On Linear Programming, Integer Programming and Cutting Planes

A Thesis
Presented to
The Academic Faculty

by

Daniel G. Espinoza

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Industrial and Systems Engineering

School of Industrial and Systems Engineering
Georgia Institute of Technology
May 2006

On Linear Programming, Integer Programming and Cutting Planes

Approved by:

William J. Cook, Adviser
School of Industrial and Systems Enginering
Georgia Institute of Technology

George L. Nemhauser
School of Industrial and Systems Enginering
Georgia Institute of Technology

Ellis Johnson
School of Industrial and Systems Enginering
Georgia Institute of Technology

Martin Savelsbergh
School of Industrial and Systems Enginering
Georgia Institute of Technology

Zonghao Gu
Ilog Inc.

Date Approved: March 29 2006

ACKNOWLEDGEMENTS

Special thanks to John Boyer, for allowing us to use his excellent implementation of his planarity test algorithm for the implementation of the Domino-Parity separation algorithm. The author was supported in part by ONR Grant N00014-03-1-0040 and by NSF Grant DMI-0245609.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In this thesis we address three related topics in the field of Operations Research.

Firstly, we discuss a problem and limitations of most common solvers for linear programming, namelly, precision. We then present a solver that generates rational optimal solutions to linear programming problems by solving a succession of (increasingly more precise) floating point approximations of the original rational problem until the rational optimality conditions are achieved. This method is shown to be (on average) only 20% slower than the common pure floating point approach, while returning true optimal solutions to the problems.

Secondly, we present an extension of the Local Cut procedure introduced by Applegate et al. 2001, for the Symmetric Traveling Salesman Problem (TSP), to the general setting of MIP problems. This extension also proves finiteness of the separation, facet and tilting procedures in the general MIP setting, and provides conditions under which the separation procedure is guaranteed to generate cuts that separate the current fractional solution from the convex hull of the mixed-integer polyhedron. We then move on to explore some configurations for local cuts, including extensive testing on the instances from MIPLIB. These results show that this technique may be useful in general MIP problems, while the experience of Applegate et al., shows that the ideas can be successfully applied to structured problems as well.

Thirdly, we present extensive computational experiments on the TSP and Domino Parity inequalities as introduced by Letchford, 2000. This work includes a safe-shrinking theorem for domino parity inequalities, heuristics to apply the planar separation algorithm introduced by Letchford to instances where the planarity requirement does not hold, and several practical speed-ups. Our computational experience shows that this class of inequalities effectively improves the lower bounds from the best relaxations obtained with `Concorde`, which is one of the state of the art solvers for the TSP. As part of these experience, we solved to optimality the (up to now) largest TSP instance, and one of the open problems from TSPLIB, they have 18,512 and 33,810 cities respectively.

CHAPTER 1

Linear Programming, The Simplex Algorithm, and Exact Solutions

1.1 Introduction

Linear programming (LP) problems are optimization problems in which the objective function and all the constraints are linear. Linear programming is an important field of optimization for several reasons. G. B. Dantzig [31] writes:

> "These problems occur in everyday life; they run the gamut from some very simple situations that confront an individual to those connected with the national economy as a whole. Typically, these problems involve a complex of different activities in which one wishes to know which activities to emphasize in order to carry out desired objectives under known limitations."

Nowadays, LP is an extensively used tool, both in industry and in academic research, with an extensive literature devoted to it. From an historical point of view, linear programming has inspired many of the central concepts of optimization theory, such as duality, decomposition, and the importance of convexity and its generalizations.

One of the most well known algorithms for linear programming is the *simplex algorithm*, introduced (as we know it today) by Dantzig [31, 32] in 1947. According to Schrijver [94], one of the earliest references of a simplex-like algorithm is due to Fourier [41] in 1826, where he described a rudimentary version of the simplex algorithm for the problem

$$\begin{aligned} \min \quad & z \\ s.t. \quad & z \geq |a_i x + b_i y + c_i| \quad i \in \{1, \ldots, m\}. \end{aligned}$$

According to Fourier's notes, his description is enough to extend his algorithm to the n dimensional case. Schrijver mentions yet another early reference for a simplex-like method, this one is due to de la Vallée [34] in 1911, where a simplex-like method is proposed to solve

$$\min \|Ax - b\|_\infty.$$

This is known as the Chebyshev approximation problem. de la Vallée's method is for A of general dimension, but it is for the particular type of objective of the Chebyshev approximation problem, and also makes the assumption that all sets of n rows of A are linearly independent.

The simplex algorithm solves LP problems by constructing an admissible solution at a vertex of the polyhedron, and then walking along edges of the polyhedron to vertices with successively better values of the objective function until the optimum is reached, or unboundedness is detected. Although this algorithm is quite efficient in practice, and can be guaranteed to find the global optimum if certain precautions against *cycling* are taken, it has poor worst-case behavior. In fact, for most *pivot rules*, it is possible to construct a linear programming problem for which the simplex method takes a number of steps exponential in the problem size. The existence of a polynomial-time version of the simplex algorithm remains an open problem to this day.

For some time it was not known whether the linear programming problem was NP-complete or solvable in polynomial time. The first worst-case polynomial-time algorithm for the linear programming problem was proposed by L. Khachiyan [65], however, the practical performance of Khachiyan's algorithm is very disappointing. Later, in 1984, N. Karmarkar [64] proposed the projective method algorithm. This was the first algorithm performing well both in theory and in practice. Its worst-case complexity is polynomial and experiments on practical problems showed that it is reasonably efficient compared to the simplex method (for more details in LP algorithmic history see Bixby [15]).

Modern versions of Karmarkar's algorithm are now the most effective method to solve large linear programming problems. However, there is one major advantage that the simplex algorithm still has over its interior point cousins; namely, when re-optimizing a slightly modified problem, the simplex algorithm can use the solution to the previous problem to compute the solution to the new one, and if the modifications are *small*, the amount of extra work is in fact very little. While some research has been done to improve these capabilities for interior point algorithms (and some improvements have been done), they remain far inferior to those of the simplex algorithm.

One prominent area where several iterations of similar LP problems are solved is in (Mixed) Integer Programming or MIP. The most successful approach to date to exactly solve MIP problems is a mixture of the branch-and-bound algorithm and the cutting-plane algorithm (also called branch-and-cut algorithms). This algorithm starts with an LP relaxation of the real MIP problem, and successively strengthens this relaxation by adding valid inequalities for the MIP problem, or branches by creating two LP relaxations which differ in only a few constraints from the original LP relaxation. In this setting the simplex algorithm remains of crucial importance.

A second important example, where this advantage plays a role, is when using Column Generation to solve really large LP problems. In this setting we add (and delete) variables from the current LP description until we find an optimal solution. Applications that use this approach range from the airline crew-scheduling problem, to the cutting-stock problem.

In this chapter we address one special question regarding LP and the simplex algorithm: How can we obtain *exact* solutions for a given LP problem, while still maintaining the re-optimization properties of the simplex algorithm?.

The rest of this chapter is organized as follows. In Section 1.2 we explain the relevance and context for this question, as well as some previously known results. In Section 1.3 we detail our implementation. In Section 1.4 we present some hard LP instances (from the point of view of optimality) and numerical results for a range of instances including the NETLIB and MIPLIB problems. Finally, in Section 1.5 we discuss some of the questions that were not answered in this research and also some related topics.

1.2 Why Exact Solutions?

Before answering the question of this section, we must explain what we mean by a solution to a LP problem. Since all but the most trivial problems need a computer code to be solved, we must also consider what are the limitations of computer implementations.

While we can represent any rational number in a computer without any error (under the memory constraints), the representation of choice for rational numbers in most LP solvers are floating point numbers. We start by defining a floating point number and a small discussion on its limitations. We then give some arguments as to why floating point representation is used in most LP solvers. Then we define what is an exact solution to a LP problem and give the context of when an exact solution to a LP problem is needed. We finish this section by providing some previously known results.

1.2.1 Binary Floating Point Representation

Floating-point representation basically represents rationals in scientific notation. Scientific notation represents numbers as a base number and an exponent. For example, 123.456 could be represented as 1.23456×10^2 . In hexadecimal, the number 123.abc might be represented as $1.23abc \times 16^2$.

Floating-point solves a number of representation problems. Fixed-point has a fixed window of representation, which limits it from representing very large or very small numbers. Also, fixed-point is prone to a loss of precision when two large numbers are divided.

Floating-point, on the other hand, employs a sort of *sliding window* of precision appropriate to the scale of the number. This allows it to represent numbers from 1,000,000,000,000 to 0.0000000000000001 with ease.

Storage Layout IEEE[1] floating point numbers have three basic components: the sign, the exponent, and the mantissa. The mantissa is composed of the fraction and an implicit leading digit (explained below). The exponent base (2) is implicit and need not be stored.

Table 1.1 shows the layout for single (32-bit) and double (64-bit) precision floating-point

[1] The IEEE Standard floating point is the most common representation used today for real numbers on computers, including Intel-based PC's, Macintoshes, and most Unix platforms. (See [57] for more detail).

values.

Table 1.1: Storage layout of IEEE-754 compliant floating point numbers, bit ranges are in square brackets.

	Sign	Exponent	Fraction	Bias
Single Precision	1 [31]	8 [30-23]	23 [22-00]	127
Double Precision	1 [63]	11 [62-52]	52 [51-00]	1023

The Sign Bit The sign bit is as simple as it gets. 0 denotes a positive number; 1 denotes a negative number. Flipping the value of this bit flips the sign of the number.

The Exponent The exponent field needs to represent both positive and negative exponents. To do this, a *bias* is added to the actual exponent in order to get the stored exponent. For IEEE single-precision floats, this value is 127. Thus, an exponent of zero means that 127 is stored in the exponent field. A stored value of 200 indicates an exponent of 200-127, or 73. For reasons discussed later, exponents of -127 (all 0s) and +128 (all 1s) are reserved for special numbers.

For double precision, the exponent field is 11 bits, and has a bias of 1023.

The Mantissa The mantissa, also known as the significand, represents the precision bits of the number. It is composed of an implicit leading bit and the fraction bits.

To find out the value of the implicit leading bit, consider that any number can be expressed in scientific notation in many different ways. For example, the number five can be represented as any of these:

- 5.00×10^0
- 0.05×10^2
- 5000×10^-3

In order to maximize the quantity of representable numbers, floating-point numbers are typically stored in normalized form. This basically puts the radix point after the first non-zero digit. In normalized form, five is represented as 5.0×10^0.

A nice little optimization is available to us in base two, since the only possible non-zero digit is 1. Thus, we can just assume a leading digit of 1, and do not need to represent it explicitly. As a result, the mantissa has effectively 24 bits of resolution, by way of 23 fraction bits for single precision floating point.

Putting it All Together To sum up:

1. The sign bit is 0 for positive, 1 for negative.
2. The exponent's base is two.
3. The exponent field contains 127 plus the true exponent for single-precision, or 1023 plus the true exponent for double precision.
4. The first bit of the mantissa is typically assumed to be $1.f$, where f is the field of fraction bits.

Ranges of Floating-Point Numbers Let us consider single-precision floats. Note that we are taking essentially a 32-bit number and re-arranging the fields to cover a much broader range. Something has to give, and it is precision. For example, regular 32-bit integers, with all precision centered around zero, can precisely store integers with 32-bits of resolution. Single-precision floating-point, on the other hand, is unable to match this resolution with its 24 bits. It does, however, approximate this value by effectively truncating from the lower end. For example:

$$\begin{array}{crrrrl} & 11110000 & 11001100 & 10101010 & 00001111 & \text{32-bit integer} \\ = & +1.1110000 & 11001100 & 10101010 & \times 2^{31} & \text{Single-Precision Float} \\ = & 11110000 & 11001100 & 10101010 & 00000000 & \text{Corresponding Value} \end{array}$$

This approximates the 32-bit value, but does not yield an exact representation. On the other hand, besides the ability to represent fractional components (which integers lack completely), the floating-point value can represent numbers around 2^{127}, compared to 32-bit integers maximum value around 2^{32}.

The range of positive floating point numbers can be split into normalized numbers (which preserve the full precision of the mantissa), and *denormalized* numbers (discussed later) which use only a portion of the fraction's precision. The actual values can be seen in Table 1.2.

Table 1.2: Ranges of representable floating-point numbers

	Single Precision	Double Precision
Denormalized	2^{-149} to $(1-2^{-23})\times 2^{-126}$	2^{-1074} to $(1-2^{-52})\times 2^{-1022}$
Normalized	2^{-126} to $(2-2^{-23})\times 2^{127}$	2^{-1022} to $(2-2^{-52})\times 2^{1023}$
Approximate Decimal	$10^{-44.85}$ to $10^{38.53}$	$10^{-323.3}$ to $10^{308.3}$

Since the sign of floating point numbers is given by a special leading bit, the range for negative numbers is given by the negation of the above values.

There are five distinct numerical ranges that single-precision floating-point numbers are not able to represent:

1. Negative numbers less than $-(2-2^{-23})\times 2^{127}$ (negative overflow).
2. Negative numbers greater than -2^{-149} (negative underflow).
3. Zero
4. Positive numbers less than 2^{-149} (positive underflow).
5. Positive numbers greater than $(2-2^{-23})\times 2^{127}$ (positive overflow).

Overflow means that values have grown too large for the representation, much in the same way that you can overflow integers. Underflow denotes a loss of precision, which is guaranteed to be closely approximated by zero.

Note that the extreme values occur (regardless of sign) when the exponent is at the maximum value for finite numbers (2^{127} for single-precision, 2^{1023} for double), and the mantissa is filled with 1s (including the normalizing 1 bit).

Special Values IEEE reserves exponent field values of all 0s and all 1s to denote special values in the floating-point scheme.

Zero As mentioned above, zero is not directly representable in the straight format, due to the assumption of a leading 1 (we would need to specify a true zero mantissa to yield a value of zero). Zero is a special value denoted with an exponent field of zero and a fraction field of zero. Note that -0 and $+0$ are distinct values, though they both compare as equal.

Denormalized If the exponent is all 0s, but the fraction is non-zero (else it would be interpreted as zero), then the value is a denormalized number, which does not have an assumed leading 1 before the binary point. Thus, this represents a number $(-1)^s \times 0.f \times 2^{-126}$, where s is the sign bit and f is the fraction. For double precision, denormalized numbers are of the form $(-1)^s \times 0.f \times 2^{-1022}$. From this you can interpret zero as a special type of denormalized number.

Infinity The values $+\infty$ and $-\infty$ are denoted with an exponent of all 1s and a fraction of all 0s. The sign bit distinguishes between negative infinity and positive infinity. Being able to denote infinity as a specific value is useful because it allows operations to continue past overflow situations[2].

Not A Number The value NaN (Not a Number) is used to represent a value that does not represent a real number. NaN's are represented by a bit pattern with an exponent of all 1s and a non-zero fraction.

1.2.2 The Limits and Errors of Floating Point Arithmetic

While floating-point representation effectively allow us to represent a wide range of values, there are inherent errors while using them. For example, if we represent numbers as double precision floating-points, then we obtain the representations shown in Table 1.3 for some common fractions.

Although the relative error between the desired number and the actual represented value is not more than $\varepsilon = 2^{-52} = 1/4,503,599,627,370,496 \approx 10^{-15.65}$. This minimum relative error puts a barrier on the confidence of the results from any computer code that performs its calculations using floating-point numbers. Note that the existence of this barrier is regardless of how many bits we use to store the mantissa, the number of bits used in the mantissa only changes the actual value of this barrier.

[2] Operations with infinite values are well defined in IEEE floating point, but a program may choose not to adhere to the strict standard.

Table 1.3: Common Fractions as Floating-Points. Mantissa values are written in hexadecimal.

$1/5$	$= \mathtt{1.999999999999a}\times 2^{-3}$	$1/3$	$= \mathtt{1.5555555555555}\times 2^{-2}$
$1/9$	$= \mathtt{1.c71c71c71c71c}\times 2^{-4}$	$1/7$	$= \mathtt{1.2492492492492}\times 2^{-3}$
$1/13$	$= \mathtt{1.3b13b13b13b14}\times 2^{-4}$	$1/11$	$= \mathtt{1.745d1745d1746}\times 2^{-4}$
$1/17$	$= \mathtt{1.e1e1e1e1e1e1e}\times 2^{-5}$	$1/15$	$= \mathtt{1.1111111111111}\times 2^{-4}$
$1/21$	$= \mathtt{1.8618618618618}\times 2^{-5}$	$1/19$	$= \mathtt{1.af286bca1af28}\times 2^{-5}$
$1/25$	$= \mathtt{1.47ae147ae147b}\times 2^{-5}$	$1/23$	$= \mathtt{1.642c8590b2164}\times 2^{-5}$
$1/29$	$= \mathtt{1.1a7b9611a7b96}\times 2^{-5}$	$1/27$	$= \mathtt{1.2f684bda12f68}\times 2^{-5}$

These errors in turn generate more errors once we start doing arithmetic in these numbers. Take for example an inner product between two vectors $x, y \in \mathbb{R}^n$ and assume that we are representing these number using double precision floating-point numbers, and that the relative error of each represented number is bounded by ε. Then, the absolute error of the inner product is bounded by $n\varepsilon\langle|x|, |y|\rangle$. Note that, on the other hand, the relative error is essentially unbounded. Although these values are upper bounds on the error, it is possible to derive lower bounds for some special cases, but such studies are outside the scope of the present work.

1.2.3 Why Floating Point representation is used at all?

Besides the fact that the use of floating point representation allows us to work with a wide range of valid values, there is a second argument that supports the use of floating point representation when solving LP:

> "The initial data (of LP problems) are in part subject to uncertainty, but even where a fraction is known it cannot necessarily be expressed in a fixed punching field. Even should the decimal data be near correct, by the time it has been converted to binary much has been lost. These errors should not be treated as having physical reality." P.M.J. Harris [54].

Nowadays, most state of the art LP solvers take advantage of the floating point nature of the numbers in an algorithmic way. Some of these algorithmic uses are *bound shifting*, the *Devex pivot rule* and *objective perturbation*. For details on these ideas see [15, 40, 54].

