal feasible solution of the subproblem*

$$\begin{array}{ll} \textit{maximize} & \underline{c}^T x \\ \textit{subject to} & \underline{A}x = b, \\ & x \geq 0, \\ & x_{H-G} = 0, \end{array} \tag{2.2}$$

which is equivalent to

$$\begin{array}{ll} \textit{maximize} & \underline{c}_G^T x_G \\ \textit{subject to} & \underline{A}_G x_G = b, \\ & x_G \geq 0, \end{array} \tag{2.3}$$

with x_G, $\underline{A}_G$ *and* $\underline{c}_G$ *as defined in (1.20), (1.22) and (1.26).*

Proof. Let us first discuss in which sense (2.2) and (2.3) are equivalent. The following holds: $\tilde{y}$ is a feasible solution to (2.2), with value $\tilde{z}$ if and only if $\tilde{y}_{H-G} = 0$ and $\tilde{y}_G$ is a feasible solution to (2.3), with value $\tilde{z}$. This immediately follows from

$$\underline{c}^T x = \underline{c}_G^T x_G + \underline{c}_{H-G}^T x_{H-G} \text{ and } \underline{A}x = \underline{A}_G x_G + \underline{A}_{H-G} x_{H-G}.$$

In particular, both problems have the same optimum value, and (2.3) arises from (2.2) by removing the zero variables x_{H-G}.

(i) $\Rightarrow$ (ii). Recall that $x(G)$ by definition is the best BFS with basic variables in G, and if there was an improving variable in G, this variable could enter the basis, resulting in a better BFS with basic variables in G. (This argument needs boundedness.)

(ii) $\Rightarrow$ (iii). Consider the BFS $\tilde{x}$ and the tableau associated with it. This tableau has some z-row

$$z = z_0 + \gamma^T x_N,$$

with $N = H - B(G)$ the set of nonbasic variables and $z_0 = z(G) = z(B(G))$. We can rewrite this as

$$z = z_0 + \gamma_{N_1}^T x_{N_1} + \gamma_{N_2}^T x_{N_2},$$

where $N_1 := N \cap G$ collects the variables which are in G, $N_2 := N - G$ the ones not in G. Since $\tilde{x}$ has no improving variable in G, nondegeneracy implies that we must have $\gamma_{N_1} \leq 0$.

Let $\tilde{x}$ be any feasible solution of (2.2), with value $\tilde{z}$. The tableau properties guarantee

$$\tilde{z} = z_0 + \gamma^T \tilde{x}_N.$$

$N_2 = N - G \subseteq H - G$ implies $\tilde{x}_{N_2} = 0$, so

$$\tilde{z} = z_0 + \gamma_{N_1}^T \tilde{x}_{N_1} \leq z_0,$$

because of $\gamma_{N_1} \leq 0, \tilde{x}_{N_1} \geq 0$. Thus $z_0 = z(G)$ is the optimum value of (2.2), attained at $\tilde{x}$.

(iii) $\Rightarrow$ (i). Any BFS $\tilde{y}$ with basic variables in G is a feasible solution of (2.2), and $\tilde{x}$ is an optimal one among them. This means, $\tilde{x} = x(G)$. □

This lemma looks quite innocent and was easy to prove. Nevertheless, it has consequences: beyond just computing an optimal BFS $x(H)$ from some initial basis B, the simplex method is capable of solving the subproblem (2.2), for any $G \supseteq B$. (How? Just run the method, keeping the variables from $H - G$ nonbasic all the time. This terminates with some BFS for which no improving variables are left in G. The lemma shows that in this case $x(G)$, an optimal solution to (2.2) has been computed.) This observation is not at all deep and not at all surprising. It is clear that (2.2) – in particular in the form (2.3) – is a linear program itself that one expects to be solved by the simplex method. Moreover, we *want* to compute $x(H)$ and not the solutions to subproblems, so one might even consider the observation as not at all useful. Exactly the opposite is true. The RANDOM-FACET pivot rule we introduce in the next section and the efficiency of the RANDOM-FACET simplex method crucially depend on the subproblem solvability, which will also be a key feature of the abstract frameworks we are going to consider later.

In general, subproblem solvability induces additional structure, making the successive improvement paradigm potentially stronger. We will exploit this to quite some extent.

2.2.3 Pivoting

Let us complete our 'shift to abstraction' by defining – in terms of subscript sets – the notion of an improving variable and the operation of entering a variable into the basis. As before, this applies to bounded and nondegenerate LPs.

Theorem 2.6 (LP pivot)

(i) If B is the current basis, a variable $j \in H - B$ is improving, if and only if

$$z(B \cup \{j\}) > z(B). \tag{2.4}$$

(ii) The new basis B' obtained by making the improving variable j basic is given by

$$B' := B(B \cup \{j\}). \tag{2.5}$$

Proof. (i) If j is improving for B, entering it into the basis gives a basis of higher z-value, contained in $B \cup \{j\}$, hence $z(B \cup \{j\}) > z(B)$. If on the other hand, $z(B \cup \{j\}) > z(B)$ holds, then $x(B) \neq x(B \cup \{j\})$, so Lemma 2.5 certifies that there exists an improving

variable for $x(B)$ in $B \cup \{j\}$, and since the variables in B are already basic, this must be j.

(ii) If j is entered, we obtain a basis $B' = B \cup \{j\} - \{i\}$, i the leaving variable. If B' was not yet the optimal basis in $B \cup \{j\}$, then i would have to be an improving variable for B'. However, 'entering' i brings us back to B, a contradiction. ◻

We conclude with a lemma that – although being merely a corollary of Lemma 2.5, formulated in the abstract view – really captures the essence of this section.

Lemma 2.7 *Consider $G \subseteq H$ with $z(G) > -\infty$ and basis $B(G)$. For any $j \in H - G$,*

$$z(G \cup \{j\}) > z(G)$$

implies

$$z(B(G) \cup \{j\}) > z(B(G)).$$

Proof. By (2.4), equivalence (i) $\Leftrightarrow$ (ii) of Lemma 2.5 can be expressed as follows. Given some basis B,

$$B = B(G) \text{ if and only if } z(B \cup \{i\}) = z(B), \text{ for all } i \in G.$$

Thus, on the one hand

$$z(B(G) \cup \{i\}) = z(B(G)), \text{ for all } i \in G.$$

On the other hand, $z(G \cup \{j\}) > z(G)$ means $B(G) \neq B(G \cup \{j\})$, so

$$z(B(G) \cup \{i\}) > z(B(G)), \quad \text{for some } i \in G \cup \{j\}.$$

Consequently, $z(B(G) \cup \{j\}) > z(B(G))$. ◻

2.3 Randomized Pivot Rules

We have now laid the grounds for describing two randomized pivot rules in the abstract view. The Random-Edge rule is local in the sense that it chooses the entering variable independent from previous computations, while Random-Facet has 'memory'.

2.3.1 The Random-Edge Rule

Random-Edge does almost the simplest possible: among all candidates j for entering the basis it chooses a *random* one, each candidate chosen with the same probability. In the geometric interpretation, this rule traverses a random out of all improving edges starting at the current vertex. Given some initial basis $B \subseteq G \subseteq H$, the following algorithm computes $B(G)$, an optimal basis contained in G.

Algorithm 2.8

```
RANDOM-EDGE-SIMPLEX (G, B)
   (* returns B(G). B ⊆ G is an initial basis *)
1  I := {j ∈ G − B | z(B ∪ {j}) > z(B)} (* compute improving variables *)
2  if I ≠ ∅
3  then (* pivot step *)
4       choose random j ∈ I
5       B' := B(B ∪ {j}) (* enter variable j *)
6       return RANDOM-EDGE-SIMPLEX (G, B') (* repeat with B' *)
7  else (* B = B(G) *)
8       return B
9  fi
```

2.3.2 The Random-Facet Rule

RANDOM-FACET is nonlocal and recursive, so its functionality is best explained by describing the complete algorithm rather than a single pivot step. Given some basis B, the generic call of RANDOM-FACET-SIMPLEX finds $B(G)$, the optimal basis contained in some set $G \supseteq B$ of currently *admissible* variables. IF $B \subsetneq G$, this is done by recursively solving the problem for $G - \{j\}$ first, with j a variable chosen at random from all admissible variables which are nonbasic (i.e. not in B), each with the same probability. If the basis B' obtained from this recursive call is not yet optimal for G, a pivot step brings j into the basis, delivering a better basis B'' from which the process repeats.

In the geometric interpretation, (the top level of) this algorithm first optimizes recursively over a random facet incident to the initial vertex, and if this does not give the global optimum yet, it 'pivots away' from this facet to a better vertex from which it repeats. Note that down the recursion RANDOM-FACET-SIMPLEX heavily exploits subproblem solvability (Subsection 2.2.2).

Algorithm 2.9

```
RANDOM-FACET-SIMPLEX (G, B)
   (* returns B(G). B ⊆ G ⊆ H is an initial basis *)
1  if B = G
2  then (* B = B(G) *)
3       return B
4  else (* recur with smaller set *)
5       choose random j ∈ G − B
6       B' := RANDOM-FACET-SIMPLEX (G − {j}, B) (* find B(G − {j}) *)
7       if z(B' ∪ {j}) > z(B')
8       then (* pivot step *)
9            B'' := B(B' ∪ {j}) (* enter variable j *)
10           return RANDOM-FACET-SIMPLEX (G, B'') (* repeat with B'' *)
11      else (* B' = B(G) *)
```

12 **return** B'
13 **fi**
14 **fi**

2.3.3 Correctness

Lemma 2.10 *Given a bounded and nondegenerate LP with initial basis B, algorithm* Random-Edge-Simplex(G, B) *and algorithm* Random-Facet-Simplex(G, B) *compute $B(G)$, for all $G \supseteq B$.*

Proof. Both algorithms only feature pivot steps resulting in a new basis with strictly larger z-value than the current one (where boundedness makes sure that the result of the pivot step *is* actually a basis, Subsection 1.3.1). Since there are only finitely many bases, induction over the quality of B (and in addition over the size of G in case of Random-Facet-Simplex) shows that both algorithms will eventually terminate. It remains to show that in case of termination, the returned result is correct.

Random-Edge-Simplex(G, B) terminates if $I = \emptyset$, i.e. if there is no improving variable in G for the current basis B. By Lemma 2.5, $B = B(G)$ in this case.

Random-Facet-Simplex(G, B) has two possibilities for termination. IF $B = G$, then $B = B(G)$ holds by definition. Otherwise, we inductively assume that the recursive call in line 6 has computed $B' := B(G - \{j\})$. The call terminates with B' if

$$z(B' \cup \{j\}) = z(B').$$

We need to show that this implies $B(G - \{j\}) = B(G)$, equivalently

$$z(G) = z(G - \{j\}).$$

This, however, is exactly the statement of Lemma 2.7. ◻

Chapter 3

LP-type Systems

We have obtained algorithms 2.8 and 2.9 as concrete instances of the generic simplex method, by supplying concrete pivot rules. This approach turns history upside down. The true story is that Sharir and Welzl developed algorithm Random-Facet-Simplex in a quite different context, where it is *not* a simplex algorithm but a randomized, incremental algorithm to solve linear programs and other – more general – optimization problems, so-called *LP-type problems* [45]. Only when specialized to LP, their algorithm becomes a dual simplex algorithm.[1] The key to the more general view lies in the abstract formulation of the algorithm that allows us to interpret its basic operations in a different way. After all, Random-Facet-Simplex just works on sets G which have some z-value assigned to it, determined by the bases contained in G. For the correctness of the algorithm, z is only required to have the property of Lemma 2.7. As long as we can guarantee this and are able to supply the basic pivoting operations (2.4) and (2.5), we are free to redefine bases and their z-values arbitrarily – the algorithm will still compute a basis in G with largest z-value, whatever meaning this has then. We can even drop the implicit assumption that all bases have the same size. By these observations, we arrive at an abstract class of problems – called *LP-type systems* – that have just enough structure to make Algorithm 2.9 (and Algorithm 2.8 Random-Edge-Simplex) work. As it turns out, LP-type systems are slightly more general than Sharir and Welzl's LP-type problems, but the difference is only marginal. We introduce both, discuss their relations and embed algorithms Random-Facet-Simplex and Random-Edge-Simplex into this more general setting. We then review Sharir and Welzl's original result and its implications for LP. Finally, we discuss another related framework, namely *abstract polytopes* as introduced by Adler [1].

3.1 Two Abstract Frameworks

Definition 3.1 *For the whole chapter let $(\mathcal{W}, \leq)$ be some linearly ordered set. Let H be a finite set, $\mathcal{B} \subseteq 2^H$ some set of subsets of H, and $z : 2^H \mapsto \mathcal{W} \cup \{-\infty\}$ some function*

[1] We will see in Subsection 3.2.3 what this means.

with $z(B) \neq -\infty$ *for all* $B \in \mathcal{B}$*, and*

$$z(G) = \max\{z(B) \mid B \in \mathcal{B}, B \subseteq G\}, \text{ for all } G \subseteq H. \tag{3.1}$$

($z(G) = -\infty$ *if and only if* G *does not contain a basis.)*

$\mathcal{B}$ *is an* (abstract) set of bases*, and* z *is an* (abstract) objective function*, determined by* $\mathcal{B}$*.*

A basis *of* $G \subseteq H$ *is any basis* $B \subseteq G$ *with* $z(B) = z(G)$*. Consider* $G \subseteq H$ *and a basis* B *of* G*. If, for all* $j \in H - G$*,*

$$z(G \cup \{j\}) > z(G) \text{ implies } z(B \cup \{j\}) > z(B), \tag{3.2}$$

then the triple $(H, \mathcal{B}, z)$ *is called an* LP-type system*.*

$(H, \mathcal{B}, z)$ *is* basis-regular *if all bases have the same cardinality.*

The only missing ingredients are now operations (2.4) and (2.5), which supply all the problem-specific information the algorithm needs. Let us explicitly name these operations and assume that they come with any LP-type system $(H, \mathcal{B}, z)$.

Improvement query. Given basis B and $j \in H - B$, test whether

$$z(B \cup \{j\}) > z(B). \tag{3.3}$$

Basis computation. Given basis B and $j \in H - B$ such that $z(B \cup \{j\}) > z(B)$, compute a basis B' of $B \cup \{j\}$,

$$B' := B(B \cup \{j\}). \tag{3.4}$$

Let us review the basis computation with respect to LP-type systems. To this end look at line 9 of Algorithm 2.9, where it is used to compute the basis B''. For correctness, we argued that progress is made, i.e. that $z(B'') > z(B')$. In LP, making progress is equivalent to computing $B(B' \cup \{j\})$ (Theorem 2.6 (ii)). This is not true in a general LP-type system. If bases need not have fixed size, $B' \cup \{j\}$ might actually contain many bases B'' with $z(B'') > z(B')$, and any of them would do. Thus, finding a basis of $B' \cup \{j\}$ is not necessary (and might even be much more difficult than finding just some improving basis).

As a consequence, we relax the basis computation to a *basis improvement*.

Basis improvement. Given basis B and $j \in H - B$ such that $z(B \cup \{j\}) > z(B)$, compute *some* basis $B' \subseteq B \cup \{j\}$ with $z(B') > z(B)$. To have a notation for B' we assume that it results from a call to a function $\uparrow$ ('improvement oracle'), with parameters B and j,

$$B' :=\uparrow (B, j). \tag{3.5}$$

Sharir and Welzl's original definition of LP-type problems has *subproblems* as the basic building blocks, while we are starting from *bases*. The fact that both definitions turn out to be in very close relation, nicely conveys a general philosophy which Lemma 2.5 captures for the case of LP: the value of an optimal *basis* contained in some set G is at the same time the optimum value of the *subproblem* defined on G.

Definition 3.2 (Sharir, Welzl). *Let H be a finite set and $z : 2^H \mapsto \mathcal{W} \cup \{-\infty\}$ some function. The pair (H, z) is called an* LP-type problem *if and only if z satisfies the following two properties.*

Monotonicity. *$z(F) \leq z(G)$, for all $F \subseteq G \subseteq H$.*
Locality. *For all $F \subseteq G \subseteq H$, if $-\infty \neq z(F) = z(G)$ and*

$$z(G \cup \{j\}) > z(G),$$

then

$$z(F \cup \{j\}) > z(F),$$

for all $j \in H - G$.

The following lemma proves that if $(H, \mathcal{B}, z)$ is an LP-type system, then (H, z) defines an LP-type problem. Moreover, given an LP-type problem (H, z), we can reconstruct a set of bases $\mathcal{B}$ such that $(H, \mathcal{B}, z)$ is an LP-type system. The set $\mathcal{B}$ is not unique (and therefore no strict one-to-one correspondence between LP-type systems and LP-type problems exists), but all possible $\mathcal{B}$'s contain a unique common 'core' $\mathcal{B}(H, z)$ with the property that $(H, \mathcal{B}(H, z), z)$ is an LP-type system. This means, (H, z) determines a unique *minimal* LP-type system.

Lemma 3.3 *Let H be a finite set and $z : 2^H \mapsto \mathcal{W} \cup \{-\infty\}$ some function. Define*

$$\mathcal{B}(H, z) := \{B \subseteq H \mid -\infty \neq z(B) > z(B'), \forall B' \subsetneq B\}. \tag{3.6}$$

(i) (H, z) is an LP-type problem if and only if $(H, \mathcal{B}(H, z), z)$ is an LP-type system.
(ii) If $(H, \mathcal{B}, z)$ is an LP-type system, then

$$\mathcal{B}(H, z) = \{B \in \mathcal{B} \mid z(B) > z(B'), \forall B' \subsetneq B\}, \tag{3.7}$$

and $(H, \mathcal{B}(H, z), z)$ is an LP-type system.

Proof. (i) Assume (H, z) is an LP-type problem, $\mathcal{B} := \mathcal{B}(H, z)$. To see property (3.1), note that by monotonicity,

$$z(G) \geq \max\{z(B) \mid B \in \mathcal{B}, B \subseteq G\}. \tag{3.8}$$

If $G \in \mathcal{B}$, we get equality in (3.8). If $G \notin \mathcal{B}$, then either $z(G) = -\infty$, or there exists a minimal proper subset G' of G with $z(G) = z(G') \neq -\infty$. G' must be in $\mathcal{B}$, and equality in

(3.8) follows. Property (3.2) is just a special case of locality, so we conclude that $(H, \mathcal{B}, z)$ is an LP-type system.

Now let $(H, \mathcal{B}(H,z), z)$ be an LP-type system. For (H, z), monotonicity is obvious by (3.1). To prove locality, fix $F \subseteq G$ with $-\infty = z(F) = z(G)$, let B be some basis of F (and therefore of G) and assume

$$z(G \cup \{j\}) > z(G),$$

for some $j \in H - G$. Then, by monotonicity and property (3.1),

$$z(F \cup \{j\}) \geq z(B \cup \{j\}) > z(B) = z(F),$$

and locality follows.

(ii) Let $(H, \mathcal{B}, z)$ be an LP-type system. In (3.7), the inclusion '$\supseteq$' follows from definition (3.6), together with $z(B) \neq -\infty$ for $B \in \mathcal{B}$. For the other direction, consider $B \in \mathcal{B}(H,z)$. Since $z(B) \neq -\infty$, some basis B' of B must exist in $\mathcal{B}$, with $z(B) = z(B')$, and since $z(B) > z(B'')$ for all proper subsets B'', B must equal B', hence $B \in \mathcal{B}$.

To see that $(H, \mathcal{B}(H,z), z)$ is again an LP-type system, observe that (3.1) holds, because for any G, the inclusion minimal basis $B \in \mathcal{B}$ defining $z(G)$ is in $\mathcal{B}(H,z)$. Property (3.2) is trivially satisfied because $\mathcal{B}(H,z)$ is a subset of $\mathcal{B}$. □

After we have convinced ourselves that the simplex variant Random-Facet-Simplex actually works for any LP-type system (and this holds for Random-Edge-Simplex as well, of course), we conclude by explicitly writing down both algorithms in this more general setting, under the names of RF-LPtype resp. RE-LPtype.

Algorithm 3.4

```
RF-LPtype (G, B)
   (* returns B(G). B ⊆ G ⊆ H is an initial basis *)
1  if B = G
2  then (* B = B(G) *)
3       return B
4  else (* recur with smaller set *)
5       choose random j ∈ G − B
6       B' := RF-LPtype (G − {j}, B) (* find B(G − {j}) *)
7       if z(B' ∪ {j}) > z(B')
8       then (* basis improvement *)
9            B'' :=↑ (B', j)
10           return RF-LPtype (G, B'') (* repeat with B'' *)
11      else (* B' = B(G) *)
12           return B'
13      fi
14 fi
```

Algorithm 3.5

RE-LPTYPE (G, B)
(* returns $B(G)$. $B \subseteq G$ is an initial basis *)
1 $I := \{j \in G - B \mid z(B \cup \{j\}) > z(B)\}$ (* improvement queries *)
2 **if** $I \neq \emptyset$
3 **then** (* basis improvement *)
4 choose random $j \in I$
5 $B' :=\uparrow (B, j)$
6 **return** RE-LPTYPE (G, B') (* repeat with B' *)
7 **else** (* $B = B(G)$ *)
8 **return** B
9 **fi**

The insights of this section can be summarized as follows.

Theorem 3.6 *Let $(H, \mathcal{B}, z)$ be an LP-type system, specified by operations (3.3)* improvement query *and (3.5)* basis improvement. *If some $B \in \mathcal{B}$ is known, algorithms* RF-LPTYPE(G, B) *and* RE-LPTYPE(G, B) *compute a basis $B(G)$ of G, for all $G \supseteq B$.*

Up to Chapter 5, algorithm RF-LPTYPE will dominate the picture. Only in chapter 6 do we derive results concerning RE-LPTYPE.

3.2 Time Bounds

Now that we have climbed a sufficiently abstract level by introducing an abstract problem class and algorithms to solve problems in this class, you might ask the following questions.

- Are there any (natural) LP-type systems other than LP itself? In other words, does the abstraction really bring about more generality?

- Even if plenty of interesting LP-type systems do exist, does algorithm RF-LPTYPE represent more than just some straightforward backtracking method to solve them? In particular, does it represent an *efficient* method?

- What is the randomization good for? So far we were mainly discussing termination and correctness issues, and for this, randomization was obviously irrelevant.

The last two questions – which are closely related – are addressed in this section. Only in the next chapter do we answer the first question by presenting other 'real' LP-type system. For LP, however, this section already leads to nontrivial results.

3.2.1 Basic Ideas

By looking at Algorithm 3.5 one realizes that it indeed has a backtracking flavor. To find the optimal basis contained in some set G, we first (recursively) search among the bases not containing some chosen element $j \in G$ (line 6), and if we have not been successful yet, we (recursively) search among the ones containing j (line 10). The latter is a 'subtle triviality': we never *explicitly* tell the algorithm that we would like only bases containing j to be considered in the second phase (and there would be no way to do this), but it *implicitly* happens anyway. Just by construction, the basis B'' computed in line 9 has higher z-value than the best basis B' not containing j. Therefore, j is an element of B'' and any subsequent, still better, basis. We say that j is *enforced* in (G, B''). While the algorithm proceeds, more and more elements become enforced, and this intuitively means that the 'dimension' of the problem gets smaller and smaller.

Of course, there can never be more than $|G|$ enforced elements, so the dimension of a pair (G, B) can actually be defined as the gap between $|G|$ and the number r of currently enforced elements in (G, B). Since all the enforced elements lie in the basis B, we can as well bound r by the maximum cardinality δ of any basis and consider the number $\delta - r$ as a measure of dimension. This is the motivation behind the following definition.

Definition 3.7 *Let $\mathcal{L} = (H, \mathcal{B}, z)$ be an LP-type system. The number*

$$\delta := \max\{|B| \mid B \in \mathcal{B}\}$$

is called the combinatorial dimension *of $\mathcal{L}$.*

Let $B \subseteq G \subseteq H$, B a basis. An element $j \in G$ is called enforced *in (G, B) if*

$$z(B) > z(G - \{j\}).$$

The number

$$\delta(G, B) := \min(\delta, |G|) - \#\{j \in G \mid j \text{ enforced in } (G, B)\}$$

is called the hidden dimension *of (G, B).*

Note that if j is enforced in (G, B), then $j \in B$, so $\delta(G, B) \geq \min(\delta, |G|) - |B| \geq 0$. If $\delta(G, B) = 0$, then $z(B) = z(G)$ and B is a basis of G, as one might have expected. To see this, consider any other basis $B' \subseteq G$. Since $|B| = \min(\delta, |G|)$ holds, which implies $|B'| \leq |B|$, there exists an element $j \in B - B'$, so $B' \subseteq G - \{j\}$, which in turn implies $z(B') \leq z(G - \{j\}) < z(B)$. Thus B is optimal (and even unique with this property) among all bases contained in G which proves the claim.

As we show next, two ingredients make algorithm RF-LPtype efficient, and this is where the randomization comes in. First, if the hidden dimension is small compared to $|G|$ (which in particular is the case if δ is small compared to $|G|$), the second recursive call in line 10 is necessary only with small probability. Second, in compensation for the unlucky event that the second call *does* become necessary, the hidden dimension decreases quickly on average.

We start with analyzing the unlucky event. To this end consider the elements $\{j_1, \ldots, j_k\}$ that actually cause a second recursive call if they are chosen in line 5, together with their respective bases of line 6 and 9,

$$\begin{array}{rcl} B'_\ell & = & B(G - \{j_\ell\}) \\ B''_\ell & = & \uparrow(B'_\ell, j_\ell) \end{array}, \ell = 1, \ldots, k.$$

Assume the j_ℓ are ordered by increasing z-values of the B'_ℓ, such that

$$z(G - \{j_1\}) \leq \cdots \leq z(G - \{j_k\}).$$

Obviously, the larger the subscript ℓ of the chosen j_ℓ, the more progress does the algorithm make in line 6, and this progress can be quantified in terms of decrease in hidden dimension of the second recursive call.

Lemma 3.8 *(G, B''_ℓ) has hidden dimension at most $\delta(G, B) - \ell$.*

Proof. Because of $z(B''_\ell) > z(B)$, all elements which are enforced in (G, B) are also enforced in (G, B''_ℓ). Moreover,

$$z(B''_\ell) > z(B'_\ell) = z(G - \{j_\ell\}) \geq z(G - \{j_{\ell-1}\}) \geq \ldots \geq z(G - \{j_1\}),$$

so (G, B''_ℓ) features at least ℓ new enforced elements $j_1, \ldots, j_\ell$. □

Since the hidden dimension cannot get negative, this also shows that no more than $\delta(G, B)$ elements $j \in G - B$ can trigger a second recursive call.

Corollary 3.9 *Consider the call* RF-LPTYPE(G, B)*, $B \neq G$, and let B' be the basis of $G - \{j\}$ computed in line 6 of the algorithm. Then*

$$\text{prob}(z(B' \cup \{j\}) > z(B')) \leq \frac{\delta(G, B)}{|G - B|},$$

where the probability is over all choices of $j \in G - B$ in line 5.

3.2.2 The Bound

We want to derive an upper bound on the maximum expected number of primitive operations (improvement queries and basis improvements) in Algorithm RF-LPTYPE. Below we bound the number of improvement queries, and since any basis improvement is preceded by such a query, the overall number of primitive operations is at most twice as large.

Now, fix some LP-type system $(H, \mathcal{B}, z)$ of combinatorial dimension δ. For $m, k \geq 0$ we let $T(m, k)$ denote the maximum expected number of improvement queries performed in a call to RF-LPTYPE(G, B), $G \subseteq H$, $B \subseteq G$ basis, with $|G| = m, \delta(G, B) \leq k$.

Theorem 3.10

(i) If $m \leq \delta + 1$, then $T(m, k) \leq 2^{k+2}$.

(ii) If $m \geq \delta + 1$, then $T(m,k) \leq 2^{k+2}(m-\delta)$.

Proof. (i) Consider the improvement queries

$$z(B_t \cup \{j_t\}) > z(B_t), \quad t = 1, \ldots$$

performed during RF-LPtype(G, B), in chronological order. First note that there are no more than 2^{k+1} distinct sets $B_t \cup \{j_t\}$, because $\min(\delta, m) - \delta(G, B) \geq m - 1 - k$ elements $E \subseteq G$ are enforced in (G, B) so that $E \subseteq B_t \subseteq B_t \cup \{j_t\}$, for all t. Thus, any $B_t \cup \{j_t\}$ (as well as any B_t) is determined by a subset of the remaining at most $k+1$ elements. In particular, there are no more than 2^{k+1} basis improvements.

Now we show that every set $B_t \cup \{j_t\}$ appears for at most two values of t (amortized). Consider some number s and the next larger number u (if it exists) such that

$$B_s \cup \{j_s\} = B_u \cup \{j_u\}. \tag{3.9}$$

By inspection of RF-LPtype we see that no basis is ever tested twice with the same element, so $B_s \neq B_u$. Since any basis change is an improvement, we have

$$z(B_s) < z(B_u) \leq z(B_u \cup \{j_u\}) = z(B_s \cup \{j_s\}),$$

which means that the improvement query at time s had a positive answer, and we can charge equality (3.9) to the basis improvement $B_{s+1} := \uparrow (B_s, s_t)$. For fixed s, u is unique, so each of the at most 2^{k+1} basis improvements is charged only once; in other words, while scanning the sequence $B_t \cup \{j_t\}, t = 1, \ldots$ we will at most 2^{k+1} times encounter a set which we have already seen before, and as argued above, we can only encounter 2^{k+1} sets that we have *not* seen before. This implies (i).

(ii) We claim that for $m > \delta$,

$$T(m,k) \leq T(m-1,k) + 1 + \frac{1}{m-\delta} \sum_{\ell=1}^{\min(k, m-\delta)} T(m, k-\ell) \tag{3.10}$$

holds. To see this, observe that the call to RF-LPtype$(G - \{j\}, B)$ in line 6 solves a problem on $m-1$ elements, with $\delta(G - \{j\}, B) \leq \delta(G, B)$ (if i is enforced in (G, B), then $z(B) > z(G - \{j\}) \geq z(G - \{j, i\})$, so i is enforced in $(G - \{j\}, B)$ as well), so the expected number of improvement queries in this call is at most $T(m-1, k)$. Then we do one improvement query in line 7. The subsequent effort has to be averaged over all $|G - B|$ choices of j in line 5. As the previous corollary shows, at most k (and of course no more than $|G - B|$) of these choices actually lead to additional calls, where by Lemma 3.8, the ℓ-th such choice gives hidden dimension at most $k - \ell$. This bounds the expected effort after line 7 by

$$\frac{1}{|G-B|} \sum_{\ell=1}^{\min(k, |G-B|)} T(m, k-\ell),$$

which is no more than the last term of (3.10) because $T(m, k-\ell)$ decreases with increasing ℓ, and so the average of $T(m, k-\ell)$ over the range $\ell = 1, \ldots, |G-B|$ is no larger than the average over the smaller range $\ell = 1, \ldots, m-\delta$.

Now we can prove the bound stated in (ii) by induction. For $k = 0$, (3.10) gives $T(m, 0) = T(m-1, 0) + 1$, and using (i) this implies

$$T(m, 0) = m - \delta - 1 + T(\delta + 1, 0) \leq m - \delta + 3 \leq 4(m - \delta),$$

so (ii) holds for $k = 0$. The case $m = \delta + 1$ has already been handled in (i), so let $k > 0, m > \delta + 1$ and assume that (ii) holds for all smaller values of m and k. We inductively get

$$\begin{aligned} T(m,k) &\leq 2^{k+2}(m-\delta-1) + 1 + \frac{1}{m-\delta} \sum_{\ell=1}^{\min(k,m-\delta)} 2^{k+2-l}(m-\delta) \\ &\leq 2^{k+2}(m-\delta-1) + 1 + \sum_{\ell=1}^{k+2} 2^{k+2-l} \\ &= 2^{k+2}(m-\delta-1) + 1 + 2^{k+2} - 1 = 2^{k+2}(m-\delta). \end{aligned}$$

□

Result 3.11 *Given an LP-type system $(H, \mathcal{B}, z)$ of combinatorial dimension at most δ, with $|H| = m > \delta$. If B is some basis, then procedure* RF-LPTYPE(H, B) *computes a basis of H with an expected number of at most*

$$2^{\delta+2}(m-\delta)$$

improvement queries, resp. basis improvements (primitive operations). (For $m < \delta$, a bound of

$$2^{m+2}$$

holds.)