Finally, note that all commercial LP solvers are based on floating-point arithmetic, and none of them provides any mathematically well defined assurance on the quality of the solutions that they provide (although they do mention tolerances and their default values).

1.2.4 What is an Exact Solution?

Choosing the correct domain The first aspect that we must take care of is to define the right domain for the input that we need to consider. And the alternatives are clear, we could choose to work with data either on $\mathbb{R}$ or in $\mathbb{Q}$.

While LP theory works in both domains, MIP theory does not. A classical example of this is the following problem:

$$\begin{aligned} (IIP) \quad \max \quad & \sqrt{2}y - x \\ s.t. \quad & \sqrt{2}y \leq x \\ & x, y \in \mathbb{Z}^{+}. \end{aligned}$$

Note that (IIP) is both feasible, and bounded, but there is no optimal solution. A second reason to stay away from irrationals is that the common notions of complexity and polynomial algorithms breaks down if we allow irrationals. For instance, what is the input size for the number $\sqrt{2}$? These considerations suggest that we restrict our attention to rational numbers.

Now that the domain question has been settled, we start by stating a purely mathematical definition of what constitutes an exact solution, and then we make this notion precise in a computer code context.

Definition 1.1 (Exact solution). Given an LP

$$\begin{aligned} P = \quad \max \quad & c \cdot x \\ s.t. \quad & Ax \leq b \\ & l \leq x \leq u \end{aligned}$$

with $A \in \mathbb{Q}^{m \times n}$, $l, u, c \in \mathbb{Q}^{n}$ and $b \in \mathbb{Q}^{m}$, we say that $(x_o, y_o) \in \mathbb{Q}^{n+m}$ is an *exact* solution to P if x is primal feasible, y is dual feasible, and complementary slackness holds.

Note that since all data is rational, all computations can be carried out (without introducing errors) in rational arithmetic, and thus the conditions of the definitions can be computed exactly. But there remain some important questions regarding the meaning of this definition inside a computer code.

Interpreting decimal fractions The first question that needs to be answered is how do we interpret fractional coefficients in a given LP?.

If the coefficient is integer, then it is clear what the value means, but if we encounter a coefficient of say 0.3333333333333333 at some point, should we interpret the rational value to be 1/3 or to be 3333333333333333/10000000000000000 ?.

Note that the question is not just a rhetorical one. In general, given a fractional value f, we can compute integers a, b such that $|f - a/b| \leq \frac{1}{b^2}$. Such integers can be computed using the *continued fraction* method, and if we stop whenever the current approximation a_i/b_i is such that $b_i \geq 2^{27}$, then $|f - a_i/b_i| \leq 2^{-54}$, which is less than the smallest number representable by a normalized double precision floating point. The advantage of such an interpretation is that the fractions used to represent the fractional number are in general much smaller than the alternative approach, but one major drawback of this approach is that if we use such an interpretation of decimal fractions, then the problem to be solved depends on the actual implementation of the conversion routine, and thus the concept of an exact solution to the problem is not well defined for a computer code. For this reason we choose the second (more extended) interpretation of decimal fractional values, and thus uniquely defining the problem to be solved. Note that this interpretation choice does not preclude us from representing values like 1/3. To do so, we only need to scale the inequality by 3 (or by the minimum common denominator among all desired fractions) and write the inequality with integer coefficients.

Representing arbitrary numbers in a computer One pressing question remains. Although in *theory* we can represent arbitrarily large integers (and thus rationals) on a computer, is there any efficient computer implementation for arbitrary precision rationals?. Fortunately, the answer is yes. In fact, there is a wide range of possible choices that provide a callable library C interface. The following list contains just a few possibilities:

mp Multiple Precision package that comes with some Unix systems.

This library provides +, -, *, /, gcd, exponentiation, sqrt.

PARI by Henri Cohen, et al., Universite Bordeaux I, Paris, FRANCE.

Multiple precision desk calculator and library routines. Contains optimized assembly code for Motorola 68020, semi-optimized code for SPARC, and a generic C version.

Arithmetic in Global Fields (Arith) by Kevin R. Coombes, David R. Grant.

Package of routines for arbitrary precision integers or polynomials over finite fields. Includes basic +, -, *, / and a few others like gcd.

Arbitrary Precision Math Library by Lloyd Zusman, Los Gatos, CA.

C package which supports basic +, -, *, /. It also provides for radix points (i.e., non-integers).

BigNum by J. Vuillemin, INRIA, FRANCE, and others, and distributed by Digital Equipment Paris Research Lab (DECPRL).

A "portable and efficient arbitrary-precision integer" package. C code, with generic C "kernel", plus assembly "kernels" for MC680x0, Intel i960, MIPS, NS32032, Pyramid, and VAX.

Lenstra's LIP package by Arjen Lenstra, Bellcore.

Portable unsigned integer package written entirely in C. Includes +, -, *, /, exponentiation, mod, primality testing, sqrt, random number generator, and a few others.

MIRACL (Shamus Software, Dublin, Ireland).

Integer and fractional multiple precision package. MIRACL is a portable C library. Full C/C++ source code included (in-line assembly support for 80x86). C++ classes for Multiprecision Integers, Modular arithmetic, and Chinese Remainder Theorem. Implementation in C/C++ of all modern methods of integer factorization, viz Brent-pollard, p-1, p+1, Elliptic Curve, MPQS.

GNU Multiple Precision (GMP) GNU (Free Software Foundation) multiple precision package.

This library is completely written in C, with assembler implementation for many of the most critical parts for several common architectures. It provides integer, rational and multiprecision floating point numbers and is actively being developed and maintained.

For our computational purposes, we choose the **GNU Multiple Precision** library because of its availability, high performance, and its license restrictions (GNU-MP is released under the Lesser GNU license).

1.2.5 When do we need exact LP solutions?

There are several reasons why one would like to consider exact solutions to a given LP.

First, there are some cases from industry where the customers demand an exact solution for a given LP problem[3].

A second setting where getting exact solutions is of importance is when LP is used to compute theoretical bounds on different problems. LP-based bounds are widely used (see [3, 71, 92]) for many problems, but almost always authors just report the values obtained from some commercial solver without ever questioning or checking the quality of the obtained results. This is a serious weak point for any conclusions drawn from these LP-based bounds.

A third setting where inaccuracies in LP solutions are of major importance is in the realm of mixed integer programming. The first reason for this is that in many instances there are no inaccuracies whatsoever in the original data nor in the representability of such data inside a computer (all problems in TSPLIB fit this description), and thus one of the key arguments for using floating-point representation is no longer valid. A second reason is that in many cases the formulation of the problem naturally forces small values for the LP solution. For example, consider the following problem:

$$\begin{aligned} \min \quad & x_1 + x_2 + y \\ s.t. \quad & x_1 + x_2 \leq 10^6 y \\ & x_1 + x_2 \geq 10^{-3} \\ & x_1, x_2, y \geq 0 \\ & y \in \mathbb{Z}. \end{aligned} \tag{1.1}$$

Note that an optimal LP solution is $(x_1, x_2, y) = (10^{-3}, 0, 10^{-9})$, and moreover, if we set

[3] Personal communications with Zonghao Gu, September 2005.

our *tolerances* to any number above 10^{-9}, this LP solution would mistakenly be taken as an optimal IP solution, which of course is not true. The choice of 10^{-9} was not accidental, this value represents the best accuracy that most commercial LP solvers can achieve. And although Problem (1.1) is clearly artificial, this kind of structure does arise in practice quite often, and in fact, more general structures leading to the same kind of numerical issues are easy to devise.

A fourth setting arises when computing valid inequalities for MIP problems, in particular when generating MIR-Gomory cuts. Although in theory it is possible to solve any IP problem by applying successive rounds of Gomory cuts, in practice this is not done. One of the reasons is that the cuts obtained from successive rounds do cut feasible/optimal solutions from the problem. This is not due to a breakdown of the theory of Gomory cuts, but rather a consequence of the floating point representation of numbers inside LP solvers, and the unavoidable errors that this representation generates.

1.2.6 Previously known results

The subject of accurate solutions for LP problems is not new, and it has received some attention in the past. One of the earliest references on this topic is from Gärtner [44], who discusses the problem of inaccurate solutions for some LP arising from computational geometry. He implements a simplex algorithm where some operations are carried out in floating-point representation, and others in exact arithmetic. The results show that for problems with either a few rows or columns, exact solutions can be computed in competitive times with commercial software like CPLEX, but the method does not seem to work well on larger instances.

Jansson [59] address the inaccuracy problem with a completely different approach. He studies methods to provide upper and lower bounds for the LP at hand, taking in consideration problems as ill-conditioning and margin errors for coefficients in the input data. Numerical tests where carried out using MATLAB on small instances with mixed results.

In the context of MIP problems, Neumaier and Shcherbina [87] show (in some seemingly innocents MIP problems), that state-of-the-art MIP solvers (based on floating point representation), like CPLEX, fail to find the actual optimal solution to the problem. One such example is the following:

$$\begin{aligned} \min \quad & -x_{20} \\ s.t. \quad & (s+1)x_1 - x_2 \geq s-1 \\ & -sx_{i-1} + (s+1)x_i - x_{i+1} \geq -1^i(s+1) \quad \forall i = 2, \ldots, 19 \\ & -sx_{18} - (3s-1)x_{19} + 3x_{20} \geq -5s+7 \\ & 0 \leq x_i \leq 10 \quad \forall i = 1, \ldots, 13 \\ & 0 \leq x_i \leq 10^6 \quad \forall i = 14, \ldots, 20 \\ & x_i \in \mathbb{Z}. \end{aligned} \tag{1.2}$$

For this example, setting $s = 6$, CPLEX 7.1, ended with the message “integer infeasible”, however, changing the upper bound of x_{20} to 10, produced the feasible solution $x = (1, 2, 1, 2, \ldots, 1, 2)$. They used techniques of rounding and interval arithmetic, preprocessing and postprocessing to detect such ill-behaving situations and to reformulate these problems. They also go on to use their techniques to generate *safe* cuts from some templates like Gomory and MIR cuts.

An interesting approach for the problem of accuracy was taken by Dhiflaoui et al. [35]. They tried to verify the quality of the *basic* solution returned by the simplex algorithm as implemented in CPLEX and other software. What they found was that, in many cases, the basic solution was indeed the optimal solution for the LP at hand, but that the objective values were miscalculated by some small factor. Nevertheless, there where some instances where the basic solution returned was not optimal, and where a small number of extra pivots were needed to get the actual optimal solution, these extra pivots where performed entirely on exact arithmetic. This approach was also employed by T. Koch [66] in his study to find the true optimal solutions for all NETLIB problems.

1.3 Getting Exact LP Solutions

1.3.1 Selecting the Right Tools

Given the constraints that we have imposed on ourselves, i.e. keep the hot-start capabilities of the simplex algorithm, the only relevant algorithmic choice is to use the simplex algorithm. Given also the fact that we need to modify internal routines of the LP solver, we may choose to implement from scratch a simplex algorithm, or to take an existing implementation and modify its source code.

Since implementing a modern simplex algorithm, with hot-start capabilities that allows adding and deleting both columns and rows, and that is competitive with current simplex implementations, would be in itself a major undertaking, we have chosen to build upon an available implementation.

Another requirement for us was that the implementation to be selected must provide a callable library interface (so that we can use the resulting tool as a subroutine on different problems), and moreover, we restrict ourselves to implementations done in either the C or C++ programming languages.

Having said that, the choices are still numerous. Simplex implementations that provide the source code include QSopt, GLPK, CLP and SOPLEX. Unfortunately we found that the license restrictions for SOPLEX were too stringent for our purposes, and this fact made us discard it as a viable alternative.

Of the three remaining choices, their performance on many LP instances seems to be comparable (see Mittelmann [77] for more details on these comparisons), and then the remaining consideration was the footprint of the source code. Looking at Table 1.4, it was clear that QSopt was a better choice. That, plus the possibility to speak with the developers of QSopt, determined our choice for QSopt.

1.3.2 A first Naïve Approach

Since our goal is to obtain exact solutions in the sense of Definition 1.1, a natural way to achieve this goal is to perform every computation in rational arithmetic. To achieve this we took the full QSopt source code, changed every floating point type to the rational type

Table 1.4: Source Code Footprint for Free LP solvers. The size for CLP is just a lower bound on the actual size, because it seems to depend on other components from the COIN suite.

Program	Lines of Code	Source Code Size (Kb)
QSopt	35,482	1,009
GLPK	56,632	1,971
CLP	+81,813	+2,492

provided by GMP, and changed every operation on the original code to use GMP operations.

The resulting code, called `mpq_QSopt` produced the results showed in Table 1.5.

Table 1.5: Running times for `mpq_QSopt` on a number of small MIPLIB and NETLIB LP instances. Time is measured in seconds. The QSopt time column refers to the time spent by the original QSopt code solving the same set of problems.

Instance	Pivots	Time		Instance	Pivots	Time	
		`mpq_QSopt`	`QSopt`			`mpq_QSopt`	`QSopt`
sc50b	50	0.06	0.00	flugpl	15	0.02	0.00
afiro	18	0.02	0.00	sc50a	45	0.05	0.00
sc105	92	0.20	0.00	p0033	28	0.02	0.00
marketshare	32	0.04	0.00	kb2	56	0.38	0.00
enigma	21	0.04	0.00	stocfor1	81	0.15	0.00
bell5	84	0.08	0.00	adlittle	79	0.43	0.00
marketshare2	44	0.06	0.00	egout	86	0.07	0.00
blend	100	2.04	0.00	stein27	40	0.04	0.00
scagr7	144	0.22	0.00	lseu	67	0.08	0.00
bell3a	101	0.08	0.00	sc205	191	1.69	0.00
gt2	29	0.03	0.00	share2b	132	0.86	0.00
rgn	52	0.06	0.00	recipe	48	0.03	0.00
pp08a	132	0.08	0.00	pk1	51	0.24	0.00
noswot	34	0.04	0.00	lotfi	178	0.36	0.00
vtp	157	0.69	0.00	share1b *	251	7.94	0.01
pp08aCUTS	237	0.63	0.00	vpm1	160	0.15	0.00
boeing2	154	0.49	0.00	stein45	67	0.14	0.00
vpm2	179	0.20	0.00	timtab1	108	0.13	0.00
modglob	240	0.26	0.00	bore3d	185	0.99	0.00
scorpion	372	0.66	0.00	mod008	29	0.2	0.00
mas76	171	1.36	0.00	blend2	171	1.38	0.00
mas74 *	167	6.53	0.00	capri	374	2.35	0.02
misc03	122	0.45	0.00	brandy *	302	27.38	0.04
sctap1	154	0.31	0.00	scagr25	560	3.76	0.05
dcmulti	438	0.70	0.00	p0201	212	0.46	0.00

Continued on Next Page...

Table 1.5 (continued)

Instance	Pivots	Time		Instance	Pivots	Time	
		`mpq_QSopt`	`QSopt`			`mpq_QSopt`	`QSopt`
opt1217	129	0.65	0.00	israel	136	1.59	0.00
set1ch	495	0.41	0.00	glass4	75	0.16	0.00
scfxm1 *	423	7.34	0.03	p0282	98	0.21	0.00
bandm *	621	161.58	0.08	timtab2	192	0.24	0.00
e226 *	292	12.45	0.02	p0548	446	0.58	0.01
grow7 *	198	247.60	0.02	etamacro	511	3.28	0.03
agg	104	0.42	0.00	finnis	464	3.33	0.02
standata	64	0.18	0.00	scsd1	119	1.41	0.01
standgub	174	0.30	0.00	beaconfd	109	0.29	0.00
danoint *	949	392.04	0.26	gen	914	1.57	0.04
rout	279	3.06	0.04	stair *	372	2083.13	0.12
gfrd-pnc	895	2.28	0.05	standmps	314	0.54	0.01
khb05250	222	0.47	0.01	scrs8 *	798	13.36	0.08
qiu	1277	13.98	0.23	boeing1	564	7.69	0.05
modszk1 *	875	1326.14	0.30	fixnet6	278	0.33	0.01
aflow30a	352	0.46	0.01	tuff *	524	105.99	0.05
degen2	555	5.06	0.18	forplan *	151	12.75	0.02
agg3	160	0.51	0.01	agg2	183	0.67	0.01
gesa3_o	1176	3.20	–	shell	726	1.10	0.04
scfxm2 *	870	21.24	0.09	pilot4 *	1200	11615.13	0.25

It is not surprising to see that `mpq_QSopt` is slower (by a factor of 100 in most instances). But what is surprising is that, despite the fact that all entries are ordered in Table 1.5 by the size of the problem, the fluctuations in the running time (specially for those problems marked with an asterisk) are not correlated with the number of simplex iterations performed or with the size of the input.

On close examination of the optimal primal/dual solution found by our algorithm, we discover a startling fact: The number of bits needed to represent the solution for the ill-behaving problems is far greater than that for most other problems (some statistics of this fact can be seen in Table 1.6). In fact, the number of bits needed to represent the rational solution found for the problem `pilot4` is amazing, 5,588 bits on average! To put that number in perspective, a conservative upper bound on the number of atoms in the whole universe is 10^{100}, and the suspected age of the universe in milliseconds is about 10^{21}; those two numbers multiplied can be represented in just 402 bits.

Table 1.6: Optimal Solution Representation. We display the average number of bits needed to represent each non-zero in the primal/dual optimal solution found by the algorithm.

Instance	Number of Bits	Instance	Number of Bits
share1b	163	mas74	551
brandy	415	scfxm1	126
bandm	635	e226	429
grow7	1605	danoint	472
stair	4026	scrs8	159
modszk1	827	tuff	150
forplan	306	scfm2	156
pilot4	5588		
others	60		

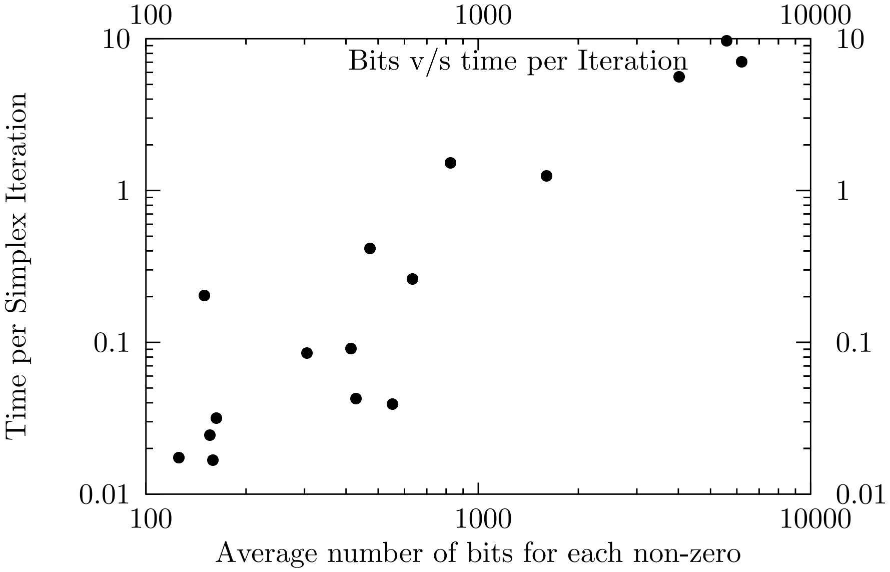


Figure 1.1: Bit representation v/s Time per simplex iteration.

If we now take in consideration the average size of the representation for the optimal solution, and also the average time spent in each simplex iteration, we obtain the graph in Figure 1.1, which shows that the final length of the solution correlates very well with the average cost of each simplex iteration.

If we assume a linear extrapolation on the results from Table 1.1, and also assume that the number of simplex iterations for the regular floating point simplex algorithm to be a good indicator of the number of iterations needed to solve a problem with rational arithmetic, then, since the average number of bits needed to represent the optimal primal/dual solution of the problem `d2q06c` is 78,990 bits, the time needed to solve it with simplex in exact arithmetic would be 870,613 seconds, against the 22 seconds it takes to *solve* the problem with floating point, a factor of 39,573 slower!.

These considerations, namely that the time to solve a problem would depend on the size of the representation for the optimal solution, which is unknown and highly unpredictable, and also memory constraints, make this näive approach highly impractical for most applications.

1.3.3 Is Everything Lost?

Although the previous section shows how hopeless it is to try to avoid errors due to the use of floating-point calculations by performing all calculations in rational arithmetic, there is some evidence that the solutions obtained by floating-point simplex implementations are quite good.

In the studies of Dhiflaoui et al. [35] and of Koch [66], they report that in most cases the reported optimal *basis* gives, in many cases, the actual optimal solution, and on a few others, only some extra pivots are needed to get the true optimal solution. Moreover, both of them show that the time for verification is usually not too large. Koch [66] says with respect to the NETLIB problems the following:

> “The current development version of SOPLEX using 10^6 as tolerance finds true optimal bases to all instances besides `d2q05c`, `etamacro`, `nesm`, `dfl001`, and `pilot4`. Changing the representation from 64 to 128 bit floating point arithmetic

allows also to solve these cases to optimality".

The last part of the quote is quite remarkable. It says that working with larger floating point representation, in his case 128 bits, allowed them to find optimal bases for all NETLIB problems.

Although not stated in his paper, in personal communications with Koch, he explained that he used the *long double* type provided on some architectures that uses 128 bit representations for his calculations. Unfortunately, this solution is not *portable* in the sense that normal long double implementations are only 64 bits for mantissa on most common architectures (which is not a big leap from doubles that use 52 bits for mantissa). Moreover, what if the basic solution found with floating points on 128 bits is not optimal?.

1.3.4 Working with Floating Point with Dynamic Precision

Building on the results of both Koch [66] and Dhiflaoui et al. [35], we propose to extend their methodology by keeping a rational representation of the problem, but solving it with floating point arithmetic with dynamic precision and only performing the optimality/unboundedness/infeasibility test in exact rational arithmetic.

For this approach to work we need to be able to perform floating point calculations with arbitrary (but fixed) precision, and moreover, we need to be able to change this precision at running time. Fortunately **GMP** directly provides those capabilities, thus giving us the tools needed for this approach. An overview of the algorithm is presented in Algorithm 1.1.

Algorithm 1.1 `Exact_LP_Solver` (basic)

```
Require: $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$, $A \in \mathbb{Q}^{m\times n}$
1: Start with some preset precision $p$ (number of bits for floating point representation)
2: Compute approximations $\bar{c}, \bar{b}, \bar{A}$ of original input in the current floating point precision.
3: Solve $\min\{\bar{c}x : \bar{A}x \leq \bar{b}\}$.
4: Test result in rational arithmetic
5: if Test fails then
6:    Increase precision $p$
7:    goto step 2
8: end if
9: return $x^*$
```

We now describe in some detail the different parts that make the full algorithm. We then describe some coding choices that greatly reduced the amount of work needed for the development of the exact LP solver, as well as some engineering questions regarding parameters and the related choices that we made.

Infeasibility Certificates in Exact Arithmetic Consider the problem

$$\begin{aligned} \min \quad & c \cdot x \\ s.t. \quad & Ax = b \\ & l \leq x \leq u. \end{aligned} \tag{1.3}$$

With $c \in \mathbb{Q}^n$, $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$ and $l, u \in (\mathbb{Q} \cup \{+\infty, -\infty\})^n$.

An infeasibility proof for problem (1.3) may assume that $c = 0$. Then, the dual problem can be written as

$$\begin{aligned} \max \quad & b \cdot y - u \cdot d_u + l \cdot d_l \\ s.t. \quad & A^t y + d_l - d_u = 0 \\ & d_u, d_l \geq 0. \end{aligned} \tag{1.4}$$

Note that problem (1.4) is always feasible, and then, a proof of infeasibility for problem (1.3) reduces to finding (y, d_l, d_u) such that it is dual feasible, and with a positive objective value.

When working with modern implementations of the simplex algorithm, it is usually the case that whenever the program stops with the message `infeasible`, it also provides an infeasibility certificate, in most cases in the form of a dual solution y that should satisfy the dual conditions. We take this dual infeasibility certificate, and compute (d_l, d_u) so that (y, d_l, d_u) is feasible for (1.4), we then check that $(d_l)_k = 0$ when $l_k = -\infty$ and that $(d_u)_k = 0$ when $u_k = \infty$, and then check that $(y, d_l, d_u) \cdot (b, l, u) > 0$[4]. If all these conditions hold, then we save the infeasibility certificate and return with success, otherwise we report a failure. An overview of this algorithm can be seen in Algorithm 1.2.

Note that step 1 of Algorithm 1.2 is subject to some freedom. Probably, the most correct

[4] Note that we are assuming that $\infty \times 0 = -\infty \times 0 = 0$

Algorithm 1.2 `Infeasibility_Certificate`

```
Require: $b \in \mathbb{Q}^m$, $A \in \mathbb{Q}^{m \times n}$, $p$ number of bits used for the mantissa for floating points.
Require: $\bar{y}$ floating-point infeasibility certificate as returned by the LP solver
Compute $y \in \mathbb{Q}^m$ such that $\|y - \bar{y}\|_\infty \leq 2^{-p-1}$.
Compute $d = A^t y$, $d_u = d^+$ and $d_l = -d^-$.
if $(d_l)_k = 0$ for all $l_k = -\infty$, $(d_u)_k = 0$ for all $u_k = \infty$ and $(y, d_l, d_u) \cdot (b, l, u) > 0$ then
    return success, problem infeasible
else
    return failure
end if
```

way to implement it would be to use simultaneous diophantine approximation; the problem with this approach is that it is extremely expensive to compute such approximations, specially when the dimension of the vector y is large.

Note that the objective of Algorithm 1.2 is not to prove infeasibility in general, but only to check whether the given certificate (in floating point arithmetic) can be translated into an infeasibility certificate in rational arithmetic. More aggressive versions of this algorithm are possible, for example, if the computed (y, d_l, d_u) is not an infeasibility certificate, we could randomly perturb y, and perform again the tests, however, in our experience, the procedure implemented as described in Algorithm 1.2 proved to be appropriate for our computational tests.

Optimality Certificates in Exact Arithmetic In this case we want to give an optimality proof of a given *basis* of problem (1.3). In this setting, a *basis* $\mathcal{B} = (B, L, U)$ indicates for each variable $x_i, i = 1, \ldots, n$ whether the variable is at its upper bound ($i \in U$), lower bound ($i \in L$), or if it is basic ($i \in B$)[5]. This defines a partition of $1, \ldots, n$ that satisfies the following:

1. $|B| = m$.
2. $A_B := (A_{\cdot i})_{i \in B}$ is full rank.

Definition 1.2 (Optimal Basis). We say that a basis defines an optimal solution if the following holds:

[5] A fourth possibility, is that a given variable is *free*, we ignore this case for the sake of simplicity, although the actual implementation does take this into account.

1. Primal feasibility:

$$l_B \leq x_B := A_B^{-1}(b - \sum_{i \in U} A_{\cdot i} u_i - \sum_{i \in L} A_{\cdot i} l_i) \leq u_B.$$

   Where a_B denotes the vector $a \in \mathbb{Q}^n$ restricted to the index set B.

2. Dual solution:

$$y := A_B^{-t} c_B$$

$$(d_l)_i := (c_i - c_B^t A_B^{-1} A_{\cdot i}), \quad \forall i \in L, \quad \text{zero otherwise.}$$

$$(d_u)_i := -(c_i - c_B^t A_B^{-1} A_{\cdot i}), \quad \forall i \in U, \quad \text{zero otherwise.}$$

3. Dual feasibility: $d_l \geq 0$, $d_u \geq 0$.

4. Complementary slackness: $(x_i - l_i)(d_l)_i = 0$, $(x_i - u_i)(d_u)_i = 0$ for all $i \in \{1, \ldots, n\}$.

Our optimality certificate function takes as an input a basis $\mathcal{B}$ for Problem (1.3), computes the primal/dual solution as described in Definition 1.2, and checks for all required conditions. If all conditions holds, it returns `optimal`, otherwise, it returns `fail` and displays a message indicating which test failed. An overview of this algorithm can be seen in Algorithm 1.3.

Algorithm 1.3 `Optimality_Certificate`

```
Require: $b \in \mathbb{Q}^m$, $A \in \mathbb{Q}^{m \times n}$, $c \in \mathbb{Q}^n$, basis $\mathcal{B} = (B, L, U)$
$x_U \leftarrow u_U$, $x_L \leftarrow l_L$
$x_B \leftarrow A_B^{-1}(b - \sum_{i \in U} A_{\cdot i} u_i - \sum_{i \in L} A_{\cdot i} l_i)$.
if exists $i \in B$ such that $x_i < l_i$ or $x_i > u_i$ then
    return fail, basis is primal infeasible.
end if
$d_u \leftarrow 0$, $d_l \leftarrow 0$.
$y \leftarrow A_B^{-t} c_B$.
$(d_l)_i \leftarrow (c_i - c_B^t A_B^{-1} A_{\cdot i}), \quad \forall i \in L$.
$(d_u)_i \leftarrow -(c_i - c_B^t A_B^{-1} A_{\cdot i}), \quad \forall i \in U$.
if exists $i \in U \cup L$ such that $(d_l)_i < 0$ or $(d_u)_i < 0$ then
    return fail, basis is dual infeasible.
end if
if exists $i \in U \cup L$ such that $(x_i - l_i)(d_l)_i \neq 0$ or $(x_i - u_i)(d_u)_i \neq 0$ then
    return fail, complementary slackness does not hold.
end if
return success $(x, y, d_l, d_u)$.
```

Note that the condition in line 13 of Algorithm 1.3 is redundant, but we keep it for one reason, in general, steps 2,7,8,9 can be expensive, and it might be that by taking the already computed floating point values $\bar{y}, \bar{x}, \bar{d_l}$ and $\bar{d_u}$ we could approximate the true rational values y, x, d_l, d_u. As in the infeasibility certificate, we do this by using the continued fraction approximation, and test the obtained solution. If this approximate solution fails to provide us with an optimality certificate, then we apply Algorithm 1.3.

The approximation procedure (where we estimate y, x, d_l, d_u from their floating point values) succeeded in almost all TSP-related tests, but failed in all MIPLIB and NETLIB instances. Probably the reason for this is that the solutions of our TSP-related test would only need a few bits to be represented, and then the continued fraction method should give us the true rational representation; while for instances like `d2q06c` we would need a very precise floating point solution to get the actual rational representation for the solutions.

Some Computational Considerations Although, from an algorithmic point of view, having an implementation of the simplex algorithm that works on variable-length floating points should be enough to implement Algorithm 1.1; when we also add the requirement of having *fast* implementations, using a general implementation of floating points is clearly a poor choice.

Table 1.7: Number Representation v/s Performance I. Here we show the impact on running time when we change the number-representation used. The algorithm being tested is a variant of the Padberg-Rinaldi (with shrinking) min-cut algorithm over a collection of fractional solutions for the TSP.

Number Representation	Time (s)
integer	162
double	190
GMP-float (128 bits)	463
GMP-rational	645

The reasons for this are numerous, but to cite a few, the fact that most modern day computers have specially designed chips that perform these floating-point operations (in either single or double precision) on hardware provides great speed-ups. These speed-ups mean that today, working with doubles is almost as fast as working with plain integers.

Moreover, if we also consider the fact that accessing a variable-length floating point (as provided by **GMP**) incurs in the extra cost of *dereferencing* the array where the actual number is stored, then the performance penalty by the effects of memory fragmentation (which effectively reduce performance in array operations, so common in most LP solvers) can be quite substantial. Some computational experiments showing these effects can be seen in Table 1.7 and in Table 1.8.

Table 1.8: Number Representation v/s Performance II. Here we show the impact on running time when we change the number-representation used. The algorithm being tested is QSopt primal simplex on the instance `pilot4`.

Number Representation	Time (s)
double	0.38
GMP-float (64 bits)	4.29
GMP-float (96 bits)	4.93
GMP-float (128 bits)	6.03
GMP-float (160 bits)	6.63
GMP-float (1,024 bits)	64.45
GMP-rational	11,615.13

Another consideration is that we need to solve in exact arithmetic linear systems $Ax = b$ in order to effectively perform optimality tests as described in Algorithm 1.3. A possible alternative would be to implement a Gaussian elimination procedure, but given that the systems to be solved may be quite large, a sparse implementation would be needed. Given that QSopt already provides such functions, an interesting alternative would be to have a version of those routines that work with rational arithmetic.

However, in principle, this approach would require us to have three versions of the same QSopt source code, which would make debugging extremely difficult.

For this reason we decided to develop a common interface to work with numbers. This common interface can be configured at compile time to generate versions of the code that use the different number representations, thus allowing us to have only one source code version, but different compiled versions using different type representations for numbers.

This interface, called `EGlpNum`[6], provides all basic arithmetic operations, multiplying, dividing and adding integer numbers to a given number, allocating (and initializing), reallocating and freeing arrays of numbers, converting to and from plain doubles and file input/output. A nice side-effect of this approach, is that we can now read and write LP problems with rational coefficients in either mps or lp format.

Tolerances in Extended Floating Point Representation It is hard to overstress the importance of setting right tolerance values. These choices effectively affect the overall performance of the code. QSopt has eleven such parameters. These parameters include primal and dual feasibility tolerances (set at 10^{-6}), absolute zero tolerance (set at 10^{-15}) and pivot tolerances (set at 10^{-12}).

These settings are the result of experimentation over many problems, but they do not tell us much about how to set them when we change the precision of the floating-point representation.

A possible way of extending these tolerances to larger precisions would be to linearly scale the given value. For example, if we have a tolerance value of 2^a in double precision floating point, then we could choose a value of $2^{a\frac{p}{53}}$ for floating points with p bits of mantissa representation. Note, however, that this approach is greedy in the sense that we are trying to extend as much as possible the achievable precision, but it does not take into account the fact that a factor of error is due to rounding while performing floating-point operations. For this reason we decided to leave some of the precision to *buffer* some of these rounding errors. Then, if we have a tolerance value of 2^a, we set the tolerance on p bits as $2^{a\frac{p}{64}}$, and then leaving $p\frac{11}{64}$ (or about 17%) bits to buffer errors due to floating point calculations.

Another important detail to be decided is how to choose the next precision after a floating-point LP solution has failed to provide us with an optimal solution. We choose to start our procedure with regular double precision floating point calculations, and then we move to 128 bits of mantissa representation, after that we grow by factors of 3/2; we try

[6] `EGlpNum` is part of EGlib, a common project of Daniel Espinoza and Marcos Goycoolea, available online at `http://www2.isye.gatech.edu/~despinoz/EGlib_doc/main.html`, and is released under the LGPL license.

Algorithm 1.4 `Exact_LP_Solver`

```
Require: $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$, $A \in \mathbb{Q}^{m \times n}$
for precision in double, 128, 192, 288, 416, 640, 960, 1440, 2176, 3264 do
  Compute approximations $\bar{c}, \bar{b}, \bar{A}$ of original input in current floating point precision.
  Solve $\min\{\bar{c}x : \bar{A}x \leq \bar{b}\}$.
  $\mathcal{B} \leftarrow$ final basis.
  if simplex status is optimal then
    $(\bar{x}, \bar{y}) \leftarrow$ optimal floating point solution.
    $x \leftarrow\approx \bar{x}$, $y \leftarrow\approx \bar{y}$.
    if success = Optimality_Certificate($\mathcal{B}, x, y$) then
      return success, $(\mathcal{B}, x, y)$.
    else
      $x \leftarrow A_B^{-1}b'$, $y \leftarrow A_B^{-t}c$.
      if success = Optimality_Certificate($\mathcal{B}, x, y$) then
        return success, $(\mathcal{B}, x, y)$.
      end if
    end if
  end if
  if simplex status is infeasible then
    $\bar{y} \leftarrow$ fractional infeasibility certificate.
    if success = Infeasibility_Certificate($\mathcal{B}, \bar{y}$). then
      return infeasible, $(\mathcal{B}, \bar{y})$.
    end if
  end if
  if simplex status is unbounded and precision $\geq$ 128 bits then
    return unsolved.
  end if
end for
return unsolved.
```

up to 12 steps where we increase the precision on the floating point representation. The resulting full algorithm can be seen in Algorithm 1.4.