The result shows that the expected performance of RF-LPTYPE(G, B) is linear in m for fixed δ, which is in sharp contrast to the trivial worst case bound of

$$\binom{m}{\leq \delta} = O(m^\delta)$$

given by the maximum number of bases. Result 3.11 has first been obtained by Sharir and Welzl for the case that an actual basis computation (3.4) is available [45]. We have proved it in presence of the weaker basis improvement primitive (3.5), and this requires a slightly different argumentation in the proof of Theorem 3.10 (i).

3.2.3 Bounds for LP

What are the implications of Result 3.11 for linear programming as the 'master' LP-type system? In the first chapter we have considered the standard form linear program

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & c^T x \\ & \text{subject to} & Ax \leq b, \\ & & x \geq 0 \end{array} \tag{3.11}$$

in d variables and n constraints (plus d nonnegativity constraints). Subsection 2.2.1 has then prepared the ground for the subsequent formulation as an LP-type system $(H, \mathcal{B}, z)$, under the boundedness and nondegeneracy Assumption 2.1. First, we have set

$$H := [d + n].$$

The set of bases $\mathcal{B}$ has then been characterized by Observation 2.2 and the function z defined in Definition 2.3. The cardinality of any basis and therefore the combinatorial dimension of LP in this setting is n.

Improvement query and basis improvement are operations (2.4) and (2.5) in Subsection 2.2.3, which we have shown to be just abstract analogs of the two basic steps into which a pivot step subdivides, namely (i) checking whether a variable is improving, by evaluating its coefficient in the z-row of the current tableau, and (ii) constructing the next tableau by entering the variable into the basis, Subsection 1.2.2. How this can be done in time $O(n^2)$ per primitive operation, has been demonstrated in Section 1.4, where we have described the revised simplex method. Finally, recall that for LP, algorithm RF-LPtype specializes to algorithm 2.9 Random-Facet-Simplex.

Now, the stage is set.

Theorem 3.12 *A bounded, nondegenerate standard form linear program (3.11) in d variables and n constraints ($n > 0$) can be solved with an expected number of*

$$O(n^2 2^{n+2} d)$$

arithmetic operations, by the randomized simplex Algorithm 2.9, Random-Facet-Simplex*, provided some initial basis is known.*

If you are a computational geometer, this bound looks probably strange to you. It is reasonable if n is very small, or $d \gg n$ but becomes quite poor otherwise. However, in computational geometry, d is usually small or even constant and n is large; quite frequently, problems in 2-d or 3-d arising in computer graphics, CAD etc., can be cast as low-dimensional LP's. In such applications, the above bound is useless – we would like it the other way round. In fact, we can get it the other way round by invoking LP duality. To motivate the concept, consider (3.11).

For any n-vector $\tilde{y} \geq 0$, the inequality

$$\tilde{y}^T Ax \leq \tilde{y}^T b$$

is a linear combination of the inequality constraints in (3.11), satisfied by all feasible solutions $\tilde{x}$ (where $\tilde{y} \geq 0$ guarantees that the inequalities do not reverse direction). If in addition

$$\tilde{y}^T A \geq c^T$$

holds, then

$$\tilde{y}^T b \geq c^T x$$

is valid for all feasible solutions $x \geq 0$. Therefore, $\tilde{y}^T b$ is an upper bound for the optimum value of (3.11), and the smallest such upper bound (if one exists at all) is obtained as the optimum value of the program

$$\begin{array}{lll} \text{(LP)} & \text{minimize} & y^T b \\ & \text{subject to} & A^T y \geq c, \\ & & y \geq 0. \end{array} \tag{3.12}$$

Our argumentation implies that the optimum value of (3.12) is at least as large as the optimum value of (3.11). In fact, if one of them has an optimum value, so has the other, and both values coincide. (3.12) is called the *dual* of (3.11), which itself is called the *primal* [10].

Therefore, one can solve (3.11) by solving its dual (after rewriting it to standard form). The dual now has n variables and d constraints and can be solved in time

$$O(d^2 2^{d+2} n), \tag{3.13}$$

which is the 'right' bound from a computational geometry point of view.

Since a natural one-to-one correspondence exists between the respective bases of primal and dual, it is not surprising that the actions taken by algorithm Random-Facet-Simplex in the course of solving (3.12) can directly be interpreted as operations on (3.11), without any reference to the dual; the algorithm then becomes a *dual simplex algorithm*, based on dual versions of the primitive operations. Under this interpretation, Sharir and Welzl [45] have first obtained bound (3.13), by describing the dual primitive operations to plug into algorithm 3.4 RF-LPtype in case of LP.

You might wonder whether in case of LP the general bound on the number of primitive operations given by Result 3.11 can be improved by using specific properties of LP. For example, in part (i) of Theorem 3.10, instead of 2^{k+2} a bound of 2 holds! This is due to the fact that for $|G| = \delta + 1 = n + 1$, the call RF-LPtype(G, B), $G = B \cup \{j\}$ boils down to a single pivot step: if j is improving for B, a basis computation by Theorem 2.6 (ii) already delivers $B(B \cup \{j\}) = B(G)$, which by ignorance of the algorithm finally undergoes another, unnecessary, improvement query, for a total of at most two queries. This seems like a dramatic saving, with implications also for the bound in part (ii) of Lemma 3.10. Indeed, there are such implications. We will see in Chapter 5 that even for LP-type systems, the bound of Result 3.11 is not the last word.

One final remark is in order here. Theorem 3.12 (and therefore bound (3.13) as well) do actually hold for *any* standard form LP. When we made Assumption 2.1 we argued – based on our 'exception handling' in Subsections 1.3.1, 1.3.2 and 1.3.3 – that all three parts of it (boundedness, nondegeneracy, existence of initial basis) can be guaranteed at (asymptotically) no extra cost, where the only part requiring real computational effort is the symbolic perturbation to achieve nondegeneracy (conceptually, this is simple as well). Thus, the material so far takes you all the way from the standard form LP (1.1) we have started with to the runtime bounds of this section, with no details (but only some proofs you really find anywhere) left out. In particular, we have shown that the well-known and widely used simplex method – under the pivot rule RANDOM-FACET – becomes a theoretically optimal, linear time algorithm for solving linear programs in fixed dimension.

3.3 Abstract Polytopes

Although the concept of LP-type systems/problems presents the most recent (and so far most general) approach to abstraction of linear programming, it is not the first one. In this section we review another, closely related, framework from the literature, namely *abstract polytopes* with objective function.

Abstract polytopes have been introduced by Adler [1] as a combinatorial abstraction of simple polytopes. Just like in case of LP-type problems, the aim was to have a framework that allows combinatorial questions to be treated on a combinatorial level, and there are many known results for polytopes that can be shown to hold for abstract polytopes as well, see [3] and the references there. Generalizing linear objective functions on simple polytopes, Adler and Saigal [3] introduced objective functions on abstract polytopes. We show that an abstract polytope together with an objective function defines an LP-type system. In particular, we will see that the defining property of objective functions on abstract polytopes is equivalent to the locality condition (3.2) of LP-type systems, so there is reason to believe that this is a natural condition. Furthermore, we examine what exactly makes LP-type systems more general than abstract polytopes.

3.3.1 Abstract Polytopes, Faces and Objective Functions

A d-dimensional polytope is *simple* (or *nondegenerate*) if any vertex lies on exactly d facets. Moreover, if we label every vertex v with the set V of facets that contain it, we arrive at an abstract polytope, defined as follows.

Definition 3.13 *Let S be a finite set, $d \leq |S|$ and*

$$\mathcal{V} \subseteq \binom{S}{d}$$

a set of vertices. *The pair $(S, \mathcal{V})$ is called an* abstract polytope *of dimension d if the following two properties hold.*

(i) Any $(d-1)$-subset of S is either contained in no or in exactly two vertices (which then are called adjacent).

(ii) For any two vertices $V, V' \in \mathcal{V}$, there exist a path of adjacent vertices

$$V = V_0, \ldots, V_k = V',$$

all containing $V \cap V'$, i.e.

$$\begin{aligned} V \cap V' &\subseteq V_i, && (3.14)\\ |V_i \cap V_{i+1}| &= d-1, && (3.15) \end{aligned}$$

for $i = 0, \ldots, k-1$.

In algebraic topology, abstract polytopes are well-known as *pseudomanifolds* without boundary [3, 46]. Note that if $\mathcal{V}$ is defined by a simple polytope, then any $(d-1)$-set U contained in two vertices corresponds to an edge of the polytope.

An objective function on $(S, \mathcal{V})$ is any injective function $w : \mathcal{V} \mapsto \mathcal{W}$, $(\mathcal{W}, \leq)$ a linearly ordered set as already considered for LP-type systems, such that for any vertex V, there exists not only some, but a *w-increasing* path of adjacent vertices to the unique w-maximal vertex V'. Moreover, this has to be true not only for the whole (abstract) polytope but for every face of it. Thus, just like LP-type problems, this framework features subproblem solvability.

Definition 3.14 *Let $(S, \mathcal{V})$ be an abstract polytope. The* face *of* $(S, \mathcal{V})$ induced *by $U \subseteq S$ is the set of vertices*

$$\mathcal{V}(U) = \{V \in \mathcal{V} \mid U \subseteq V\}.$$

For a simple polytope, $\mathcal{V}(U)$ collects the vertices on the face induced by the (intersection of) facets in U.

Thus, $\mathcal{V}(\emptyset) = \mathcal{V}$, and $\mathcal{V}(U) = \emptyset$ for $|U| > d$. If $U \in \mathcal{V}$, then $\mathcal{V}(U) = \{U\}$. Note that every face defines itself an abstract polytope

$$(S - U, \{V - U \mid V \in \mathcal{V}(U)\}) \qquad (3.16)$$

of dimension $d - |U|$ (or empty) because (3.14) guarantees that for $V, V' \in \mathcal{V}(U)$, the path connecting them lies entirely in $\mathcal{V}(U)$.

Now let $w : \mathcal{V} \mapsto \mathcal{W}$ be an injective function, and let $V(U)$ denote the unique vertex of $\mathcal{V}(U)$ satisfying

$$w(V(U)) = \max\{w(V) \mid V \in \mathcal{V}(U)\}.$$

Definition 3.15 *w is an* objective function *on $(S, \mathcal{V})$ if it satisfies the following property for all $U \subseteq S$.*

For any vertex $V \in \mathcal{V}(U)$, there exists a path

$$V = V_0, \ldots, V_k = V(U)$$

satisfying (3.14), (3.15), and in addition $w(V_i) < w(V_{i+1})$, for $i = 0, \ldots, k-1$, i.e. the path is strictly w-increasing.

To be able to maximize *and* minimize w over $(S, \mathcal{V})$, Adler and Saigal originally demanded the existence of a *w-decreasing* path to the *minimal* vertex of every face as well. We won't do that here.

If $\mathcal{V}$ is obtained from a simple polytope, then any linear function in general position (with respect to the polytope) defines an objective function on $(S, \mathcal{V})$.

3.3.2 Relation to LP-type Systems

In the following we elaborate on the connections between abstract polytopes and LP-type systems, and we start with a lemma that lets Definition 3.15 appear in a strong LP-type flavor.

Lemma 3.16 *Let $(S, \mathcal{V})$ be an abstract polytope. w is an objective function if and only if*

$$V = V(U), V \neq V(U - \{j\}) \text{ implies } V \neq V(V - \{j\}), \tag{3.17}$$

for all $j \in U \subseteq V \subseteq S$.

Proof. Let w be an objective function. If $V \neq V(U - \{j\})$, then consider the path

$$V = V_0, \ldots, V_k = V(U - \{j\})$$

guaranteed by Definition 3.15. In particular $w(V_1) > w(V)$ holds, with

$$V_1 \supseteq V \cap V(U - \{j\}) \supseteq U - \{j\}$$

and

$$V_1 \not\supseteq U,$$

since V was already optimal in $\mathcal{V}(U)$. This implies $j \notin V_1$, and since V_1 is adjacent to V, we get

$$V_1 \supseteq V \cap V_1 = V - \{j\}.$$

This in turn means $V \neq V(V - \{j\})$.

Now suppose (3.17) holds, and let V be any vertex in $\mathcal{V}(U)$, $V' := V(U)$. We can assume that $V \cap V' = U$, otherwise we argue with $U := V \cap V'$.

If $V = V(U)$, there is nothing to prove. Otherwise, we need to find a w-increasing path

$$V = V_0, \ldots, V_k = V'$$

of adjacent vertices, all containing U. Let us construct V_1. To this end choose U' of minimal size such that

$$V = V(U') \text{ and } U \subseteq U' \subseteq V.$$

Such an U' exists because $V = V(V)$. Moreover, $U' \neq U$ because $V \neq V(U)$. Choose $j \in U' - U$. By choice of U',

$$V \neq V(U' - \{j\}),$$

and (3.17) implies

$$V \neq V(V - \{j\}),$$

so there exists $V_1 \in \mathcal{V}(V - \{j\})$ with

$$w(V_1) > w(V) \text{ and } V_1 \supseteq V - \{j\} \supseteq U' - \{j\} \supseteq U.$$

Then repeat with $V := V_1$. After a finite number of steps this process must reach $V(U)$, and the desired path has been constructed. ◻

Now define

$$\begin{aligned} \mathcal{B} &:= \{S - V \mid V \in \mathcal{V}\}, && (3.18)\\ z(S-U) &:= w(V(U)), \text{ for all } U \subseteq S. && (3.19) \end{aligned}$$

Thus, z assigns to $S - U$ the w-value corresponding to the optimal vertex in the abstract polytope (3.16) defined on $S - U$. In particular, $z(S - V) = w(V)$, for all $V \in \mathcal{V}$ and $z(S - U) = -\infty$ for $|U| > d$. z is completely determined by $\mathcal{B}$, because

$$\begin{aligned} z(S-U) &= w(V(U)) \\ &= \max\{w(V) \mid V \supseteq U, V \in \mathcal{V}\} \\ &= \max\{z(S-V) \mid S - V \subseteq S - U, S - V \in \mathcal{B}\}. \end{aligned}$$

Hence, in the sense of Definition 3.1, z is an abstract objective function determined by the abstract set of bases $\mathcal{B}$. We remark that this would also hold if we had defined $\mathcal{B}$ and z in the most straightforward way, namely by

$$\begin{aligned} \mathcal{B} &:= \mathcal{V}, \\ z(U) &:= \max\{w(V) \mid V \subseteq U, V \in \mathcal{V}\}. \end{aligned}$$

However, while the set of vertices *containing* U nicely defines an abstract polytope again, the vertices *contained* in some set U have no particular structure, so this definition would lead to nothing.

On the other hand, if $\mathcal{B}$ and z are determined by (3.18) and (3.19), then the condition of Lemma 3.16 is directly equivalent to the defining condition (3.2) of LP-type systems; more precisely, if $G = S - U, B = S - V$, then

$$\begin{array}{ccccc} z(B) = z(G), & z(G \cup \{j\}) > z(G) & \Rightarrow & z(B \cup \{j\}) > z(B) \\ \Updownarrow & \Updownarrow & & \Updownarrow \\ V = V(U), & V(U) \neq V(U - \{j\}) & \Rightarrow & V \neq V(V - \{j\}). \end{array} \qquad (3.20)$$

Consequently, an abstract polytope with objective function defines an LP-type system.

Theorem 3.17 *Let $(S, \mathcal{V})$ be a d-dimensional abstract polytope with objective function w. Then $(S, \mathcal{B}, z)$ with $\mathcal{B}$ and z as defined in (3.18) and (3.19), is an LP-type system of combinatorial dimension $\delta = |S| - d$.*

The LP-type systems obtained in this way are of special nature. First, all bases have the same size (we called this property *basis-regularity*), and second, all bases have distinct z-values (this is called *nondegeneracy*). The obvious question is whether *any* basis-regular, nondegenerate LP-type system $\mathcal{L}$ actually arises from an abstract polytope in this way.

To answer this, we first observe that, given such a system $\mathcal{L} = (S, \mathcal{B}, z)$, (3.18) and (3.19) can be applied backwards to give unique candidates $\mathcal{V}$ and w for the vertices and the objective function. Moreover, via (3.20) and Lemma 3.16, the LP-type system property (3.2) implies the existence of w-increasing paths as demanded by Definition 3.15, and by each time pasting together two such paths, we also obtain the paths required by Definition 3.13 (ii) – just connect any pair of vertices V, V' by their increasing paths to $V(V \cap V')$. Via (3.18), part (i) of Definition 3.13 is equivalent to the statement that any $\delta + 1$-subset of S, $\delta = |S| - d$, contains no or exactly two bases $B \in \mathcal{B}$. This, however, does not hold in general, even if $\mathcal{L}$ is basis-regular and nondegenerate. In fact, a $\delta+1$ set might contain *any* number of bases, up to $\delta + 1$. In this respect, even basis-regular, nondegenerate LP-type systems are truly more general than abstract polytopes.

Example 3.18 *Let H be a finite set, $\delta < |H|$. Choose $F \subseteq H$, $|F| = \delta + 1$ and let $\mathcal{B}$ consist of k arbitrary δ-subsets of F, $1 \leq k \leq \delta + 1$. Let z assign distinct values to all the k bases. Then $(H, \mathcal{B}, z)$ is a basis-regular, nondegenerate LP-type system of combinatorial dimension δ, with F containing exactly k bases.*

Chapter 4

Convex Optimization

In this chapter we consider a class of nonlinear optimization problems. We show them to be LP-type problems and present primitive operations *improvement query* and *basis improvement* for any problem in this class. Thus – keeping our promise from the previous chapter – we present 'real' LP-type problems different from LP.

The class consists of special *convex programming problems* (which in general are problems of minimizing a convex function subject to convex constraints), and it has been chosen according to two objectives.

(i) It should be general enough to cover two concrete problems that we are particularly interested in, namely the *polytope distance* problem and the *minimum spanning ball* problem.

(ii) It should be specific enough to allow a detailed treatment without drowning in technicalities.

The class we get covers problems of minimizing a convex function subject to *linear* equality and nonnegativity constraints.

We start off by introducing the polytope distance problem and derive from this the class of convex programs we are going to consider. We present a solution method for the generic problem in this class, along with time bounds, where we keep track of what concretely happens for the polytope distance problem. Having done this, we apply the developed machinery to obtain a solution for the minimum spanning ball problem. The major result is that for fixed dimension, both problems allow linear-time bounds similar to the ones we have derived for LP in (3.13).

4.1 The Polytope Distance Problem

Given two (disjoint and finite) sets $\mathcal{P}, \mathcal{Q}$ of points in $\mathbb{R}^d$, the objective is to find a pair of points (p, q), p in the convex hull of $\mathcal{P}$, q in the convex hull of $\mathcal{Q}$, such that the length of

the difference vector $v = p - q$ is minimized. Formally, if

$$\begin{aligned} \mathcal{P} &:= \{P_1, \ldots, P_r\}, \\ \mathcal{Q} &:= \{Q_1, \ldots, Q_s\}, \end{aligned}$$

we let P (resp. Q) denote the $(d \times r)$-matrix (resp. $(d \times s)$-matrix) containing as columns the points of $\mathcal{P}$ (resp. $\mathcal{Q}$). PD is the following problem in the variables $x = x_1, \ldots, x_r$, $y = y_1, \ldots, y_s$.

$$\begin{array}{lll} \text{(PD)} & \text{minimize} & v^T v \\ & \text{subject to} & v = Px - Qy, \\ & & \sum_{i=1}^{r} x_i = 1, \\ & & \sum_{j=1}^{s} y_j = 1, \\ & & x, y \geq 0. \end{array} \tag{4.1}$$

This is a quadratic programming problem (this means quadratic objective function and linear constraints) whose optimum value is the square of the euclidean distance between $\text{conv}(\mathcal{P})$ and $\text{conv}(\mathcal{Q})$.

4.2 A Class of Convex Programs

The general pattern behind PD is the following: minimize a real-valued convex function $f(\rho), \rho \in \mathbb{R}^d$, with its argument ρ ranging over the nonnegative span of some $(d \times n)$-matrix M , where the coefficients satisfy additional q equality constraints $Ax = b$. In case of PD, we have $\rho = v$, $f(v) = v^T v$, $M = (P| - Q)$ (thus $n = r + s$) and $q = 2$ with

$$A = \begin{pmatrix} 1 \cdots 1 & 0 \cdots 0 \\ \underbrace{0 \cdots 0}_{r} & \underbrace{1 \cdots 1}_{s} \end{pmatrix},$$

$$b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

In general, our convex programming problems look as follows.

$$\begin{array}{lll} \text{(CP)} & \text{minimize} & f(\rho) \\ & \text{subject to} & \rho = Mx, \\ & & Ax = b, \\ & & x \geq 0, \end{array} \tag{4.2}$$

where ρ is a d-vector, M a $(d \times n)$-matrix, A a $(q \times n)$-matrix, b a q-vector and x an n-vector. As in LP, inequality constraints $Ax \leq b$ can be handled by introducing slack variables. We restrict ourselves to equality constraints here. No condition is imposed on M. (For PD, this in particular means that we do not require the point sets $\mathcal{P}$ and $\mathcal{Q}$ to be in any kind of general position.) There will be a condition on A, however, see below.

We demand f to be a differentiable function with continuous partial derivatives, which is convex, i.e. for ρ, ρ' and $0 \leq t \leq 1$,

$$f((1-t)\rho + t\rho') \leq (1-t)f(\rho) + tf(\rho') \tag{4.3}$$

holds. In addition, conditions on *subproblems* of CP are assumed to hold, and they concern nondegeneracy and uniqueness of solution. For any $G \subseteq H := [n]$, consider the problems CP(G) and UCP(G), defined as follows.

$$\begin{array}{lll} (\mathrm{CP}(G)) & \text{minimize} & f(\rho) \\ & \text{subject to} & \rho = M_G x_G, \\ & & A_G x_G = b, \\ & & x_G \geq 0. \end{array} \tag{4.4}$$

$$\begin{array}{lll} (\mathrm{UCP}(G)) & \text{minimize} & f(\rho) \\ & \text{subject to} & \rho = M_G x_G, \\ & & A_G x_G = b. \end{array} \tag{4.5}$$

The subproblem CP(G) minimizes $f(\rho)$ over all feasible x which satisfy the additional restrictions $x_{H-G} = 0$. This is the same kind of subproblem as we considered for LP back in Chapter 2. In case of PD, this corresponds to a distance computation between subpolytopes of $\mathcal{P}$ and $\mathcal{Q}$ specified by G.

Where CP(G) asks for the minimum of f over all *nonnegative* linear combinations with coefficients in some affine subspace, the relaxation UPD(G) considers any linear combination, and this problem is obtained by simply dropping the nonnegativity constraints from CP(G). The 'U' stands for 'unconstrained', although UCP is not an unconstrained problem in the strict sense. The equality constraints of CP are still present, but later we will see that they are no 'real' constraints and that UCP is substantially easier to solve than CP. In case of PD, UCP(G) amounts to a distance computation between the *affine* hulls of subpolytopes.

Here are the conditions on $CP(G)$ that we need to be satisfied.

(i) Any CP(G) has a *unique* optimum solution ρ (or is infeasible).

(ii) If CP(G) is feasible, then the rows of A_G are linearly independent.

Condition (ii) is a nondegeneracy assumption, essentially equivalent to the one discussed for LP back in Chapter 1, only that here the condition is more explicitly linked to the constraint matrix. In case of PD, the rows of A_G are linearly independent, unless all columns of A_G are chosen from the first r (resp. the last s) ones of A. Then, however, the problem is infeasible.

Finally, there is one more condition on subproblems UCP(G). G is called *basic*[1] if for any feasible solution ρ to UCP(G), a *unique* vector x_G exists that determines ρ. We demand that

[1] not to be confused with the concept of a *basis* in LP-type problems

if G is basic, then UCP(G) has a unique optimum solution ρ.

A sufficient condition for this to hold is that f is *strictly* convex, i.e. strict inequality holds in (4.3) for $\rho \neq \rho$ and $0 < t < 1$, and that the 'largest' problem UCP(H) already has an optimum. Again, in case of PD, this is the case. In general, the optimum assumption for UCP is necessary because one step in our solution process consists of solving problems UCP(G) when G is basic. This assumption is convenient and leads to a uniform method – later we will see that we can get around it in some cases. Note that LP is out of the game by now: linear programs subject only to equality constraints are typically unbounded, and even if they are not, no unique solution exists.

4.2.1 Basics about Convex Functions

We require f to be differentiable with continuous partial derivatives in order to apply calculus. For example, the following very useful fact holds then, see [41, p. 52].

Fact 4.1 *For any two points ρ, ρ',*

$$f(\rho) + \nabla f(\rho)(\rho' - \rho) \leq f(\rho'), \tag{4.6}$$

where ∇ is the gradient operator.

Recall that the value $f(\rho)+\nabla f(\rho)(\rho'-\rho)$ is the first order estimate of $f(\rho')$ obtained from approximating the graph of f by the tangent hyperplane at ρ. The formula above states that the graph of f lies above all its tangent hyperplanes – just what you expect in the convex case. This fact can be used to prove an optimality criterion for f over any convex set.

Lemma 4.2 *Let $C \subseteq \mathbb{R}^d$ be convex. $\rho \in C$ is a (global) minimum of f over C if and only if*

$$\nabla(\rho)(\rho' - \rho) \geq 0$$

for all $\rho' \in C$.

Proof. Assume ρ is a minimum and let $\rho' \in C$. Consider the convex combination

$$\rho(t) := \rho + t(\rho' - \rho),\ t \in [0, 1].$$

$\rho(t)$ is again in C. Then

$$\frac{\partial}{\partial t} f(\rho(t))|_{t=0} \geq 0$$

must hold, otherwise we had

$$\frac{\partial}{\partial t} f(\rho(t))|_{t=0} = \lim_{t \to 0} \frac{f(\rho(t)) - f(\rho)}{t} < 0,$$

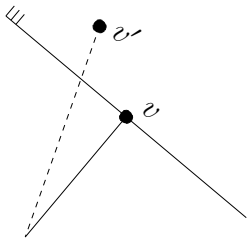


Figure 4.1: The optimal vector for PD

and $f(\rho(t)) < f(\rho)$ would hold for some small t. On the other hand,

$$\frac{\partial}{\partial t} f(\rho(t))|_{t=0} = \nabla f(\rho)(\rho' - \rho),$$

by the chain rule.

If ρ is not a minimum, let ρ' be any better solution. Then, by Fact 4.6,

$$\nabla f(\rho)(\rho' - \rho) \leq f(\rho') - f(\rho) < 0.$$

□

If in this proof ρ' itself happens to be a unique minimum of f over C, then the latter argument also shows that for $0 \leq t_0 < 1$,

$$\frac{\partial}{\partial t} f(\rho(t))|_{t=t_0} = \nabla f(\rho(t_0))(\rho' - \rho) = \frac{1}{t_0} \nabla f(\rho(t_0))(\rho(t_0) - \rho) < 0,$$

since $\rho(t_0)$ is not optimal. This gives

Corollary 4.3 *Let $C \subseteq \mathbb{R}^d$ be convex and assume that ρ' is the unique minimum of f over C, $\rho \in C, \rho \neq \rho'$. With*

$$\rho(t) := \rho + t(\rho' - \rho),$$

$f(\rho(t))$ is strictly monotone decreasing in the interval $[0, 1]$.

What does the criterion of Lemma 4.2 mean for PD? We have $f(v) = v^T v$, i.e. $\nabla f(v) = 2v^T$, so v is optimal if and only if

$$v^T(v' - v) \geq 0, \tag{4.7}$$

for all feasible solutions v'. The set of vectors v' satisfying (4.7) is a halfspace ('pointing away' from the origin), whose supporting hyperplane contains v and is perpendicular to it, Figure 4.1. In other words, v is optimal if and only if it spans an empty parallel slab between the polytopes $\mathcal{P}$ and $\mathcal{Q}$, Figure 4.2 (a). Note that although v is unique, the points of $\mathcal{P}$ and $\mathcal{Q}$ that determine v need not be unique, Figure 4.2 (b) gives an example.

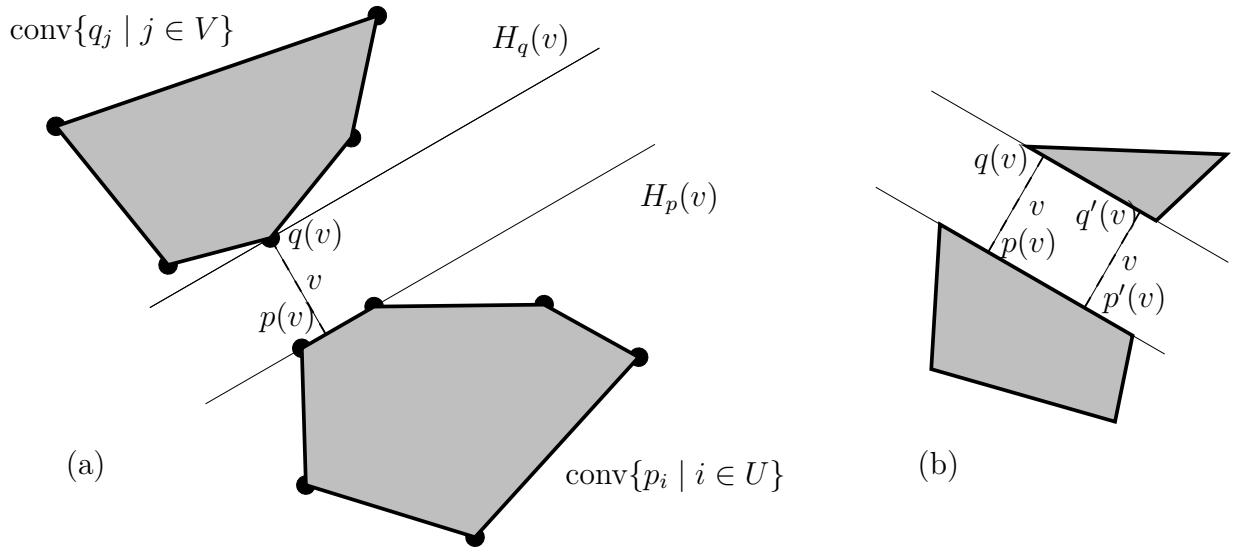


Figure 4.2: The optimality criterion for PD

4.2.2 Convex Programming as LP-type Problem

While LP (as we have described it) gives rise to an LP-type system in the form of Definition 3.1, CP presents itself most naturally as an LP-type problem in form of Definition 3.2, i.e. via z-values for subproblems CP(G) as defined in (4.4).

Let $z(G)$ be the optimum value of CP(G) and define $\rho(G)$ to be the unique optimal solution of CP(G), i.e. $\rho(G)$ is the unique vector satisfying

$$f(\rho(G)) = z(G).$$

Unless CP(G) is infeasible (in which case we set $z(G) := \infty$), $z(G)$ is some finite value.

A major step in proving that a problem is LP-type often consists of a good characterization of optimality. For our convex programming problems, one characterization is given by Lemma 4.2. We supplement it with another, more powerful tool, namely the *Karush-Kuhn-Tucker condition* for mixed constraints (equality and inequality constraints), and this beautiful result does not even rely on convexity.

Proposition 4.4 (Karush-Kuhn-Tucker Condition) *Consider the problem*

$$\begin{array}{lll} \text{(OPT)} & \text{minimize} & f(x) \\ & \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots . g \\ & & h_j(x) = 0, \quad j = 1, \ldots, h, \end{array}$$

where f, g_i, h_j have continuous partial derivatives. Let $\tilde{x}$ be a feasible point of OPT such that the set of vectors

$$\{\nabla g_i(\tilde{x}) \mid g_i(\tilde{x}) = 0, i = 1, \ldots, g\} \cup \{\nabla h_j(\tilde{x}) \mid j = 1, \ldots, h\}$$

is linearly independent. (Such a point is called regular.*) If $\tilde{x}$ is a local minimum of OPT, then there exists a g-vector μ and a h-vector λ such that*

(i) $\mu \geq 0$,

(ii) $\mu_i g_i(\tilde{x}) = 0, i = 1, \ldots, g$, *and*

(iii) $\nabla f(\tilde{x}) + \sum_{i=1}^{g} \mu_i \nabla g_i(\tilde{x}) + \sum_{j=1}^{h} \lambda_j \nabla h_j(\tilde{x}) = 0$.

The following is a specialization of this very general theorem to our setting (the general formulation and a proof can be found in [41, chapter 7]). If only equality constraints are present (like in the unconstrained version UCP), the first two conditions are empty, and the Karush-Kuhn-Tucker condition boils down to the classical theory of *Lagrange multipliers* (we get to that later).

Theorem 4.5 *$\rho = M\tilde{x}_G$ minimizes f over CP(G) if and only if there exists a q-vector λ and a $|G|$-vector μ_G and such that*

(i) $\mu_G \geq 0$,

(ii) $\mu_i \tilde{x}_i = 0, \ \ i \in G$ *and*

(iii) $\nabla f(\rho) M_G = \mu_G^T - \lambda^T A_G$.

Proof. The 'only if' part is a direct application of Proposition 4.4, which in general gives only a necessary condition for the existence of a minimum. Due to the special nature of our problem, this condition is also sufficient here. To see this, let $\rho = M_G \tilde{x}_G$ be any feasible point satisfying (i), (ii) and (iii) and consider any other feasible solution $\rho' = M_G \tilde{x}'_G$. Then

$$\nabla f(\rho)(\rho' - \rho) = \nabla f(\rho) M_G(\tilde{x}'_G - \tilde{x}_G) = \mu_G^T(\tilde{x}'_G - \tilde{x}_G) - \lambda^T A_G(\tilde{x}'_G - \tilde{x}_G) = \mu_G^T \tilde{x}'_G \geq 0,$$

because $\mu_G, \tilde{x}'_G \geq 0$, so ρ is a minimum by Lemma 4.2.

All this requires the (gradients of the) constraints that are *active* at ρ (i.e. that hold with equality at ρ) to be linearly independent, so that $\rho = M_G \tilde{x}_G$ is a regular point. This is achieved by our nondegeneracy assumption. Assume $\{i \mid \tilde{x}_i = 0\} = F \subseteq G$. Then $\tilde{x}$ is actually feasible for CP$(G - F)$, therefore the rows of A_{G-F} are linearly independent by nondegeneracy. This implies that the rows of A_G, together with the unit (row) vectors $\mathbf{e}_i, i \in F$, are linearly independent, and these are exactly the gradients under consideration. □

Now we can prove the main lemma that allows us to fit CP into the LP-type framework.

Lemma 4.6 *Let $\rho := \rho(G) = M_G \tilde{x}_G$, μ_G, λ according to Theorem 4.5. Then*

$$z(G \cup \{j\}) > z(G) \text{ if and only if } \lambda^T A_j + \nabla f(\rho) M_j < 0.$$

Proof. Let $\rho' = M_{G\cup\{j\}}\tilde{x}'_{G\cup\{j\}}$ be any feasible solution to $\mathrm{CP}(G\cup\{j\})$. Then we can write

$$\begin{aligned}
\nabla f(\rho)(\rho'-\rho) &= \nabla f(\rho)(M_{G\cup\{j\}}\tilde{x}'_{G\cup\{j\}} - M_G\tilde{x}_G)\\
&= \nabla f(\rho)M_G(\tilde{x}'_G - \tilde{x}_G) + \nabla f(\rho)M_j\tilde{x}'_j\\
&= (\mu_G^T - \lambda^T A_G)(\tilde{x}'_G - \tilde{x}_G) + \nabla f(\rho)M_j\tilde{x}'_j\\
&= (\mu_G^T - \lambda^T A_G)\tilde{x}'_G + \lambda^T b + \nabla f(\rho)M_j\tilde{x}'_j\\
&= \mu_G^T\tilde{x}'_G + \lambda^T A_j\tilde{x}'_j + \nabla f(\rho)M_j\tilde{x}'_j. \qquad (4.8)
\end{aligned}$$

If $z(G\cup\{j\}) > z(G)$, then ρ is not optimal for $\mathrm{CP}(G\cup\{j\})$, so by Lemma 4.2 we can choose ρ' (with $\tilde{x}'_j > 0$) such that

$$0 > \nabla f(\rho)(\rho'-\rho) = \mu_G^T\tilde{x}'_G + \lambda^T A_j\tilde{x}'_j + \nabla f(\rho)M_j\tilde{x}'_j \geq \lambda^T A_j\tilde{x}'_j + \nabla f(\rho)M_j\tilde{x}'_j,$$

since $\mu_G, \tilde{x}'_G \geq 0$. This in turn means $0 > \lambda^T A_j + \nabla f(\rho)M_j$.

Now assume that $z(G\cup\{j\}) = z(G)$ and let $B \subseteq G$ be inclusion minimal with $z(B) = z(G)$. This implies that ρ has a representation

$$\rho = M_G\tilde{x}_G = M_B\tilde{x}_B + M_{G-B}\tilde{x}_{G-B}, \text{ with } \tilde{x}_B > 0, \tilde{x}_{G-B} = 0,$$

and if we enter the lemma with this representation, we get $\mu_B = 0$. Let $\rho' = M_{G\cup\{j\}}\tilde{x}'_{G\cup\{j\}} = M_{B\cup\{j\}}\tilde{x}'_{B\cup\{j\}}$ be any feasible solution of $\mathrm{CP}(B\cup\{j\})$ with $\tilde{x}'_j > 0$. Such a ρ' exists by our nondegeneracy assumption: $\{A_B x_B = b\}$ is a proper subspace of $\{A_{B\cup\{j\}}x_{B\cup\{j\}} = b\}$, so $\tilde{x}'_{B\cup\{j\}}$ can be obtained by slightly perturbing $\tilde{x}_B$, keeping the values at coordinates with subscripts in B positive.

ρ' is in particular feasible for $\mathrm{CP}(G\cup\{j\})$, and since ρ was assumed to be optimal for $\mathrm{CP}(G\cup\{j\})$, Lemma 4.2 gives us

$$\begin{aligned}
0 \leq \nabla f(\rho)(\rho'-\rho) &\overset{(4.8)}{=} \mu_G^T\tilde{x}'_G + \lambda^T A_j\tilde{x}'_j + \nabla f(\rho)M_j\tilde{x}'_j\\
&= \mu_B^T\tilde{x}'_B + \lambda^T A_j\tilde{x}'_j + \nabla f(\rho)M_j\tilde{x}'_j\\
&= \lambda^T A_j\tilde{x}'_j + \nabla f(\rho)M_j\tilde{x}'_j,
\end{aligned}$$

because $\mu_B = 0$. Now $\tilde{x}'_j > 0$ implies $\lambda^T A_j + \nabla f(\rho)M_j \geq 0$. ◻

Theorem 4.7 *$(H, -z)$ is an LP-type problem, according to Definition 3.2.*

Proof. Since for $F \subseteq G$, any feasible solution to $\mathrm{CP}(F)$ is feasible for $\mathrm{CP}(G)$, we get $z(F) \geq z(G)$, and this is *monotonicity*. (Note that the inequalities are not in the usual direction, since we are actually talking about $-z$.)

To see *locality*, fix $F \subseteq G \subseteq H$ with $\infty \neq z(F) = z(G)$. Then $\rho(G) = \rho(F) = M_G\tilde{x}_G = M_F\tilde{x}_F$. Consider λ and μ_G as guaranteed by Theorem 4.5 for $\mathrm{CP}(G)$. Then λ and μ_F are good for $\mathrm{CP}(F)$, and via Lemma 4.6, $z(G\cup\{j\}) < z(G)$ is equivalent to $z(F\cup\{j\}) < z(F)$. ◻

To apply Algorithm RF-LPtype to CP, two more ingredients are necessary: improvement query and basis improvement. Although we have not explicitly characterized the bases of $(H, -z)$, these are the inclusion minimal sets B that define a certain finite z-value (see Lemma 3.3). Note that we have already made use of a basis in the proof of Lemma 4.6.

4.2.3 Improvement Query

Given basis B and j, we know that some vector λ exists with $z(B \cup \{j\}) < z(B)$ if and only if $\lambda^T A_j + \nabla f(\rho) M_j < 0$, and this criterion seems to be the most natural choice for the improvement query, provided such a λ is given to us. This is the case if we simply store a suitable λ with every basis B, and ensure that the basis improvement does not only deliver a new basis B' but a corresponding vector λ' as well. In fact, this will happen automatically, because we are going to apply Theorem 4.5 during basis computation as a tool for actually finding minima.

4.2.4 Basis Improvement

This procedure repeatedly solves relaxations UCP of CP as introduced in (4.5).

As for CP(G) above, we define $z_u(G)$ as the optimum value of UCP(G) and let $\rho_u(G)$ be the unique optimal solution to UCP(G), i.e. $\rho_u(G)$ is the unique feasible point such that

$$f(\rho_u(G)) = z_u(G).$$

Again, $z_u(G)$ is some finite value, unless UCP(G) is infeasible, and obviously,

$$z_u(G) \leq z(G).$$

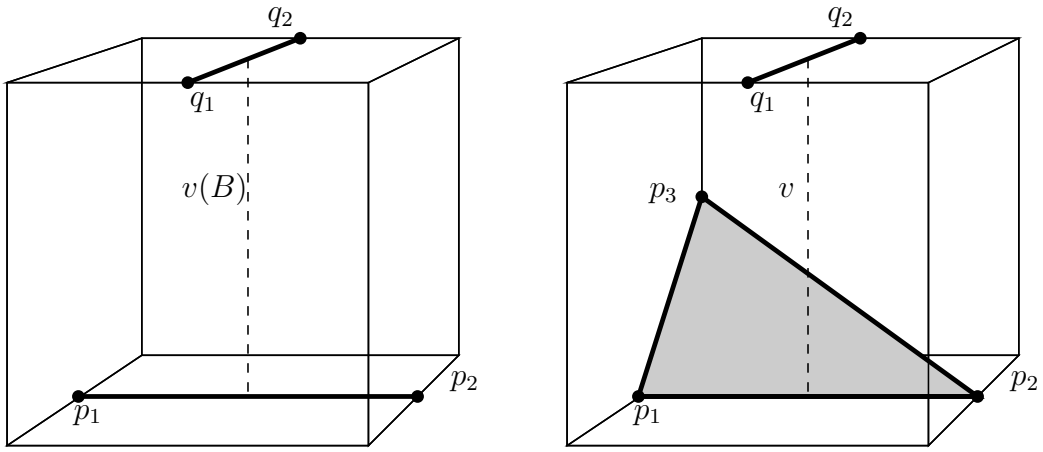
Note that all this presumes that G is basic (i.e. the representation of $\rho_u(G)$ in the variables x_i is unique), otherwise UCP(G) does not need to have an optimum. Below we prove that this holds during basis improvement.

We shall first describe the improvement in informal terms, where we keep track of its geometric interpretation in case of PD to get across the main idea. A rigorous treatment of the remaining details follows.

Geometry

We start with some basis B and its optimal point $\rho(B)$ (which is an optimal difference vector $v(B)$ in case of PD, Figure 4.3, left) that we store together with B . Given some j such that $z(B \cup \{j\}) < z(B)$, we enter a loop in which we maintain a set B' and a point ρ. Initially, $B' := B \cup \{j\}$, $\rho := \rho(B)$, Figure 4.3, right.

During any execution of the loop body, $f(\rho)$ is improved and B' gets smaller (in size), where the invariant is maintained that ρ is feasible for CP(B'). The loop terminates if



$B = (\{1,2\},\{1,2\})$ $B' = (\{1,2,3\},\{1,2\})$

Figure 4.3: Starting off: $v := v(B), B' := B \cup \{j\}$

$f(\rho)$ cannot be improved anymore, i.e. if $\rho = \rho(B')$. In this case, B' will be delivered as the new basis.

To improve on $f(\rho)$, first compute (by some direct method to be described) the optimal solution $\rho' := \rho_u(B')$ of the relaxation UCP(B'), Figure 4.4 left, and start moving ρ towards ρ', by linearly interpolating between the respective values of $x_{B'}$ at ρ and ρ'. Feasibility requires the movement to stop as soon as one (or more) variables $\tilde{x}_i$ determining the current ρ become zero, Figure 4.4, right. The corresponding indices are dropped from B' and if ρ' has not been reached yet, the next iteration is entered with smaller B' and improved $f(\rho)$, Figure 4.5.

Since B' cannot get empty (for $|B'| = 1$, either CP(B') is infeasible or CP(B') and UCP(B') coincide), ρ' is eventually reached by ρ in some iteration (Figure 4.6, left). This means that $\rho' = \rho_u(B')$ is feasible for CP(B'), so $\rho = \rho' = \rho(B')$ holds and B' is returned as the new basis (Figure 4.6, right).

Here is the algorithm, written out formally. In every loop execution we assume that ρ and ρ' are represented as

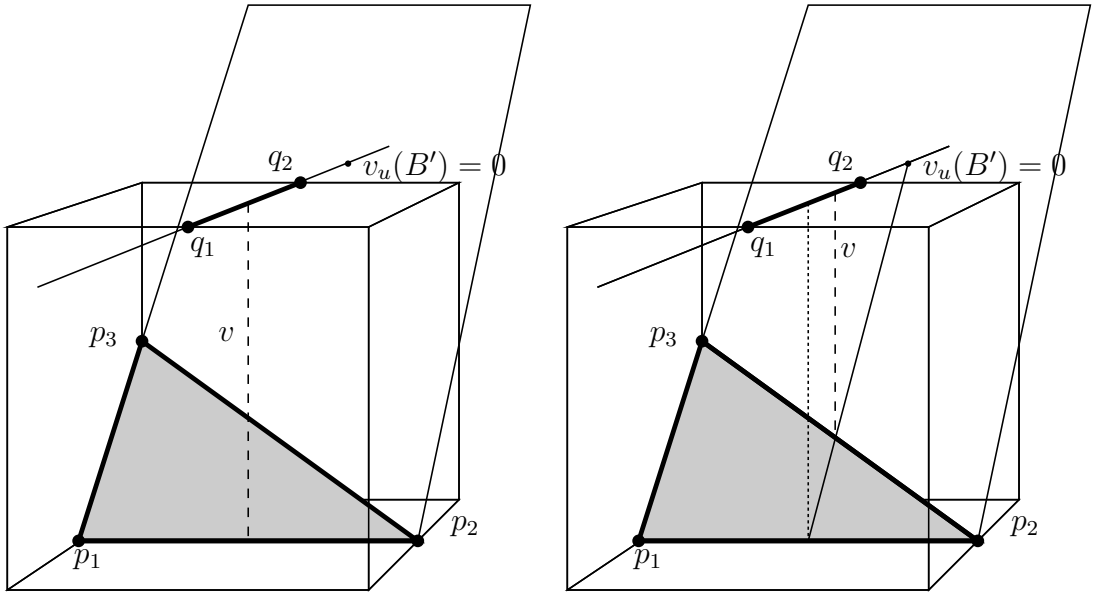
$$\begin{aligned} \rho &= M_{B'}\tilde{x}_{B'}, \\ \rho' &= M_{B'}\tilde{x}'_{B'} \end{aligned}$$

and that $\tilde{x}(t)$ is the vector

$$\tilde{x}(t) := \tilde{x}_{B'} + t(\tilde{x}'_{B'} - \tilde{x}_{B'}).$$

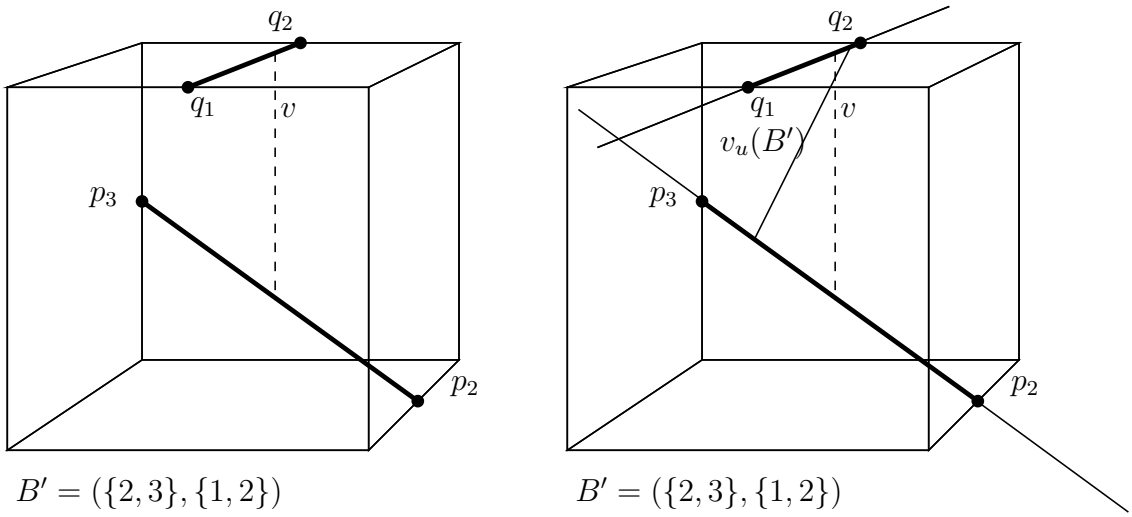
Algorithm 4.8

BASIS (B, j) (* B basis, $z(B \cup \{j\}) < z(B)$ *)
1 $\rho := \rho(B)$
2 $B' := B \cup \{j\}$



$B' = (\{1,2,3\},\{1,2\})$ $B' = (\{1,2,3\},\{1,2\})$

Figure 4.4: First iteration: $v' := v_u(B')$, v is moved towards v'



$B' = (\{2,3\},\{1,2\})$ $B' = (\{2,3\},\{1,2\})$

Figure 4.5: v has not reached v'; update B', enter next iteration with $v' = v_u(B')$

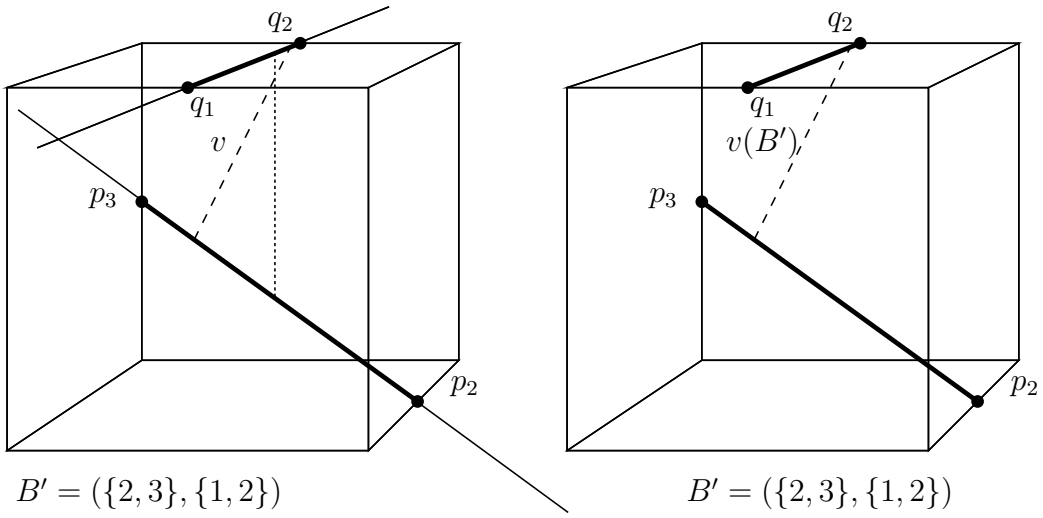


Figure 4.6: $v = v_u(B') = v(B'), B'$ is the new basis

3 **repeat** (* invariant: ρ is feasible for CP(B') *)
4 $\quad \rho' := \rho_u(B')$
5 $\quad t_0 := \max\{0 \le t \le 1 \mid \tilde{x}(t) \ge 0\}$
6 $\quad \rho := \rho + t_0(\rho' - \rho)$
7 $\quad B' := \{i \mid \tilde{x}_i(t_0) > 0\}$
8 **until** $t_0 = 1$
9 **return** B'

The loop invariant is guaranteed by construction, and if $t_0 < 1$ in line 8, then B' has become smaller in line 7, so the loop eventually terminates. At this time,

- $\rho = \rho_u(B')$ (because of $t_0 = 1$), and
- ρ is feasible for CP(B') (because of the invariant).

Thus, the final B' is a set with $\rho(B') = \rho_u(B')$, or $z(B') = z_u(B')$. The questions we still need to answer are

(i) Did we ever decrease $f(\rho)$?

(ii) Is B' a basis?

(iii) What is the 'direct method' for computing $\rho_u(B')$ in line 4 ?

The following presents the necessary (linear) algebra to answer these questions. The first part leads to a correctness proof of Algorithm 4.8, the second part is devoted to the third question above.

Correctness

This is the key lemma, and here we really use the fact that the set B we started with is a basis. We show that the set B' maintained during the improvement loop is basic, and only this justifies the whole approach.

Lemma 4.9 *Let B be a basis, $B' := B \cup \{j\}$ with $z(B') > z(B)$, and let ρ be any point. Then the system of equations*

$$\rho = M_{B'} x_{B'}, \tag{4.9}$$
$$b = A_{B'} x_{B'} \tag{4.10}$$

has either a unique solution $\tilde{x}_{B'}$ or no solution at all.

Proof. We equivalently prove that there are no numbers $\tilde{x}_{B'}$, not all zero, with

$$0 = M_{B'} \tilde{x}_{B'}, \tag{4.11}$$
$$0 = A_{B'} \tilde{x}_{B'}. \tag{4.12}$$

This is done in two stages. First, we prove that there are no such numbers with $\tilde{x}_j = 0$ (thus showing that B is basic). Second we prove that $\tilde{x}_j = 0$ must hold. (so B' is basic as well).

First stage. Consider

$$\rho(B) = M_{B'} \tilde{x}'_{B'},$$

$\tilde{x}'_{B'} \geq 0, \tilde{x}'_j = 0$, and assume there exist nontrivial numbers $\tilde{x}_{B'}$ satisfying (4.11) and (4.12), with $\tilde{x}_j = 0$. Suitably scaled, these numbers can be added to $\tilde{x}'_{B'}$ to yield still nonnegative numbers $\tilde{x}''_{B'}$ with some number $\tilde{x}''_i$ attaining zero value. Because of (4.11) and (4.12), these numbers are a feasible representation of $\rho(B)$ again, proving that $\rho(B)$ is feasible (and therefore optimal) for CP$(B - \{i\})$. But then $z(B) = z(B - \{i\})$ holds, a contradiction to B being a basis.

Since this already shows that B is basic, $\rho_u(B)$ must exist. Moreover, it follows that the $\tilde{x}'_{B'}$ defining $\rho(B)$ are strictly positive (except that $\tilde{x}'_j = 0$). We claim that this implies $\rho(B) = \rho_u(B)$, hence

$$z(B) = z_u(B). \tag{4.13}$$

To see this, consider

$$\rho(t) := \rho(B) + t(\rho_u(B) - \rho(B)).$$

If we had $\rho(B) \neq \rho_u(B)$, then for some small t, $\rho(t)$ was still feasible for CP(B) by strict positiveness, and Corollary 4.3 would yield $f(\rho(t)) < f(\rho(B))$, a contradiction to $\rho(B)$ being optimal.

Second stage. We proceed as above, but this time consider

$$\rho(B') = M_{B'} \tilde{x}'_{B'}.$$

Assume there exist numbers $\tilde{x}_{B'}$ satisfying (4.11) and (4.12), with $\tilde{x}_j \neq 0$. Suitably scaled, these numbers can be added to $\tilde{x}'_{B'}$ to yield numbers $\tilde{x}''_{B'}$ with $\tilde{x}''_j = 0$. This proves that $\rho(B')$ is feasible for $\mathrm{UCP}(B' - \{j\}) = \mathrm{UCP}(B)$, which implies $z_u(B) \leq z(B')$. Together with the condition of the lemma and (4.13) this gives

$$z_u(B) \leq z(B') < z(B) = z_u(B),$$

a contradiction. □

Now we are ready to prove correctness of the basis improvement.

Theorem 4.10 *If B is a basis and $z(B \cup \{j\}) < z(B)$, algorithm 4.8* Basis(B, j) *computes a basis $B' \subseteq B \cup \{j\}$ with $z(B') < z(B)$.*

Proof. We first show that the set B' returned in line 9 satisfies $z(B') < z(B)$. To this end we prove that $f(\rho)$ decreases during the first execution of the loop, when $B' = B \cup \{j\}$, so $\rho = \rho(B)$, $\rho' = \rho_u(B \cup \{j\})$. Because of $z_u(B \cup \{j\}) \leq z(B \cup \{j\}) < z(B)$ we know that $\rho \neq \rho'$ holds, so by Corollary 4.3 $f(\rho)$ decreases in line 6, provided that $t_0 > 0$. The unique $\tilde{x}_{B'}$ defining ρ are strictly positive as we have seen in the proof of Lemma 4.9, except that $\tilde{x}_j = 0$. Thus, $t' > 0$ if and only if $\tilde{x}'_j > 0$, where $\tilde{x}'_{B'}$ are the unique numbers determining ρ'. Assume $\tilde{x}'_j \leq 0$ would hold. Then let $\tilde{x}''_{B'}$ be the numbers defining $\rho(B')$, where $\rho(B) \neq \rho(B')$ implies $\tilde{x}''_j > 0$. It follows that some number $t_0 \leq 1$ must exist such that in the representation of

$$\rho(B') + t_0(\rho' - \rho(B')),$$

variable x_j has zero value. Then $\sigma := \rho(B') + t_0(\rho' - \rho(B'))$ is feasible for $\mathrm{UCP}(B)$, with

$$f(\sigma) < f(\rho(B')) = z(B') < z(B) = z_u(B),$$

a contradiction. The first inequality holds by Corollary 4.3, the last equality is (4.13).

Next, we show that the final B' is a basis. Assume this is not the case and $z(B') = z(B'')$ holds for some proper subset B''. Then let $\tilde{x}''_{B'}$ be the unique numbers defining $\rho(B'') = \rho(B') = \rho_u(B')$ after the final iteration. The variables with subscripts corresponding to $B' - B''$ have zero value which is a contradiction because such subscripts have been removed from B' in line 7 of the algorithm. □

Solving UCP

Finally, we have arrived at the last open question associated with the basis improvement of CP, namely how the arising subroblems $\mathrm{UCP}(B')$, B' basic, are solved. Note that for the unconstrained problem

$$\begin{array}{lll} (\mathrm{UCP}(B')) & \text{minimize} & f(\rho) \\ & \text{subject to} & \rho = M_{B'} x_{B'}, \\ & & A_{B'} x_{B'} = b, \end{array}$$

the Karush-Kuhn-Tucker condition (Proposition 4.4) shows that if $\rho := \rho(B') = M_{B'}\tilde{x}_{B'}$, then there exists a q-vector λ such that $\tilde{x}_{B'}, \lambda$ solve the equation

$$\nabla f(M_{B'}x_{B'})M_{B'} = \lambda^T A_{B'}. \tag{4.14}$$

The λ_i are called *Lagrange multipliers*. As in case of CP, this condition is already sufficient for ρ being a minimum. Let $\tilde{x}_{B'}, \lambda$ be any solution of (4.14), $\tilde{x}_{B'}$ feasible for UCP(B'), and define $\rho := M_{B'}\tilde{x}_{B'}$. Let $\rho' := M_{B'}\tilde{x}'_{B'}$ be any other feasible solution. Multiplying (4.14) from the right with $\tilde{x}'_{B'} - \tilde{x}_{B'}$ gives

$$\nabla f(\rho)(\rho' - \rho) = 0,$$

so ρ is optimal by Corollary 4.2. Since we have assumed UCP(B') to have a unique optimum whenever B' is basis, this shows that (4.14) must have a unique solution $\tilde{x}_{B'}$ with $\tilde{x}_{B'}$ feasible. Moreover, by nondegeneracy the rows of $A_{B'}$ are linearly independent, and this implies that (4.14) determines a unique vector λ as well.

When solved the last time during basis improvement, (4.14) provides us with the multipliers λ for the new basis B', which enables us to perform subsequent improvement queries on B'. Note that for B' a basis, CP(B') and UCP(B') coincide, because $z(B') = z_u(B')$ as we have shown before. Therefore, the multiplier λ we get from solving UCP(B') as described here serves as a legal multiplier for CP(B') as well, with $\mu_{B'} = 0$ in Theorem 4.5.

4.3 Combinatorial Dimension and Time Bounds

To have the basic parameters at hand, recall that we are solving a CP problem on $|H| = n$ points, in dimension d, with q equality constraints.

If we store multipliers λ with every basis, an *improvement query* (Subsection 4.2.3) can be performed in time $O(d + q)$ by evaluating two inner products, one involving two q-vectors, and one involving two d-vectors. This presumes that we have a primitive at hand to evaluate the gradient of f at a particular point in constant time per entry. For many reasonable functions, this is the case, otherwise we can account for the necessary effort by an extra term for the following operation.

Gradient primitive. Evaluate $\nabla f(\rho)$, for any ρ. (4.15)

The time for *basis improvement* depends on the number of times the loop of Algorithm 4.8 is executed and on the time required to solve the UPD problem in line 4. Let us first examine the number of loop executions. We have already argued that $|B'|$ gets smaller in every iteration, is always positive and initially no larger than $|B| + 1$, B the basis we have started with. Thus, an upper bound on the number of loop executions is δ, where δ is the maximum cardinality of any basis, equivalently the combinatorial dimension of CP as an LP-type problem.

Lemma 4.11 *CP has combinatorial dimension $\delta \leq d + q$.*

Proof. The proof of Lemma 4.9 in particular shows that for B a basis, unique numbers $\tilde{x}_B$ exist that solve the system

$$\rho(B) = M_B x_B,\ A_B x_B = b$$

of $d+q$ linear equations in $|B|$ unknowns, and this can have a unique solution only for $|B| \leq d+q$. □

Let us denote by t_L the time necessary to solve a single UCP problem during basis improvement, equivalently the time necessary to perform the following computational primitive.

$$\textit{Lagrangian primitive.}\ \text{Solve} \left\{ \begin{array}{l} \nabla f(M_{B'} x_{B'}) M_{B'} = \lambda^T A_{B'}, \\ A_{B'} \tilde{x}_{B'} = b \end{array} \right\} \text{for } x_{B'}, \lambda. \tag{4.16}$$

Then we have the following result, derived from the general Result 3.11.

Result 4.12 *Let CP be a convex programming problem (4.2) over $\mathbb{R}^d$ in n variables, with q equality constraints, $n > d+q$. Using Algorithm 3.4* RF-LPtype*, CP can be solved with an expected number of no more than*

$$O((d+q)t_L)2^{d+q+2}(n-d-q)$$

arithmetic operations, where t_L is the time necessary to perform the Lagrangian primitive (4.16). (For $n < d+q$, a bound of

$$O((d+q)t_L)2^{n+2}$$

holds.)

Before we (re)turn to specific problems, let us conclude our treatment of the general case with two remarks. First, the Lagrangian primitive is easy whenever the objective function is quadratic. In this case, the gradient is linear and (4.16) boils down to a (uniquely solvable) system of linear equations. Thus, we get explicit time bounds for quadratic programming problems. The polytope distance problem falls into this class and therefore allows for a 'real' bound, see below. However, there is one restriction, and this brings us to the second remark. Our approach fails when the unconstrained problem UCP(B') we attempt to solve during basis improvement turns out to be unbounded, thus we needed to rule out this case explicitly in the beginning. Consequently, LP – and probably other interesting problems – cannot be handled. As we will see below, we cannot even handle the minimum spanning ball problem then! Fortunately, there is a cure in this (and probably other) cases. Carefully reconsidering the improvement algorithm, one will find that the sole purpose of the optimal solution $\rho_u(B')$ to UCP(B') was to determine a *feasible direction* along which the objective function improves until either the optimum is reached or the boundary of the feasible region is hit.