Some Final Remarks on the Implementation Note that in Algorithm 1.4 we do not handle the case of unbounded problems, this is not because is not possible in theory to provide such certificates, but rather to the fact that QSopt does not provide unboundedness certificates. While we could formulate two auxiliary LP to generate first a feasible solution, and then a ray, we decided to discard such an approach in the hope that future versions of QSopt will provide this functionality.

Another detail is that the choice to stop after trying 3,264 bits is an arbitrary one, and in fact we could keep iterating beyond this point. However our experience shows that we never reach such levels of required floating point precision to actually find the optimal basis for a given LP problem.

1.4 Some Applications and Numerical Results

1.4.1 Orthogonal Arrays with Mixed Levels

What are Orthogonal Arrays? Orthogonal arrays are extensively used in statistical experiments that call for a fractional factorial design. In such applications, columns correspond to the factors or variables in the experiment, and the rows specify the settings or level combinations at which observations are to be made.

We consider two sets of factors F_1 and F_2, with $|F_1| = k_1$, $|F_2| = k_2$, where all factors in F_1 can have s_1 different values (also called levels), and all factors on F_2 can have s_2 levels. The objective is to find a matrix M where each row is a $(k := k_1 + k_2)$-tuple in $\{1, \ldots, s_1\}^{k_1} \times \{1, \ldots, s_2\}^{k_2}$ in such a way that in any submatrix M' of M with t columns, all possible t-tuples that could occur as rows appear equally often. If we call the number of rows of M n, then we say that M is an orthogonal array for $F_1 \cup F_2$ of strength t and size n, also called $OA(n, s_1^{k_1} s_2^{k_2}, t)$.

The Sloan LP Sloan et al. [95] introduced a linear programing problem that produces a lower bound on the size of these arrays, i.e., a lower bound on the number of combinations that are necessary to consider. In their paper, they where able to compute bounds for configurations having $s_1 = 2$, $s_2 = 3$, $t = 3$, $k_1 <= 60$ and $k_1 + 2k_2 \leq 70$, but they found that "outside this range the coefficients in the linear program get too large". The solver that they used was CPLEX 1.2.

We now define the actual LP problem: given s_1, s_2, k_1, k_2 and t positive integers, we define $SL(s_1, k_1, s_2, k_2, t)$ as

$$\begin{aligned}
\min \quad & \sum_{i=0}^{k_1}\sum_{j=0}^{k_2} x_{i,j} & \\
s.t. \quad & x_{0,0} \geq 1 & \\
& x_{i,j} \geq 0 & \text{for } 0 \leq i \leq k_1, \quad 0 \leq j \leq k_2 \\
& \sum_{i'=0}^{k_1}\sum_{j'=0}^{k_2} P_{s_1}^{k_1}(i,i')P_{s_2}^{k_2}(j,j')x_{i',j'} \geq 0 & \text{for } 0 \leq i \leq k_1, \quad 0 \leq j \leq k_2 \\
& \sum_{i'=0}^{k_1}\sum_{j'=0}^{k_2} P_{s_1}^{k_1}(i,i')P_{s_2}^{k_2}(j,j')x_{i',j'} = 0 & \text{for } 1 \leq i + j \leq t.
\end{aligned} \tag{1.5}$$

Where

$$P_s^k(a,b) = \sum_{j=0}^{b}(-1)^j(s-1)^{b-j}\binom{a}{j}\binom{k-a}{b-j}.$$

The Experiments Our experiments have two objectives. First see what are the limits of CPLEX regarding these instances, and to see how our exact LP solver performs in these instances.

We tried several configurations. These configurations, and the results obtained, can be seen in Table 1.9. To put the results in perspective, note that we did not check primal/dual feasibility for the CPLEX solutions (thus, although the objective value may be close to the true optimal, it may well be the case the solution is not feasible). Another detail is that the CPLEX solutions were both over and below the true optimal, which again shows the problem of trusting the obtained bounds. On the upside, CPLEX now is able to *solve* larger problems than those reported by Sloan et al. [95] with ease.

A surprise was instance $(3, 20, 5, 20, 35)$; in this instance the ratio between the smaller and larger coefficient in each constraint is no more than 10^{16}, and the overall ratio between

Table 1.9: Runs on the Sloan LP problems. We present the optimal value, running time and maximum floating point precision required for our exact lp solver. We also display results for CPLEX 9.1 primal simplex, dual simplex and barrier solvers. These results are either the percentage error on the *optimal* solution found, or infeasible if no solution was found.

Instance					QSopt exact			CPLEX		
s_1	k_1	s_2	k_2	t	Value	Time	Precision	Primal	Dual	Barrier
2	20	3	1	4	302.022	0.04	double	$10^{-10.429}$	$10^{-10.429}$	$10^{-10.429}$
2	20	3	2	4	348.336	0.06	double	$10^{-8.875}$	$10^{-8.875}$	$10^{-8.875}$
2	30	3	1	4	584.797	0.05	double	$10^{-9.116}$	$10^{-9.116}$	$10^{-9.116}$
2	40	3	2	4	1084.219	0.13	double	$10^{-9.364}$	$10^{-9.364}$	$10^{-9.364}$
2	60	3	2	4	2169.515	0.22	double	$10^{-9.993}$	$10^{-9.993}$	infeasible
2	100	3	2	4	5621.167	0.56	double	$10^{-9.544}$	$10^{-9.544}$	infeasible
3	18	5	18	35	$10^{20.692}$	131.43	double	$10^{-1.734}$	$10^{-1.734}$	infeasible
3	20	5	20	35	$10^{22.789}$	402.68	double	infeasible	infeasible	infeasible
5	18	7	18	35	$10^{27.094}$	157.54	double	infeasible	infeasible	infeasible
11	20	13	20	20	$10^{23.126}$	2224.33	128	infeasible	infeasible	infeasible
11	20	13	20	30	$10^{34.370}$	3691.24	192	infeasible	infeasible	infeasible
17	20	19	20	10	$10^{13.008}$	3.41	double	$10^{-9.326}$	$10^{-4.453}$	$10^{-9.326}$
19	20	23	20	10	$10^{13.764}$	16.55	double	$10^{-7.768}$	infeasible	$10^{-8.446}$
20	10	30	10	5	$10^{7.385}$	0.07	double	0	0	0
31	20	37	20	10	$10^{15.682}$	16.17	128	infeasible	infeasible	infeasible
40	10	80	10	5	$10^{9.515}$	0.07	double	0	0	0
40	10	80	10	19	$10^{33.449}$	2.34	double	infeasible	infeasible	infeasible
43	20	47	20	10	$10^{16.720}$	13.18	128	infeasible	infeasible	infeasible
61	20	73	20	10	$10^{18.633}$	12.73	128	infeasible	infeasible	infeasible
80	8	90	8	15	$10^{28.955}$	0.77	double	infeasible	infeasible	$10^{-7.898}$
80	9	90	9	15	$10^{29.006}$	4.69	double	infeasible	99.99	infeasible
80	10	90	10	15	$10^{29.057}$	14.42	double	infeasible	99.99	infeasible
200	20	250	20	10	$10^{23.979}$	11.52	128	infeasible	infeasible	infeasible

the largest and smallest coefficient is 10^{23}. These numbers are big enough to make floating point calculations very prone to rounding errors (a testimony of this is that CPLEX can not find a feasible solution), but QSopt is able to find the actual optimal basis in plain double precision floating point arithmetic.

Note also that the running time for the two slowest instances is only 138.8 and 483.47, respectively, if we disregard the time spent in the rational certificate process. This opens the possibility of having better running times if we find a more efficient implementation of our exact certificates.

Finally, note that these instances have no more than 441 variables and constraints, but they are completely dense, i.e. the number of non-zeros is exactly the product of number of rows and columns.

1.4.2 The NETLIB, MIPLIB and other instances

In response to the needs of researchers for access to real-world mixed integer programs, a group of researchers R.E. Bixby, E.A. Boyd and R.R. Indovina created in 1992 the MIPLIB repository, an electronically available library of both pure and mixed integer programs.

This was updated in 1996 by Robert E. Bixby, Sebastian Ceria, Cassandra M. McZeal, and Martin W.P. Savelsbergh. This version of MIPLIB is known as MIPLIB 3.0, and is available on-line at Rice University.

A second update was done in 2003 by Alexander Martin, Tobias Achterberg, and Thorsten Koch. Their intention was to update the set of instances with *harder* MIP examples. This version of MIPLIB is known as MIPLIB 2003. This test-set is available on-line at `http://miplib.zib.de`.

The NETLIB repository contains freely available software, documents, and databases of interest to the numerical, scientific computing, and other communities. This repository has a set of well known LP instances that has also come to be known as the NETLIB problems.

We also consider problems from the Kennington LP test sets from NETLIB, the LP test sets from Hungarian Academy of Sciences, including the set of stochastic LPs, the set of miscellaneous LP, and also the set of problematic LP. We also consider the LP test sets from Hans D. Mittelmann. Most of these problems can be found at `http://www.gamsworld.org/performance/plib/origin.htm`.

We took the union of all these set of instances as a test-set for our exact LP solver, containing 625 problems in total. All comparisons are against the original version of QSopt, the reason being that since our code is a transformation of QSopt, and no work has been done to improve its performance; we can not hope to obtain better running times than the original code.

The reader should also bear in mind that our code is a *proof of concept* and not a commercial grade implementation. One of the problems that we have not addressed yet is how to hot-start the simplex algorithm in extended floating point precision when the previous precision run of simplex fails. This has a very important impact in the performance of the overall algorithm, because when the simplex code fails in plain double precision, then the extended precision code must completely solve the LP from scratch, which means a penalty factor of about 10-20 times slower than the original version of QSopt. We split our results into problems where hot-start does occur, and into problems where no hot-start is done.

Table 1.10: Comparison on Infeasible Instances. Here we compare running times for both the original QSopt code (QSopt) and our exact version (QSopt_ex) primal simplex, as well as the precision required to prove infeasibility.

Instance	QSopt	QSopt_ex		Instance	QSopt	QSopt_ex	
	Time	Precision	Time		Time	Precision	Time
bgdbg1	0.01	double	0.04	bgetam	0.04	double	0.14
bgindy	0.14	double	0.45	bgprtr	0.00	double	0.00
box1	0.00	double	0.01	ceria3d	0.84	double	1.49
chemcom	0.01	double	0.02	cplex1	1.02	double	2.05
cplex2	0.09	128	1.12	ex72a	0.01	double	0.01
ex73a	0.00	double	0.01	forest6	0.00	double	0.01
galenet	0.00	double	0.00	gosh	4.29	double	6.39
gran	5561.30	128	714.27	greenbea	2.80	double	5.39
itest2	0.00	double	0.00	itest6	0.00	double	0.00
klein1	0.01	double	0.05	klein2	0.25	double	0.35
klein3	0.72	double	1.39	mondou2	0.01	double	0.02
pang	0.04	double	0.12	pilot4i	0.08	double	0.55
qual	0.04	128	0.32	reactor	0.01	double	0.03
refinery	0.03	double	0.09	vol1	0.05	128	0.52
woodinfe	0.00	double	0.01				

We do not report on problems where the sum of running time of primal and dual simplex for both the exact LP solver and the original QSopt code is less than one second, this leaves us with only 364 problems. Furthermore, some problems could not be solved in less than five days of running time, these problems are nug20, nug30, cont11, cont1, cont11_l, and cont1_l, these problems were removed from the overall list of problems that we report. Infeasible problems are reported separately from feasible LP problems, and feasible LP problems are split into three categories, problems with less than 1,000 constraints are considered small, problems with 1,000-10,000 constraints are considered of medium size, and problems with over 10,000 constraints are considered large.

In Table 1.10 we compare results for QSopt and our exact version on all problems from NETLIB that are infeasible. Note that for the instance `gran` QSopt ends without any answer (it reaches the iteration limit), while QSopt_ex gives us the correct answer. The reason is that since QSopt_ex can change precision, each precision that we try has a much lower limit of iterations than the default QSopt, because if any floating point precision fails, we can always move to the next precision. In fact, after less than 800 iterations in 128-bits precision, it is able to find a proof of infeasibility, while QSopt seems to be stuck in an infinite loop.

Table 1.11: QSopt_ex performance on MIPLIB, NETLIB and other problems. Column Size is the number of problems that fall within the corresponding category, column R1 is the geometric average of all running time ratios computed as QSopt_ex running time over QSopt running time, column R2 is the same but we discount the time spend computing rational certificates from the QSopt_ex running time, column R3 is the ratio of the sum of all running time for QSopt_ex and the sum of all running time for QSopt, and the last column show the average running time for QSopt in the set of instances

Problem Set	Alg.	Size	Running time ratio			Running
			R1	R2	R3	time
Large with restart	primal	68	1.17	1.09	1.04	4000.86
Large with restart	dual	62	1.53	1.43	1.09	1932.69
Medium with restart	primal	181	3.65	2.86	2.82	459.84
Medium with restart	dual	182	4.59	3.28	1.12	409.44
Small with restart	primal	59	3.96	2.75	1.42	6.78
Small with restart	dual	62	3.99	2.55	1.48	7.37
Large no restart	primal	15	14.42	13.71	13.66	1901.96
Large no restart	dual	21	9.51	9.24	6.19	4290.02
Medium no restart	primal	33	37.38	27.11	59.64	166.35
Medium no restart	dual	32	70.92	40.66	68.51	164.43
Small no restart	primal	7	120.77	102.21	383.48	91.96
Small no restart	dual	4	217.39	175.75	403.09	43.22

In Appendix A, Table A.2 shows the results for the primal simplex algorithm on instances where hot-start could not be done, while Table A.1 shows the results on instances where hot-start was done. In Table A.4 we present the results for the dual simplex algorithm on instances where hot-start could not be done, while Table A.3 shows the results on instances where hot-start was done. Table 1.11 present a summary of these results showing the geometric average of the running time ratios for the original code and our exact version for small, medium and large problems. We also show the geometric average of the running time of the exact code versus the original code without considering the time spent on the rational check, the ratio of total running time computed as the ratio between the sum of all running time for the exact code and the sum of all running time for the original QSopt implementation, and the number of instances that falls within each category. All runs where done using a Linux workstation with a 3GHz Intel Pentium 4 CPU and with 4GB of RAM.

It is interesting to note that for those problems where we successfully did hot-start the simplex procedure with higher precision, the running time ratios for large instances are in between 1.04 and 1.53 depending on how we measure it. The situation among all problems where hot-start was possible is similar, we obtain ratios ranging from 4.59 down to 1.04 depending on the set and how we measure it. The case of problems where we could not hot-start subsequent runs of simplex, is as we expected much worst, varying from ratios between 403.09 to 6.19. Fortunately, only on 18.24% of the problems we could not hot-start

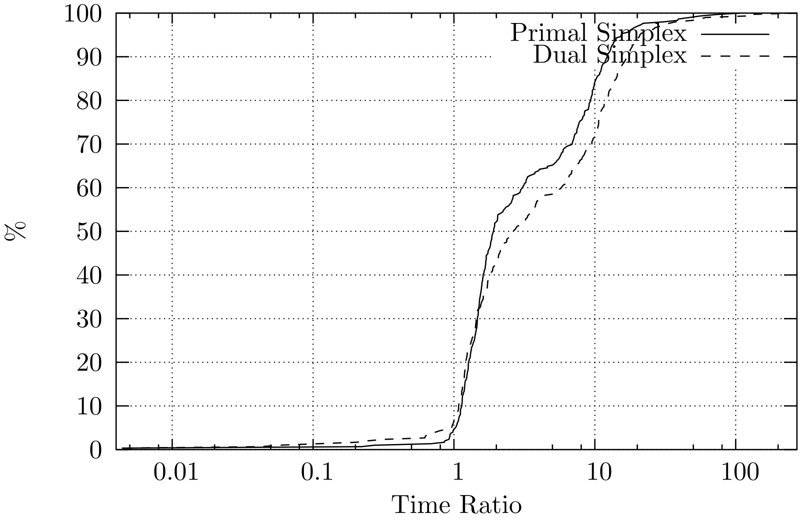


Figure 1.2: Running time ratio distribution I. Here we show the running time ratio distribution over all instances where re-start was possible for both primal and dual simplex methods. This gave us a total of 299 problems.

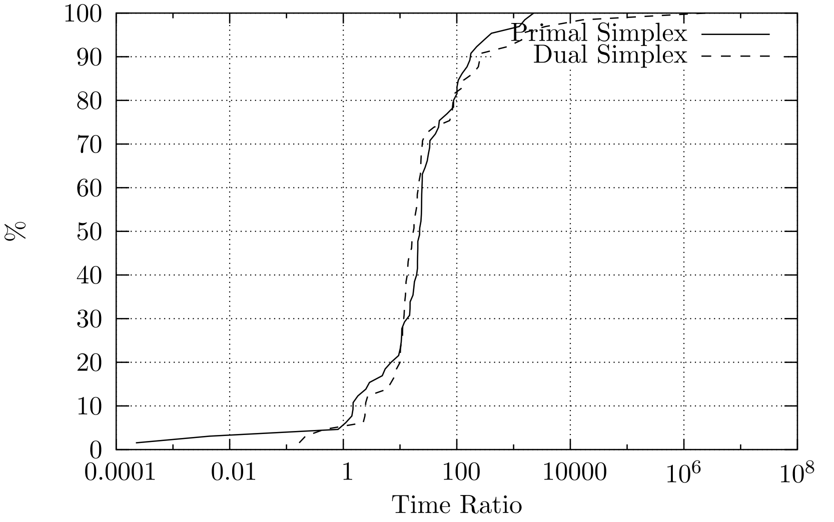


Figure 1.3: Running time ratio distribution II. Here we show the running time ratio distribution over all instances where re-start was not possible for either primal or dual simplex method. This gave us a total of 65 problems.

the successive simplex calls.