$\rho_u(B')$ itself does not really enter the picture before the very last improvement step where it determines the new feasible optimum $\rho(B')$ (and in *this* step, $\rho_u(B')$ surely exists in any case because $\rho(B')$ exists). In the previous steps, we could do without $\rho_u(B')$; all that is actually needed is the following routine.

Given a feasible solution to CP(B'), find an improved feasible solution to CP($B' - \{i\}$), for some i.

Solving the unconstrained problem provides a convenient and generally applicable way of realizing this routine but is not necessarily the only way. For specific problems, there might be another way that does not rely on solvability of UCP. Keep this in mind for the minimum spanning ball problem.

4.3.1 Polytope Distance Revisited

As we have seen in the beginning of this chapter, PD over $n := r + s$ points in $\mathbb{R}^d$ (4.1) can be formulated as CP, with $q = 2$. Moreover, $f(v) = v^T v$ is a quadratic function, so (4.16) is a system of $d+2$ linear equations in $d+2$ unknowns which can be solved in time $O(d^3)$, such that we obtain

Theorem 4.13 *Let PD be a polytope distance problem (4.1) over $n = r + s$ points in $\mathbb{R}^d$, $n > d + 2$. In the LP-type framework, PD can be solved with an expected number of no more than*

$$O(d^4 2^{d+4}(n - d - 2))$$

arithmetic operations. (For $n \leq d + 2$, a bound of

$$O(d^4 2^{n+2})$$

holds.)

4.3.2 Minimum Spanning Ball

Applying the convex programming machinery that we have developed, the solution to another interesting problem – the *minimum spanning ball* problem – comes (almost) for free now. Given a finite point set $\mathcal{P}$ in $\mathbb{R}^d$, the objective is to find the ball of smallest radius containing all the points. Formally, if

$$\mathcal{P} = \{P_1, \ldots, P_n\},$$

then MB is the following problem in $d + 1$ variables $p = (p_1, \ldots, p_d)^T, r^2$.

$$\begin{array}{lll} \text{(MB)} & \text{minimize} & r^2 \\ & \text{subject to} & (P_i - p)^T (P_i - p) \leq r^2, \quad i = 1, \ldots, n. \end{array} \tag{4.17}$$

In contrast to PD, this is a problem with linear objective function and quadratic constraints. Its optimum value is the square of the radius of the smallest ball enclosing P, and the values of $p_1, \ldots, p_d$ at the optimum determine the center of this ball. In the form of (4.17), the problem obviously does not constitute a convex programming problem (CP) as defined in (4.2). Fortunately, it can be stated another way.

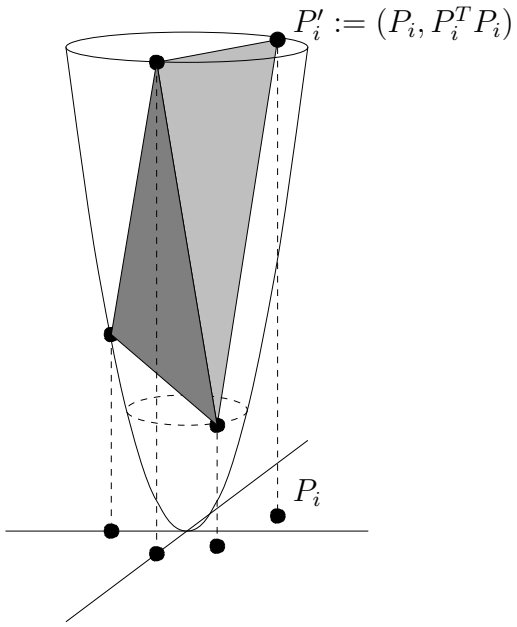


Figure 4.7: The lifting transform

Lemma 4.14 *Consider the problem*

$$\begin{array}{lll} (MB') & \textit{maximize} & \sum_{i=1}^n P_i^T P_i x_i - p^T p \\ & \textit{subject to} & p = \sum_{i=1}^n P_i x_i, \\ & & \sum_{i=1}^n x_i = 1, \\ & & x \geq 0 \end{array} \tag{4.18}$$

in the variables $x_1, \ldots, x_n$. MB and MB ′ have the same optimum value and the same (unique) point p realizing the optimum value.

Proof. We do not give a strict algebraic proof but argue geometrically, applying the following lifting transform, well-knwon in computational geometry, see e.g. [17]. Project any point $P_i \in I\!R^d$ onto the unit paraboloid

$$U_{d+1} = \{(\lambda_1, \ldots, \lambda_d, \sum_{\ell=1}^{d} \lambda_\ell^2\}$$

in $I\!R^{d+1}$, i.e. assign to it the image point $P_i' := (P_i, P_i^T P_i) \in I\!R^{d+1}$, Figure 4.7. Any feasible, fixed values of $x = (x_1, \ldots, x_n)$ specify a point

$$p' = (p, \sum_{i=1}^{n} P_i^T P_i x_i)$$

in the convex hull of the image points P_i', and the objective function value of MB′ at x is given by the (vertical) distance between this point p' and the point $(p, p^T p)$ on the

paraboloid. For fixed p, this distance is maximum if p' is above p as far as possible, i.e. if it lies on the upper boundary of the convex hull of the P'_i. In this case, p' (resp. p) can be specified as convex combinations of at most $d+1$ affinely independent points P'_i (resp. P_i). This means, we can assume without loss of generality that the optimum of MB$'$ is determined by a point set $\mathcal{Q} \subseteq \mathcal{P}$ such that

- $|\mathcal{Q}| \leq d+1$, and
- $\mathcal{Q}$ is affinely independent.

For any such $\mathcal{Q}$, Rajan proves that the optimum of MB$'$ appears at the center of the minimum spanning ball of $\mathcal{Q}$, the objective function value being the squared radius of this ball [42]. Thus, the optimum of MB$'$ for the whole set $\mathcal{P}$ appears at the center of the largest minimum spanning ball over all $\mathcal{Q}$. On the one hand, this ball is obviously no larger than the minimum spanning ball of $\mathcal{P}$; on the other hand, it is no smaller because the latter is itself determined by at most $d+1$ points, as proved e.g. by Welzl [47].

Uniqueness of the optimal p for MB (and hence for MB$'$) follows by observing that the left-hand sides

$$(P_i - p)^T (P_i - p)$$

of the constraints in (4.17) are strictly convex functions in p, so if two distinct p would exist with the same optimum radius, any convex combination of them would allow for a strictly smaller radius. □

Now, let $f : \mathbb{R}^{d+1} \mapsto \mathbb{R}$ be the function

$$f(p_1, \ldots, p_{d+1}) = \sum_{\ell=1}^{d} p_\ell^2 - p_{d+1},$$

M the $(d+1) \times n$-matrix

$$M := \begin{pmatrix} P_1 & \cdots & P_n \\ P_1^T P_1 & \cdots & P_n^T P_n, \end{pmatrix},$$

A the $(1 \times n)$-matrix

$$A := (1, \ldots, 1)$$

and b the 1-vector $b := (1)$. Then MB$'$ appears as a problem of type CP in the form of (4.2) as follows.

$$\begin{array}{lll} \text{(MB}') & \text{minimize} & f(p) \\ & \text{subject to} & p = Mx, \\ & & Ax = b, \\ & & x \geq 0, \end{array} \tag{4.19}$$

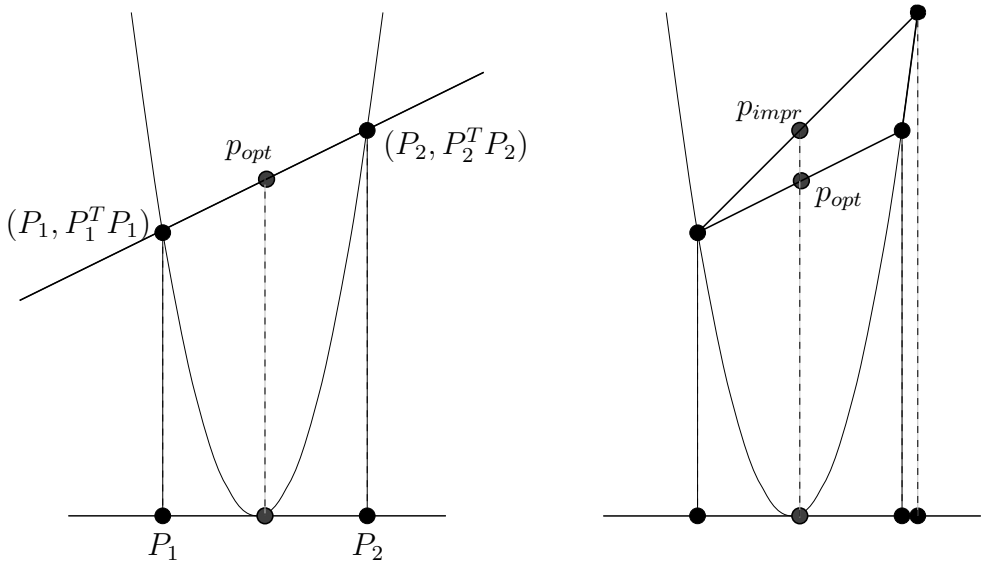


Figure 4.8: MB

with f being the sum of convex functions (quadratic and linear), therefore convex itself (and differentiable with continuous partial derivatives, of course). The matrix A has only one (all-one) row, so the nondegeneracy assumptions are trivially fulfilled.

Uniqueness of the optimal solution for $\mathrm{MB}'(G)$, $G \subseteq H := [n]$ follows from Lemma 4.14 by observing that $\mathrm{MB}'(G)$ is itself a problem in the form of (4.18), for a subset of the points. It remains to prove that the unconstrained problem

$$\begin{array}{lll} (\mathrm{UMB}'(G)) & \text{minimize} & f(p) \\ & \text{subject to} & p = M_G x_G, \\ & & A_G x_G = b \end{array} \tag{4.20}$$

has a unique optimum whenever G is basic, equivalently, if for any feasible solution p of $\mathrm{UMB}'(G)$ the system of equality constraints in $\mathrm{UMB}'(G)$ is uniquely solvable for x_G. Going back to the definition of M and A, this is the case if and only if the lifted point set $\{(P_i, P_i^T P_i), i \in G\} \subseteq U_{d+1}$ is affinely independent. In particular, $|G| \leq d+2$ holds.

Let us first assume $|G| \leq d+1$. The feasible points p of $\mathrm{UMB}'(G)$ are exactly the points on the hyperplane or lower-dimensional flat in $\mathbb{R}^{d+1}$ spanned by the lifted points, and the value of $f(p)$ at any such point is given by its vertical distance to the unit paraboloid. This distance is nonpositive exactly in the compact subset of the flat that lies above the paraboloid. This proves that $\mathrm{UMB}'(G)$ has an optimum, attained at a point p_{opt} above the paraboloid, Figure 4.8, left. Since the vertical distance is the sum of a quadratic, strictly convex function and a linear function, it is itself strictly convex, and the optimal point must be unique. (Rajan proves that p_{opt} projects to the center of the circumsphere of the P_i [42].)

If $|G| = d+2$, the lifted points span the whole space $\mathbb{R}^{d+1}$, and $\mathrm{UMB}'(G)$ is unbounded. However, in the remark after the statement of Result 4.12 we have noted that the optimal

solution to UMB′(G) is just a convenient 'gadget' that helps to perform another operation, namely

> given a feasible solution to MB′(G), find an improved feasible solution to MB′($G - \{i\}$), for some i.

We have argued that there might be a way to find such an improved solution without resorting to UMB′(G). This is exactly the case here. Given the previous optimal solution p_{opt} (or any other point in the convex hull of the lifted points), we can improve the objective function value by moving the point vertically upwards until the upper boundary of the convex hull is hit in a point p_{impr} (Figure 4.8, right). This point is indeed a feasible solution to some MB′($G - \{i\}$). In particular, $|G - \{i\}| = d + 1$, so we are back in the above case where solutions to UMB *can* be used to proceed further. To compute the point p_{impr} and its representation in the x_i, determine the vertical line through p_{opt} as the solution of the linear system in (4.20) that we get by setting $p := p_{opt}$ and then removing the restriction

$$p_{d+1} = \sum_{i \in G} x_i P_i^T P_i$$

from the system. This takes $O(d^3)$ time. Then, for any $x_i, i \in G$ determine the point on the line whose representation satisfies $x_i = 0$. This takes $O(d)$ time per i and determines p_{impr} as the lowest of these points above p_{opt}. Thus, this first improvement step (before we proceed as usual) does not take longer than solving an UMB′ problem (which again comes down to a linear system because f is quadratic). The time bound of Result 4.12 therefore applies with $t_L = O(d^3)$. As already noted, $q = 1$ (as opposed to PD where $q = 2$); in return, M has $d + 1$ rows this time (and not d as in PD). Adding up, both problems lead to exactly the same bound.

Theorem 4.15 *Let MB be a minimum spanning ball problem (4.17) over $n > d+2$ points in $\mathbb{R}^d$. In the LP-type framework, MB can be solved with an expected number of no more than*

$$O(d^4 2^{d+4}(n - d - 2))$$

arithmetic operations. (If $n \leq d + 2$, a bound of

$$O(d^4 2^{n+2})$$

holds.)

4.4 Discussion

We have shown that special convex programming problems are LP-type problems and that efficient (i.e. polynomial) basis improvement routines exist for them. As noted in [33], even convex programming in full generality satisfies the defining properties *monotonicity*

and *locality* of LP-type problems, Definition 3.2, but efficient primitive operations are not known to exist in the general situation. For problems PD and MB, such primitives had been established by the author in [18], and Amenta generalized this to prove that problems of minimizing a smooth, strictly convex function subject to linear constraints allow an efficient treatment [4]. Our method supplements this statement with an explicit realization of the primitive routines, modulo the Lagrangian primitive (4.16). For quadratic programming problems (like PD and MB) this leads to a 'real' linear time bound for fixed dimension. When specialized to the PD problem where one polytope is just a single point, the method is essentially the one Wolfe applied to solve this problem [48]. For the general PD problem, Sekitani and Yamamoto have recently given an interior-point method, without time bounds [44].

Unlike LP, problem PD is apparently well-behaved in many respects. The objective function is strictly convex (therefore, the optimal solution is unique); the problem is bounded, and even the unconstrained version is always uniquely solvable. This might lead to the the impression that PD is actually an easier problem than LP. As shown by Meyer auf der Heide, this is not the case, at least when we go for strongly polynomial bounds. One can reduce any instance of LP in strongly polynomial time to a PD problem with one polytope being a simplex, the other one just a single point [37].

The final remark leads us to the next chapter. The careful reader might have noticed that in order to obtain the bound of Result 4.12 (with a similar behavior in $d+q$), it would not have been necessary to perform the primitive operations in polynomial time. Since any basis has size at most $d+q$, we could actually afford to implement them by brute-force techniques. In particular, during basis improvement, one could imagine testing all the at most 2^{d+q} subsets of $B \cup \{j\}$, B the old basis, j improving, for being the new basis, and this would only double the exponent.

The reason why we have bothered to obtain polynomial bounds becomes clear in the next chapter whose main objective is to reduce the required number of primitive operations to a *subexponential* bound in $d+q$, roughly of the order $2^{\sqrt{d+q}}$. Of course, such a bound would be ruined by the brute-force primitive suggested above.

Chapter 5

Upper Bounds for Abstract Optimization

In the previous chapter, Section 3.2 we have shown that an LP-type system $\mathcal{L} = (H, \mathcal{B}, z)$ can be solved (i.e. a basis $B \in \mathcal{B}$ with $z(B) = z(H)$ can be found) with an expected number of no more than

$$2^{\delta+2}(m - \delta) \tag{5.1}$$

primitive operations (improvement queries and basis improvements), if $m > \delta$, where

$$\delta := \max\{|B| \mid B \in \mathcal{B}\}$$

is the combinatorial dimension of $\mathcal{L}$ and $m = |H|$. This was Result 3.11. (For $m \leq \delta$, a bound of

$$2^{m+2} \tag{5.2}$$

was shown to hold.)

We have shown that for linear programming and other specific convex optimization problems, the primitive operations can be performed in time polynomial in the respective combinatorial dimensions, so that the actual runtimes are basically determined by the bounds (5.1) and (5.2).

The goal of this chapter is to develop *improved* bounds in an even *more general* context, where we restrict ourselves to the 'absolute' basics contained in the LP-type framework. The improvement proceeds in two independent stages, resulting in a substantial improvement over (5.1) for a large range of values m, δ. The first stage refines the methods of Section 3.2, and the second stage develops a better (subexponential) bound for the *small* problems ($m \leq \delta$), where (5.2) gives hardly more than a trivial estimate.

5.1 Abstract Optimization Problems

Let us try to extract the 'spirit' of LP-type systems and find out what we really needed for the approach so far.

The heart of an LP-type systems $(H, \mathcal{B}, z)$ is an (ordered) set of bases $\mathcal{B}$, on top of which we have *subproblem solvability* via *successive improvement*; that's all. In its most general form, the paradigm can be described as follows.

> For every basis B contained in $G \subseteq H$ we need to determine whether there is a better basis B' contained in G (improvement query), and if so, deliver such a basis B' (basis improvement).

This enables us to find the optimal basis contained in G (solving the subproblem on G) by successively improving a candidate basis, provided some initial basis is known.

Of course, LP-type systems present a more specific realization of this paradigm. First of all, the quality of bases is measured by some function z. In itself, this gives no more than an arbitrary ordering among the bases, but in addition we have required z to extend to all subsets of the ground set H. In particular, unlike the general paradigm above, this requires every basis B to be an optimal basis contained in B.

Even more important, we know that if B is not the optimal basis in G, then it is already not the optimal basis in $B \cup \{j\}$ for some $j \in G - B$ (this feature follows directly from property (3.2) and is a great help in actually finding an improving basis). However, by looking at Algorithm 3.4 RF-LPtype it becomes apparent that the improved basis B'' computed in line 9 needs to be a subset of G, but *not* necessarily a subset of $B' \cup \{j\}$, where B' is the previous basis and j an improving variable. Hence, the main defining property (3.2) of LP-type systems is not enforced by the general paradigm of subproblem solvability by successive improvement. It rather introduces additional structure that facilitates the process of testing for improvement and finding an improved basis. We shall assume no additional structure and instead require the existence of an oracle that exactly realizes the general paradigm. Obviously, such an oracle introduces additional structure itself, and for any concrete problem it needs an explicit realization. At this point, one probably arrives at the conclusion that the structure of LP-type systems leads to the most natural realization, and this is actually the case for any concrete problem in this thesis. We therefore do not claim that the new level of abstraction we are going to climb immediately leads to practical applications. Rather, the goal is

- to investigate how far we can get by requiring just a minimal set of axioms (and these axioms will already take us to a subexponential randomized bound, proved in this chapter), and

- to show where this will *not* lead us (namely provably to nothing better than an exponential deterministic bound, derived in the next chapter).

These introductory remarks motivate the following definition.

Definition 5.1 *Let H be a finite set, $\mathcal{B} \subseteq 2^H$ a set of bases. We assume that $\mathcal{B}$ is equipped with a linear ordering $\preceq$ by which bases are compared. Equality ($\simeq$) is allowed and applies to bases with the same quality.*

For $G \subseteq H$ define

$$B(G) := \max_{\preceq}\{B \in \mathcal{B}, B \subseteq G\}.$$

Thus, $B(G)$ is some largest basis contained in G. As in case of LP (definition 2.4), the possible ambiguity presents no problem.

Let Φ be a function defined on pairs (G, B), $B \in \mathcal{B}, B \subseteq G$ with the following property.

$$\Phi(G,B) = \begin{cases} B, & \text{if } B \simeq B(G), \\ B' > B, B' \subseteq G, & \text{otherwise.} \end{cases} \tag{5.3}$$

We refer to Φ as an improvement oracle.
The quadruple $(H, \mathcal{B}, \preceq, \Phi)$ is called an abstract optimization problem *(AOP).*

For ease of handling, we no longer distinguish between improvement query and basis improvement but collect both operations under the improvement oracle. This, in turn, becomes an integral part of the AOP.

If $\mathcal{L} = (H, \mathcal{B}, z)$ is an LP-type system, this defines an AOP $(H, \mathcal{B}, \preceq, \Phi)$ quite in the obvious way. The linear ordering $\preceq$ compares bases by their z-values, and the improvement oracle is obtained from the primitive operations associated with $\mathcal{L}$. There is just one subtle point: the oracle asks for more than what the primitive operations directly give. Recall that for any set G, improvement query and basis improvement apply to specific bases only, namely optimal bases $B = B(G - \{j\})$ of some one-element-less subproblem, while the improvement oracle needs to apply them to *any* basis $B \subseteq G$. Two points let us cope with this.

(i) In case of LP-type systems, $B = B(G)$ if and only if $B = B(B \cup \{j\})$, for all $j \in G - B$. This observation has already been mentioned above and reduces the generic call $\Phi(G, B)$ to at most $|G - B|$ improvement queries and one basis improvement (if a query was successful).

(ii) The algorithms we devise for AOPs are similar in spirit to Algorithm RF-LPtype, and they will as well apply the improvement oracle only in the specific situation where $B = B(G - \{j\})$. The additional freedom might come handy later; for the moment it is no more than an artifact of our wish to have a clean and succinct definition.

As for LP-type systems, we define the *combinatorial dimension* δ of an AOP as the maximum cardinality of any basis, and again this turns out to be a crucial parameter.

The following two sections present algorithms to solve AOPs, where the first algorithm Aop – for large problems, $|H| > \delta$ – is basically Algorithm RF-LPtype, with the second algorithm Small-Aop – for small problems, $|H| \leq \delta$ – plugged in as a subroutine.

As we mentioned back in Section 3.2, Small-Aop comes for free in case of LP (and, more general, for any basis-regular LP-type system): if $|H| = \delta$, then H itself is a basis of H, and no more work has to be done. This is not the case for a general LP-type system, let

alone for an AOP. Thus, we proceed in two stages. First we prove a bound for algorithm Aop which is better than (5.1) for many values of m, but which still has to be multiplied by the cost for a call to Small-Aop to give a real bound. For LP and basis-regular LP-type system, this preliminary bound *is* already a real bound. For all other problems, the second stage presents the algorithm Small-Aop, along with a bound that leads to some – but tolerable – slowdown in Algorithm Aop.

5.2 Large Problems – Algorithm Aop

To make Algorithm RF-LPtype work for an AOP of combinatorial dimension δ, we change it in two respects. First, if $|G| \leq \delta$, we call the subroutine Small-Aop which for the purpose of this section can be treated as a black box. This replaces the previous termination criterion '$B = G$', which is no longer a legal termination criterion, because B need not be optimal for G. Second, improvement query and basis improvement are replaced by a single call to the improvement oracle Φ.

Algorithm 5.2

```
Aop (G, B)
   (* returns B(G). B ⊆ G ⊆ H is an initial basis *)
1  if |G| ≤ δ
2  then
3      return Small-Aop(G, B) (* delivers B(G) *)
4  else (* recur with smaller set *)
5      choose random j ∈ G − B
6      B' := Aop (G − {j}, B) (* find B(G − {j}) *)
7      B'' := Φ(G, B')
8      if B' ≠ B''
9      then (* B'' is improved basis *)
10         return Aop (G, B'') (* repeat with B'' *)
11     else (* B' ≃ B(G) *)
12         return B'
13     fi
14 fi
```

The algorithm is obviously correct (provided that Small-Aop is correct). Observe that for $|G| > \delta$, there is always a choice for j in line 5.

To analyze the performance of Aop, we proceed very similar to Section 3.2, only that now we (need to) invest substantially more effort in solving the main recurrence (3.10) that will turn up again. Previously, any attempts to do so would immediately have been killed by the bad bound for the small problems. Here we are in another situation, since we are going to analyze the performance just in terms of the number of calls to Small-Aop.

The following definition and the lemma have no other objective than to take you to recurrence (3.10) again, in the more general context of AOPs. Besides from the fact that we formally need to do this, we repeat the arguments of Section 3.2 in shorter form to get you back into game, to acquaint you with the AOP terminology, and to convince you that back in Section 3.2 we really did nothing that would not immediately work for AOPs. We start with the hidden dimension.

Definition 5.3 *Let $\mathcal{P} = (H, \mathcal{B}, \preceq, \Phi)$ be an AOP, and consider a basis $B \subseteq G \subseteq H$. An element $j \in G$ is called* enforced *in (G, B) if $B \succ B(G - \{j\})$. The number*

$$\delta(G, B) := \min(\delta, |G|) - \#\{j \in G \mid j \text{ enforced in } (G, B)\}$$

is called the hidden dimension *of (G, B). (δ is the combinatorial dimension of $\mathcal{P}$.)*

Recall from earlier that the enforced elements in (G, B) are elements which must lie in every basis traced by the call $\textsc{Aop}(G, B)$. Since the algorithm does not know about this explicitly, the term hidden dimension is used to measure the 'difficulty' of its task. As before, we can easily show that the hidden dimension is always nonnegative (all enforced elements have to fit into B) and that $\delta(G, B) = 0$ implies $B = B(G)$.

Now comes the lemma that concerns the average hidden dimension of the second recursive call in line 10 of Aop, in other words, the expected progress the algorithms makes in the first recursive call. In particular, for $|G|$ large with respect to δ, with high probability there will be no second recursive call.

Lemma 5.4 *Fix (G, B) and consider the elements $j_1, \ldots, j_k \in G - B$ that cause a second recursive call if they are chosen in line 5 of* Aop*, together with their respective bases of line 6 and 7,*

$$\begin{array}{rcl} B'_\ell & = & B(G - \{j_\ell\}), \\ B''_\ell & = & \Phi(B'_\ell, G), \end{array} \quad \ell = 1, \ldots, k.$$

Assume the j_ℓ are ordered such that

$$B'_1 \preceq \cdots \preceq B'_k.$$

Then (G, B''_ℓ) has hidden dimension at most $\delta(G, B) - \ell$.

Proof. Because of $B''_\ell \succ B$, all elements which are enforced in (G, B) are also enforced in (G, B''_ℓ). Moreover,

$$B''_\ell \succ B'_\ell \simeq B(G - \{j_\ell\}) \succeq B(G - \{j_{\ell-1}\}) \succeq \cdots \succeq B(G - \{j_1\}),$$

so (G, B''_ℓ) features at least ℓ new enforced elements $j_1, \ldots, j_\ell$. Since the hidden dimension stays nonnegative, this implies $k \leq \delta(G, B)$ as well. ◻

Now we can give a recurrence for the maximum expected number of oracle calls needed by Algorithm AOP. Again, the derivation is completely similar to the one in Section 3.2, but we promise that you encounter new material afterwards.

Fix some AOP $(H, \mathcal{B}, \preceq, \Phi)$ of combinatorial dimension δ. For $m, k \geq 0$ let $T(m,k)$ (resp. $T_S(m,k)$, for $m \leq \delta$) denote the maximum expected number of oracle queries performed in a call to AOP(G,B) (resp. SMALL-AOP(G,B)), with basis $B \subseteq G \subseteq H$, $|G| = m$ and $\delta(G,B) \leq k$.

Lemma 5.5

$$T(m,k) = T_S(m,k),\ m \leq \delta, \tag{5.4}$$

$$T(m,k) \leq T(m-1,k) + 1 + \frac{1}{m-\delta} \sum_{\ell=1}^{\min(k,m-\delta)} T(m,k-\ell),\ m > \delta. \tag{5.5}$$

Proof. For (5.4) there is nothing to prove. For the first term of (5.5) we observe that the recursive call in line 6 solves a problem on $m-1$ elements, with hidden dimension at most $\delta(G,B)$ (if i is enforced in (G,B), then i is enforced in $(G-\{j\},B)$). One query is done in line 7. Finally, the third term bounds the effort for the second recursive call, averaged over all $|G-B|$ choices of j in line 5. As Lemma 5.4 shows, at most k (and no more than $|G-B|$) of these choices actually trigger an additional call, where the ℓ-th such choice leads to hidden dimension at most $k-\ell$. This gives a bound of

$$\frac{1}{|G-B|} \sum_{\ell=1}^{\min(k,|G-B|)} T(m,k-\ell), \tag{5.6}$$

which is no more than the last term of (5.5) because $T(m,k-\ell)$ decreases with increasing ℓ, and so the average of $T(m,k-\ell)$ over the range $\ell = 1,\ldots,|G-B|$ is no larger than the average over the smaller range $\ell = 1,\ldots,m-\delta$. □

To get an upper bound for $T(m,k)$ we define an auxiliary function $t(m,k)$ by $t(0,k) = t(m,0) = 1$, for all $m,k \geq 0$ and

$$t(m,k) = t(m-1,k) + \frac{1}{m} \sum_{\ell=1}^{\min(m,k)} t(m,k-\ell),\ k,m \geq 1. \tag{5.7}$$

By the following lemma, a bound on $t(m,k)$ leads to a bound on $T(m,k)$, with a multiplicative overhead of $(m-\delta)(T_S(\delta,k)+1)$. Thus, $t(m,k)$ bounds the performance 'modulo the small problems'.

Lemma 5.6 *For $m > \delta$,*

$$T(m,k) \leq (m-\delta)t(m-\delta,k)(T_S(\delta,k)+1). \tag{5.8}$$

Proof. Completely straightforward, but convince yourself. For $k = 0$ we have

$$\begin{aligned} T(m,0) &\leq T_S(\delta,0) + (m-\delta) \leq (m-\delta)(T_S(\delta,0)+1) \\ &= (m-\delta)t(m-\delta,0)(T_S(\delta,0)+1), \end{aligned}$$

for $m = \delta + 1$ we get

$$\begin{aligned} T(m,k) &\leq \sum_{i=0}^{k} T_S(\delta,i) + k + 1 \leq (k+1)(T_S(\delta,k)+1) \\ &= (m-\delta)t(m-\delta,k)(T_S(\delta,k)+1). \end{aligned}$$

Finally, for $k > 0$ and $m > \delta + 1$ we inductively obtain

$$\begin{aligned} T(m,k) &\leq (m-\delta-1)t(m-\delta-1,k)(T_S(\delta,k)+1)+1 \\ &+ \frac{1}{m-\delta} \sum_{\ell=1}^{\min(m-\delta,k)} (m-\delta)t(m-\delta,k-\ell)(T_S(\delta,k-\ell)+1) \\ &\leq (m-\delta)t(m-\delta-1,k)(T_S(\delta,k)+1) \\ &+ (m-\delta)\frac{1}{m-\delta} \sum_{\ell=1}^{\min(m-\delta,k)} t(m-\delta,k-\ell)(T_S(\delta,k)+1) \\ &= (m-\delta)(T_S(\delta,k)+1)\left(t(m-\delta-1,k) + \frac{1}{m-\delta}\sum_{\ell=1}^{\min(m-\delta,k)} t(m-\delta,k-\ell)\right) \\ &= (m-\delta)t(m-\delta,k)(T_S(\delta,k)+1). \end{aligned}$$

□

5.2.1 The Results

To let you know what we are going for, we anticipate the result of the analysis and state the bound and its implications right here.

Theorem 5.7 *For all $m, k > 0$,*

$$t(m,k) \leq \exp\left(2\sqrt{k\underline{\ln}\frac{m}{\sqrt{k}}} + (\ln 3 + 2)\sqrt{k} + \underline{\ln}\frac{m}{\sqrt{k}}\right) \leq \exp\left(2\sqrt{k\ln m} + O(\sqrt{k} + \ln m)\right),$$

where $\underline{\ln}x := \max(\ln x, 1)$.

Via Lemma 5.6, this implies the following result on abstract optimization problems.

Result 5.8 *Any AOP $(H, \mathcal{B}, \preceq, \Phi)$ of combinatorial dimension δ on $|H| = m > \delta$ elements can be solved with an expected number of no more than*

$$(m-\delta)\exp\left(2\sqrt{\delta\ \underline{\ln}\frac{m-\delta}{\sqrt{\delta}}} + (\ln 3 + 2)\sqrt{\delta} + \underline{\ln}\frac{m-\delta}{\sqrt{\delta}}\right)(T_S(\delta,\delta)+1)$$

oracle queries, where $T_S(\delta,\delta)$ denotes the expected number of oracle queries needed to solve any sub-AOP on δ elements.