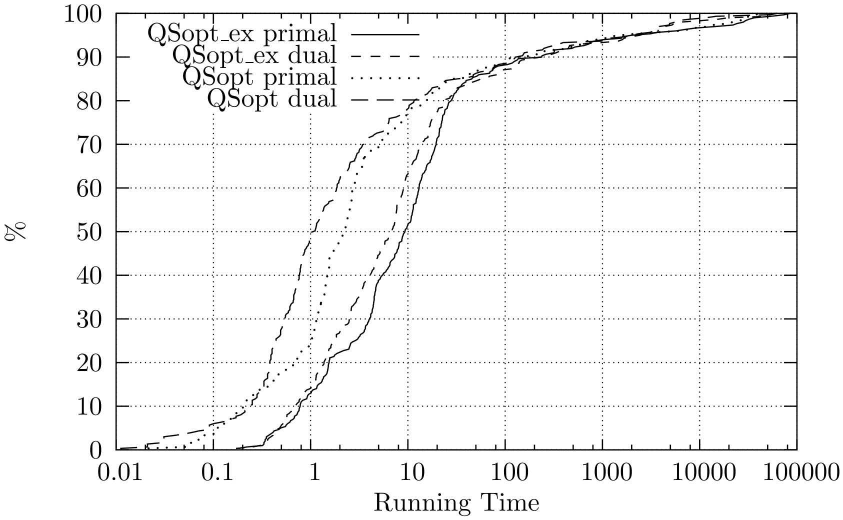


Figure 1.4: Running time distribution. Here we show the running time distribution over all instances where re-start was possible for both primal and dual simplex method, we also show the running time distribution for the original QSopt code.

Note that on the other hand, Figure 1.2 and Figure 1.3, suggest a slightly different story, they seem to indicate that the ratios are in the rage of 1-10 for problems with hot-start, and in the range 1-100 for problems where no hot-start was done, how does this fit with the overall running time ratio for medium and large instances shown in Table 1.11? To try to understand this disparity, we take a look at Figure 1.4, where the running time distribution for primal and dual simplex is shown for both QSopt and our exact version QSopt_ex. Note that for running times below 100 seconds there is a substantial difference for running times between the exact and the original versions of QSopt, but those differences tend to disappear on problems requiring more than 100 seconds to solve. With this knowledge,

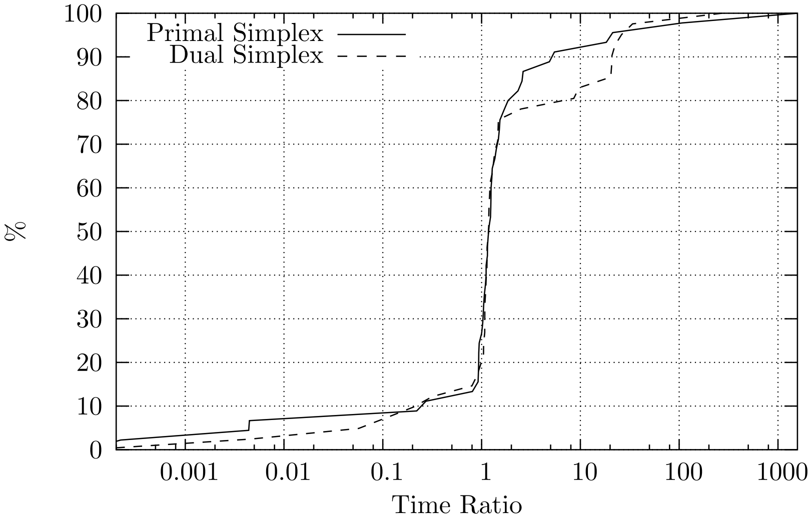


Figure 1.5: Running time ratio distribution III. Here we show the running time ratio distribution over all instances where re-start was possible for both primal and dual simplex methods, and that took over 100 seconds to solve for either QSopt, or for QSopt_ex. This gave us a total of 45 problems for primal simplex, and 41 problems for dual simplex.

we repeat our running time ratio distribution for those instances where hot-start was done and where either QSopt or our exact version took over 100 seconds to solve, this leave us with 45 problems for the primal simplex algorithm, and 41 problems for the dual simplex algorithm, Figure 1.5 shows the distribution of the running time ratios, and now we can see that they are closely distributed around 1, which is a consistent result with the results shown in Table 1.11.

Another interesting observation that we can make is that LP problems can have quite large encoding for their optimal solutions, Figure 1.6 shows the empirical distribution of the encoding length of each non-zero coefficient of optimal solutions, the data was drawn from all optimal solutions collected during our experiments. The problem with the largest

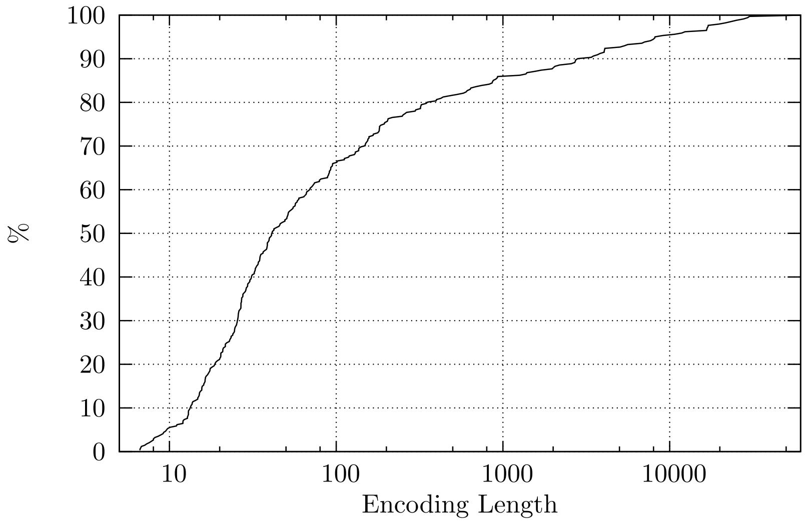


Figure 1.6: Encoding length distribution. Here we show the experimental distribution of the average encoding length for nonzero coefficients in optimal solutions. The data consists of 341 LP problems from MIPLIB, NETLIB and other sources. Note that the x axis is in logarithmic scale.

average encoding that we solved was stat96v3, which require on average 64,000 bits to represent each coefficient, and the largest coefficient representation required 152,000 bits, this problem is part of the miscellaneous set of LP from the Hungarian Academy of Sciences. Fortunately, it seems that most problems require much less precision to be represented, note that from Figure 1.6, 80% of the problems required less than 256 bits to exactly represent their optimal solution.

In fact, if we look at the required precision to find an optimal basis, 51.92% were solved in plain double arithmetic, 74.03% could we solved using at most 128 bits for mantissa representation, 81.59% could be solved using 192 bits for mantissa representation, and 98.90% of the problems could be solved using 256 bits of mantissa representation. Figure 1.7 shows the optimal value for problem maros-r7, which only requires on average 16,715 bits to represent the optimal solution found by QSopt_ex, about a quarter of the average size required by problem stat96v3.

One final question that we address is how close are the objective values reported by QSopt to the true optimal value computed by QSopt_ex. We present the absolute value of the relative error (in percentage) between the true optimal solution, and the value reported

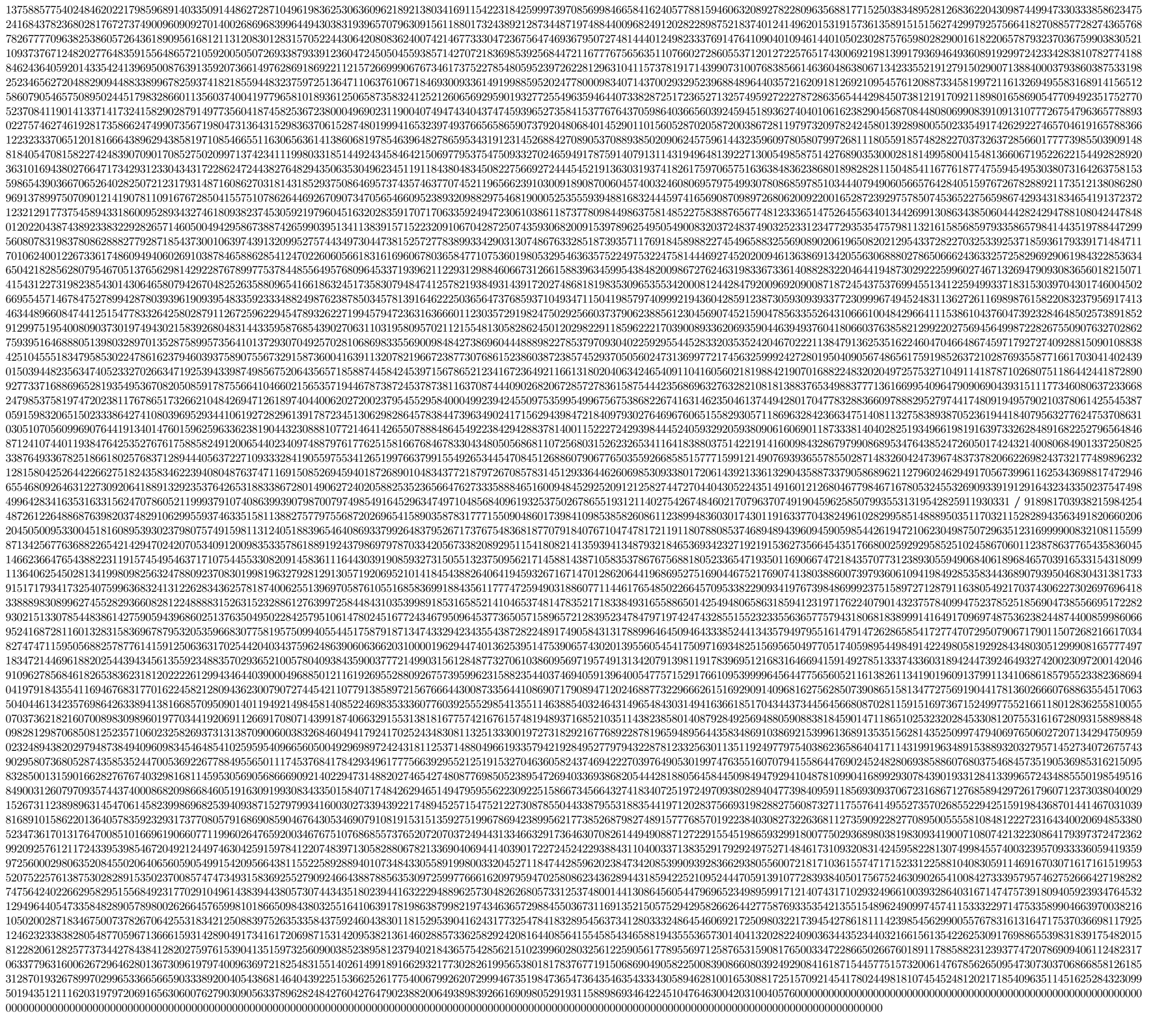


Figure 1.7: A Long Optimal Value. This figure shows the optimal value for problem maros-r7, its primal/dual solution has 12,545 non zeros entries, each one needing an average of 16,715 bits to be represented.

by the original QSopt code. Note that as reported by many before us, it seems that the errors tend to be small but for a few problems. Figure 1.8 shows the empirical distribution derived from our experiments, note that 98% of the problems have relative error of less than 1% in the objective value.

1.4.3 TSP-related Tests

One interesting standing conjecture related to the STSP, is whether the optimal optimal value of the subtour-elimination polytope (SEP)[7] is within 4/3 of the true optimal value for

[7] This relaxation was introduced by Held and Karp [55]

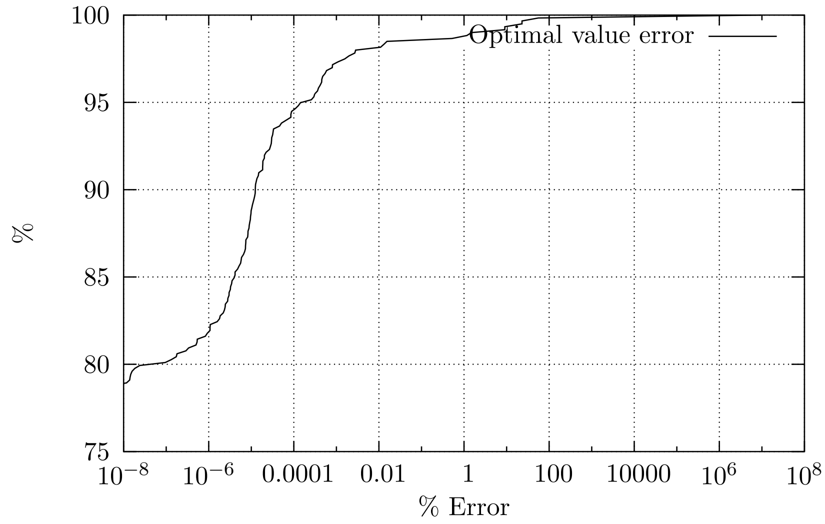


Figure 1.8: Relative objective value error distribution. Here we show the relative error in objective value as reported by QSopt and our exact version QSopt_ex, the test bed includes all our models, i.e. 595 problems in total.

the STSP in the metric case (i.e. when the cost matrix satisfies the triangular inequality).

Boyd and Labonté [21] presented exact values for the integrality gap for instances with up to 10 cities. We extend their results to include all instances in the TSPLIB with at least 50 cities, providing the exact solution for the SEP, the integrality gap to the optimal solution[8], and the length of the encoding for the primal/dual solution that we obtained. We also provide the same results for a million city TSP problem.

Our implementation relies on CONCORDE [7] to obtain a floating point approximation of the SEP polytope, and then we perform the pricing of edges, the generation of violated subtours in exact rational arithmetic, and we use our exact LP solver for the LP part, iterating the cutting and the column generation process until we prove optimality. We implemented a version of the Padberg and Rinaldi [89] minimum-cut algorithm in exact rational arithmetic to obtain any remaining violated subtour inequality. A summary of our results can be seen in Table 1.12.

[8] In the case of pla85900 we only provide the gap to the best known upper bound, since the optimal solution for it remains an open problem.

Table 1.12: Subtour elimination polytope integrality gap. We present for each instance the number of bits to encode the largest number in the optimal primal/dual solution (including objective value), total running time, optimal value for the SEP relaxation, optimal value for the instance or best upper bound known, and the percentage gap.

Instance	Encoding length	Time (seconds)	SEP Optimal value	Optimal value / Best bound	Gap
a280	12	0.36	2566	2579	0.506%
ali535	22	2.05	804949/4	202339	0.547%
att48	14	0.04	10604	10628	0.226%
att532	21	1.40	164515/6	27686	0.973%
berlin52	13	0.03	7542	7542	0.000%
bier127	17	0.18	117431	118282	0.724%
brazil58	18	0.05	50709/2	25395	0.159%
brd14051	30	340.97	21020743/45	469385	0.483%
brg180	19	3.19	1950	1950	0.000%
burma14	12	0.01	3323	3323	0.000%
ch130	15	0.12	12151/2	6110	0.567%
ch150	19	0.17	51921/8	6528	0.583%
d1291	30	8.00	7029201/140	50801	1.179%
d15112	37	451.49	375571207/240	1573084	0.524%
d1655	21	10.46	246197/4	62128	0.940%
d18512	34	587.06	92464823/144	645238	0.486%
d198	14	0.60	15712	15780	0.432%
d2103	18	12.07	79307	80450	1.441%
d493	18	1.26	69657/2	35002	0.498%
d657	24	1.59	775283/16	48912	0.942%
dsj1000	32	4.25	222563723/12	18659688	0.607%
eil101	12	0.13	1255/2	629	0.239%
eil51	12	0.04	845/2	426	0.828%
eil76	10	0.05	537	538	0.186%
fl1400	15	21.15	19783	20127	1.738%
fl1577	15	31.49	21886	22249	1.658%
fl3795	20	124.69	113909/4	28772	1.035%
fl417	17	2.16	23579/2	11861	0.606%
fnl4461	27	33.99	4357661/24	182566	0.548%
gil262	14	0.33	4709/2	2378	0.998%
gr120	18	0.11	27645/4	6942	0.444%
gr137	21	0.26	276481/4	69853	1.060%
gr202	16	0.38	40055	40160	0.262%
gr229	28	0.53	5332681/40	134602	0.963%
gr431	24	1.53	510916/3	171414	0.650%
gr666	29	2.75	10529761/36	294358	0.637%
gr96	19	0.15	109139/2	55209	1.171%
kroA100	17	0.07	41873/2	21282	1.650%
kroA150	15	0.15	26299	26524	0.855%
kroA200	15	0.22	29065	29368	1.042%

Continued on Next Page...