The remarkable property of this bound is that in contrast to the previous $2^{\delta+2}(m-\delta)$ bound of Result 3.11 for LP-type systems, the exponent grows only with $\sqrt{\delta}$ – it is *subexponential*. In return, however, the bound is no longer linear in m. Still, for m not too large compared to δ, it is a substantial improvement.

For LP, $T_S(\delta,\delta)=1$ in the AOP framework (one oracle query is needed to certify that a basis defines its own optimum), so the bound is a 'real' bound, and this holds for any basis-regular LP-type system when it is cast as an AOP, as described above. In this case, the analysis applies verbatim to Algorithm RF-LPtype (resp. Random-Facet-Simplex for LP) where it even yields $T(\delta,\delta)=0$.

This implies the following bounds (for LP we know the complexity of an oracle query $\approx$ pivot step – it is $O(d^2)$ when we solve the dual as described in Subsection 3.2.3).

Theorem 5.9 *Any basis-regular LP-type system on $m>\delta$ elements, of combinatorial dimension δ, can be solved with an expected number of no more than*

$$(m-\delta)\exp\left(2\sqrt{\delta\ \underline{\ln}\frac{m-\delta}{\sqrt{\delta}}}+(\ln 3+2)\sqrt{\delta}+\underline{\ln}\frac{m-\delta}{\sqrt{\delta}}\right)$$

improvement queries, resp. basis improvements by Algorithm 3.4 RF-LPtype.

Theorem 5.10 *Any linear program in d variables and n constraints (plus d nonnegativity constraints) can be solved with an expected number of no more than*

$$O(d^2)\ n\ \exp\left(2\sqrt{d\ \underline{\ln}\frac{n}{\sqrt{d}}}+(\ln 3+2)\sqrt{d}+\underline{\ln}\frac{n}{\sqrt{\delta}}\right)\le\exp(2\sqrt{d\ln n}+O(\sqrt{d}+\ln n))$$

arithmetic operations, by applying Algorithm 2.9 Random-Facet-Simplex *to its dual.*

These two bounds are the main results of Matoušek, Sharir and Welzl [33].

5.2.2 The Analysis

The proof for a good bound on $t(m,k)$ as defined in (5.7) follows the one in [33], where additional care is taken for explicit constants. The proof relies on the method of *generating functions* and – although being somewhat technical – it is a clean application of the method.

To begin with, we define for all $m\ge 0$

$$\tau_m(z):=\sum_{k=0}^{\infty}t(m,k)z^k. \tag{5.9}$$

The prower series τ is called the *generating function* of the sequence $(t(m,k))_{k\ge 0}$. The goal is to develop an explicit formula for τ, by using the recurrence relation (5.7) defining $t(m,k)$. From this explicit formula for τ we then try to retrieve information about the coefficients $t(m,k)$.

By multiplying (5.7) with z^k and summing over all k, for $m \geq 1$, we obtain

$$\begin{aligned}
\tau_m(z) &= \tau_{m-1}(z) + \frac{1}{m}\sum_{k=0}^{\infty}\sum_{\ell=1}^{\min(m,k)} t(m,k-\ell)z^k \\
&= \tau_{m-1}(z) + \frac{1}{m}\sum_{\ell=1}^{m}\sum_{k=\ell}^{\infty} t(m,k-\ell)z^k \\
&= \tau_{m-1}(z) + \frac{1}{m}\sum_{\ell=1}^{m} z^\ell \sum_{k=\ell}^{\infty} t(m,k-\ell)z^{k-\ell} \\
&= \tau_{m-1}(z) + \frac{1}{m}\sum_{\ell=1}^{m} z^\ell \sum_{k=0}^{\infty} t(m,k)z^k \\
&= \tau_{m-1}(z) + (\frac{1}{m}\sum_{\ell=1}^{m} z^\ell)\tau_m(z).
\end{aligned}$$

This gives

$$\tau_m(z) = \tau_{m-1}(z)\frac{1}{1-\frac{1}{m}\sum_{\ell=1}^{m} z^\ell},$$

and together with

$$\tau_0(z) = \sum_{k=0}^{\infty} t(0,k)z^k = \sum_{k=0}^{\infty} z^k = \frac{1}{1-z},$$

we obtain

$$\tau_m(z) = \frac{1}{1-z}\prod_{j=1}^{m}\frac{1}{1-\frac{1}{j}\sum_{\ell=1}^{j} z^\ell}. \tag{5.10}$$

Since all coefficients $t(m,k)$ of τ_m are nonnegative, we get

$$t(m,k)s^k \leq \tau_m(s), \tag{5.11}$$

for all k and all nonnegative s for which $\tau_m(s)$ exists. This is the case for $0 \leq s < 1$, because then

$$1 - \frac{1}{j}\sum_{\ell=1}^{j} s^\ell > 1 - \frac{1}{j}\sum_{\ell=1}^{j} 1 = 0, \tag{5.12}$$

so none of the denominators in (5.10) become zero. Consequently, (5.11) shows that for $0 < s < 1$,

$$t(m,k) \leq \frac{1}{s^k(1-s)}\prod_{j=1}^{m}\frac{1}{1-\frac{1}{j}\sum_{\ell=1}^{j} s^\ell} = \frac{1}{s^k(1-s)}\prod_{j=1}^{m}\frac{1}{R_j}, \tag{5.13}$$

$$R_j := 1 - \frac{1}{j}\sum_{\ell=1}^{j} s^\ell = 1 - \frac{s-s^{j+1}}{j(1-s)}.$$

This looks like a rather crude estimate, because (5.11) seems to be the most trivial bound one could possibly come up with. Luckily this is not the case, and by a careful choice of s we will be able to obtain an almost optimal bound. Assume for the rest of the analysis that $m, k > 0$. We choose s of the form

$$s = 1 - \frac{1}{q},$$

where $q \geq 2$ is an integer. We derive a bound on (5.13) in terms of q and then determine a value of q that minimizes this bound. The two main steps independently develop bounds for

$$\prod_{j=1}^{q-1} \frac{1}{R_j} \text{ (bound for small } j\text{), and } \prod_{j=q}^{m} \frac{1}{R_j} \text{ (bound for large } j\text{).}$$

The bound for small j. Let us assume that $j < q = 1/(1-s)$. Exploiting that

$$s^{j+1} = (1-(1-s))^{j+1} = \sum_{\ell=0}^{j+1} \binom{j+1}{\ell} (-1)^\ell (1-s)^\ell,$$

we can rewrite R_j as

$$\begin{aligned} R_j &= \frac{j(1-s) - s + s^{j+1}}{j(1-s)} = \frac{-1 + (j+1)(1-s) + s^{j+1}}{j(1-s)} \\ &= \frac{1}{j} \sum_{\ell=2}^{j+1} \binom{j+1}{\ell} (-1)^\ell (1-s)^{\ell-1} = \frac{1}{j} \sum_{\ell=2}^{j+1} \binom{j+1}{\ell} (-1)^\ell q^{-\ell+1}, \end{aligned}$$

so by some magic, the 'rest' of R_j exactly cancels the first two terms of s^{j+1}'s expansion. The ratio between the absolute values of consecutive summands is

$$q \frac{\binom{j+1}{\ell}}{\binom{j+1}{\ell+1}} = q \frac{\ell+1}{j+1-\ell} \geq \frac{(j+1)(\ell+1)}{j+1-\ell} \geq 1,$$

for all $l \geq 0$. This means, the sequence of summands is decreasing, and we get a lower bound for the sum by omitting any tail that starts with a positive summand. In particular,

$$\begin{aligned} R_j &\geq \frac{1}{j} \left(\binom{j+1}{2} q^{-1} - \binom{j+1}{3} q^{-2} \right) \\ &= \frac{j+1}{2q} - \frac{(j+1)(j-1)}{6q^2} = \frac{j+1}{2q}\left(1 - \frac{j-1}{3q}\right) \geq \frac{2}{3}\frac{j+1}{2q} = \frac{j+1}{3q}. \end{aligned}$$

Using the estimate

$$\frac{n^k}{k!} \leq \frac{1}{\sqrt{2\pi k}} \left(\frac{en}{k}\right)^k, \; k \geq 1 \tag{5.14}$$

obtained from Stirling's formula, Appendix (7.2), we get

$$\prod_{j=1}^{q-1} \frac{1}{R_j} \leq \prod_{j=1}^{q-1} \frac{3q}{j+1} = \frac{(3q)^{q-1}}{q!} = \frac{1}{3q}\frac{(3q)^q}{q!} \leq \frac{1}{3q\sqrt{2\pi q}}(3e)^q \leq \frac{(3e)^{q-1}}{q}, \tag{5.15}$$

because $q \geq 2$ and $\sqrt{4\pi} > e$.

The bound for large j. Now assume that $j \geq q = 1/(1-s)$. Then we get

$$R_j = 1 - \frac{s - s^{j+1}}{j(1-s)} \geq 1 - \frac{s}{j(1-s)} = 1 - \frac{q-1}{j} = \frac{j-q+1}{j}.$$

Using the estimate

$$\binom{n}{k} \leq \frac{n^k}{k!}$$

combined with the weaker variant of (5.14) that we obtain by omitting the $1/\sqrt{2\pi k}$ factor, this gives

$$\prod_{j=q}^{m} \frac{1}{R_j} \leq \prod_{j=q}^{m} \frac{j}{j-q+1} = \frac{q(q+1)\cdots m}{(m-q+1)!} = \binom{m}{q-1} \leq (\frac{em}{q-1})^{q-1}, \tag{5.16}$$

if $q \leq m$ (otherwise we have an empty product and a bound of 1 holds).

Putting things together. We still need an estimate for the factor

$$\frac{1}{s^k(1-s)} = q\left(1 - \frac{1}{q}\right)^{-k}$$

in (5.13). For $x \leq 0.6838\ldots$ $(1-x) \geq \exp(-x-x^2)$ holds so that we get

$$\left(1 - \frac{1}{q}\right)^{k} \geq \exp(-k/q - k/q^2),$$

and therefore

$$q\left(1 - \frac{1}{q}\right)^{-k} \leq q\exp(k/q + k/q^2). \tag{5.17}$$

Putting together (5.17), (5.15) and (5.16) gives a bound of

$$\begin{aligned} t(m,k) &\leq \exp(k/q + k/q^2)(3e)^{q-1}(\frac{em}{q-1})^{q-1} \\ &= \exp\left(k/q + k/q^2 + (\ln 3 + 1)(q-1) + (q-1)\ln(em/(q-1))\right), \end{aligned} \tag{5.18}$$

for $q \leq m$. For $q > m$, the last term in the exponent just disappears. Now is the time to choose q. Let $\underline{\ln}x := \max(\ln x, 1)$. We set

$$q := \lceil p \rceil,\ p := \sqrt{\frac{k}{\underline{\ln}\frac{m}{\sqrt{k}}}}.$$

In all previous considerations we have assumed $q \geq 2$, and to guarantee this, p should be larger than 1, equivalently

$$m < e^k\sqrt{k}. \tag{5.19}$$

Let us assume that this holds. Then we can bound the first two terms in the exponent of (5.18) as follows.

$$\frac{k}{q} \leq \frac{k}{p} = \sqrt{k \underline{\ln} \frac{m}{\sqrt{k}}}, \tag{5.20}$$

$$\frac{k}{q^2} \leq \frac{k}{p^2} = \underline{\ln} \frac{m}{\sqrt{k}}. \tag{5.21}$$

The remaining two terms in the exponent get larger if we replace $q-1$ with $p \geq q-1$. This is obvious for the third term. For the last one it follows from the fact that

$$\frac{\partial}{\partial x} x(\ln \frac{em}{x}) = \ln \frac{em}{x} - 1 \geq 0,$$

for all $x \leq m$, and if the term is present at all we have $q-1 \leq p \leq q \leq m$.

This gives

$$\begin{aligned}(q-1)\ln\frac{em}{q-1} &\leq p \ln \frac{em}{p} = \sqrt{\frac{k}{\underline{\ln}\frac{m}{\sqrt{k}}}} \ln \frac{em\sqrt{\underline{\ln}\frac{m}{\sqrt{k}}}}{\sqrt{k}} \\ &= \sqrt{\frac{k}{\underline{\ln}\frac{m}{\sqrt{k}}}} \left(\ln \frac{m}{\sqrt{k}} + \ln \sqrt{\underline{\ln}\frac{m}{\sqrt{k}}} + 1 \right) \\ &\leq \sqrt{k \underline{\ln} \frac{m}{\sqrt{k}}} + \sqrt{k}. \end{aligned} \tag{5.22}$$

The last inequality uses $\ln x \leq \underline{\ln} x$ and $\ln x + 1 \leq x$, for all x.

Finally, the third term in the exponent of (5.18) is bounded by

$$(\ln 3 + 1)(q-1) \leq (\ln 3 + 1)p = (\ln 3 + 1)\sqrt{\frac{k}{\underline{\ln}\frac{m}{\sqrt{k}}}} \leq (\ln 3 + 1)\sqrt{k}. \tag{5.23}$$

The four estimates (5.20), (5.21), (5.22) and (5.23) together finally imply a bound of

$$t(m,k) \leq \exp\left(2\sqrt{k \underline{\ln} \frac{m}{\sqrt{k}}} + (\ln 3 + 2)\sqrt{k} + \underline{\ln} \frac{m}{\sqrt{k}} \right), \tag{5.24}$$

for $m < e^k \sqrt{k}$. In [33] it is shown that this bound is essentially best possible, for a large range of values m, k (not quite covering the whole range $m < e^k\sqrt{k}$). Only this result fully justifies all the technical effort we spent. It was tough, but it was worth it.

What happens if $m \geq e^k\sqrt{k}$? Then we set $q = 2$ and directly evaluate (5.18). This gives

$$t(m,k) \leq \exp\left(\frac{3}{4}k + \ln m + \ln 3 + 2 \right).$$

We claim that this is dominated by the bound of (5.24). To this end we observe that

$$\begin{aligned} \frac{3}{4}k &\leq \frac{3}{4}\sqrt{k}\sqrt{\ln\frac{n}{\sqrt{k}}} = \frac{3}{4}\sqrt{k\underline{\ln}\frac{m}{\sqrt{k}}}, \\ \ln m &= \underline{\ln}\frac{m}{\sqrt{k}} + \ln\sqrt{k}, \\ \ln\sqrt{k} + \ln 3 + 2 &\leq \sqrt{k}(\ln 3 + 2), \end{aligned}$$

for all $k \geq 1$. Thus, (5.24) holds for all $m, k > 0$, and this is Theorem 5.7.

5.3 Small Problems – Algorithm Small-Aop

Recall that Result 5.8 bounds the expected number of oracle queries needed to solve an AOP on $|H| = m$ elements, of combinatorial dimension $\delta < m$, by

$$(m-\delta)\exp\left(2\sqrt{\delta\ \underline{\ln}\frac{m-\delta}{\sqrt{\delta}}} + (\ln 3 + 2)\sqrt{\delta} + \underline{\ln}\frac{m-\delta}{\sqrt{\delta}}\right)(T_S(\delta,\delta)+1),$$

where $T_S(\delta,\delta)$ bounds the performance of a still hypothetic algorithm Small-Aop on pairs (G,B) with $|G| = \delta$ elements. There are two reasons why such pairs require a different treatment. The first one is quite obvious and has been mentioned before. In case of $G = B$ we neither can argue that $B = B(G)$ nor is it possible to recur with a set $G - \{j\} \supseteq B$ like Algorithm Aop does it in line 6. Of course, there is an easy way around this. By calling the oracle Φ once on (G,B), we either find that $B = B(G)$ or we obtain a basis $B' \subsetneq G$ by which B is replaced. This would enable Algorithm Aop to proceed regularly, for any size of G. The extra call is not even necessary if $|B| < |G|$ holds right in the beginning – δ is the *maximum* cardinality of any basis, but a particular one can be smaller.

The second (and deeper) reason for treating size-δ (and smaller) problems different is efficiency. Recall that for $|G| > \delta$, the expected number of oracle queries performed during the second recursive call in algorithm Aop can be bounded – according to (5.6) – by

$$\frac{1}{|G-B|}\sum_{\ell=1}^{\min(k,|G-B|)} T(m, k-\ell),$$

where k is the hidden dimension of (G,B). This bound is the better the larger $|G-B|$ is, in other words, the more random choices the algorithm has in line 5. While $|G-B|$ gets small, the bound gets worse. Luckily, in this case the first recursion (line 6) bottoms out after as few as $|G-B|$ levels with a call to Small-Aop which we account for separately; this compensates for the inefficiency of the second call. The way around described above makes algorithm Aop applicable to sets G of any size, but then the first recursive call does no longer compensate an inefficient second one if $|G-B|$ is small because it will typically bottom out only after $|G|$ levels instead of $|G-B|$. In the consequence, this method leads to an expected number of oracle calls which we cannot prove to be better than exponential

in δ, for $|G| = \delta$. It therefore ruins the subexponential behavior of Result 5.8 that we have deduced with so much effort.

On the other hand, these bad news do not mean that we need to abandon the whole idea behind algorithm AOP in order to handle pairs (G, B) with small $|G|$. In fact, the algorithm SMALL-AOP we are going to develop proceeds along the same lines as AOP does. Just like AOP, it will have a first and a second recursive call, the first one solving the problem on $G - \{j\}$, for some element $j \in G$. The crucial difference is that if $|G - B|$ is small (which means that there are only few elements to choose from), *some preliminary computations generate more choices in order to make the second call efficient on average.*

5.3.1 The Basic Idea.

Fix some pair (G, B). If we would (like it is done in AOP) insist on using the same basis B for the first recursive call on the set $G - \{j\}$ (thus starting the algorithm on the pair $(G - \{j\}, B)$), then of course, j would be restricted to lie in $G - B$, and there would be no way to generate more choices. But we might as well start a call on $(G - \{j\}, B')$, where B' is some other basis, and as long as B' is no worse than B, i.e. $B' \succeq B$ holds, such a choice can make the whole procedure only more efficient (we formalize this intuition later). Typically $G - B'$ contains elements j which are not yet in $G - B$, so the set of elements j which allow a call with $(G - \{j\}, B)$ *or* with $(G - \{j\}, B')$ is larger than the set of choices we had before. Of course, a suitable basis B' is easily obtained via $B' := \Phi(G, B)$ [1], and by calling the oracle several times, we might hope to collect enough bases such that many elements j exist that can be chosen as the pivot element for the first recursive call with at least one of these bases. The 'preliminary computations' of SMALL-AOP addressed above just represent a way of collecting enough such bases without spending too much oracle calls on this task. Some terminology is necessary to explain the details.

Definition 5.11 *Consider a pair (G, B), $B \subseteq G$. $j \in G$ is called* free *with respect to (G, B) if it is not enforced, equivalently if there exists a basis $B' \subseteq G - \{j\}$ with $B' \succeq B$. In this case B' is called a* witness *for j (with respect to (G, B)).*

A witness of j proves that j is free. Note that the elements of $G - B$ are free with witness B. With this terminology, the task described above consists of finding free elements (in addition to the ones in $G - B$, if these are too few), along with corresponding witnesses. For $|G| \leq \delta$ (which is the case we are dealing with here), the hidden dimension of (G, B) is given by

$$\delta(G, B) = |G| - \#\{j \in G \mid j \text{ enforced in } (G, B)\}, \tag{5.25}$$

so the total number of free elements in (G, B) equals $\delta(G, B)$. For the performance of the algorithm it would be best to identify as many as possible; on the other hand, the more

[1] or we encounter that B is already optimal

we require, the harder is the task of actually finding them. As we will see, a reasonable balance is achieved by requiring witnesses for at least

$$\lceil \alpha \ \delta(G, B) \rceil$$

free elements, where $0 < \alpha < 1$ is a constant to be determined later. Since it is the nature of the hidden dimension that its value is not known to the algorithm, we majorize it by

$$\lceil \alpha |G| \rceil$$

and make this the lower bound for the number of free elements we need to find. Of course, it may happen that $\delta(G, B) < \lceil \alpha |G| \rceil$ in which case we will not succeed. Then, however, we already find the optimal basis in G during the search process.

5.3.2 The Algorithm

Again, fix some pair (G, B) and denote by D the set of elements which have currently been identified as free. This set is constructed incrementally, in the beginning we set $D := G - B$. In case D is already large enough, i.e. $|D| \geq \lceil \alpha |G| \rceil$, there is nothing to do. Otherwise we will have to enlarge D by at least one more free element 'hidden' in B; to this end we will step through a sequence of improving oracle queries (in a way to be described in a minute), until a witness for a yet unknown free element is found (or we already end up in $B(G)$).

Suppose that in the generic step certain elements in G have already been identified as free in (G, B), and we need to find another free one. Here is the way to do it: call the algorithm recursively with (G, B'), where B' is the current basis, but supply an additional parameter E that contains all the elements of G whose status is yet unknown (note that $E \subseteq B'$). This recursive call now has two ways to terminate: either it finds the desired solution $B(G)$ or – while improving its basis – discovers some $B'' \subseteq G$ which fails to contain E. This, however, means that the elements in $E - B''$ have been uncovered as free elements with witness B'', so the call has found at least one new free element and has therefore accomplished its task. The key observation is that as long as D is small, the set E of elements with unknown status will be large, and since the recursive call with parameter E terminates as soon as the first basis B'' appears that is no longer wedged between E and G, it actually operates only on $G - E$ instead of G, which makes it substantially cheaper. In other words, the algorithm tentatively 'enforces' the elements of E and either finds the optimal basis in G under this tentative hypothesis, or it disproves the hypothesis by delivering (at least) one new free element. If enough free elements have been found, the algorithm proceeds as before: it removes a random free element j and recurs on $(G - \{j\}, B^j)$, where B^j is a witness of j.

Consequently, the problem Small-Aop(G, B) is solved by a call to an auxiliary algorithm Small-Aop-E which has three parameters $G \supseteq B \supseteq E$.

Algorithm 5.12

SMALL-AOP(G, B)
(* returns $B(G)$ *)
1 **return** SMALL-AOP-E$(G, B, \emptyset)$

The top-level of SMALL-AOP-E (where $E = \emptyset$) matches the description given above. Down the recursion the size of the problem is measured by $|G - E|$, and the quantity $\lceil \alpha |G - E| \rceil$ replaces $\lceil \alpha |G| \rceil$ as the critical value for the required number of free elements.

The procedure SMALL-AOP-E(G, B, E) either returns $B(G)$ or delivers a basis $B' \succ B$ with $E \not\subseteq B' \subseteq G$ (if $E = \emptyset$, only the former alternative is possible, therefore SMALL-AOP(G, B) indeed returns $B(G)$). The set D of currently free elements is implicitly maintained, comments will refer to it. The witness for en element j will be denoted by B^j.

Algorithm 5.13

SMALL-AOP-E (G, B, E)
(* returns basis $B_{out} \subseteq G$ satisfying either $B_{out} = B(G)$ or $B_{out} \succ B,\ E \not\subseteq B_{out} \subseteq G$ *)
1 **if** $G = E$ **then**
2 **return** $\Phi(G, B)$
3 **else** (* find free elements *)
4 $E' := B$ (* elements of unknown status *)
5 **for all** $j \in G - E'$ **do** (* set witness for set $D := G - E'$ of initially free elements *)
6 $B^j := B$
7 **od**
8 **while** $|G - E'| < \lceil \alpha |G - E| \rceil$ **do** (* D still too small, try to enlarge it *)
9 B:=SMALL-AOP-E(G, B, E')
10 **if** $E \not\subseteq B$ **then** (* $B_{out} := B$ *)
11 **return** B
12 **elsif** $E' \not\subseteq B$ **then** (* new free element(s) found *)
13 **for all** $j \in E' - B$ **do** (* set witness *)
14 $B^j := B$
15 **od**
16 $E' := E' \cap B$ (* shrink E'; $D = G - E'$ gets larger *)
17 **else** (* line 9 has already computed $B(G)$ *)
18 **return** B
19 **fi**
20 **elihw**
21 choose random $j \in G - E'$ (* random element from D *)
22 B':=SMALL-AOP-E$(G - \{j\}, B^j, E)$
23 **if** $E \not\subseteq B'$ **then** (* $B_{out} := B'$ *)
24 **return** B'
25 **else** (* once we get here, $B' = B(G - \{j\})$ holds *)
26 $B'' := \Phi(G, B')$
27 **if** $B' = B''$ **or** $E \not\subseteq B''$ **then** (* $B_{out} := B''$ *)
28 **return** B''

```
29          else (* repeat with better estimate *)
30              return Small-Aop-E(G, B'', E ∪ {j})
31          fi
32      fi
33 fi
```

Termination of Small-Aop-E follows by observing that the recursive calls in line 9, 22 and 30 solve smaller problems, measured in terms of $|G - E|$ (note that the call in line 9 is not executed if $E' = E$ because then the algorithm does not enter the **while**-loop). If $|G - E| = 0$ then the call terminates in line 2. The **while**-loop terminates because E' gets smaller in line 16.

The correctness is not hard to see but since the algorithm is somewhat involved, we give a detailed proof.

Theorem 5.14 *Given $G \supseteq B_0 \supseteq E$, the basis B_{out} returned by the call* Small-Aop-E *(G, B_0, E) either satisfies*

$$B_{out} = B(G) \tag{5.26}$$

or

$$B_{out} \succ B_0,\ E \not\subseteq B_{out} \subseteq G. \tag{5.27}$$

Proof. By induction over the problem size $m := |G - E|$. For $m = 0$, i.e. $G = E$, line 2 returns

$$B_{out} := \Phi(G, B_0).$$

Either $B_{out} = B_0$ holds in which case (5.26) holds by definition of Φ, or $B_{out} \subseteq G = E$, with $B_{out} \succ B_0$. In particular, $B_{out} \neq B_0$, so $E \not\subseteq B_{out}$ and (5.27) follows.

For $m > 0$, line 6 declares $B = B_0$ a witness for all elements in $G - E' = G - B_0$ which is obviously correct. Next consider the **while**-loop. We have already argued above that it terminates. Moreover, at the beginning of each iteration, the invariant $G \supseteq B \supseteq E'$ holds – line 4 guarantees this in the first execution, line 16 for the subsequent ones. Let B_i denote the basis obtained in line 9 of the i-th loop iteration. By the inductive hypothesis, we either have

$$B_i = B(G) \tag{5.28}$$

or

$$B_i \succ B_{i-1} \succ \cdots \succ B_0,\ E' \not\subseteq B_i \subseteq G. \tag{5.29}$$

Hence, if $E \not\subseteq B_i$ holds, then $B_{out} := B_i$ satisfies (5.27) and is a correct answer in line 11. If (5.29) holds, then $B_i \subseteq G - \{j\}$ for all $j \in E' - B_i$, so B_i is legally established as witness for j in line 14. In case of (5.28), $B_{out} := B_i$ satisfies (5.26) and is correctly returned in line 18.

After the **while**-loop we have legal witnesses B^j for all elements $j \in G - E'$, so $G - \{j\} \supseteq B^j$ holds in line 22; moreover, $B^j \supseteq E$ must hold, otherwise B^j would have been returned during the execution of the **while**-loop in which is was discovered. Hence the inductive hypothesis applies again and gives either

$$B' = B(G - \{j\}) \tag{5.30}$$

or

$$B' \succ B^j,\ E \not\subseteq B' \subseteq G - \{j\}. \tag{5.31}$$

In case of (5.31), $B_{out} := B'$ satisfies (5.27) and is therefore a legal answer in line 24, because

$$B' \succ B^j \succeq B_0,$$

where $B^j \succ B_0$ for all witnesses different from B_0 by (5.29). In case of (5.30), line 27 checks exactly the conditions (5.26) and (5.27) for the basis B'' computed in line 26 (observe that $B'' \succ B' \succeq B_0$ in case of $B'' \neq B'$). If both fail, we have

$$E \subseteq B''$$

and

$$B'' \succ B' = B(G - \{j\}).$$

This means that j is enforced in (G, B'') – in particular, j is an element of B'', so $E \cup \{j\} \subseteq B''$ holds, and the inductive hypothesis shows that the basis B_{out} computed in line 30 either satisfies

$$B_{out} = B(G)$$

or

$$B_{out} \succ B'' \succ B' \succeq B_0,\ E \cup \{j\} \not\subseteq B_{out} \subseteq G.$$

Since j was enforced in (G, B''), we have $j \in B_{out}$, so $E \cup \{j\} \not\subseteq B_{out}$ is equivalent to $E \not\subseteq B_{out}$. Hence B_{out} satisfies one of (5.26) and (5.27), as required. □

5.3.3 The Results

As in case of Algorithm AOP in the last section, let us state the result before we dive into the (again somewhat technical) analysis.

Result 5.15 *Any AOP $(H, \mathcal{B}, \preceq, \Phi)$ on $|H| = \delta$ elements can be solved with an expected number of no more than*

$$T_S(\delta, \delta) \leq 2\delta \exp\left(2\sqrt{\delta} + 2\sqrt[4]{\delta}\ln\delta + \ln^2\delta\right) = \exp(O(\sqrt{\delta}))$$

oracle queries by Algorithm SMALL-AOP.

The exponent of $T_S(\delta,\delta)$ is asymptotically neglegible compared to the $O(\sqrt{\delta}\ln(m-\delta))$ exponent of the bound for the large problems given by Result 5.8. This means, we achieve the goal of keeping the overhead introduced by subroutine Small-Aop small. (Note that 'small' only refers to the exponent. The actual slowdown is still superpolynomial!)

By plugging the bound into Result 5.8 (using $T_S(\delta,\delta)+1 \le 2T_S(\delta,\delta)$), we finally obtain the joint result of the previous and the present section, a 'real' bound, valid for all AOPs, with an exponent asymptotically no worse than the one in Result 5.8.

Result 5.16 *Any AOP $(H,\mathcal{B},\preceq,\Phi)$ of combinatorial dimension δ on $|H|=m\ge\delta$ elements can be solved by Algorithm 5.2* Aop *with an expected number of no more than*

$$4\delta(m-\delta)\exp\left(2\sqrt{\delta\ \underline{\ln}\frac{m-\delta}{\sqrt{\delta}}}+(\ln 3+4)\sqrt{\delta}+2\sqrt[4]{\delta}\ln\delta+\underline{\ln}\frac{m-\delta}{\sqrt{\delta}}+\ln^2\delta\right)$$

oracle queries, $\underline{\ln}x := \max(\ln x, 1)$.

As already anticipated in the Discussion concluding the previous chapter, this leads to subexponential bounds for the concrete convex programming problems CP that we have defined in (4.2). Namely, these problems are LP-type problems and therefore AOPs. Specifically, for the polytope distance problem PD, (4.1) and the minimum spanning ball problem MB, (4.17), the complexity of a single oracle query has been bounded by $O(d^3)$, d the dimension of the problem so that we get the following (asymptotic) results.

Theorem 5.17 *Any polytope distance problem (4.1) over n points in $\mathbb{R}^d$ and any minimum spanning ball problem (4.17) over n points in $\mathbb{R}^d$ can be solved in the AOP framework with an expected number of no more than*

$$\exp\left(2\sqrt{d\ln n}+O(\sqrt{d}+\ln n)\right)$$

oracle queries.

5.3.4 The Analysis

Just like the quantity $|G-E|$ replaces $|G|$ as a measure of size when the third parameter E is introduced to obtain Algorithm Small-Aop-E, the hidden dimension becomes a slightly different meaning for triples (G,B,E). Recall that the elements in E have the status of enforced elements during a call to Small-Aop-E(G,B,E). Therefore we define the hidden dimension of such a triple (for $|G|\le\delta$) by

$$\delta(G,B,E) := |G| - \#\{j\in G \mid j \text{ enforced in } (G,B) \text{ or } j\in E\}. \tag{5.32}$$

Therefore, just like $\delta(G,B)\le|G|$ holds under (5.25), $\delta(G,B,E)\le|G-E|$ holds under (5.32). This also guarantees that the first phase of Small-Aop-E achieves the goal described in the informal description above, namely to compute a number of free elements which is at least proportional to the hidden dimension.