Table 1.12 (continued)

Instance	Encoding length	Time (seconds)	SEP Optimal value	Optimal value / Best bound	Gap
kroB100	15	0.09	21834	22141	1.406%
kroB150	18	0.16	51465/2	26130	1.544%
kroB200	15	0.25	29165	29437	0.932%
kroC100	17	0.08	40945/2	20749	1.350%
kroD100	17	0.10	42283/2	21294	0.721%
kroE100	17	0.08	43599/2	22068	1.231%
lin105	17	0.09	28741/2	14379	0.059%
lin318	20	0.67	167555/4	42029	0.334%
million	82	3.88×10^6	$\frac{41679386539494215}{58810752}$	709412217	0.100%
nrw1379	26	4.29	1353509/24	56638	0.428%
p654	16	4.74	34596	34643	0.135%
pa561	15	1.15	5479/2	2763	0.857%
pcb1173	16	2.74	56351	56892	0.960%
pcb3038	20	17.50	273175/2	137694	0.810%
pcb442	19	0.63	100999/2	50778	0.551%
pla33810	33	4527.63	525643505/8	66048945	0.522%
pla7397	32	206.08	277517567/12	23260728	0.580%
pla85900	28	32106.46	141806385	142382641	0.406%
pr1002	26	3.76	3081191/12	259045	0.887%
pr107	16	0.26	44303	44303	0.000%
pr124	19	0.22	116135/2	59030	1.657%
pr136	20	0.28	191869/2	96772	0.872%
pr144	21	0.33	232757/4	58537	0.597%
pr152	19	0.51	146417/2	73682	0.646%
pr226	17	0.81	80092	80369	0.345%
pr2392	23	10.82	1120469/3	378032	1.216%
pr264	19	0.95	98041/2	49135	0.233%
pr299	16	0.40	47380	48191	1.711%
pr439	21	1.28	317785/3	107217	1.216%
pr76	17	0.09	105120	108159	2.890%
rat195	16	0.30	9197/4	2323	1.032%
rat575	13	0.83	6724	6773	0.728%
rat783	18	1.55	35091/4	8806	0.379%
rat99	11	0.06	1206	1211	0.414%
rd100	17	0.09	23698/3	7910	0.135%
rd400	14	0.64	15157	15281	0.818%
rl11849	30	365.01	21935527/24	923288	1.018%
rl1304	24	6.00	1494563/6	252948	1.547%
rl1323	21	5.11	531629/2	270199	1.649%
rl1889	21	11.06	623409/2	316536	1.550%
rl5915	25	83.96	3341093/6	565530	1.558%
rl5934	28	87.61	10969411/20	556045	1.381%
si1032	17	18.98	92579	92650	0.076%
si175	19	0.57	85499/4	21407	0.150%

Continued on Next Page…

Table 1.12 (continued)

Instance	Encoding length	Time (seconds)	SEP Optimal value	Optimal value / Best bound	Gap
si535	23	7.06	581011/12	48450	0.066%
st70	10	0.06	671	675	0.596%
ts225	17	0.17	115605	126643	9.548%
tsp225	17	0.31	15513/4	3916	0.973%
u1060	24	4.77	1781207/8	224094	0.648%
u1432	18	9.90	152535	152970	0.285%
u159	16	0.21	41925	42080	0.369%
u1817	21	10.80	226753/4	57201	0.904%
u2152	25	14.81	1021729/16	64253	0.618%
u2319	36	215.23	234215	234256	0.017%
u574	16	1.29	36714	36905	0.520%
u724	19	1.56	124958/3	41910	0.617%
ulysses16	13	0.02	6859	6859	0.000%
ulysses22	13	0.03	7013	7013	0.000%
usa13509	29	387.33	79405855/4	19982859	0.661%
vm1084	26	4.37	2833949/12	239297	1.327%
vm1748	27	7.43	5977093/18	336556	1.353%

It is interesting to note that the worst integrality gap that we found was around 9% (for instance ts225). A second remark is that most of the time was spent doing the full pricing for all edges not in the LP, operations like solving the current LP relaxation, or finding minimum-cuts took well under 100 seconds, even in the case of the million city TSP instance.

The million city TSP is a random-Euclidean data set with the coordinates being integers drawn from the $1,000,000 \times 1,000,000$ grid. The problem was created by David S. Johnson (AT&T), who used it as part of his test-bed of instances in [42]. The instance is called E1M.0 in the DIMACS challenge page, and can be generated using the code available at http://www.research.att.com/~dsj/chtsp/download.html.

1.5 Final Comments and Missing Links

1.5.1 Shortcomings and Possible Improvements

Rational Certificates It is clear that one of the most expensive parts of the proposed methodology is to obtain rational certificates of optimality/infeasibility (see Figure 1.9 for

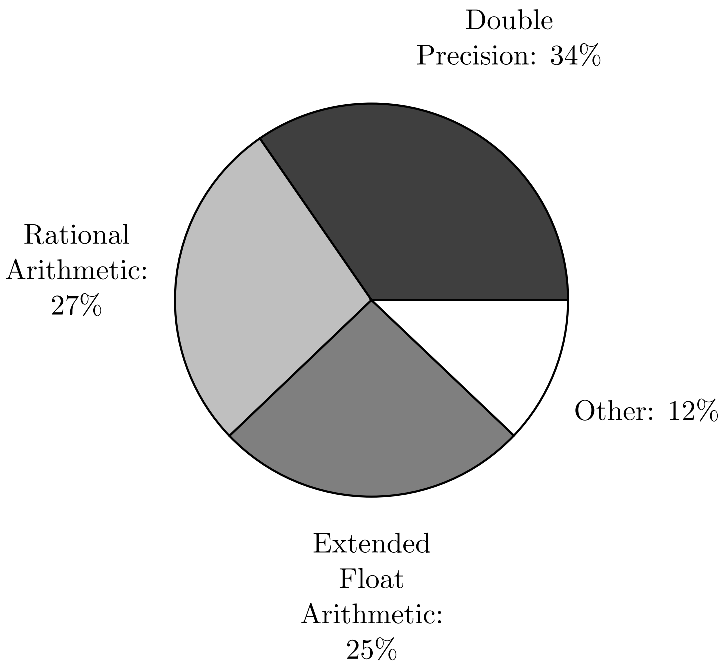


Figure 1.9: Profile of QSopt_exact I. Here we summarize the time spent in each type of arithmetic (double, extended floats and rationals) for 199 problems running both primal and dual simplex. The total accumulated time was 250,886.46 seconds. The graph shows the percentage of the time spend in each type of arithmetic.

more details). Our approach solved this problem by using QSopt in rational arithmetic to solve the rational systems, which perform first a factorization of the constraint matrix, and then compute the primal and dual solution using this factorization.

Although this approach does give us the desired result, there is reason to believe that this is not the best approach. Some alternatives are to directly solve the linear system by Gaussian elimination, or to use Wiedemann's method [97] to solve the rational system. This last method has the advantage of being parallelizable, and has been used by Dhiflaoui et al. [35], obtaining almost linear speed-up on a cluster with 40 PC's workstations.

Hot-Starts or How to use Previous Results In the case where a floating point approximation *solves* the given approximation to the problem, but the certificate step fails, we use the resulting basis as a starting point for the next floating point approximation. This results in the number of simplex iterations performed in extended arithmetic being quite low in these cases.

Unfortunately we can not take advantage of the results of the LP-solver when an unbounded or infeasible problem has been detected by the floating-point simplex method but the certificate method fails. The reason is that in many cases the ending basis is quite ill-behaving from a numerical point of view, and in many cases is not even close to the optimal solution. In this situation we are forced to re-start from scratch with increased floating point precision, the effects in running time can be seen in Figures 1.9 and 1.10. However, it could be possible to use the basis obtained in such situations if we had some knowledge of their properties, or if we know that the ending basis is *good* in some sense.

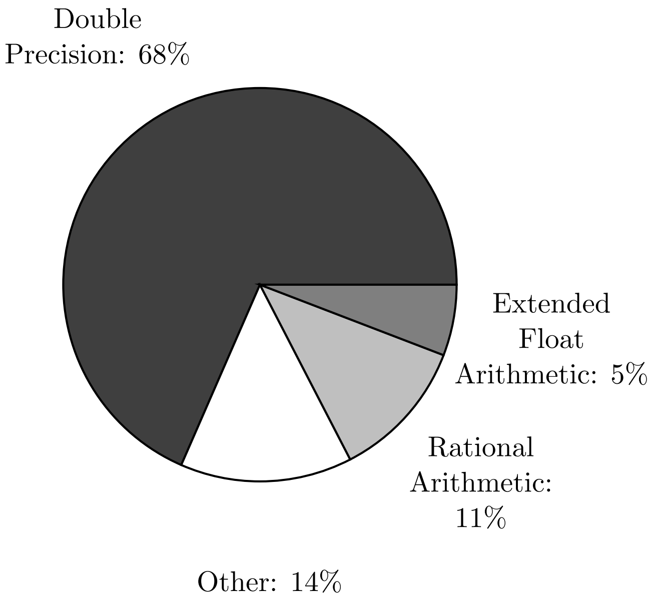


Figure 1.10: Profile of QSopt_exact II. Here we summarize the time spent in each type of arithmetic (double, extended floats and rationals) for 191 problems running both primal and dual simplex, but we exclude instances where we could not hot-start simplex iterations done in extended arithmetic. The total accumulated time was 122,852.61 seconds. The graph shows the percentage of the time spend in each type of arithmetic.

Handling Tolerances in a Multiprecision Environment This forces us to question how to handle numerical problems when we have multiple levels of precision available to us. QSopt (like other simplex implementations in floating-point arithmetic) plays with its tolerances to both improve speed and maintain numerical stability, but having the possibility to improve numerical accuracy changes this goal in the sense that we could try to get good solutions but with better numerical stability, and only when this criteria forbids any further simplex steps change to higher precision floating points to hopefully perform just a few more pivots.

Floating Point Implementations Also from a programming perspective, the GMP multiple-precision floating point implementation has some drawbacks, mainly that it introduces memory fragmentation on arrays that degrade performance. While their choice seems to be the best one if we truly want multiple precision floating point calculations with run-time determined precision, we could have a more compact and efficient implementation for *lower* floating point precisions like 96,128 and 192 bits of mantissa lengths, and further improve the actual running time.

Just for illustration purposes, we took an implementation of IEEE floating points on 128 bits using 15 bits for exponent and 112 bits for the mantissa due to John Hauser and available on-line at `http://www.jhauser.us/arithmetic/SoftFloat.html`. This implementation uses contiguous memory to represent floating-point numbers (thus avoiding part of the memory fragmentation problems described before), but lacks any assembler code (which GMP uses heavily to improve performance). To show the impact on performance of using different implementations of floating points, we took 135 of the smallest problems in our collection, and ran from scratch a version of QSopt using double representation, another using Hauser's float128 floating-point number, and another using GMP general floating point with 128 bits of mantissa, that we call mpf128.

Table 1.13: Floating point calculations speed. Here we show the total running time of QSopt using different floating point number representations over a set of 135 small LP instances.

Representation	Total Time (s)
double	17.966
float128	213.840
mpf128	272.689

In Table 1.13 we see that Hauser's floating point implementation is indeed faster than the general floating point implementation, showing that even in this area there is some time to be gained by clever implementations of the floating-point arithmetic.

Errors in the Program One regrettable outcome of our code conversion is that the resulting code is far less stable than the original QSopt code, even if we run the transformed code with plain double precision arithmetic. The reason for this is not a problem with the idea of converting the source code of the LP solver, but rather to errors while doing the conversion.

Good programming practices (like comparing the performance of the code with the original version every time an important function is changed, or when a suitable number of lines of code have been changed) should prevent this.

What surprises the author is that despite these errors the code still is able to solve almost all problems (but with some penalty on the performance).

Table 1.14: Coding Errors. Here we show running times for some LP instances for the original QSopt code (QSopt_ori), and the transformed QSopt code compiled with double arithmetic (QSopt_dbl).

Instance	QSopt_ori	QSopt_dbl
baxter	12.32	13.46
qap12	475.85	511.57
maros-r7	13.49	29.92
greenbeb	9.28	102.00

Some examples showing the effects of these suspected errors can be seen in Table 1.14.

1.5.2 Final Remarks

Taking the results obtained during this research, it seems reasonable to expect to have simplex implementations that can either produce high quality solutions, or actual rational certificates of optimality/infeasibility/unboundedness when required, without too much performance penalty.

It is also clear from Section 1.5.1 that much can be done to improve this approach, both from a programming perspective, and from an algorithmic point of view, specially when handling problems with numerical instabilities.

Although our code has still some issues regarding numerical stability, we are making it available to the academic community in the hope that it will be a good tool for solving LP problems exactly. The code can be found at `http:/www.isye.gatech.edu/~despinoz`.

CHAPTER 2

Mixed Integer Problems and Local Cuts

2.1 Introduction

Mixed integer (linear) programming (or MIP), are optimization problems in which the objective function and all the constraints are linear, and where some or all variables are restricted to have integer or discrete values.

The range of applications is wide, and as Nemhauser and Wolsey write:

> "... Because of the robustness of the general model, a remarkably rich variety of problems can be represented by discrete optimization models.
>
> An important and widespread area of application concerns the management and efficient use of scarce resources to increase productivity. These applications include operational problems such as the distribution of goods, production scheduling, and machine sequencing. They also include (a) planning problems such as capital budgeting, facility location, and portfolio analysis and (b) design problems as communication and transportation network design, VLSI circuit design, and the design of automated production systems.
>
> In mathematics there are applications to the subjects of combinatorics, graph theory, and logic. Statistical applications include problems of data analysis and reliability. Recent scientific applications involve problems in molecular biology, high-energy physics, and x-ray crystallography. A political application concerns the division of a region into election districts". Nemhauser and Wolsey [86].

Although based in LP, much of the theory available to LP problems does not apply to MIP problems. Moreover it remains an open question whether or not MIP is solvable in polynomial time. In fact MIP is $\mathcal{NP}$-complete, and is widely believed that no such algorithm exists.

The strength of the MIP modeling paradigm has been recognized almost from the beginning of its introduction, but it has been the developments of the last 20 years that has truly made it an applicable one.

These developments include both theoretical discoveries, and also the growth in computer power, for example Bixby [15] shows that the advances of the last 20 years translate in a total speed-up of six orders of magnitude while solving LP problems! It is these kind of improvements that makes it possible today to solve problems like the traveling salesman problem (TSP) on large instances[1].

The standard approach to solve MIP problems is a mixture of two algorithms. The first algorithm was introduced by Lang and Doig [68] and further improved by Little et al. [73] and Dakin [30], and is commonly known as the *branch-and-bound* algorithm. The second algorithm, introduced by Dantzig, Fulkerson and Johnson [33], is known as the *cutting plane* algorithm. The resulting hybrid method is known as the *branch-and-cut* algorithm, and is arguably the most successful approach to-date to tackle general MIP problems.

One of the key ingredients to the success of general purposes MIP solvers, is the cut generation procedure (or cutting plane phase). Bixby et al. [16] shows that by disabling (mixed) Gomory cuts in CPLEX 6.5, the overall performance goes down by a 35%, and if on the other hand we disable all other cuts and use just (mixed) Gomory cuts, CPLEX 6.5 achieves 97% improvement against the performance of using no cuts at all. This is a clear indication of how important cutting planes are for today's MIP solvers.

The subject has received wide attention in the academic literature, and there is an ever growing list of valid inequalities and facets for most well-known MIP problems, including the TSP (for a survey on valid inequalities for the TSP, see Naddef and Pochet [79]), the vehicle routing problem (VRP), bin-packing problems, the knapsack problem, the stable set

[1] The largest TSP problem solved to optimality today has 33,810 cities.

problem and many others.

While the literature for valid inequalities (and facets) for special problems is abundant, the case of general MIP problems has received less attention. Some notable exceptions are the Gomory cuts [46, 47, 48], mixed integer rounding inequalities (MIR) [85], the disjunctive cuts of Balas [9, 10] and others, the lift and project cuts from Lovász and Schrijver [74, 75] and Balas et al. [11], the split cuts of Cook et al. [27], and the lifted cover [51, 53] and flow cover [52] inequalities of Gu et al. Note however that MIR cuts, disjunctive cuts, lift and project cuts and split cuts are closely related, in fact Balas and Perregaard [12] proved that, for the case of 0-1 mixed integer programming, there is a one to one correspondence between lift-and-project cuts, simple disjunctive cuts (also called intersection cuts) and MIR cuts.

An interesting new approach to the cutting-generation procedure was introduced by Applegate et al. [5, 7]. They introduce a procedure to generate cuts that relies on the equivalence between optimization and separation to get cuts resulting from small GTSP problems that are the result of a *mapping* of the original problem. This methodology to find cuts for the TSP allowed the authors to greatly improve both the total running time on all instances (by 26% on medium size instances, and by over 50% on larger instances), increase the number of problems solved at the root node of the branch and cut procedure (from 42 without using local cuts, to 66), and also to solve the then largest TSP instances with 13,509 and 15,112 cities respectively.

In this chapter we extend this approach to the case of general MIP problems. The rest of the chapter is organized as follows. In Section 2.2 we provide some motivation for this approach, and extend the separation algorithm proposed by Applegate et al. [5, 6, 7] to the general MIP setting. In Section 2.3 we show how to get facets or high-dimensional faces from valid inequalities. In Section 2.4 we give a mathematical definition of a *mapping*, and give general conditions for them to ensure separation. In Section 2.5 we present the framework for local cuts on general MIP problems, and discuss some implementation details of the procedure, as well as some modifications that we use in our actual implementation. In Section 2.6 we present several of our choices for the *mapping* part, and Section 2.7 shows our experience with the procedure on the MIPLIB instances, and on some problems from

Atamtürk [8]. Finally, Section 2.8 presents our conclusions, thoughts, and hopes on local cuts.

2.2 The Separation Problem

The separation problem can be stated as follows:

> **Definition 2.1 (Separation Problem).** Given $P \subseteq \mathbb{R}^n$ a polyhedron with rational data, and $x \in \mathbb{Q}^n$, prove that $x \in P$, or show that there exists a linear inequality $ay \leq b$ with $(a, b) \in \mathbb{Q}^{n+1}$ such that $P \subseteq \{y \in \mathbb{R}^n : ay \leq b\}$ and that $ax > b$.

The separation problem is the heart of the cutting plane method, and also an important part of the branch-and-cut algorithm. However, most of the inequalities known today are special classes of inequalities for different special structures, and in most cases, they only provide a partial description of the special structure for which they are facets or faces. A novel approach was used by Applegate et al. [5, 6, 7], where they find violated inequalities *on the fly* for the TSP. In this section we start by providing evidence that trying to find all facets (or classes of facets) for MIP problems seems to be hopeless. We then move on to explain the method proposed by Applegate et al. but in the case of general MIP problems. We end by showing that in theory, we can get any violated face of a polyhedron by this approach, and show some examples where this approach may work well in practice.

2.2.1 Facet Description of MIPs

In this section we try to provide some evidence that finding complete descriptions of the convex hull of mixed-integer sets is *hard* in the sense that the number of facets needed to describe the convex hull of a problem may grow exponentially with the dimension of the problem.

Christof and Reinelt [25] computed all (or a large subset of the) facets for several small MIP problems. We reproduce in Table 2.1 their results for the symmetric traveling salesman problem with up to 10 nodes (TSP_n). In Table 2.2 we reproduce their results for the *linear ordering* polytope, which is the convex hull of all characteristic vectors of acyclic tournaments on a complete directed graph on n nodes (LOP_n). Table 2.3 reproduces their results for the *cut* polytope, which is the convex hull of all characteristic vectors of edge cuts for a complete undirected graph on n nodes ($CUTP_n$). Finally, Table 2.4 reproduces their results on some 0-1 polytopes with many facets.

Table 2.1: Facet Structure for TSP_n.

n	Vertices	Facets	Classes
3	1	0	0
4	3	3	1
5	12	20	2
6	60	100	4
7	360	3,437	6
8	2,520	194,187	24
9	20,160	42,104,442	192
10	181,440	$\geq 51,043,900,866$	$\geq 15,379$

Table 2.2: Facet Structure for LOP_n.

n	Vertices	Facets	Classes
3	6	8	2
4	24	20	2
5	120	40	2
6	720	910	5
7	5,040	87,472	27
8	40,320	$\geq 488,602,996$	$\geq 12,231$

Table 2.3: Facet Structure for $CUTP_n$.

n	Vertices	Facets	Classes
3	4	4	1
4	8	16	1
5	16	56	2
6	32	368	3
7	64	111,764	11
8	128	$\geq 217,093,472$	≥ 147
9	256	$\geq 12,246,651,158,320$	$\geq 164,506$

It is interesting to note that in all examples presented by Christof and Reinelt, the number of both facets and classes of facets grow quite rapidly, making one presume that the situation is worse as we look at larger instances. But a natural question to ask is if this

Table 2.4: 0-1 polytopes with many facets

Dimension d	Vertices v	Facets f	$\sqrt[d]{f}$
6	18	121	2.22
7	30	432	2.37
8	38	1,675	2.52
9	48	6,875	2.66
10	83	41,591	2.89
11	106	250,279	3.09
12	152	$\geq 1,975,937$	≥ 3.34
13	254	$\geq 17,464,365$	≥ 3.60

behavior is restricted to structured problems, or is it a common behavior of the convex hull of general combinatorial sets?

To give a partial answer to this question, consider $P \subseteq \{0,1\}^d$, and define $f(P)$ as the number of facets defining its convex hull. Call $\mathcal{P}_n$ the set of all 0-1 polytopes in dimension n, i.e. $\mathcal{P}_n = \{\text{Conv}(P) : P \subseteq \{0,1\}^n\}$, and define $f_n = \max\{f(P) : P \in \mathcal{P}_n\}$. It is easy to see that $f_n \leq 2n!$. This trivial upper bound was improved to $30(n-2)!$ by Fleiner, Kaibel and Rote [37]. The problem of obtaining lower bounds for f_n was open until recently, Bárány and Pór [13] proved that

$$f_n \geq \left(\frac{cn}{\log n}\right)^{n/4}$$

for some positive constant c. This bound was further improved by Gatzouras et al. [45] to

$$f_n \geq \left(\frac{cn}{\log^2 n}\right)^{n/2}$$

for some positive constant c.

Note that these bounds reflect worst case bounds, and again, one may ask whether this behavior is common or if it is the result of a few pathological cases.

Fortunately, we can give a partial answer to this question. Let us define $P_{n,N}$ as the convex hull of N independently chosen random points drawn from the n-dimensional sphere. Bushta et al. [23] showed that there exists two constants $c_1, c_2 > 0$ such that

$$\left(c_1 \log \frac{N}{n}\right)^{n/2} \leq \mathbb{E}[f(P_{n,N})] \leq \left(c_2 \log \frac{N}{n}\right)^{n/2}$$

for all n and N satisfying $2n \leq N \leq 2^n$. Thus, if we take $N \approx 2^{\alpha n}$ for some $\alpha < 1$, then we have that

$$\left(\frac{\tilde{c_1} n}{\log n}\right)^{n/2} \leq \mathbb{E}[f(P_{n,N})] \leq \left(\frac{\tilde{c_2} n}{\log n}\right)^{n/2}$$

for some $\tilde{c_1}, \tilde{c_2} > 0$.

Although these previous results are not an actual proof that in general the number of facets grows exponentially with respect of the number of vertices or the dimension of general mixed-integer sets, they provide circumstantial evidence of this. This hints towards the idea that the separation problem over the convex hull of mixed-integer sets should be solved through an *optimization oracle*.

2.2.2 The Optimization Oracle and the Separation Problem

Here we present how through an optimization oracle, one can construct a separation algorithm. We start by defining a theoretical optimization oracle:

> **Definition 2.2 (Optimization Oracle).** We say that OPT is an optimization oracle for a rational polyhedron $P \subseteq \mathbb{R}^n$, if for any $c \in \mathbb{Q}^n$ it asserts that P is the empty set, or provides $x^* \in P \cap \mathbb{Q}^n$ such that $c \cdot x^* \geq c \cdot x$ for all $x \in P$, or provides $r^* \in \mathbb{Q}^n$ a ray in P such that $c \cdot r^* > 0$, $\|r^*\| = 1$, and $c \cdot r^* \geq c \cdot r$ for each ray r of P with $\|r\| \leq 1$.
>
> The output of the algorithm is of the form $(status, \beta, y)$, where $status$ is one of `empty, unbounded` or `optimal`; β contains the optimal value of $\max\{c^t x : x \in P\}$ if the problem has an optimal solution, and y contains the optimal solution or an unbounded ray if the status is `optimal` or `unbounded` respectively.

A Linear Programming Formulation We now give a linear programming formulation to the separation problem. Consider P a polyhedron. Note that there exist P_c and P_r such that $P = P_c + P_r$ with P_c a bounded polyhedron, where

$$P_c = \left\{x \in \mathbb{Q}^n : \exists \lambda \in \mathbb{Q}_+^{I_c},\ \sum(\lambda_i v_i = x : i \in I_c),\ e \cdot \lambda = 1\right\} \tag{2.1}$$

with $\{v_i : i \in I_c\} \subset \mathbb{Q}^n$, e the vector of all ones whose number of components is determined by the context, and where

$$P_r = \left\{ x \in \mathbb{Q}^n : \exists \lambda \in \mathbb{Q}_+^{I_r}, \sum (\lambda_i r_i = x : i \in I_r) \right\} \tag{2.2}$$

with $\{r_i : i \in I_r\} \subset \mathbb{Q}^n$.

Then, by definition, a given $x^* \in \mathbb{Q}^n$ is an element of P if and only if there is a solution (λ^c, λ^r) of the system

$$\sum_{i \in I_c} \lambda_i^c v_i + \sum_{i \in I_r} \lambda_i^r r_i = x^*, \quad e^t \lambda^c = 1, \quad (\lambda^c, \lambda^r) \geq 0. \tag{2.3}$$

By duality, system (2.3) has no solution if and only if there is a vector $a \in \mathbb{Q}^n$ and $b \in \mathbb{Q}$ such that

$$\begin{aligned} a^t v_i - b &\leq 0 \qquad \forall i \in I_c && \text{(2.4a)} \\ a^t r_i &\leq 0 \qquad \forall i \in I_r && \text{(2.4b)} \\ a^t x^* - b &> 0. && \text{(2.4c)} \end{aligned}$$

Note that any (rational) cut that separates x^* from P is in one-to-one correspondence with solutions of (2.4). This opens the possibility of *choosing* our cut under some criteria, for example, we could choose to minimize the number of non-zeros in a (i.e. try to find a sparse cut), or minimize the sum of the absolute values of the components of a that correspond to continuous variables, or simply find the most violated cut. Unfortunately, there is no single criteria to identify the *best* cut (or a set of cuts) among all cuts, and moreover, some criteria (such as minimizing the number of non-zeros) seem to require integer variables for their formulation. For the sake of simplicity, we choose to maximize the violation of the cut subject to the normalization $\|a\|_1 = 1$. This problem may be formulated as

$$\begin{array}{lll} \max & a^t x^* - b & \\ s.t. & a^t v_i - b \quad \leq 0, & \forall i \in I_c \\ & a^t r_i \quad \leq 0, & \forall i \in I_r \\ & a - u + v \quad = 0 & \\ & e^t(u + v) \quad = 1 & \\ & u, v \geq 0. & \end{array} \tag{2.5}$$

A drawback of formulation (2.5) is that the number of constraints is $|I_r| + |I_c| + n + 1$, which might be quite large. This, plus the knowledge that in practice the number of simplex iterations tends to grow linearly with respect to the number of constraints other than bounds and logarithmically with respect to the number of variables, suggests that instead of solving (2.5) we should solve its dual,

$$\begin{array}{lll} \min & s & \\ s.t. & \sum\limits_{i\in I_c} \lambda_i^c v_i + \sum\limits_{i\in I_r} \lambda_i^r r_i + w & = x^* \\ & e^t\lambda^c & = 1 \\ & -w + se^t & \geq 0 \\ & w + se^t & \geq 0 \\ & \lambda^r, \lambda^c \geq 0. & \end{array} \tag{2.6}$$

Note that (2.6) has only $3n+1$ constraints other than bounds on individual variables, thus improving (in most cases) from the bound $|I_r| + |I_c| + n + 1$ in the case of (2.5). However, if we choose to minimize $\|a\|_1$ subject to the constraint $a^t x^* = b + 1$, then we obtain the following formulation for the problem:

$$\begin{array}{llll} \min & e^t(u+v) & & \\ s.t. & a^t v_i - b & \leq 0, & \forall i \in I_c \\ & a^t r_i & \leq 0, & \forall i \in I_r \\ & a - v + u & = 0 & \\ & a^t x^* - b & = 1 & \\ & u, v \geq 0, & & \end{array} \tag{2.7}$$

whose dual is

$$\begin{array}{lll} \max & s & \\ s.t. & sx^* - \sum\limits_{i\in I_c} \lambda_i^c v_i - \sum\limits_{i\in I_r} \lambda_i^r r_i + w & = 0 \\ & -s + e^t\lambda^c & = 0 \\ & \lambda^c, \lambda^r \geq 0, \quad -e \leq w \leq e. & \end{array} \tag{2.8}$$

Several observations should be made, first, problem (2.8) has only $n+1$ constraints other than bounds on individual variables, which is a notable improvement from formulation (2.6), which has $3n+1$ constraints. Second, problem (2.8) is trivially feasible (the all zero solution is always feasible), and then, if problem (2.8) has an optimal solution, its dual gives us a separating inequality for P and x^*, and if the problem is unbounded, then the unbounded ray provides us with a decomposition of x^* into elements in P_c and P_r, thus providing us with a proof that $x^* \in P$. Finally, note that problems (2.5) and (2.7) are essentially the same. More precisely, assuming that the separation problem is feasible, then if (a, b) is an optimal solution for (2.5), then $a^t x^* - b > 0$, and then $(\tilde{a}, \tilde{b}) = (a/(a^t x^* - b), b/(a^t x^* - b))$ is feasible for (2.7). Moreover, $(\tilde{a}, \tilde{b})$ is optimal for (2.7), if not, then there exists a separating cut (a', b')

such that $a'^t x^* - b' = 1$ and such that $\|a'\|_1 < \|\tilde{a}\|_1 = \|a\|_1/(a^t x^* - b) = 1/(a^t x^* - b)$, but this implies that $a^t x^* - b < 1/\|a'\|_1 = (a'^t x^* - b)/\|a'\|_1$, which contradicts the optimality of (a, b) for (2.5). This proves that any optimal solution of (2.5) is an optimal solution of (2.7) (after scaling) and any optimal solution of (2.7) is an optimal solution for (2.5) (after scaling).

In the remaining of the text we will always assume that the separation problem is formulated as (2.8), unless otherwise stated.

Using the Optimization Oracle One of the obvious problems with our linear programming formulation of the separation problem is that the number of variables can be quite large. It is in this part that the optimization oracle plays an important role.

Fortunately, it is possible to solve (2.8) even when writing down all columns is impossible. The technique was introduced by Ford and Fulkerson [62] and by Jewell [60], and is known as *delayed column generation*, or *column generation* for short.

Algorithm 2.1 Separation through optimization oracle SEP(OPT, x^*, I_c, I_r)

```
Require: $OPT(c)$ Optimization oracle for $P$.
    $x^*$ point to be separated from $P$.
    $\{I_c\}$ Initial set of feasible points in $P$.
    $\{I_r\}$ Initial set of rays of $P$.
loop
    Solve (2.8) over $I_c$ and $I_r$.
    if (2.8) is unbounded then
        return $x^* \in P$.
    end if
    let $a, b$ be an optimal dual solution to (2.8)
    $(status, \beta, y) \leftarrow OPT(a)$.
    if $status =$ unbounded then
        $I_r \leftarrow I_r \cup \{y\}$.
    else if $status =$ optimal and $\beta > b$ then
        $I_c \leftarrow I_c \cup \{y\}$.
    else
        return $x^* \notin P$, $(a, b)$
    end if
end loop
```

The idea is to start with some set $I'_c \subseteq I_c$ and $I'_r \subseteq I_r$ (both of which may be empty), and solve (2.8) under the restricted set of columns I'_c and I'_r. If the problem is unbounded, then we have a proof that $x^* \in P$. Otherwise, we obtain a tentative inequality $a^tx \leq b$ that is violated by x^*, We then call the optimization oracle to check whether $P \subseteq \{x : a^tx \leq b\}$ by maximizing a^tx over P. If $P \subseteq \{x : a^tx \leq b\}$, we end with the inequality $a^tx \leq b$. Otherwise, we add a point x' in P such that $a^tx' > b$ to the set I'_c, or a ray r' to I'_r such that $a^tr' > 0$ and repeat the process. An overview of this algorithm can be seen in Algorithm 2.1.

2.2.3 On the Polynomiality and Finiteness of the Separation Algorithm

A first question to answer is whether Algorithm 2.1 is polynomial. Unfortunately the answer is not easy. Since the algorithm relies on an LP problem to find a separating inequality, we need to invoke the ellipsoid method as our algorithm to solve this linear programming problem. Moreover, an inspection of (2.7), suggests to use again the ellipsoid method to solve it, with our optimization oracle as the feasibility oracle for the problem, and then, allowing us to say that the scheme is polynomial as long as the optimization oracle is polynomial.

However, we do not use the ellipsoid method as our algorithm of choice, instead we use the simplex algorithm. Under this setting, polynomiality is not a valid question anymore, but we may still want to know under which conditions the algorithm will terminate. To provide such a guarantee is enough to add some conditions on the output of our optimization oracle.

Condition 2.1 (Extreme Ray Condition). We say that an optimization oracle OPT for a polyhedron P satisfies the *extreme ray condition* if and only if whenever it outputs $(\texttt{unbounded}, \beta, y)$, y is an extreme ray of P.

Condition 2.2 (Extreme Point Condition). We say that an optimization oracle OPT for a polyhedron P satisfies the *extreme point condition* if and only if whenever it output $(\texttt{optimal}, \beta, y)$, y is an extreme point of P.

It is easy to see that whenever an optimization oracle satisfies Condition 2.1 and Condition 2.2, then Algorithm 2.1 will terminate after a finite number of iterations, this is because both the set of all extreme points and extreme rays of any polyhedron P are finite, and because each extreme point and ray can be obtained at most once during the execution of the algorithm.

Note that Condition 2.2 seems to be too restrictive for mixed-integer polyhedrons, where an optimal solution obtained by either dynamic programming, or by branch and bound may not satisfy such a requirement. Fortunately, we can relax Condition 2.2, but we need some definitions.

Definition 2.3 (Integer Projection). Let $P \subseteq \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}$ be a polyhedron with rational data, we define its integer projection

$$P_{int} := \{z \in \mathbb{Z}^{n_2} : \exists x \in \mathbb{R}^{n_1}, (x, z) \in P\}.$$

Moreover, for every $z \in P_{int}$ we define

$$P_z := \left\{(x', z') \in P : z' = z\right\}.$$

Now we present an alternative for Condition 2.2:

Condition 2.3 (Mixed Integer Extreme Point Condition). We say that an optimization oracle OPT for a polyhedron P satisfies the *mixed integer extreme point condition* if and only if whenever it outputs $(\texttt{optimal}, \beta, y)$, $y = (y_1, y_2)$ is an extreme point of P_{y_2}, and $|P_{int}|$ is finite.

Note that the requirement of $|P_{int}|$ in Condition 2.3 is not really restrictive, because P_{int} can always be assumed to be a sub-set of P_c for a suitable choice of P_c.

Final Remarks on the performance of the Separation Algorithm We should note that whenever the separation algorithm returns a separating inequality, the inequality is a face of P, and moreover, the separation algorithm also return a set of points in P satisfying it at equality.

A drawback of the previous results is that they only show that the algorithm terminates after a finite number of steps if we use the simplex algorithm, but also show that if we use the ellipsoid method, then the number of steps (or calls to the optimization oracle) is not only finite, but also polynomial in the size of the problem input. However, in our experiments, the number of calls to the optimization oracle only grows linearly in the dimension of the problem P, with a constant factor of less than 3. This suggests that it may be possible to prove that in general, regardless of the algorithm used to solve the linear programming and the column generation problems, the number of calls to the optimization oracle grows polynomially on the size of the problem. Finally, note that Conditions 2.2,2.1 and Condition 2.3 are sufficient conditions to ensure finiteness of Algorithm 2.1 while using the simplex algorithm. However, the author feels that a proof of finiteness for the simplex-implementation of the algorithm does not need those extra conditions.

2.3 Obtaining high-dimensional Faces

Up to now, we have given a general form to to *solve* the separation problem, provided that we have an optimization oracle for our polyhedron. Although the algorithm guarantees that the obtained cut (if there is one) is *good* in some sense, it does not guarantee it to be a *facet* of P. An example of such a situation can be seen in Figure 2.1.