Now fix some AOP $(H, \mathcal{B}, \preceq, \Phi)$ with combinatorial dimension δ. For $m \geq k \geq 0$ let $T_E(m,k)$ denote the maximum expected number of oracle queries performed in a call to SMALL-AOP-E(G, B, E), $E \subseteq B \subseteq G \subseteq H$, B basis, of size $|G - E| = m, \delta(G, B, E) \leq k$. By Algorithm 5.12, the quantity $T_S(\delta, \delta)$ we are still missing in Result 5.8 (and more general, the quantity $T_S(m,k)$ bounding the number of oracle queries during a call to SMALL-AOP(G, B) with $|G| = m$ and $\delta(G, B) \leq k$), computes as

$$T_S(m,k) = T_E(m,k). \tag{5.33}$$

Lemma 5.18 *$T_E(m,0) = 1$. For $m \geq k > 0$, $T_E(m,k)$ is bounded by*

$$\sum_{\ell=0}^{\lceil \alpha m \rceil - 1} T_E(\ell, \min(k,\ell)) + T_E(m-1, k-1) + 1 + \frac{1}{\lceil \alpha k \rceil} \sum_{\ell=1}^{\lceil \alpha k \rceil} T_E(m-1, k-\ell). \tag{5.34}$$

Proof. If $k = 0$, then $B = B(G)$ must hold; moreover, $G = B$ because any elements in $G - B$ would count for $\delta(G, B, E)$. If $m = 0$ (equivalently $G = E$), the algorithm terminates in line 2. Otherwise it enters the first iteration of the **while**-loop with $G = B = E'$. The recursive call in line 9 then immediately delivers $B = \Phi(G, B)$ from its line 2; subsequently, the **else**-branch in line 17 is entered from which B is finally returned. In either case, one oracle query has been spent.

For $m \geq k > 0$, the first term of (5.34) bounds the effort during the **while**-loop which is executed at most once for every value of $\ell := |G - E'|$ between 0 and $\lceil \alpha |G - E| \rceil - 1 = \lceil \alpha m \rceil - 1$. The hidden dimension of (G, B, E') in line 9 is at most k in every iteration, because B keeps improving all the time, so enforced elements stay enforced. Moreover, $E' \supseteq E$. On the other hand, the hidden dimension is always bounded by the size $\ell = |G - E'|$.

If processed at all (i.e. if the algorithm gets beyond the **while**-loop), the triple in line 22 has size $m - 1$ and hidden dimension at most $k - 1$: if i is enforced in (G, B_0), then i is enforced in $(G - \{j\}, B^j)$ because of $B^j \succeq B_0$ (as before, B_0 is the original basis the algorithm started from). Moreover, according to (5.32), j itself no longer counts for $\delta(G - \{j\}, B^j, E)$.

Line 26 of the algorithm contributes one more oracle call, and the last term estimates the expected effort, averaged over the final recursive call in line 30 (always assuming that the algorithm reaches this line and does not terminate earlier). Here we invoke the same kind of argument as we did in deriving recurrence (5.5) of Lemma 5.5 for the large problems. Consider the elements $j_1, \ldots, j_r$ that were at choice in line 21, ordered in such a way that

$$B(G - \{j_1\}) \preceq \cdots \preceq B(G - \{j_r\}).$$

Assume j_ℓ was chosen. Then

$$B'' \succ B(G - \{j_\ell\}) \succeq \cdots \succeq B(G - \{j_1\}),$$

so the elements $j_1, \ldots, j_\ell$ are enforced in (G, B''). Since none of them are enforced in (G, B_0) (but free by construction) nor contained in E (otherwise we would not have found a witness for it containing E), we have

$$\delta(G, B'', E \cup \{j_\ell\}) \leq \delta(G, B_0, E) - \ell \leq k - \ell.$$

Since ℓ is equally likely to be any of the numbers $1, \ldots, r$, this consideration bounds the expected effort in line 30 by

$$\frac{1}{r} \sum_{\ell=1}^{r} T_E(m-1, k-\ell),$$

which is no more than the last term of (5.34), because $T_E(m-1, k-\ell)$ decreases with increasing ℓ so that the average of $T_E(m-1, k-\ell)$ over the range $\ell = 1, \ldots, r$ is no larger than the average over the smaller range $\ell = 1, \ldots, \lceil \alpha k \rceil$. To this end observe that

$$r \geq \lceil \alpha |G - E| \rceil = \lceil \alpha m \rceil \geq \lceil \alpha k \rceil,$$

because $k \leq m$ by definition. □

Although the recurrence for $T_E(m, k)$ given by Lemma 5.18 looks rather complicated, the next lemma shows that we can analyze its dependence on m and k separately. This makes it simpler to solve than recurrence (5.5) for the large problems. Let us define two auxiliary single-parameter functions f and g by

$$f(k) = f(k-1) + \frac{1}{\alpha k} \sum_{\ell=1}^{k} f(k-\ell), \tag{5.35}$$

$$g(m) = g(m-1) + \sum_{\ell=0}^{\lceil \alpha m \rceil - 1} g(\ell), \tag{5.36}$$

with $f(0) = g(0) = 1$.

Lemma 5.19 $T_E(m, k) \leq g(m) f(k)$.

Proof. By induction. For $k = 0$, $T_E(m, k) = 1 \leq g(m) f(k)$ holds for all $m \geq 0$. Now assume $m \geq k > 0$. Inductively we get

$$T_E(m, k) \leq \sum_{\ell=0}^{\lceil \alpha m \rceil - 1} g(\ell) f(\min(k, \ell)) + g(m-1) f(k-1) + 1 + \frac{1}{\lceil \alpha k \rceil} \sum_{\ell=1}^{\lceil \alpha k \rceil} g(m-1) f(k-\ell).$$

f is monotone in k, so $g(\ell) f(\min(k, \ell))$ is majorized by $g(\ell) f(k)$. For $\ell = 0$ we can even save one, i.e. $g(0) f(0) \leq g(0) f(k) - 1$, because $f(k) \geq f(1) \geq 2$. This lets us get rid of the '+1' and we further obtain

$$\begin{aligned}
T_E(m, k) &\leq f(k) \sum_{\ell=0}^{\lceil \alpha m \rceil - 1} g(\ell) + g(m-1) f(k-1) + \frac{1}{\lceil \alpha k \rceil} \sum_{\ell=1}^{\lceil \alpha k \rceil} g(m-1) f(k-\ell) \\
&= f(k) \left(g(m) - g(m-1) \right) + g(m-1) \left(f(k-1) + \frac{1}{\lceil \alpha k \rceil} \sum_{\ell=1}^{\lceil \alpha k \rceil} f(k-\ell) \right) \\
&\leq f(k) \left(g(m) - g(m-1) \right) + g(m-1) \left(f(k-1) + \frac{1}{\alpha k} \sum_{\ell=1}^{k} f(k-\ell) \right) \\
&= f(k) \left(g(m) - g(m-1) \right) + g(m-1) f(k) \\
&= g(m) f(k).
\end{aligned}$$

□

It remains to bound $f(k)$ and $g(m)$. Let us start with $g(m)$.

Lemma 5.20 *$g(m) \leq 2m^{\log_{1/\alpha} m+1}$, for $m > 0$.*

Proof. By induction on m. Since $g(1) = 2$, the bound holds with equality for $m = 1$. Now assume $m > 1$. g is monotone in m, so we can majorize any $g(\ell)$ in (5.36) by $g(\lceil \alpha m \rceil - 1)$ to obtain

$$g(m) \leq g(m-1) + \lceil \alpha m \rceil g(\lceil \alpha m \rceil - 1),$$

which by expanding $g(m-1)$ gives

$$g(m) \leq \sum_{\ell=1}^{m} \lceil \alpha \ell \rceil g(\lceil \alpha \ell \rceil - 1) + 1.$$

Majorizing $\lceil \alpha \ell \rceil g(\lceil \alpha \ell \rceil - 1)$ by $mg(\lceil \alpha m \rceil - 1)$ (as above, for $\ell = 1$ we save one, because $g(0) \leq mg(\lceil \alpha m \rceil) - 1) - 1$, we get

$$g(m) \leq m^2 g(\lceil \alpha m \rceil - 1).$$

By the inductive hypothesis, this yields

$$\begin{aligned} g(m) &\leq m^2 2(\lceil \alpha m \rceil - 1)^{\log_{1/\alpha}(\lceil \alpha m \rceil - 1)+1} \leq m^2 2(\alpha m)^{\log_{1/\alpha} \alpha m + 1} \\ &= 2m^2 (\alpha m)^{\log_{1/\alpha} m} \\ &= 2m^{\log_{1/\alpha} m + 2} \alpha^{\log_{1/\alpha} m} \\ &= 2m^{\log_{1/\alpha} m + 1}. \end{aligned}$$

□

The lemma shows that the dependence of $T_E(m,k)$ on the problem size m is only *quasi-polynomial*. The major contribution comes from k.

Lemma 5.21 *$f(k) \leq e^2 \sqrt{k/\alpha}$.*

Proof. We are going to prove an explicit formula for f, namely

$$f(k) = \sum_{\ell=0}^{k} \frac{(1/\alpha)^\ell}{\ell!} \binom{k}{\ell}. \tag{5.37}$$

From this, the bound of the lemma can be derived as follows (using the estimate $\binom{k}{\ell} \leq k^\ell/\ell!$ in the first inequality).

$$\begin{aligned} f(k) &= \sum_{\ell=1}^{k} \binom{k}{\ell} \frac{(1/\alpha)^\ell}{\ell!} \leq \sum_{\ell=1}^{k} \frac{(k/\alpha)^\ell}{\ell!^2} \\ &= \sum_{\ell=1}^{k} (\frac{\sqrt{(k/\alpha)}^{\,\ell}}{\ell!})^2 \leq (\sum_{\ell=1}^{k} \frac{\sqrt{(k/\alpha)}^{\,\ell}}{\ell!})^2 \leq (\sum_{\ell=1}^{\infty} \frac{\sqrt{(k/\alpha)}^{\,\ell}}{\ell!})^2 = e^2 \sqrt{k/\alpha}. \end{aligned}$$

(5.37) can systematically be obtained by applying the method of generating functions; it can also be verified by induction. Here we give a combinatorial proof, originally suggested by P. Flajolet for the case $\alpha = 1$ (extension due to G. Rote).

Consider a permutation $\pi \in S_k$. A subset $R \subseteq [k]$ is called an *increasing chain* in π iff $\forall\, x, y \in R : x < y$ implies $\pi(x) < \pi(y)$. Let $\mathcal{I}(\pi)$ denote the set of increasing chains in π. The *weight* of a chain R is defined as

$$w(R) := \left(\frac{1}{\alpha}\right)^{|R|}.$$

The *total weight* of π is obtained by summing up the weights of all increasing chains,

$$w(\pi) := \sum_{R \in \mathcal{I}(\pi)} w(R).$$

We claim that

$$f(k) = W(k) := \frac{1}{k!} \sum_{\pi \in S_k} w(\pi),$$

i.e. $f(k)$ is the expected total weight of a random permutation. To prove this we show that $W(k)$ satisfies the same recurrence as f, along with $W(0) = f(0) = 1$.

To see the latter observe that there is only one permutation $\pi \in S_0$, and it has exactly one increasing chain, of weight 1, namely the empty chain. For $k > 0$ let us write

$$k!\, W(k) = \sum_{\pi \in S_k} \sum_{R \in \mathcal{I}(\pi)} w(R) = \sum_{\pi \in S_k} \sum_{\substack{R \in \mathcal{I}(\pi) \\ k \notin R}} w(R) + \sum_{\ell=1}^{k} \sum_{\substack{\pi \in S_k \\ \pi(\ell) = k}} \sum_{\substack{R \in \mathcal{I}(\pi) \\ k \in R}} w(R). \tag{5.38}$$

In (5.38), the first term of the right hand side equals

$$k \sum_{\pi \in S_{k-1}} \sum_{R \in \mathcal{I}(\pi)} w(R) = k!\, W(k-1), \tag{5.39}$$

because $\pi \in S_{k-1}$ extends to exactly k permutations of $[k]$ with the same set of increasing chains not containing k.

Now consider a summand

$$\sum_{\substack{\pi \in S_k \\ \pi(\ell) = k}} \sum_{\substack{R \in \mathcal{I}(\pi) \\ k \in R}} w(R) \tag{5.40}$$

of the second term. Because of $\pi(\ell) = k$, any sequence R counted in the second sum must be a subset of $\pi(1), \ldots, \pi(\ell)$ that contains $\pi(\ell)$, equivalently,

$$R = \pi(S) \cup \{k\}, S \subseteq \{1, \ldots, \ell - 1\}.$$

Which sets S define increasing chains (and therefore the total weight of π) does not depend on the particular values of $\pi(1), \ldots, \pi(\ell - 1)$ but on their ordering only (and it does not

depend at all on $\pi(\ell+1),\ldots,\pi(k)$). Thus, the possible orderings can be indexed by $S_{\ell-1}$, where a particular ordering appears for

$$\binom{k-1}{\ell-1}(k-\ell)!$$

permutations $\pi \in S_k, \pi(\ell) = k$. Consequently, (5.40) can be rewritten as

$$\begin{aligned}\binom{k-1}{\ell-1}(k-\ell)! \sum_{\pi\in S_{\ell-1}} \sum_{R\in\mathcal{I}(\pi)} w(R\cup\{k\}) &= \frac{1}{\alpha}\binom{k-1}{\ell-1}(k-\ell)!\ (\ell-1)!\ W(\ell-1)\\ &= \frac{1}{\alpha}(k-1)!\ W(\ell-1).\end{aligned}$$

Plugging this and (5.39) into (5.38) and dividing by $k!$ gives

$$W(k) = W(k-1) + \frac{1}{\alpha k}\sum_{\ell=1}^{k} W(\ell-1),$$

which equals recurrence (5.35) for f. This proves $W(k) = f(k)$ for all k.

The explicit formula (5.37) for $f(k) = W(k)$ is now easily derived as follows.

$$\begin{aligned}W(k) &= \frac{1}{k!}\sum_{\pi\in S_k}\sum_{R\in\mathcal{I}(\pi)} w(R) = \sum_R w(R)\operatorname*{prob}_\pi(R\in\mathcal{I}(\pi))\\ &= \sum_{\ell=0}^{k}(1/\alpha)^\ell \sum_{|R|=\ell}\operatorname*{prob}_\pi(R\in\mathcal{I}(\pi)) = \sum_{\ell=0}^{k}(1/\alpha)^\ell\sum_{|R|=\ell}\frac{1}{\ell!} = \sum_{\ell=0}^{k}\frac{(1/\alpha)^\ell}{\ell!}\binom{k}{\ell}.\end{aligned}$$

□

The upper bound of $e^{2\sqrt{k/\alpha}}$ is tight up to a polynomial factor. In Appendix (7.5) we establish a lower bound of

$$f(k) \geq \frac{e^{-e\sqrt{2}/\alpha}}{4(k+1)}e^{2\sqrt{k/\alpha}}.$$

The true asymptotic value is still slightly closer to $e^{2\sqrt{k/\alpha}}$, see [34].

Putting together the last three Lemmas 5.19, 5.20 and 5.21, we obtain a bound of

$$T_E(m,k) \leq 2\exp\left(2\sqrt{k/\alpha} + \ln m(\log_{1/\alpha} m + 1)\right)$$

for the maximum expected number of oracle queries performed during Algorithm 5.13 Small-Aop-E on triples (G,B,E) of size m and hidden dimension at most k, for any $0<\alpha<1$. Via (5.33), the expected number $T_S(\delta,\delta)$ of oracle calls needed to solve any small AOP on δ elements by algorithm 5.12 Small-Aop is at most

$$T_S(\delta,\delta) \leq 2\exp\left(2\sqrt{\delta/\alpha} + \ln\delta(\log_{1/\alpha}\delta + 1)\right). \tag{5.41}$$

It remains to choose α to make this a real bound which can be plugged into the bound for the large problems given by Result 5.8. Defining

$$\alpha := \frac{1}{1+\omega},\ \omega > 0$$

and rewriting (5.41) gives

$$T_S(\delta, \delta) \leq 2\delta \exp\left(2\sqrt{\delta + \delta\omega} + \frac{\ln^2 \delta}{\ln(1+\omega)}\right). \tag{5.42}$$

Now we choose

$$\omega := \frac{\ln \delta}{\sqrt[4]{\delta}}$$

and estimate both terms in the exponent of (5.42) separately. First, using the mean value theorem, Appendix (7.1), we obtain

$$\sqrt{\delta + \delta\omega} \leq \sqrt{\delta} + \frac{\omega}{2}\sqrt{\delta} = \sqrt{\delta} + \frac{1}{2}\sqrt[4]{\delta}\ln\delta.$$

Second, by comparing derivatives we see that $\ln(1+\omega) \geq \omega/(\omega+1)$ for $\omega \geq 0$, so we get

$$\frac{\ln^2 \delta}{\ln(1+\omega)} \leq \frac{\ln^2 \delta}{\omega} + \ln^2 \delta = \sqrt[4]{\delta}\ln\delta + \ln^2 \delta.$$

Together this gives

$$T_S(\delta, \delta) \leq 2\delta \exp\left(2\sqrt{\delta} + 2\sqrt[4]{\delta}\ln\delta + \ln^2 \delta\right),$$

which finally is the bound of Result 5.15 this whole section was devoted to.

5.4 Discussion

We have developed two algorithms, Aop and Small-Aop, who can be combined to solve any AOP on m elements and combinatorial dimension δ with an expected number of oracle queries that is subexponential in δ and quasi-polynomial in m (this means, the exponent depends on m in a logarithmic fashion). This leads to subexponential algorithms for concrete geometric problems like linear programming, polytope distance and minimum spanning ball. For linear programming, such a bound had already been shown by Matoušek, Sharir and Welzl in the framework of LP-type problems [33]. For the other two problems, subexponential solutions have first been established by the author in [18].

Usually we apply algorithm Small-Aop as a subroutine to solve the small instances that arise in algorithm Aop. However, for an interesting special case, Small-Aop (and the bounds of the previous chapter) serve as a standalone solution, namely if we have an

AOP on a set H where all the subsets of H are bases, and this is the setting originally studied in [18]. Such an AOP can be specified as a triple

$$(H, \preceq, \Phi), \tag{5.43}$$

where $\preceq$ arranges the subsets of H in some arbitrary order, and Φ is an improvement oracle as before. The combinatorial dimension δ in this case coincides with $m = |H|$, the problem size, and Result 5.15 states that (5.43) can be solved with an expected number of no more than

$$2m \exp\left(2\sqrt{m} + 2\sqrt[4]{m} \ln m + \ln^2 m\right) \tag{5.44}$$

oracle queries. Compared to the trivial bound of

$$2^m - 1 \tag{5.45}$$

that we would get by stepping through all the subsets, this is a dramatic saving. However, in all fairness it should be admitted that this saving is purely theoretical, since (5.44) gets better than (5.45) only for $m = 117$ where both bounds evaluate to roughly 1.6×10^{35}.

A somewhat annoying feature of the subexponential bound for the large problems given by Result 5.8 is that the dependence on the problem size m becomes superlinear. However, at least for LP-type problems (and therefore the concrete geometric problems we have considered), there is a way of avoiding this. Consider an LP on n constraints and d variables, $n \gg d$. Let $T(d^2)$ denote the runtime[2] of your favorite algorithm for solving a linear program in d variables on at most $O(d^2)$ constraints. Then Clarkson has an algorithm (applying your favorite one as a subroutine) that solves the whole problem with an expected number of no more than

$$O(d^2 n + d^4 \sqrt{n} \log n + T(d^2) \log n) \tag{5.46}$$

arithmetic operations[12, 20, 38]. As noted in [20], the algorithm and the result extend to LP-type problems of size $n = m$ and combinatorial dimension $d = \delta$, in particular to the polytope distance and the minimum spanning ball problem. For these problems (and for LP), you might choose Algorithm Aop as your favorite algorithm, and obtain

$$T(d^2) = \exp\left(O(\sqrt{d \log d})\right)$$

for all three problems, by Theorem 5.10 resp. Theorem 5.17. Plugging this into (5.46), you will find that the middle term is asymptotically always dominated either by the first or the last one (and in the latter case, the $\log n$ factor gets gobbled up as well), so that the following result is obtained.

[2] number of arithmetic operations

Theorem 5.22 *Any linear programmming, polytope distance, or minimum spanning ball problem over n constraints (resp. points) in dimension d can be solved with an expected number of no more than*

$$O(d^2 n + \exp\left(O(\sqrt{d \log d})\right)$$

arithmetic operations.

It is worthwhile to note that Clarkson's algorithm cannot be applied to AOPs, since it makes crucial use of the locality property (3.2) of LP-type systems/problems. Finding a way around this could be possible but no further investigations have been pursued in this direction.

Geometric problems seem to be the most natural candidates for being presentable as LP-type problems or AOPs. However, Ludwig has given a subexponential algorithm for the problem of finding optimal strategies in *simple stochastic games* [32]. This problem allows for a particularly strong oracle and therefore can be solved along the lines of Algorithm Aop, with no need to invoke the complicated mechanism of Algorithm Small-Aop.

Chapter 6

Lower Bounds for Abstract Optimization

In the previous chapter we have developed *upper bounds* for the worst case expected number of primitive operations (calls to an improvement oracle) that are needed to solve any given abstract optimization problem (AOP). Some obvious questions that arise in a discussion of these bounds are the following.

(i) Are the upper bounds tight, i.e. do AOPs exist that require essentially that many oracle queries no matter what algorithm is used?

(ii) At least, do the given algorithms behave according to the bounds or can their analyses be further improved?

(iii) What is the effect of randomization? Are there deterministic algorithms that attain the same upper bounds?

To start with the bad news: we are not able to answer the first question. On the one hand, to the author's knowledge no bounds better than the ones developed in the previous chapter are known, even in the stronger model of the LP-type systems. On the other hand, even in the rather weak framework of the AOPs we do not know of any nontrivial lower bound. In this respect, the chapter title might be misleading – the present chapter is not a symmetric counterpart to the previous one in the sense that it presents absolute bounds. It rather collects lower bound results for *specific* algorithms or *specific* problems.

We start with question (iii) above. Here we prove that randomization is indeed crucial in the AOP-framework. To be more precise, we show that *any* deterministic algorithm that solves AOPs must in the worst case examine *all* the bases of an AOP in order to find the best one. The lower bound is not difficult to obtain, it follows from an adversary argument. Although this result does not mean much for any concrete AOP, it shows that the general AOP framework is still much weaker than the LP-type framework, where no such lower bound is known to exist. This in turn might let the subexponential, randomized upper bound for AOPs from the previous chapter appear in an even stronger light.

The remaining material deals with question (ii) above. Reviewing a result by Matoušek [34], we prove that the kind of analysis that led to a subexponential bound for Algorithms 3.4 RF-LPtype is essentially best possible for this algorithm: a typical (i.e. random) problem chosen from a special class of LP-type systems forces Algorithm RF-LPtype to behave not (much) better than analyzed. This result obviously extends to Algorithm Aop in the weaker framework of the AOPs.

Finally, we show that Matoušek's examples can be 'recycled' to give a lower bound for the expected behavior of Algorithm 3.5 RE-LPtype as well, and here this algorithm finally comes into play. Unlike the subexponential lower bound for RF-LPtype, this bound (although nontrivial) is polynomial and therefore much weaker. However, no nontrivial upper bound (let alone a matching one) is known even for the concrete Algorithm 2.8 Random-Edge-Simplex on linear programs, indicating that this algorithm is difficult to analyze. However, our lower bound has interesting consequences for the behavior of Random-Edge-Simplex on a special linear program whose feasible region is the so-called *Klee-Minty cube*, which has been studied in this context to some extent.

This final chapter once again goes through all the concepts we have considered in this thesis, backwards in time. Starting with AOPs, it proceeds to the more concrete LP-type systems and finally ends where we originally started: with linear programming and the simplex method.

6.1 The Deterministic Lower Bound for AOPs

The goal of this section is to prove that any deterministic algorithm for solving AOPs must in the worst case examine all bases of the input AOP to find the optimal one. This implies that the randomization used in the previous chapter to obtain subexponential bounds was indeed crucial. In other words, while the subexponential algorithm can only 'fool' itself by coming up with 'bad' coin flips (and in the expectation, this does not happen), any deterministic algorithm can be fooled by an adversary who will supply just the problem the algorithm cannot handle efficiently.

Theorem 6.1 *Let H be a finite set, $\mathcal{B} \subseteq 2^H$ a set of bases. For any deterministic algorithm $\mathcal{A}$, there exists an ordering $\preceq$ of the bases and an improvement oracle Φ such that $\mathcal{A}$ needs at least $|\mathcal{B}| - 1$ oracle queries to solve $(H, \mathcal{B}, \preceq, \Phi)$.*

Proof. We start $\mathcal{A}$ on some problem $(H, \mathcal{B}, \preceq, \Phi)$ with $\preceq$ and Φ not yet determined, and we argue that an adversary answering the oracle queries can construct $\preceq$ and Φ 'online' in such a way that the algorithm is forced to step through all the bases. When supplied with a query pair (G, B), the adversary will output an answer $B' = \Phi(G, B)$ according to two simple rules:

(i) the answer B' is consistent with the previous ones, i.e. there exists an AOP on H and $\mathcal{B}$ such that the present and all previous queries have been answered correctly with respect to this AOP.

(ii) The answer $B' = B$ (proving that B is optimal among all subsets of G) is given only if there is no other consistent answer.

It is clear by induction that the adversary always has a consistent answer, so the algorithm $\mathcal{A}$ steps through a sequence of queries with pairs (G, B) and finally stops. Suppose that less than $|\mathcal{B}| - 1$ queries have been performed. Then there are two bases B_1 and B_2 which have never been the second component of a query pair. We will show that it is consistent with all answers to assume that $B_1 \simeq B(H)$, i.e. B_1 is an optimal basis. The same holds for B_2, so whatever the algorithm outputs, there is an AOP that is not correctly solved, a contradiction to $\mathcal{A}$ being an algorithm for solving any AOP.

We are left to prove that $B_1 \simeq B(H)$ is a consistent choice. Clearly, this can fail only if some answer has revealed the existence of a larger basis with respect to $\preceq$. Since there was no query pair (G, B_1), the only remaining possibility for this to happen is that some query (G, B) with $B_1 \subseteq G$ has been answered by B, thus establishing $B_1 \preceq B$. But in the first query of this type, B_1 was still 'free' and could have been returned instead of B, a contradiction to rule (ii). □

6.2 Matoušek's LP-type Systems

In this section we present a class of LP-type systems that has been introduced by Matoušek [34] in order to obtain a lower bound on the runtime of Algorithm RF-LPtype (the bound itself is subject of the next section). As it turns out, the bases of these LP-type systems as well as the z-values that we are going to assign to them can be identified with $GF(2)^\delta$, the vector space of all 0/1-vectors of length δ over the two-element field $GF(2)$. This on the one hand allows to reinterpret algorithms RF-LPtype and RE-LPtype as flipping games over $GF(2)^\delta$ and on the other hand makes linear algebra over $GF(2)$ available as a very useful tool for the analysis of these games.

6.2.1 Definition

Let

$$\begin{aligned} H &:= \{h_1, \ldots, h_\delta\}, \\ \bar{H} &:= \{\bar{h_1}, \ldots, \bar{h_\delta}\} \end{aligned}$$

be two disjoint δ-element sets. For any ℓ, h_ℓ and $\bar{h_\ell}$ are called *conjugate* elements. We consider LP-type systems

$$\mathcal{L}_A = (H \cup \bar{H}, \mathcal{B}, z_A) \tag{6.1}$$

over $H \cup \bar{H}$, parameterized by a certain class of matrices A. The set of bases $\mathcal{B}$ is the same for all problems and is defined by

$$\mathcal{B} := \{B \subseteq H \cup \bar{H} \mid |B \cap \{h_\ell, \bar{h_\ell}\}| = 1, \text{ for all } \ell\}, \tag{6.2}$$

i.e. the bases are all the sets containing *exactly one* of any two conjugate elements. Every basis has δ elements and can be identified with a 0/1 vector $\vec{B}$ whose ℓ-th component is defined by

$$\vec{B}_\ell := \begin{array}{ll} 0, & \text{if } h_\ell \in B \\ 1, & \text{if } \bar{h_\ell} \in B. \end{array}$$

The mapping $B \mapsto \vec{B}$ defines a bijection between $\mathcal{B}$ and $GF(2)^\delta$.

The class of matrices A that appear as parameter in (6.1) consists of all *lower-diagonal* $\delta \times \delta$-matrices

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ a_{\delta 1} & \cdots & \cdots & a_{\delta\delta} \end{pmatrix} \in GF(2)^{\delta\times\delta}$$

with $a_{\ell\ell} = 1$ for all ℓ (equivalently, A is nonsingular). The objective function z_A assigns to any basis B the vector

$$z_A(B) := A\vec{B} \in GF(2)^\delta, \tag{6.3}$$

where $GF(2)^\delta$ becomes an ordered set $\mathcal{W}$ via *decreasing lexicographic* order, i.e. a vector is larger than another if it has a zero-entry in the first component where they differ. This agrees with the usual decreasing order of the natural numbers when 0/1-vectors are interpreted as binary numbers in the obvious way ($11\ldots1$ being smallest, $00\ldots0$ largest). The reason for using decreasing instead of the natural increasing order is merely technical and will become clear soon.

Since A is nonsingular, all the bases get distinct values and can uniquely be ordered according to these values.

Example 6.2 Let $\delta = 3$,

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

The following table lists the 0/1-vectors corresponding to the bases, ordered by increasing z_A-value.

$\vec{B}$	$z_A(B) = A\vec{B}$	binary value
100	111	7
101	110	6
111	101	5
110	100	4
010	011	3
011	010	2
001	001	1
000	000	0

Thus, the all-zero vector is the optimal basis in all problems $\mathcal{L}_A$. The following lemma prepares the proof that $\mathcal{L}_A$ indeed defines an LP-type system. Recall that z_A extends to all subsets G of $H \cup \bar{H}$ via

$$z_A(G) := \max\{z_A(B) \mid B \in \mathcal{B}, B \subseteq G\}. \tag{6.4}$$

Lemma 6.3 *For any $G \subseteq H \cup \bar{H}$, $B \subseteq G$ basis, the following are equivalent.*

(i) $B = B(G)$, *i.e.* $z_A(B) = z_A(G)$.

(ii) $z_A(B)_\ell = 0$ *for all* ℓ *with* $h_\ell, \bar{h_\ell} \in G$.

Proof. Assume (i) holds and consider ℓ with $h_\ell, \bar{h_\ell} \in G$. Then the basis

$$B' := B \triangle \{h_\ell, \bar{h_\ell}\}$$

is in G as well, with

$$z_A(B') = A\vec{B'} = A(\vec{B} + \mathbf{e}_\ell) = z_A(B) + A\mathbf{e}_\ell,$$

where $A\mathbf{e}_\ell$ coincides with the ℓ-th column of A. Since A is lower-diagonal with $a_{\ell\ell} = 1$, $z_A(B)$ and $z_A(B')$ agree in the first $\ell - 1$ components but disagree in the ℓ-th one. With $z_A(B) > z_A(B')$ it follows that $z_A(B)_\ell = 0$.

Now suppose (ii) is satisfied. Choose any basis $B' \subseteq G$, $B' \neq B$ and let ℓ be minimal such that $\vec{B}_\ell \neq \vec{B'}_\ell$. Then $h_\ell, \bar{h_\ell} \in G$, so

$$z_A(B)_\ell = 0.$$

Moreover, because of A's shape, ℓ is also minimal with

$$z_A(B)_\ell = (A\vec{B})_\ell \neq (A\vec{B'})_\ell = z_A(B')_\ell,$$

and this shows $z_A(B) > z_A(B')$. Since B' was any basis, $B = B(G)$ follows. □

Theorem 6.4 *For any lower-diagonal, nonsingular matrix $A \in GF(2)^{\delta \times \delta}$,*

$$\mathcal{L}_A = (H \cup \bar{H}, \mathcal{B}, z_A)$$

with $\mathcal{B}$ and z_A defined according to (6.2) and (6.3) is an LP-type system of combinatorial dimension δ.

Proof. We need to check the two conditions (3.1) and (3.2) of Definition 3.1. The former, demanding that z_A extends to $2^{H \cup \bar{H}}$ as given by (6.4), is automatically satisfied if no basis contains another which is the case because all bases have the same size δ (basis-regularity). Condition (3.2) demands that

$$z_A(G \cup \{j\}) > z_A(G) \text{ implies } z_A(B \cup \{j\}) > z_A(B),$$

for any basis $B \subseteq G$ with $z(B) = z(G)$ and any $j \in H \cup \bar{H} - G$. To see this, fix such B and G. By the previous lemma,

$$z_A(B)_\ell = 0 \text{ for all } \ell \text{ with } h_\ell, \bar{h_\ell} \in G,$$

and if $z_A(G \cup \{j\}) > z_A(G) = z_A(B)$, then $B \neq B(G \cup \{j\})$, so

$$z_A(B)_{\ell'} = 1 \text{ for some } \ell' \text{ with } h_{\ell'}, \bar{h_{\ell'}} \in G \cup \{j\}.$$

This means $j \in \{h_{\ell'}, \bar{h_{\ell'}}\}$, and since $j \notin B$, $B \cup \{j\}$ must contain both $h_{\ell'}$ and $\bar{h_{\ell'}}$. Applying Lemma 6.3 backwards then gives $z_A(B \cup \{j\}) > z_A(B)$. □

6.2.2 Flipping Games

Recall that the primitive operations needed to solve an LP-type system by Algorithm RF-LPtype are the improvement query,

$$z_A(B \cup \{j\}) > z_A(B) \ ?$$

and the basis computation,

$$B' := B(B \cup \{j\}).$$

Actually, a basis improvement primitive (3.5) suffices but in this case we do have a basis computation available. The following lemma shows what the operations look like when we interpret them as operations on the 0/1-vector $z_A(B) = A\vec{B}$ rather than on B itself. This reinterpretation subsequently leads to an equivalent formulation of Algorithm RF-LPtype as a flipping game on objective function values.

Lemma 6.5 *Consider a basis B, $j \in \{h_\ell, \bar{h_\ell}\}$, $j \notin B$. Then*

$$z_A(B \cup \{j\}) > z_A(B) \text{ if and only if } (A\vec{B})_\ell = 1$$

and

$$B' = B(B \cup \{j\}) \text{ if and only if } A\vec{B'} = F_A^\ell A\vec{B},$$

where $F_A^\ell := E + (\underbrace{0, \ldots, 0}_{\ell-1}, A\mathbf{e}_\ell, 0, \ldots, 0) \in GF(2)^{\delta \times \delta}$ is called the flip at ℓ.

Proof. The first equivalence follows directly from the proof of Lemma 6.3. For the second one, let us distinguish two cases. If $B' = B$, then $B' = B(B \cup \{j\})$ equivalently means $z_A(B \cup \{j\}) = z_A(B)$, resp. $(A\vec{B})_\ell = 0$. This in turn holds if and only if $(0, \ldots, 0, A\mathbf{e}_\ell, 0, \ldots, 0)A\vec{B} = 0$, equivalently $A\vec{B'} = A\vec{B} = F_A^\ell A\vec{B}$.

If $B' \neq B$, then $B' = B(B \cup \{j\})$ if and only if $(A\vec{B'})_\ell = 0$ (Lemma 6.3), where $B' = B \triangle \{h_\ell, \bar{h_\ell}\}$ (this is the only other basis in $B \cup \{j\}$). But then we equivalently get $(A\vec{B})_\ell = 1$ and

$$A\vec{B'} = A(\vec{B} + \mathbf{e}_\ell) = A\vec{B} + A\mathbf{e}_\ell = A\vec{B} + (0, \ldots, 0, A\mathbf{e}_\ell, 0, \ldots, 0)A\vec{B} = F_A^\ell A\vec{B}.$$

□

To make this convenient interpretation of basis computations as improving linear transformations of objective function values work, decreasing lexicographic order has been chosen (no linear transformation can improve on the value $00\ldots0$, therefore this has to be the optimum value). In Example 6.2 we get

$$F_A^1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, F_A^2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}, F_A^3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

and the step from basis B to B' with $\vec{B} = 100$, $\vec{B'} = \vec{B} + \mathbf{e}_2 = 110$ takes $z_A(B) = A\vec{B} = 111$ to

$$z_A(B') = A\vec{B'} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = F_A^2 A\vec{B}.$$

In general, when the flip at ℓ is applied to a vector whose ℓ-th entry has value one, it zeroes this entry and flips values at any other position where $A\mathbf{e}_\ell$ equals one. Since A was lower-diagonal, all these positions are to the right of ℓ, so the vector becomes lexicographically smaller. Note that whether a value to the right of ℓ gets flipped does not depend on the value itself but only on its position in the vector. Moreover, if the flip at ℓ is applied to a vector whose ℓ-th entry has value zero, this vector remains unchanged.

Algorithm RF-LPtype works on pairs (G, B) with $B \subseteq G \subseteq H \cup \bar{H}$, B basis. The procedure RF-Flip$_A$ equivalently represents B by

$$V(B) := A\vec{B}$$

and G by the set

$$S(G) := \{\ell \in G \mid \{h_\ell, \bar{h_\ell}\} \subseteq G\}.$$

$S(G)$ contains exactly the subscripts ℓ that appear in $G - B$ but does not keep the information whether h_ℓ or its conjugate $\bar{h_\ell}$ is in $G - B$. However, given B (which uniquely corresponds to $V(B)$), this information is deducible and G can be recovered via

$$G := \{h_\ell, \bar{h_\ell} \mid \ell \in S(G)\} \cup B,\ A\vec{B} = V(B).$$

Hence the mapping

$$(G, B) \mapsto (S(G), V(G))$$

establishes a bijection between all possible pairs (G, B) and all pairs (S, V) with $S \subseteq [\delta]$ and $V \in GF(2)^\delta$. Invoking Lemma 6.5, the following game RF-Flip$_A$ therefore exactly parallels Algorithm RF-LPtype when applied on problem $\mathcal{L}_A$. To make the relations clear, the latter appears to the right, exactly in parallel to RF-Flip.

Game 6.6

RF-FLIP$_A$ (S, V)
(* returns $V(B(G)), V = V(B), S = S(G)$ *)
1 **if** $S = \emptyset$
2 **then**
3 **return** V
4 **else**
5 choose random $\ell \in S$
6 $V' :=$ RF-FLIP$_A$ $(S - \{\ell\}, V)$
7 **if** $V'_\ell = 1$
8 **then**
9 $V'' := F^\ell_A V'$
10 **return** RF-FLIP$_A$ (S, V'')
11 **else**
12 **return** V'
13 **fi**
14 **fi**

RF-LPTYPE (G, B)
(* returns $B(G)$. *)
1 **if** $B = G$
2 **then**
3 **return** B
4 **else**
5 choose random $j \in G - B$
6 $B' :=$ RF-LPTYPE $(G - \{j\}, B)$
7 **if** $z_A(B' \cup \{j\}) > z_A(B')$
8 **then**
9 $B'' := B(B' \cup \{j\})$
10 **return** RF-LPTYPE (G, B'')
11 **else**
12 **return** B'
13 **fi**
14 **fi**

Since the algorithm RE-LPTYPE which we are going to consider soon relies on the same primitives as RF-LPTYPE does, its actions on Matoušek's LP-type systems $\mathcal{L}_A$ can be modeled as a vector flipping game in exactly the same way as it was done for RF-LPTYPE. Therefore, we introduce the corresponding flipping game RE-FLIP$_A$ right here.

Game 6.7

RE-FLIP$_A(S, V)$
(* returns $V(B(G)), V = V(B), S = S(G)$. *)
1 $I := \{\ell \in S \mid V_\ell = 1\}$
2 **if** $I \neq \emptyset$
3 **then**
4 choose random $\ell \in I$
5 $V' := F^\ell_A V$
6 **return** RE-FLIP$_A(S, V')$
7 **else**
8 **return** V
9 **fi**

RE-LPTYPE (G, B)
(* returns $B(G)$. *)
1 $I := \{j \in G - B \mid z(B \cup \{j\}) > z(B)\}$
2 **if** $I \neq \emptyset$
3 **then**
4 choose random $j \in I$
5 $B' := B(B \cup \{j\})$
6 **return** RE-LPTYPE (G, B')
7 **else**
8 **return** B
9 **fi**

Both flipping games can be interpreted as games on binary numbers; in each round, the present number is replaced with a smaller one, according to a randomized rule, until the number zero is reached where the game finally ends. The goal of the two subsequent sections is to develop lower bounds on the expected number of rounds played in these games.

6.3 A Lower Bound for RF-FLIP$_A$

Let $\mathcal{A}$ denote the set of all nonsingular, lower-diagonal $\delta \times \delta$-matrices over $GF(2)$. For any $A \in \mathcal{A}$, any $S \subseteq [\delta]$ and any $V \in GF(2)^\delta$ we let

$$T_A(S, V)$$

denote the expected number of times line 9 is executed (equivalently, the expected number of flips) during Game RF-FLIP$_A$ when started on (S, V). The expectation is over the random choices in line 5. The goal is to prove that there exists a matrix A and a vector V such that $T_A(S, V)$ is large. To this end we are going to prove that the expected value of $T_A(S, V)$ is large where the expectation is over all choices of A and V. In other words, when supplied with a random matrix A and a random initial vector V, Game RF-FLIP will be slow.

Consequently, we define

$$T(S) := \frac{1}{|\mathcal{A}|\ 2^\delta} \sum_{A \in \mathcal{A}} \sum_{V \in GF(2)^\delta} T_A(S, V).$$

Lemma 6.8 *$T(S)$ only depends on $|S|$.*

Proof. For any vector W that arises during the computations, the values W_S at its coordinates with subscripts in S only depend on V_S and on the $|S| \times |S|$-submatrix A_{SS} of A that consists of the rows and columns of A with indices in S; this means, V_S and A_{SS} determine $T_A(S, V)$. However, averaged over A and V, V_S as well as A_{SS} are random and therefore independent of the specific S. □

The lemma allows us to define

$$T(m) := T(S), \text{ for some } S \text{ with } |S| = m.$$

The following is the crucial result.

Theorem 6.9

$$\begin{aligned} T(0) &= 0, \\ T(m) &= T(m-1) + \frac{1}{2}\left(1 + \frac{1}{m}\sum_{\ell=1}^{m} T(m-\ell)\right), \ m > 0. \end{aligned}$$

Proof. Consider the call RF-FLIP$_A(S, V)$ with random A and V, chosen independently and without loss assume $S = [m]$. If $m = |S| = 0$, RF-FLIP$_A$ returns with no flip. For $m > 0$, line 6 solves the problem for a set $S - \{\ell\}$ of size $m - 1$, where A and V are still random. By definition, this contributes $T(m-1)$ flips. By construction of RF-FLIP, the vector V' obtained from line 6 is equal to $z_A(G - \{j\})$, $j = h_\ell$ or $j = \bar{h_\ell}$, and by Lemma 6.3 it must satisfy

$$V'_k = 0, \text{ for all } k \in S, k \neq \ell.$$

V'_ℓ has been obtained by flipping V_ℓ's original value a certain number of times, where the number of these flips only depends on $V_{S-\{\ell\}}$, hence not on V_ℓ. Therefore, if V_ℓ was random, so is V'_ℓ. This means, with probability $1/2$, $V'_\ell = 0$ and the game terminates in line 12. The remaining terms account for the other half of the cases where

$$V'_S = \underbrace{0\ldots0}_{\ell-1}10\ldots0,$$

and the flip in line 9 brings us to

$$V''_S = F^\ell_A V'_S = \underbrace{0\ldots0}_{\ell}\ a_{\ell+1,\ell}\ldots a_{m,\ell}.$$

Since all vectors computed during the final recursive call RF-FLIP$_A(S, V'')$ are lexicographically smaller than V'', the ℓ leading zeroes are fixed once and for all; therefore, no more flips will be performed at these positions and S could actually be replaced by $S_\ell = \{\ell+1,\ldots,m\}$ without changing the expected number of flips. This shows

$$T_A(S, V'') = T_A(S_\ell, V''),$$

the latter depending only

- on the vector $V''_{S_\ell} = a_{\ell+1,\ell}\ldots a_{m,\ell}$, i.e. the ℓ-th column of A, and
- on the sub-matrix A_{S_ℓ,S_ℓ} of A which is missing the ℓ-th column.

Averaged over A, both are random and independent of each other which proves that the recursive call in line 10 contributes

$$T(S_\ell) = T(m-\ell)$$

flips. Finally, observe that ℓ is equally likely to be any of the values $1,\ldots,m$. □

Setting $t(m) := T(m)+1$ gives the recurrence

$$\begin{aligned} t(0) &= 1, \\ t(m) &= t(m-1) + \frac{1}{2m}\sum_{\ell=1}^{m} t(m-\ell), \end{aligned}$$

which is nothing else than a specialization of recurrence (5.35) from the previous chapter to the case $\alpha = 2$. Thus we obtain

$$T(m) = \sum_{\ell=0}^{m} \frac{(1/2)^\ell}{\ell!}\binom{m}{\ell} - 1 \tag{6.5}$$

from the formula for general α developed in the proof of Lemma 5.21.

Recall that $T(m)$ counts the number of flips in Game RF-FLIP, averaged over all matrices A and start vectors V. In particular, for every $|S| = m$ there exists a matrix A and a start vector V such that RF-FLIP$_A(S, V)$ performs at least that many flips (this time averaged over the internal random choices).

To get a lower bound on $T(m)$, note that in Appendix (7.5) resp. Lemma 5.21, bounds of

$$\frac{e^{-e\sqrt{2}/\alpha}}{4(m+1)} \exp\left(2\sqrt{m/\alpha}\right) \leq \sum_{\ell=0}^{m} \frac{(1/\alpha)^\ell}{\ell!} \binom{m}{\ell} \leq \exp\left(2\sqrt{m/\alpha}\right) \tag{6.6}$$

are established, for any $\alpha > 0$ (where the lower bound presumes that $m \geq 2e\sqrt{m/\alpha}$ holds).

When translated back to LP-type systems and Algorithm RF-LPTYPE, the previous discussion implies the following result.

Result 6.10 *For any δ with $\delta \geq 2e\sqrt{\delta/2}$ ($\delta \geq 8$ suffices) there exists an LP-type system (of the form $\mathcal{L}_A$) on 2δ elements, of combinatorial dimension δ, such that the expected number of basis computations in Algorithm* RF-LPTYPE*, when applied to the problem is at least*

$$\frac{e^{-e/\sqrt{2}}}{4(\delta+1)} \exp\left(\sqrt{2\delta}\right) - 1 = \exp\left(\Omega(\sqrt{\delta})\right)$$

for some initial basis.

Just for comparison: the upper bound on the number of basis computations derived from Theorem 5.9 for the basis regular LP-type systems $\mathcal{L}_A$ is

$$\delta \exp\left(2\sqrt{\delta\ \ln\sqrt{\delta}} + (\ln 3 + 2)\sqrt{\delta} + \ln\sqrt{\delta}\right) = \exp\left(O(\sqrt{\delta \log \delta})\right)$$

which is asymptotically worse (even in the exponent) than the lower bound. However, Algorithm RF-LPTYPE cannot beat the subexponential behavior in δ which is the 'heart' of the upper bound. This is what we were aiming at; in particular, to solve LP-type systems substantially faster, a different algorithm would have to be developed.

6.4 A Lower Bound for RE-FLIP$_A$

In this section we derive a lower bound for the expected number of basis computations that algorithm RE-LPTYPE performs on problem $\mathcal{L}_A$. Recall that RE-LPTYPE is an abstraction of the RANDOM_EDGE_SIMPLEX Algorithm 2.8. As above, the analysis exploits the one-to-one correspondence between the actions taken by algorithm RE-LPTYPE on problem $\mathcal{L}_A$ and the game RE-FLIP$_A$.

As in the previous section, let

$$T_A(S, V)$$

denote the expected number of times line 5 is executed (equivalently, the expected number of flips), this time during Game 6.7 RE-FLIP$_A$ when started on (S, V). As before, A is taken from the set $\mathcal{A}$ of all nonsingular, lower-diagonal $\delta \times \delta$-matrix over $GF(2)$, and V is any vector from $GF(2)^\delta$. The expectation is over the random choices in line 4.

Unlike in RF-Flip, the set S remains invariant during the game, and since we are actually interested only in the case $S = [\delta]$, we restrict our considerations to that case and define

$$T_A(V) := T_A([\delta], V).$$

To get an idea about the magnitude of a possible lower bound on $T_A(V)$, we start with an *upper* bound; for RF-Flip$_A$, we had the subexponential upper bound from Chapter 5 that we were trying to approach from below. Here the situation is radically different: Game RE-Flip, equivalently Algorithm RE-LPtype is polynomial on problem $\mathcal{L}_A$.

Lemma 6.11 *For all A and V,*

$$T_A(V) \leq \frac{\delta(\delta+1)}{2}.$$

Proof. We subdivide the game into phases, depending on how the current vector evolves. Let V' denote this vector, at any stage of the game. A phase ends whenever the leftmost one-entry in V' gets flipped – which at the same time advances the leftmost one-entry to the right, because entries to the left are unaffected by the flip. It follows that during phase k, V' has at most $\delta - k + 1$ one-entries, thus the probability of the leftmost one-entry getting flipped in a specific round is at least $1/(\delta - k + 1)$. Interpreting this as a sequence of Bernoulli-trials with probability of success at least $1/(\delta - k + 1)$ independently in every trial, we see that the expected length of phase k is no more than $\delta - k + 1$. Summing over all phases gives

$$T_A(V) \leq \delta + (\delta - 1) + \cdots + 1 = \frac{\delta(\delta+1)}{2}.$$

□

Thus, the best we can hope for is an $\Omega(\delta^2)$ lower bound for game RE-Flip. The best we can prove (by considering a specific matrix A and random vector V) is a lower bound of

$$\Omega(\frac{\delta^2}{\ln \delta}), \tag{6.7}$$

which is worse by a logarithmic factor. Whether game RE-Flip can actually take a quadratic number of rounds for some matrix A and vector V remains an open problem.

The analysis is not easy. The major difficulty is that the number of random choices the game has in each round, varies and depends on previous choices. This is overcome by relating $T_A(V)$ to the performance of another game that draws its random choices uniformly from the whole set S in each round. As a consequence of our method we obtain a lower bound on the performance of the Random-Edge-Simplex algorithm on the *Klee-Minty cube*, which is a linear program that can be formulated as an LP-type system of the form $\mathcal{L}_A$, for exactly the matrix A that we are going to consider for (6.7).

Let us start by introducing the above mentioned 'uniform' flipping game and examine its relations to the original game.

6.4.1 Game RE-FLIP$^*_A$

As already noted in the remarks to Example 6.2, the flip at ℓ leaves a vector V invariant if $V_\ell = 0$. We call this a *void* flip. Thus, game RE-FLIP (which does not contain void flips) can be simulated by flipping in each round with a truly random F^ℓ_A (i.e. with ℓ chosen uniformly from the whole set S) – the additional void flips do not influence how the current vector evolves, they only slow down the game. Let us denote the resulting game by RE-FLIP*.

Game 6.12

RE-FLIP$^*_A(S, V)$
(* returns $V(B(G)), V = V(B), S = S(G)$. *)
1 **while** $V_S \neq 0$ **do**
2 choose random $\ell \in S$
3 $V' := F^\ell_A V$
4 **od**
5 **return** V

If we denote the expected number of rounds (i.e. the expected number of flips, including void flips) played in RE-FLIP$^*_A([\delta], V)$ by $T^*_A(V)$, then

$$T_A(V) \leq T^*_A(V),$$

where $T_A(V)$ counts exactly the *nonvoid* flips. As we show next, by averaging over V we are able to quantify by how much RE-FLIP* is slower than RE-FLIP. In particular, a lower bound on the (easier-to-analyze) game RE-FLIP will then yield a lower bound on the game RE-FLIP that we are really interested in.

Consider the set of all infinite sequences

$$L = \ell_1, \ell_2, \ldots$$

with elements in $[\delta]$. We refer to them as *flip sequences*, because the game RE-FLIP* runs through (finite prefixes of) such sequences, where the random choice in line 2 induces the probability distribution

$$\operatorname*{prob}_L(\ell_k = \ell) = \frac{1}{\delta},$$

for all $\ell \in [\delta]$ and all k.

For a flip sequence $L = \ell_1, \ell_2, \ldots$ and an integer k we let

$$V^{(L,k)} := F^{(L,k)}_A V, \text{ with } F^{(L,k)}_A := F^{\ell_k}_A \cdots F^{\ell_2}_A F^{\ell_1}_A$$

be the result of 'applying' the first k flips of L to the vector V. With this notation we get

Lemma 6.13

$$T_A(V) = \sum_{k=1}^{\infty} \operatorname*{prob}_L(V^{(L,k)} \neq V^{(L,k-1)}), \tag{6.8}$$

$$T_A^*(V) = \sum_{k=0}^{\infty} \operatorname*{prob}_L(V^{(L,k)} \neq 0). \tag{6.9}$$

Proof. The right-hand side of (6.8) sums over all k the probability that the k-th round of RE-FLIP$_A^*$ performs a nonvoid flip. This is just the expected number of nonvoid flips, hence the expected number of rounds in the original game RE-FLIP$_A$. The right-hand side of (6.9) sums over all k the probability that the game RE-FLIP$_A^*$ consists of more than k rounds, and this evaluates to the expected number of rounds. □

By averaging over the vector V, we are going to establish a relation between (6.8) and (6.9). To begin with, recall that $V^{(L,k)} \neq V^{(L,k-1)}$ if and only if the current vector $V^{(L,k-1)}$ has a one-entry at position ℓ_k, which implies that the probability for a nonvoid flip in round k is just the expected number of one-entries in $V^{(L,k-1)}$, divided by δ, hence

$$\operatorname*{prob}_L(V^{(L,k)} \neq V^{(L,k-1)}) = \frac{1}{\delta} \sum_{\ell=1}^{\delta} \operatorname*{prob}_L(V_\ell^{(L,k-1)} = 1). \tag{6.10}$$

Now consider

$$T_A := \frac{1}{2^\delta} \sum_{V \in GF(2)^\delta} T_A(V),$$

the expected number of flips in RE-FLIP$_A$, averaged over all vectors V. Via (6.10) we obtain

$$\begin{aligned} T_A &= \sum_{k=1}^{\infty} \frac{1}{\delta} \sum_{\ell=1}^{\delta} \operatorname*{prob}_{L,V}(V_\ell^{(L,k-1)} = 1) \\ &= \frac{1}{\delta} \sum_{\ell=1}^{\delta} \sum_{k=1}^{\infty} \operatorname*{prob}_{L,V}((F_A^{(L,k-1)} V)_\ell = 1) \\ &= \frac{1}{\delta} \sum_{\ell=1}^{\delta} \sum_{k=1}^{\infty} \operatorname*{prob}_{L,V}(\mathbf{e}_\ell{}^T F_A^{(L,k-1)} V = 1) \\ &= \frac{1}{2\delta} \sum_{\ell=1}^{\delta} \sum_{k=1}^{\infty} \operatorname*{prob}_{L}(\mathbf{e}_\ell{}^T F_A^{(L,k-1)} \neq 0), \end{aligned} \tag{6.11}$$

since $\operatorname{prob}_V(\mathbf{e}_\ell{}^T F_A^{(L,k-1)} V = 1)$ equals $1/2$ if $\mathbf{e}_l{}^T F_A^{(L,k-1)} \neq 0$ (and is 0 otherwise). Note that if $\mathbf{e}_\ell{}^T F_A^{(L,k-1)}$ was equal to $F_A^{(L,k-1)} \mathbf{e}_\ell$, then (6.11) would under (6.9) give

$$T_A = \frac{1}{2\delta} \sum_{\ell=1}^{\delta} T_A^*(\mathbf{e}_\ell), \tag{6.12}$$

and we had obtained a nice relation between games RE-FLIP$_A$ and RE-FLIP$^*_A$. Unfortunately all this does not quite hold. Nevertheless, the following technical lemma allows us to 'pull' $\mathbf{e}_\ell{}^T$ through $F_A^{(L,k-1)}$, at the cost of doing some minor damage both to the vector and the matrix.

For a matrix M (not necessarily quadratic), ${}_TM$ denotes the *co-transpose* of M, which is obtained by reflecting M along its co-diagonal. ${}_TM$ can be obtained by turning M^T upside down (i.e. rotating it by 180°, equivalently reversing the row and column orders). Just like for the ordinary transpose, ${}_T(MM') = {}_TM'{}_TM$ holds. Note that if A is a nonsingular, lower-diagonal matrix that defines a flipping game, so is its co-transpose, and the respective flipping matrices are related as follows.

Lemma 6.14 *For any nonsingular matrix A and $\ell \in [\delta]$,*

$$A^{-1}F_A^\ell A = {}_TF_{{}_TA}^{\delta-\ell+1}.$$

Proof.

$$\begin{aligned}
A^{-1}F_A^\ell A &= A^{-1}(E + (\underbrace{0,\ldots,0}_{\ell-1}, A\mathbf{e}_\ell, 0,\ldots,0))A \\
&= (A^{-1} + (\underbrace{0,\ldots,0}_{\ell-1}, \mathbf{e}_\ell, 0,\ldots,0))A \\
&= E + \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ a_{\ell 1} & \cdots & a_{\ell\delta} \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix} = {}_T\left(E + \begin{pmatrix} 0 & \cdots & a_{\ell\delta} & \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \cdots & \underbrace{a_{\ell 1}}_{\text{column } \delta-\ell+1} & \cdots & 0 \end{pmatrix}\right) \\
&= {}_T(E + (\underbrace{0,\ldots,0}_{\delta-\ell}, {}_TA\ \mathbf{e}_{\delta-\ell+1}, 0,\ldots,0)) \\
&= {}_TF_{{}_TA}^{\delta-\ell+1}.
\end{aligned}$$

□

The following corollary will let us rewrite (6.11) into a form that gives us something similar to the 'would-be' result (6.12).

Corollary 6.15

$$\operatorname*{prob}_L(\mathbf{e}_\ell{}^T F_A^{(L,k-1)} \neq 0) = \operatorname*{prob}_L(F_{{}_TA}^{(L,k-1)}{}_TA\ \mathbf{e}_{\delta-\ell+1} \neq 0).$$

Proof. For $L = \ell_1, \ell_2, \ldots \ell_k, \ell_{k+1}, \ldots$ define $L' = \delta-\ell_k+1, \ldots, \delta-\ell_2+1, \delta-\ell_1+1, \ell_{k+1}, \ldots$ From the Lemma it follows that

$$\mathbf{e}_\ell{}^T F_A^{(L,k-1)} = \mathbf{e}_\ell{}^T A\ {}_TF_{{}_TA}^{(L',k)}\ A^{-1},$$

and this is nonzero if and only if its cotranspose

$$({}_TA)^{-1} F_{{}_TA}^{(L',k)} {}_TA \ \mathbf{e}_{\delta-\ell+1}$$

is nonzero. Observing that $({}_TA)^{-1}$ is an automorphism of $GF(2)^\delta$ we obtain

$$\mathbf{e}_\ell{}^T F_A^{(L,k-1)} \neq 0 \ \Leftrightarrow \ F_{{}_TA}^{(L',k)} {}_TA \ \mathbf{e}_{\delta-\ell+1} \neq 0.$$

Since $L \leftrightarrow L'$ defines a bijection between flip sequences, we get

$$\operatorname*{prob}_L(F_{{}_TA}^{(L',k-1)} {}_TA \ \mathbf{e}_{\delta-\ell+1} \neq 0) = \operatorname*{prob}_L(F_{{}_TA}^{(L,k-1)} {}_TA \ \mathbf{e}_{\delta-\ell+1} \neq 0),$$

and this proves the corollary. □

Plugging this into (6.11) gives the main theorem of this subsection.

Theorem 6.16

$$T_A = \frac{1}{2\delta} \sum_{\ell=1}^{\delta} T^*_{{}_TA}({}_TA \ \mathbf{e}_\ell).$$

Let us illustrate this theorem on the simplest possible example, namely for $A = {}_TA = E$. In this case, a round in game RE-FLIP$_A$ consists of zeroing a random one-entry of the current vector – all other entries remain unchanged. It follows that the number of rounds equals the number of one-entries in the start vector; for a random start vector, $\delta/2$ rounds will be played, thus

$$T_A = \frac{\delta}{2}.$$

On the other hand, when started on ${}_TA\mathbf{e}_\ell = \mathbf{e}_\ell$, game RE-FLIP$^*_{{}_TA}$ needs an expected number of δ rounds until it gets to flip the single one-entry at position ℓ and stops. This gives

$$\frac{1}{2\delta} \sum_{\ell=1}^{\delta} T^*_{{}_TA}({}_TA \ \mathbf{e}_\ell) = \frac{1}{2\delta} \sum_{\ell=1}^{\delta} \delta = \frac{\delta}{2}.$$

6.4.2 Analysis of RE-FLIP$^*_A$

Theorem 6.16 relates the average performance of game RE-FLIP$_A$ over *all* vectors to the average performance of game RE-FLIP$^*_{{}_TA}$ over a few *specific* vectors. Thus, we have a simpler game to analyze, but in return give up the freedom of chosing a random start vector.

We are going to concentrate on a specific matrix A now, namely

$$A := \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}, \tag{6.13}$$

which is the *all-one lower-diagonal* matrix. We have $A = {}_TA$, so Theorem 6.16 tells us that in order to compute T_A, we need to determine – for all ℓ – how many rounds are played in game RE-FLIP$^*_A$ on the average to reduce the vector

$$A\mathbf{e}_\ell = (\underbrace{0,\ldots,0}_{\ell-1},1,\ldots,1)$$

to zero. To this end we will analyze how the vector evolves during the game. Actually, the analysis will only trace the *dimension* which records the one-entry with lowest index. Therefore, the considerations for $A\mathbf{e}_\ell$ are valid as well for any other vector V with the same lowest one-entry.

For a vector V let $d(V)$ denote the *dimension* of V, which is defined as the number of significant bits in the binary number interpretation, i.e.

$$d(V) := \delta + 1 - \min\{\ell \mid V_\ell = 1\}.$$

(For $V = 0$ we let $d(V) = 0$.) For example, $d(A\mathbf{e}_\ell) = \delta - \ell + 1$.

Now define

$$T^*(d) := \min\{T^*_A(V) \mid d(V) = d\}. \tag{6.14}$$

We get $T^*(0) = 0$ and $T^*_A(A\mathbf{e}_\ell) \geq T^*(\delta - \ell + 1)$, and our objective will be to bound

$$\sum_{\ell=1}^{\delta} T^*(\delta - \ell + 1) = \sum_{d=1}^{\delta} T^*(d)$$

from below. To this end fix any vector V of dimension $d > 0$, with $T^*_A(V) = T^*(d)$, and start playing the game on it. Eventually the leading one-entry will be flipped, thereby decreasing the dimension of the vector currently under consideration. The expected number of flips performed until this happens is exactly δ. The expected number of flips performed after the dimension has decreased depends on the actual dimension obtained. For $k < d$ let p_k denote the probability that the dimension goes down from d to k. Then

$$T^*(d) \geq \delta + \sum_{k=0}^{d-1} p_k T^*(k). \tag{6.15}$$

Lemma 6.17 *Let $k < d$. Then*

$$p_0 + \cdots + p_k \leq \frac{1}{d-k}.$$

Proof. $p_0 + \cdots + p_k$ is the probability that the dimension goes down by at least $d-k$, and if this event is possible at all, the flip sequence applied during the game must necessarily hit the leftmost one-entry before it hits any of the next $d-k-1$ positions to the right – otherwise there is a zero-entry at the leftmost such position which was hit, and this entry turns into one by the time the leading position gets flipped, [1] preventing the dimension from advancing by more than $d-k-1$. However, the leading one-entry gets hit first with probability exactly $1/(d-k)$. □

To proceed further, we need a simple fact.

[1] Here we use the fact that A is the all-one lower-diagonal matrix.

Lemma 6.18 *$T^*(d)$ is monotone increasing with d.*

Proof. As above, let

$$V = (\underbrace{0, \ldots, 0,}_{\delta-d} 1, V_{\delta-d+2}, \ldots, V_\delta)$$

be a vector with $T^*_A(V) = T^*(d)$ and consider the shifted vector

$$V' = (\underbrace{0, \ldots, 0,}_{\delta-d+1} 1, V_{\delta-d+2}, \ldots, V_{\delta-1})$$

of dimension $d-1$. We claim that $T^*_A(V') \le T^*_A(V)$ holds which proves the lemma. To see this, consider the following bijection $L \leftrightarrow L'$ between flip sequences. $L = \ell_1, \ell_2, \ldots$ is mapped to $L' = \ell'_1, \ell'_2, \ldots$ defined by

$$\ell'_k := \begin{cases} \ell_k, & \text{if } \ell_k = 1, \ldots, \delta - d \\ \ell_k + 1, & \text{if } \ell_k = \delta - d + 1, \ldots, \delta - 1 \\ \delta - d + 1, & \text{if } \ell_k = \delta \end{cases}$$

Now, if playing the game with L reduces V to zero, then playing the game with L' reduces V' to zero. Consequently, on the average over all L, the game on V does not end before the game on V'. □

Monotonicity of T^* implies that the right hand side of (6.15) is minimized if the tuple $(p_{d-1}, \ldots, p_0)$ is lexicographically smallest subject to $\sum_{k=0}^{d-1} p_i = 1$ and the inequalities established by Lemma 6.17. This is the case if

$$p_k = \frac{1}{d-k} - \frac{1}{d-k+1}$$

for $k > 0$, $p_0 = 1/d$. Recalling that $T^*(0) = 0$, we get

Lemma 6.19 *For $d > 0$,*

$$T^*(d) \ge \delta + \sum_{k=1}^{d-1} \left(\frac{1}{d-k} - \frac{1}{d-k+1} \right) T^*(k).$$

Now we are ready to prove a lower bound on $\sum_{d=1}^{\delta} T^*(d)$.

Theorem 6.20 *Let $H_n := 1 + \frac{1}{2} + \cdots + \frac{1}{n}$ denote the n-th* harmonic number. *Then*

$$\sum_{d=1}^{\delta} T^*(d) \ge \frac{\delta^3}{2(H_{\delta+1} - 1)}.$$

Proof. The inequality of Lemma 6.19 can be rewritten as

$$\sum_{k=1}^{d} \frac{T^*(k)}{d-k+1} \geq \delta + \sum_{k=1}^{d-1} \frac{T^*(k)}{d-k},$$

and after setting $f(d) := \sum_{k=1}^{d} T^*(k)/(d-k+1)$ reads as

$$f(d) \geq \delta + f(d-1)$$

with $f(0) = 0$. This implies $f(d) \geq d\ \delta$ for all $d \leq \delta$, so

$$\sum_{k=1}^{d} \frac{T^*(k)}{d-k+1} \geq d\ \delta. \tag{6.16}$$

Summing up these inequalities for all values of d up to δ gives

$$\begin{aligned} \frac{\delta^2(\delta+1)}{2} &\leq \sum_{d=1}^{\delta} \sum_{k=1}^{d} \frac{T^*(k)}{d-k+1} \\ &= \sum_{k=1}^{\delta} T^*(k) \sum_{d=k}^{\delta} \frac{1}{d-k+1} \\ &= \sum_{k=1}^{\delta} T^*(k) H_{\delta-k+1}. \end{aligned} \tag{6.17}$$

While $T^*(k)$ increases with k, $H_{\delta-k+1}$ decreases, and Chebyshev's first summation inequality, Appendix (7.9), can be applied to yield

$$\begin{aligned} \sum_{k=1}^{\delta} T^*(k) H_{\delta-k+1} &\leq \frac{1}{\delta} \left(\sum_{k=1}^{\delta} T^*(k) \right) \left(\sum_{k=1}^{\delta} H_{\delta-k+1} \right) \\ &= \frac{1}{\delta}(\delta+1)(H_{\delta+1}-1) \sum_{k=1}^{\delta} T^*(i), \end{aligned} \tag{6.18}$$

because

$$\begin{aligned} \sum_{k=1}^{\delta} H_{\delta-k+1} &= \sum_{k=1}^{\delta} H_k = \sum_{k=1}^{\delta} \sum_{m=1}^{k} \frac{1}{m} = \sum_{m=1}^{\delta} \sum_{k=m}^{\delta} \frac{1}{m} = \sum_{m=1}^{\delta} \frac{\delta-m+1}{m} \\ &= \sum_{m=1}^{\delta} \frac{\delta+1}{m} - \delta = (\delta+1)H_\delta - \delta = (\delta+1)(H_{\delta+1}-1). \end{aligned}$$

Putting together (6.17) and (6.18) gives

$$\frac{\delta^3}{2} \leq (H_{\delta+1}-1) \sum_{k=1}^{\delta} T^*(k),$$

and this implies the result. □

We remark that within a factor of 2, the bound of Theorem 6.20 is the best one can deduce from the recurrence in Lemma 6.19. Namely, solving the recurrence with equality leads to

$$\sum_{k=1}^{\delta} \frac{T^*(k)}{\delta - k + 1} = \delta^2$$

in (6.16). This time both $T^*(k)$ and $1/(\delta - k + 1)$ increase with k, and Chebyshev's second summation inequality, Appendix 7.10 gives

$$\delta^2 = \sum_{k=1}^{\delta} \frac{T^*(k)}{\delta - k + 1} \geq \frac{1}{\delta} \left(\sum_{k=1}^{\delta} T^*(k) \right) \left(\sum_{k=1}^{\delta} \frac{1}{\delta - k + 1} \right) = \frac{1}{\delta} \left(\sum_{k=1}^{\delta} T^*(k) \right) H_\delta.$$

This implies

$$\sum_{k=1}^{\delta} T^*(k) \leq \frac{\delta^3}{H_\delta}.$$

6.4.3 The Bound

Now, let us collect our achievements. Originally we were aiming at a lower bound on $T_A(V)$, the expected number of rounds in game RE-FLIP$_A([\delta], V)$. In Theorem 6.16 we have shown that this expectation can be expressed in terms of the easier game RE-FLIP* when we average over V, more precisely,

$$T_A := \frac{1}{2^\delta} \sum_{V \in GF(2)^\delta} T_A(V) = \frac{1}{2\delta} \sum_{\ell=1}^{\delta} T^*_{{}_TA}({}_TA\ \mathbf{e}_\ell), \tag{6.19}$$

where $T^*_{{}_TA}({}_TA\ \mathbf{e}_\ell)$ is the expected number of rounds in game RE-FLIP$^*_{{}_TA}([\delta], {}_TA\ \mathbf{e}_\ell)$. For the all-one lower-diagonal matrix A defined in (6.13), $A = {}_TA$ holds, and the previous subsection shows that for this choice of A, the sum on the right hand side of (6.19) can be bounded from below by

$$\sum_{\ell=1}^{\delta} T^*_A(A\mathbf{e}_\ell) \geq \sum_{d=1}^{\delta} T^*(d) \geq \frac{\delta^3}{2(H_{\delta+1} - 1)}.$$

Pasting this into (6.19) gives the final bound.

Theorem 6.21 *For the all-one lower-diagonal matrix A, the expected number of rounds in game* RE-FLIP$_A$*, averaged over all start vectors, is at least*

$$T_A \geq \frac{\delta^2}{4(H_{\delta+1} - 1)}.$$

When translated back to LP-type systems and algorithm RE-LPTYPE, this gives the following result.

Result 6.22 *For any $\delta > 0$ the all-one lower-diagonal matrix A defines an LP-type system $\mathcal{L}_A$ on 2δ elements, of combinatorial dimension δ, such that the expected number of basis computations in Algorithm* RE-LPtype*, when applied to the problem is at least*

$$\frac{\delta^2}{4(H_{\delta+1}-1)} \geq \frac{\delta^2}{4\ln(\delta+1)},$$

for some initial basis.

Recall that from Lemma 6.11 we get an upper bound of

$$\frac{\delta(\delta+1)}{2}$$

for the expected number of basis computations, for all problems $\mathcal{L}_A$ and any initial basis. Thus, there is still a logarithmic gap between the lower and the upper bound.

6.5 The Klee-Minty Cube

In the previous section we have developed a lower bound for the number of basis computations performed by the abstract algorithm RE-LPtype on an abstract LP-type problem $\mathcal{L}_A$. In this section we show that this has actually an interpretation as a lower bound on the number of pivot steps performed by the 'real' Random-Edge-Simplex Algorithm 2.8 on a 'real' linear program whose feasible region is the so-called *Klee-Minty cube*, a deformed unit cube in $\mathbb{R}^d$.

Fix $0 < \varepsilon < 1/2$. For any $d \geq 1$, the d-dimensional Klee-Minty LP is the following (bounded, feasible and nondegenerate) linear program in d variables and $2d$ constraints.

$$\begin{array}{lrcccl} \text{minimize } x_d & & & & & \\ \text{subject to} & 0 & \leq & x_1 & \leq & 1, \\ & \varepsilon x_{i-1} & \leq & x_i & \leq & 1-\varepsilon x_{i-1}, \quad 2 \leq i \leq d. \end{array} \tag{6.20}$$

The feasible region of this LP is called the d-dimensional *Klee-Minty cube*. Figure 6.1 shows this 'cube' in 3 dimensions, for $\varepsilon = 1/3$. In the limit $\varepsilon \to 0$, the Klee-Minty cube approaches the regular unit cube. Since $\varepsilon < 1/2$, we may set any of the variables to its lower resp. upper bound, independently in any of the d inequalities of (6.20). The 2^d choices specify the vertices of the cube, and we can encode every vertex v by a 0/1-vector $V = (V_1, \ldots, V_d) \in GF(2)^d$, with $V_i = 1$ if and only if the value of variable x_i is equal to its upper bound. For $d = 3$, Figure 6.1 shows the vertex-vector assignment that we get.

In order to analyze the actions taken by the Random-Edge-Simplex algorithm on the LP (6.20), we do not intend to argue on the level of basic feasible solutions and tableaus again. Instead, let us recall the geometric interpretation of the simplex method, and in particular, of a pivot step, as given in Subsection 1.2.3 of Chapter 1. In this interpretation, the simplex method traces a path of vertices in the feasible region, starting with some initial vertex. At any point in time it 'sits' at a vertex and performs a pivot step. In case of Random-Edge-Simplex, this step consists of

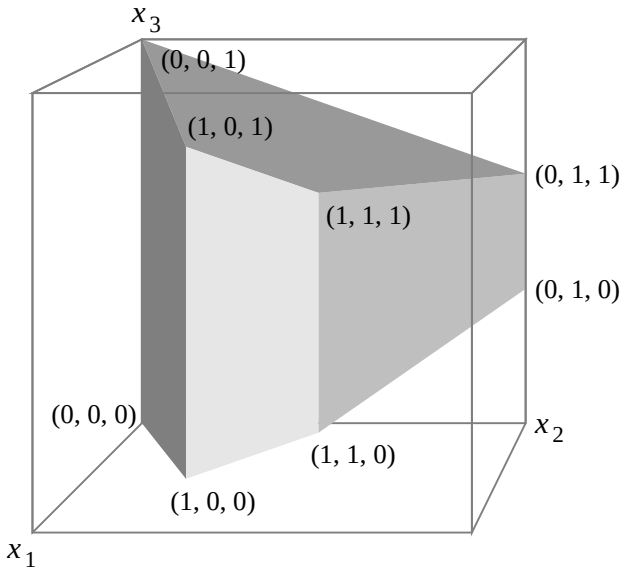


Figure 6.1: Klee-Minty cube for $d = 3, \varepsilon = 1/3$

- choosing a random out of the incident edges along which the objective function improves, and
- following this edge to its other incident vertex.

If no improving edge exists, the current vertex is reported as optimal. Under boundedness and nondegeneracy, this geometric description exactly matches the algebraic description involving basic feasible solutions and tableaus.

In case of the Klee-Minty cube, an edge connects two vertices v, v' if and only if the associated 0/1-vectors V, V' differ in exactly one entry, and the x_d-coordinates (heights) of v and v' determine in which direction the edge is an improving edge. The following lemma gives a combinatorial characterization of the edge orientations, i.e. given two adjacent vertices v, v', we can decide which one has smaller x_d-coordinate by just looking at the corresponding 0/1-vectors V, V'.

Lemma 6.23 *Let v, v' be two adjacent vertices of the d-dimensional Klee-Minty cube, of height $\tilde{x}_d$ resp. $\tilde{x}'_d$, with associated 0/1-vectors*

$$\begin{aligned} V &= (V_1, \ldots, V_i, \ldots, V_d), \\ V' &= (V_1, \ldots, 1 - V_i, \ldots, V_d). \end{aligned}$$

Then $\tilde{x}'_d < \tilde{x}_d$ if and only if $\sum_{k=i}^{d} V_k = 1$, summation being over $GF(2)$.

Proof. If every variable either assumes its upper or its lower bound in (6.20), then x_d is a linear expression in x_i, where the sign of x_i in this expression is given by

$$(-1)^{\sum_{k=i+1}^{d} V_k}.$$

Consequently, x_d gets smaller by going from v to v' if and only if

- x_i gets smaller (i.e. $V_i = 1$) and $\sum_{k=i+1}^{d} V_k = 0$, or
- x_i gets larger (i.e. $V_i = 0$) and $\sum_{k=i+1}^{d} V_k = 1$,

and this is equivalent to $\sum_{k=i}^{d} V_k = 1$. ◻

With this lemma, we can reinterpret the RANDOM-EDGE-SIMPLEX algorithm on the Klee-Minty cube as a combinatorial game on 0/1-vectors, without referring to the geometry any longer.

> Given any vector $V = (V_1, \ldots, V_d)$, choose at random a position i such that $\sum_{k=i}^{d} V_k = 1$ (random improving edge) and flip it (follow the edge to its other incident vertex). Repeat until $V = 0$.

Let us compare this with the actions taken by Game 6.7 RE-FLIP$_A$, with A equal to the all-one lower-diagonal matrix (6.13).

> Given any vector $W = (W_1, \ldots, W_d)$, choose at random a position i such that $W_i = 1$ and flip it, together with all positions to the right of i. Repeat until $W = 0$.

The point is now that both flipping processes are equivalent up to a coordinate transformation. If we define

$$W_i := \sum_{k=d-i+1}^{d} V_k,$$

for all i, the same game is played in two different coordinate systems: we obviously have $\sum_{k=i}^{d} V_k = 1$ if and only if $W_{d-i+1} = 1$. Moreover, flipping V_i exactly corresponds to flipping $W_{d-i+1}, \ldots, W_d$. Consequently, the expected number of pivot steps performed by RANDOM-EDGE-SIMPLEX (when started on a random vertex of the d-dimensional Klee-Minty cube) equals the expected number of rounds in game RE-FLIP$_A$ (when started on a random vector of $GF(2)^d$). This number of rounds has been bounded from below in Theorem 6.21, and the previous discussion establishes the same bound for RANDOM-EDGE-SIMPLEX.

Theorem 6.24 *When started on a random vertex of the d-dimensional Klee-Minty cube, the expected number of pivot steps performed by Algorithm 2.8* RANDOM-EDGE-SIMPLEX *is at least*

$$\frac{d^2}{4\ln(d+1)}.$$

6.6 Discussion

We have proved a nearly quadratic lower bound on the number of pivot steps in Algorithm Random-Edge-Simplex. This result appeared first in [21], here we have derived it from the more general setting of Algorithm RE-LPtype and Matoušek's LP-type systems $\mathcal{L}_A$ which we have shown to give a subexponential lower bound for Algorithm RF-LPtype.

It was Exercise 8.10* in the book by Papadimitriou and Steiglitz [40, p. 188] that first raised the question about the performance of Random-Edge-Simplex on the Klee-Minty cube. As already mentioned in Chapter 1, the Klee-Minty linear program (and variants of it) serve as worst case examples for many deterministic simplex algorithms, leading to exponential behavior in the dimension d. These LPs are therefore natural 'test problems' for randomized variants. While an easy upper bound of expected $O(d^2)$ pivots for Random-Edge-Simplex on the Klee-Minty cube follows from Lemma 6.11 (this bound has explicitly been stated first by Kelly [29]), no other bounds were known. In the above mentioned exercise, a wrong upper bound of $O(d)$ was claimed, for all start vertices. Motivated by computational results for small cases, Kelly [29] conjectured an upper bound of $O(d \log^2 d)$. Our nearly quadratic lower bound disproves these bounds, but is interesting beyond that, in the following context. As already mentioned in the introduction to this thesis, it has first been shown by Borgwardt (followed by others) that a *deterministic* simplex variant (namely the *shadow vertex* algorithm) is polynomial on a *random* input [9]. Here we have an inverse setting; we study a *randomized* simplex variant on a *fixed* input, and it is not at all clear whether this setting leads to polynomial bounds. Even worse, to the author's knowledge, not even superquadratic lower bound are known for any reasonable randomized simplex variant, and we have only just established a near quadratic lower bound. Incidentally, the best bounds known nowadays for the average complexity (i.e. for random input) are quadratic [2]. A superquadratic lower bound for Random-Edge-Simplex therefore would indicate that internal randomization (in the algorithm) is weaker than external randomization (of the input). Still, it is not impossible that Random-Edge-Simplex or Random-Facet-Simplex turn out to be polynomial on all linear programs in the end.

Chapter 7

Appendix

To make this thesis mostly self-contained, we collect in this Appendix some terminology, facts and bounds that appear in the other chapters. These are either referenced at least twice, or they are too specific at the point where they are referenced.

If the material is standard, we include a reference, otherwise we give a proof.

Terminology. The following lists some terminology that might not be completely standardized.

x_G	subvector of x collecting the entries with subscripts in set G
A_G	submatrix of A collecting the columns with subscripts in set G
$\tilde{x}$	vector of numerical values assigned to the vector variable x
E	unit matrix; dimension clear from the context
$\mathbf{e}_\ell$	ℓ-th unit vector, dimension clear from the context
0	zero vector; dimension clear from the context
M^T	transpose of matrix M
${}_TM$	co-transpose of M, obtained by rotating the transpose by 180°
$[m]$	$\{1,\ldots,m\}$, $m \geq 0$ an integer
∇f	gradient of multivariate function f
$\frac{\partial}{\partial t}f$ or f'	derivative of univariate function f
$\triangle$	symmetric set difference
$GF(2)$	two-element field $(\{0,1\},+,\times)$ with arithmetic modulo 2

The mean value theorem. If $f : I\!R \mapsto I\!R$ is a function which is differentiable in an interval $[a,b]$, then there exists a value $\xi \in [a,b]$ such that

$$\frac{f(b)-f(a)}{b-a} = f'(\xi), \tag{7.1}$$

and if f' is monotone increasing (decreasing), then the values $f'(a)$ resp. $f'(b)$ establish lower (upper) resp. upper (lower) bounds on the left-hand side's value, equivalently on $f(b)-f(a)$.

Stirling's approximation. [13, p. 35] For all natural numbers n,

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \le n! \le \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n+\frac{1}{12n}}. \tag{7.2}$$

The L_1-L_2-inequality. For any real numbers $r_1, \ldots, r_n$,

$$\sum_{\ell=1}^{n} |r_\ell| \le \sqrt{n}\sqrt{\sum_{\ell=1}^{n} r_\ell^2}. \tag{7.3}$$

Proof. After taking squares we equivalently need to prove

$$\left(\sum_{\ell=1}^{n} |r_\ell|\right)^2 \le n \sum_{\ell=1}^{n} r_\ell^2,$$

and this follows from $2|r_k||r_\ell| \le r_k^2 + r_\ell^2$. □

A tail estimate for the exponential function. For all real numbers $r \ge 0$,

$$\sum_{\ell=\lceil er\rceil+1}^{\infty} \frac{r^\ell}{\ell!} \le r. \tag{7.4}$$

Proof. For $r = 0$, equality holds. If $0 < r \le 1$ we get

$$\sum_{\ell=\lceil er\rceil+1}^{\infty} \frac{r^\ell}{\ell!} \le \sum_{\ell=2}^{\infty} \frac{r^\ell}{\ell!} \le \sum_{\ell=2}^{\infty} \frac{r^2}{\ell!} = r^2(e-2) \le r,$$

which holds up to $r \le 1/(e-2) \approx 1.392$. If $r > 1$, a weakening of Stirling's approximation (7.2) gives

$$\ell! \ge 2\left(\frac{\ell}{e}\right)^\ell,$$

for all ℓ. Let $q := \lceil er\rceil + 1$. Then

$$\begin{aligned}
\sum_{\ell=q}^{\infty} \frac{r^\ell}{\ell!} &\le \frac{1}{2}\sum_{\ell=q}^{\infty}\left(\frac{er}{\ell}\right)^\ell \le \frac{1}{2}\sum_{\ell=q}^{\infty}\left(\frac{er}{q}\right)^\ell = \frac{q/2}{q-er}\left(\frac{er}{q}\right)^q \le \frac{q/2}{q-\lceil er\rceil}\left(\frac{\lceil er\rceil}{q}\right)^q \\
&= \frac{1}{2}(\lceil er\rceil+1)\left(1-\frac{1}{\lceil er\rceil+1}\right)^{\lceil er\rceil+1} \\
&\le \frac{1}{2e}(\lceil er\rceil+1) \le \frac{1}{2e}(er+2) \le r,
\end{aligned}$$

which is valid for $r \ge 2/e \approx .735$.

A lower bound on a special sum. For any natural number k such that $k \geq 2e\sqrt{k/\alpha}$,

$$f(k) := \sum_{\ell=0}^{k} \frac{(1/\alpha)^\ell}{\ell!}\binom{k}{\ell} \geq \frac{e^{-e\sqrt{2}/\alpha}}{4(k+1)} \exp\left(2\sqrt{k/\alpha}\right), \tag{7.5}$$

where $\alpha > 0$ is some constant.

Proof.

$$\begin{aligned} f(k) &= \sum_{\ell=0}^{k} \frac{(1/\alpha)^\ell}{\ell!^2} k(k-1)\cdots(k-\ell+1) \\ &\geq \sum_{\ell=0}^{k} \frac{(1/\alpha)^\ell (k-\ell+1)^\ell}{\ell!^2} = \sum_{\ell=0}^{k} \left(\frac{\sqrt{(k-\ell+1)/\alpha}^{\,\ell}}{\ell!}\right)^2 \\ &\geq \frac{1}{k+1}\left(\sum_{\ell=0}^{k} \frac{\sqrt{(k-\ell+1)/\alpha}^{\,\ell}}{\ell!}\right)^2 =: \frac{1}{k+1} h(k)^2, \end{aligned} \tag{7.6}$$

where the last inequality is the L_1-L_2-inequality (7.3). Let $q := e\sqrt{k/\alpha}$. Because of $k \geq 2q > q$ we further obtain

$$\begin{aligned} h(k) &\geq \sum_{\ell=0}^{\lceil q \rceil} \frac{\sqrt{(k-\ell+1)/\alpha}^{\,\ell}}{\ell!} \geq \sum_{\ell=0}^{\lceil q \rceil} \frac{\sqrt{(k-q)/\alpha}^{\,\ell}}{\ell!} \\ &= \exp\left(\sqrt{(k-q)/\alpha}\right) - \sum_{\ell=\lceil q \rceil+1}^{\infty} \frac{\sqrt{(k-q)/\alpha}^{\,\ell}}{\ell!} \\ &\geq \exp\left(\sqrt{(k-q)/\alpha}\right) - \sum_{\ell=\left\lceil e\sqrt{(k-q)/\alpha}\right\rceil+1}^{\infty} \frac{\sqrt{(k-q)/\alpha}^{\,\ell}}{\ell!} \\ &\geq \exp\left(\sqrt{(k-q)/\alpha}\right) - \sqrt{(k-q)/\alpha}, \end{aligned}$$

by the tail estimate (7.4). Plugging this into (7.6) – observing that $e^x - x \geq e^x/2$, for all x – gives

$$f(k) \geq \frac{1}{4(k+1)} \exp\left(2\sqrt{(k-q)/\alpha}\right). \tag{7.7}$$

To estimate the exponent, we use the mean value theorem (7.1), giving

$$\frac{2\sqrt{k/\alpha} - 2\sqrt{(k-q)/\alpha}}{q} \leq \frac{1}{\alpha\sqrt{(k-q)/\alpha}},$$

hence

$$2\sqrt{(k-q)/\alpha} \geq 2\sqrt{k/\alpha} - \frac{q}{\alpha\sqrt{(k-q)/\alpha}} = 2\sqrt{k/\alpha} - \underbrace{\frac{e\sqrt{k}}{\alpha\sqrt{k-e\sqrt{k/\alpha}}}}_{g(k)}. \tag{7.8}$$

Because of

$$\frac{\partial}{\partial k} g(k) = -\frac{e^2}{4\alpha^{3/2}\sqrt{k-e\sqrt{k/\alpha}}^{\,3}} < 0,$$

g is monotone decreasing in k, and with $k \geq 2e\sqrt{k/\alpha}$ it follows that

$$g(k) \leq g(2e\sqrt{k/\alpha}) = \frac{e\sqrt{2e\sqrt{k/\alpha}}}{\alpha\sqrt{e\sqrt{k/\alpha}}} = \frac{e\sqrt{2}}{\alpha}.$$

In (7.8), this leads to a bound of

$$2\sqrt{(k-q)/\alpha} \geq 2\sqrt{k/\alpha} - \frac{e\sqrt{2}}{\alpha},$$

and plugged into (7.7), the required bound of

$$f(k) \geq \frac{e^{-e\sqrt{2}/\alpha}}{4(k+1)} \exp\left(2\sqrt{k/\alpha}\right)$$

follows. □

Chebyshev's summation inequalities. [24, p. 38] If $a_1 \leq a_2 \cdots \leq a_n$ and $b_1 \geq b_2 \cdots \geq b_n$, then

$$\left(\sum_{k=1}^{n} a_k\right)\left(\sum_{k=1}^{n} b_k\right) \geq n \sum_{k=1}^{n} a_k b_k. \tag{7.9}$$

If $a_1 \leq a_2 \cdots \leq a_n$ and $b_1 \leq b_2 \cdots \leq b_n$, then

$$\left(\sum_{k=1}^{n} a_k\right)\left(\sum_{k=1}^{n} b_k\right) \leq n \sum_{k=1}^{n} a_k b_k. \tag{7.10}$$

Note that (7.10) implies the L_1-L_2-inequality (7.3) if $a_k = b_k$, for all k.

Bibliography

[1] I. Adler. *Abstract Polytopes*. PhD thesis, Dept. Operations Research, Stanford University, 1971.

[2] I. Adler and N. Megiddo. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *J. ACM*, 32:871–895, 1985.

[3] I. Adler and R. Saigal. Long monotone paths in abstract polytopes. *Math. Operations Research*, 1(1):89–95, 1976.

[4] N. Amenta. *Helly Theorems and Generalized Linear Programming*. PhD thesis, University of Berkeley, California, 1993.

[5] D. Avis and V. Chvátal. Notes on Bland's pivoting rule. *Math. Programming Study*, 8:24–34, 1978.

[6] B. Bixby. Personal communication, 1995.

[7] J. Blömer. Computing sums of radicals in polynomial time. In *Proc. 32nd annu. IEEE Symp. on Foundations of Computer Science*, pages 670–677, 1991.

[8] J. Blömer. How to denest Ramanujan's nested radicals. In *Proc. 33nd annu. IEEE Symp. on Foundations of Computer Science*, pages 447–456, 1992.

[9] K. H. Borgwardt. *The Simplex Method. A Probabilistic Analysis*, volume 1 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin Heidelberg, 1987.

[10] V. Chvátal. *Linear Programming*. W. H. Freeman, New York, NY, 1983.

[11] K. L. Clarkson. Linear programming in $O(n3^{d^2})$ time. *Inform. Process. Lett.*, 22:21–24, 1986.

[12] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 452–456, 1988.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA., 1990.

[14] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[15] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean one-center problem. *SIAM J. Comput.*, 15:725–738, 1986.

[16] M. E. Dyer and A. M. Frieze. A randomized algorithm for fixed-dimensional linear programming. *Math. Programming*, 44:203–212, 1989.

[17] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete Comput. Geom.*, 1:25–44, 1986.

[18] B. Gärtner. A subexponential algorithm for abstract optimization problems. In *Proc. 33rd annu. IEEE Symp. on Foundations of Computer Science*, pages 464–472, 1992.

[19] B. Gärtner and E. Welzl. Vapnik-Chervonenkis dimension and (pseudo-)hyperplane arrangements. *Discrete Comput. Geom.*, 12:399–432, 1994.

[20] B. Gärtner and E. Welzl. Linear programming – randomization and abstract frameworks. In *Proc. 13th annu. Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 1046 of *Lecture Notes in Computer Science*, pages 669–687. Springer-Verlag, 1996.

[21] B. Gärtner and G. M. Ziegler. Randomized simplex algorithms on Klee-Minty cubes. In *Proc. 35th annu. IEEE Symp. on Foundations of Computer Science*, pages 502–510, 1994.

[22] D. Goldfarb and W. Y. Sit. Worst case behavior of the steepest edge simplex method. *Discr. Applied Math.*, 1:277–285, 1979.

[23] M. Goldwasser. A survey of linear programming in randomized subexponential time. *ACM-SIGACT News*, 26(2):96–104, 1995.

[24] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, 1989.

[25] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin Heidelberg, 1988.

[26] R. G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discr. Math.*, 4:367–377, 1973.

[27] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24th annu. ACM Symp. on Theory of Computing.*, pages 475–482, 1992.

[28] G. Kalai and D. J. Kleitman. A quasi-polynomial bound for the diameter of graphs of polyhedra. *Bulletin Amer. Math. Soc.*, 26:315–316, 1992.

[29] D. G. Kelly. Some results on random linear programs. *Methods of Operations Research*, 40:351–355, 1981.

[30] L. G. Khachiyan. Polynomial algorithms in linear programming. *U.S.S.R. Comput. Math. and Math. Phys*, 20:53–72, 1980.

[31] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972.

[32] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117(1):151–155, 1995.

[33] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. In *Proc. 8th annu. ACM Symp. on Comput. Geom.*, pages 1–8, 1992.

[34] J. Matoušek. Lower bound for a subexponential optimization algorithm. *Random Structures & Algorithms*, 5(4):591–607, 1994.

[35] P. McMullen. The maximal number of faces of a convex polytope. *Mathematica*, 17:179–184, 1970.

[36] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31:114–127, 1984.

[37] F. Meyer auf der Heide. Unpublished notes, 1994.

[38] R. Motwani and R. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[39] K. Murty. *Linear and Combinatorial Programming*. John Wiley & Sons, New York, 1976.

[40] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[41] A. L. Peressini, F. E. Sullivan, and J. J. Uhl. *The Mathematics of Nonlinear Programming*. Undergraduate Texts in Mathematics. Springer-Verlag, 1988.

[42] V. T. Rajan. Optimality of the Delaunay triangulation in R^d. In *Proc. 7th annu. ACM Symp. on Comput. Geom.*, pages 357–363, 1991.

[43] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6:423–434, 1991.

[44] K. Sekitani and Y. Yamamoto. A recursive algorithm for finding the minimum norm point in a polytope and a pair of closest points in two polytopes. *Math. Programming*, 61(2):233–249, 1993.

[45] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Symp. on Theoretical Aspects of Computer Science.*, volume 577 of *Lecture Notes in Computer Science*, pages 569–579. Springer-Verlag, 1992.

[46] E. H. Spanier. *Algebraic Topology.* McGraw-Hill Book Company, New York, 1966.

[47] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes in Computer Science*, pages 359–370. Springer-Verlag, 1991.

[48] P. Wolfe. Finding the nearest point in a polytope. *Math. Programming*, 11:128–149, 1976.

[49] N. Zadeh. What is the worst case behavior of the simplex algorithm? Technical Report 27, Dept. Operations Research, Stanford University, 1980.

[50] G. M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics.* Springer-Verlag, 1994.