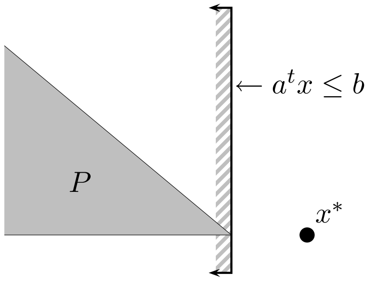


Figure 2.1: Example of a solution of Algorithm 2.1. In this example, the optimal solution to (2.8) is the shown inequality $a^t x \leq b$, which is a face of the polyhedron P, but is not a facet.

The appeal of facets is that they are inequalities that can not be replaced by any other inequality (but for a scaling factor and by adding multiples of equality constraints) in the description of a polyhedron, and also, they are *maximal* (by inclusion) among all possible faces.

In this section we describe an algorithm that transforms the inequality found by Algorithm 2.1 (or any other face of a polyhedron) into a facet of P. This is an extension of a similar algorithm described by Applegate et al. [5, 6, 7] in the context of the TSP.

2.3.1 An Algorithmic Approach

Let us start by defining some notation. Let P be our working polyhedron, and assume it to be non-empty. Let x^* be the point being separated from P, satisfying $x^* \notin P$. Let

$$a^t x \leq b \tag{2.9}$$

be the inequality found by Algorithm 2.1. Let OPT be our optimization oracle for P, and define P_o as the set of feasible points of P satisfying (2.9) at equality found at the end of Algorithm 2.1. Note that since we are assuming that $P \neq \emptyset$, that (2.9) is the result of Algorithm 2.1, and also $x^* \notin P$, then we have that $P_o \neq \emptyset$.

Another detail is that, since we are not assuming that P is full-dimensional, we need to take this into account while looking for facets. In order to do this, we define $P^\perp$, the orthogonal of P, as

$$P^\perp := \left\{x \in \mathbb{R}^n : x^t(y - y_o) = 0, \, \forall y \in P, \text{ and some fixed } y_o \in P\right\}.$$

Note that the choice of y_o is arbitrary, and $P^\perp$ is independent of it. Let $P_o^\perp := \{p_1, \ldots, p_r\}$ be a generating set for $P^\perp$, and let $\bar{x}$ be a point in P such that $a^t\bar{x} < b$, then, by definition, inequality (2.9) is a facet of P if and only if the system

$$\begin{aligned}
w(p_i^t y_o) + v^t p_i &= 0 && \forall p_i \in P_o^\perp && \text{(2.10a)}\\
w - v^t x_i &= 0 && \forall x_i \in P_o && \text{(2.10b)}\\
w - v^t \bar{x} &= 0 && && \text{(2.10c)}\\
(w, v) &\in [-1, 1]^{n+1} && && \text{(2.10d)}
\end{aligned}$$

has as unique solution the all zero vector. Note that condition (2.10d) is not really needed, but it helps to make the feasible region of Problem (2.10) a compact set. Note also that condition (2.10c) ensures that $P \nsubseteq \{x : a^t x = b\}$, i.e. that (2.9) is a proper face of P.

But if there exists a non-trivial solution (w, v) to (2.10), what should we do? One possibility would be to try to find more affine independent points that satisfy (2.9) at equality. Unfortunately, there are some drawbacks for this approach, first, we would need a more deep knowledge of P, in fact, the provided optimization oracle would not help us in this endeavor, even worst, this deeper knowledge of P may not be available. A second problem is that there may be no more affine independent points in P that satisfy (2.9) at equality, just because $a^t x \leq b$ is not a facet of P.

Instead, we take an algorithmic approach. This algorithm allow us to start with a partial description of P_o and of $P_o^\perp$, and ensures that at every iteration it will either increase the dimension of $P_o^\perp$, or increase the dimension of P_o while possibly modifying the current inequality, or finish with the result that (2.9) is in fact a facet for P, or proves that (2.9) is a valid *equation* for P. We do this in such a way as to ensure that we have a *good* violation, and also that all access to P is through our optimization oracle OPT.

Ensuring a Proper Face Our first problem is to either find out that our current separating hyperplane $a^t x \leq b$ is a proper face of P, or if it is a valid equation for P. Fortunately, this has an easy answer, we just maximize over P the function $-a$; if the problem is unbounded, then we can easily find a point $\bar{x} \in P$ such that $a^t \bar{x} < b$; if on the other hand the problem has an optimal solution with value different from $-b$, then such an optimal solution provide us with the sought $\bar{x} \in P$; otherwise, the optimal value is $-b$, thus proving that $a^t x = b$ is a valid equation for P that is violated by x^*.

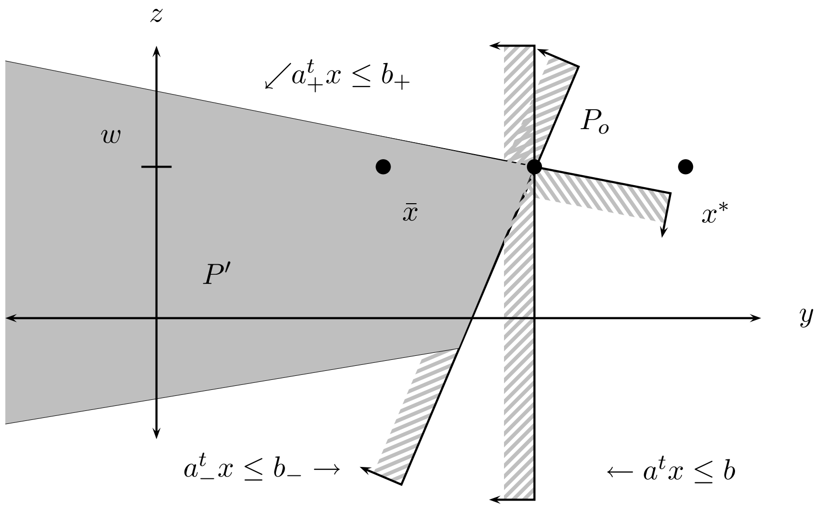
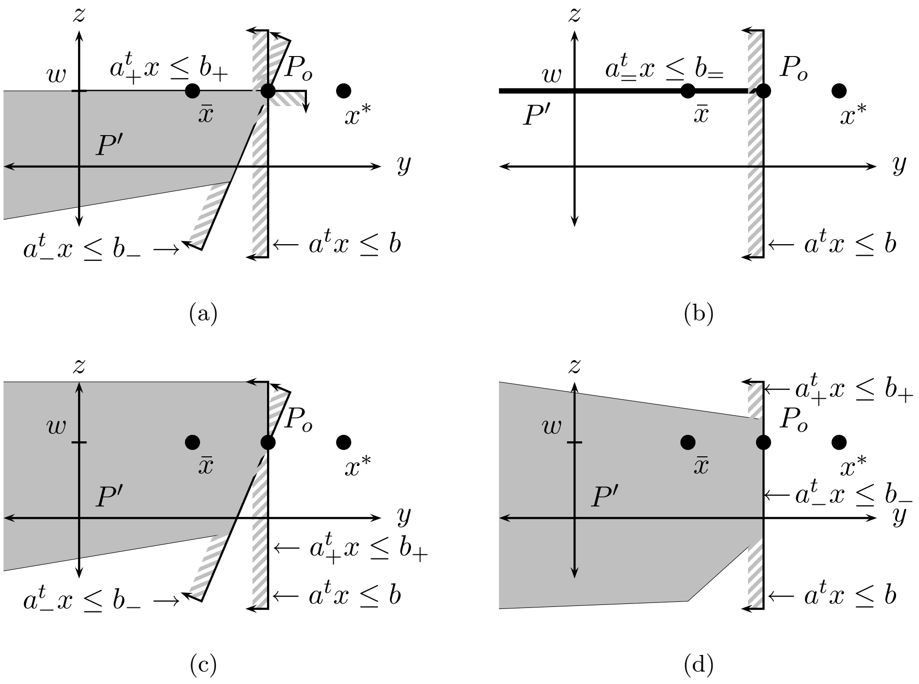

z
$\swarrow a_+^t x \leq b_+$
P_o
w
$\bar{x}$
x^*
P'
y
$a_-^t x \leq b_- \rightarrow$
$\leftarrow a^t x \leq b$

Figure 2.2: Non-Facet Certificates I. Here we show a possible outcome for the mapping of P through $a^t x$ and $v^t x$, the points $\bar{x}, P_o$ and x^* refer to their projection into P', as well as all the inequalities. The gray area represent P'.

The (non) Facet Certificate Once we have $\bar{x}$, and a proof that the only solution to (2.10) is the all-zero vector, we finish with a certificate that the current inequality is a facet.

If, on the other hand, we find a non-zero solution (w, v), the natural question to ask is whether this inequality can be used in any form to achieve our goal of either increasing the dimension of P_o or of $P_o^{\perp}$. Before answering this question, let us give some intuition: Consider $P' := \{(y, z) \in \mathbb{R}^2 : \exists x \in P, y = a^t x, z = v^t x\}$. Figure 2.2 shows how P' might look like. The idea that we will use, is to *tilt* our current inequality $a^t x \leq b$, using as pivot the set P_o, using as rotating direction the vector (v, w), until we *touch* the border of P, identifying then a new linearly independent point. Figure 2.2 also shows two resulting inequalities $a_+^t x \leq b_+$ and $a_-^t x \leq b_-$, both of them with dimension one more than the original constraint $a^t x \leq b$. It is easy to see that, if we restrict ourselves to move in this two-dimensional space, these two inequalities are all inequalities that we can find by *rotating* our original constraint $a^t x \leq b$.



z
w
$a_+^t x \leq b_+$
P_o
$\bar{x}$
x^*
P'
y
$a_-^t x \leq b_- \rightarrow$
$\leftarrow a^t x \leq b$
(a)
z
w
$a_=^t x \leq b_=$
P_o
P'
$\bar{x}$
x^*
y
$\leftarrow a^t x \leq b$
(b)
z
w
P_o
$\bar{x}$
x^*
P'
y
$\leftarrow a_+^t x \leq b_+$
$a_-^t x \leq b_- \rightarrow$
$\leftarrow a^t x \leq b$
(c)
z
$\leftarrow a_+^t x \leq b_+$
w
P_o
$\bar{x}$
x^*
P'
$\leftarrow a_-^t x \leq b_-$
y
$\leftarrow a^t x \leq b$
(d)

Figure 2.3: Non-Facet Certificates II. Here we show some of the possible ill-behaving outcomes for the mapping of P through $a^t x$ and $v^t x$, the points $\bar{x}, P_o$ and x^* refer to their projection into P', as well as all the inequalities. The gray area represent P'.

The Facet Procedure Assuming that we can perform this tilting procedure, we still must bear in mind that Figure 2.2 is just one possible outcome for P', Figure 2.3 shows four ill-behaving outcomes: Figure 2.3(a) is an example where one of the resulting inequalities coincides with $v^t x \leq w$. Figure 2.3(b) shows an example where $v^t x = w$ is in fact a valid equation for P, and then giving us a new point v to add to P_o^t. Figure 2.3(c) shows an example where one side of the tilting is in fact our original inequality $a^t x \leq b$, and Figure 2.3(d) shows an example where both sides of the tilting are our original inequality. We will provide a tilting algorithm in Section 2.3.2, but in the meantime, let us assume that we do have a tilting routine with the following characteristics:

Condition 2.4 (Abstract Tilting Procedure: $\texttt{TILT}(a, b, v, w, \bar{x}, P_o, OPT)$**).**

Input The input of the algorithm should satisfy all of the following:

- $a^t x \leq b$ a face of P, and $P_o \subset P$ set of points satisfying it at equality.
- $v^t x \leq w$ an inequality linearly independent from $a^t x \leq b$, satisfied at equality by all points in P_o.
- $\bar{x} \in P$ such that $a^t \bar{x} < b$ and $v^t \bar{x} = w$.
- OPT optimization oracle for P.

Output The output should be one of the following:

- ($\texttt{unbounded}$,a, b, r), where r is a ray for P that satisfies $a^t r = 0$ and $v^t r > 0$.
- ($\texttt{optimal}$,$v', w', \bar{x}'$) where (v', w') is a non-negative combination of (v, w) and of (a, b) and such that $\{\max v'^t x : x \in P\} = w'$, and where $\bar{x}' \in P$ such that $v'^t \bar{x}' = w'$ and such that $\bar{x}'$ is affine independent from P_o.

Note that if we have a non-facet certificate $(v, w, \bar{x})$ for the inequality $a^t x \leq b$ with point set P_o and equation set $P_o^\perp$, then we can obtain both a_+, b_+ and a_-, b_- with the calls

$$(status_1, a_+, b_+, \bar{x}_1) = \texttt{TILT}(a, b, v, w, \bar{x}, P_o, OPT), \quad \text{and}$$
$$(status_2, a_-, b_-, \bar{x}_2) = \texttt{TILT}(a, b, -v, -w, \bar{x}, P_o, OPT).$$

With this information, we can the finish our facet procedure. If $status_1 = \texttt{optimal}$ and $(v_+, w_+) = (v, w)$, and $status_2 = \texttt{optimal}$ and $(v_-, w_-) = (v, w)$, then we are in the situation depicted in Figure 2.3(b), and we can add v to $P_o^\perp$, increasing its dimension by one, and repeat the process.

In any other situation we have two inequalities (possibly the same one), and we pick the one which is most violated by x^*; assuming that a_+, b_+ is the most violated one, then, we replace a, b with a_+, b_+. If $status_1 = \texttt{optimal}$ we add $\bar{x}_1$ to P_o, otherwise, we add $\bar{x}_1 + x$

Algorithm 2.2 FACET$(a, b, x^*, P_o, P_o^\perp, OPT)$

```
Require: $a^t x \leq b$ face of $P$ such that $a^t x^* > b$.
    $\emptyset \neq P_o \subset P$ such that $a^t x = b$ for all $x \in P_o$.
    $P_o^\perp \subset P^\perp$ and $OPT$ optimization oracle for $P$.
$x_o \leftarrow x \in P_o$   /* select some point of $P_o$.   */
loop
    /*   find proper face certificate   */
    $(status, \beta, y) \leftarrow OPT(-a)$.
    if $status$ = optimal and $\beta = -b$ then
        return (equation, $a, b$)   /* $P \subseteq \{x : a^t x = b\}$   */
    else if $status$ = unbounded then
        $\bar{x} \leftarrow y + x_o$.
    else
        $\bar{x} \leftarrow y$.
    end if
    /*   get (non-)facet certificate   */
    if $(0,0)$ is the unique solution for Problem (2.10) then
        return (facet, $a, b$)
    end if
    $(v, w) \leftarrow$ a non-trivial solution for Problem (2.10).
    $(status_+, a_+, b_+, \bar{x}_+) \leftarrow$ TILT$(a, b, v, w, \bar{x}, P_o, OPT)$.
    $(status_-, a_-, b_-, \bar{x}_-) \leftarrow$ TILT$(a, b, -v, -w, \bar{x}, P_o, OPT)$.
    /*   update $a, b, P_o, P_o^\perp$   */
    if $status_+ = status_-$ = unbounded then
        $P_o \leftarrow P_o \cup \{x_o + \bar{x}_+\}$.   /* grow dimension of $P_o$   */
    else if $status_+ = status_-$ = optimal and $(a_+, b_+) = (a_-, b_-) = (v, w)$ then
        $P_o^\perp \leftarrow P_o^\perp \cup \{v\}$.   /* grow dimension of $P_o^\perp$   */
    else
        if $status_\pm$ = unbounded then
            $\bar{x}_\pm \leftarrow x_o + \bar{x}_\pm$.
        end if
        $\lambda_+ \leftarrow \left(a_+^t x^* - b_+\right) / \|a_+\|_1$.
        $\lambda_- \leftarrow \left(a_-^t x^* - b_-\right) / \|a_-\|_1$.
        if $\lambda_+ > \lambda_-$ then
            $(a, b) \leftarrow (a_+, b_+)$.
            $P_o \leftarrow P_o \cup \{\bar{x}_+\}$.   /* grow dimension of $P_o$   */
        else
            $(a, b) \leftarrow (a_-, b_-)$.
            $P_o \leftarrow P_o \cup \{\bar{x}_-\}$.   /* grow dimension of $P_o$   */
        end if
    end if
end loop
```

to P_o where x is any point in P_o, note that since in any case the newly added point to P_o increase its dimension by one.

Since at every step we either increase the dimension of $P_o^{\perp}$ or the dimension of P_o, the algorithm performs at most n iterations, where $P \subset \mathbb{R}^n$. Moreover, in the case where we add one point to P_o, since $a^t x^* - b > 0$, and $\{x : a_+^t x \leq b_+, \, a_-^t x \leq b_-\} \subseteq \{x : a^t x \leq b\}$, then at least one of the tilted inequalities separates x^* from P. In the case where we add a point to $P_o^{\perp}$, since we keep our original constraint, then we still have a separating inequality for P and x^*.

Algorithm 2.2 shows an outline of the complete algorithm, that starts with a face of P, and returns a facet for P or a separating equation for x^*.

2.3.2 Solving the Tilting Problem

We now show an algorithm that performs the tilting procedure as specified by Condition 2.4. We assume that we have some set P_o feasible for P and satisfying at equality the inequality $a^t x \leq b$. We also have another inequality (although it might not be valid for P) $v^t x \leq w$ such that P_o and $\bar{x}$ satisfy it at equality, but also $a^t \bar{x} < b$ and $\bar{x} \in P$. Note that we are interested in obtaining only a_+, b_+, the procedure to obtain a_-, b_- is completely analogous.

Our objective is to find a valid inequality $v'^t x \leq w'$ for P, such that P_o satisfies it at equality, and a new affine independent point $\bar{x}'$ from P_o that also satisfies it at equality. The idea is to use $v^t x \leq w$ as our candidate output constraint, and $\bar{x}$ as our candidate for an affine independent point, but before we can claim this, we must show that $\{\max v^t x : x \in P\} = w$.

If we maximize v over P, and there is an optimal solution with value w, then we are in the situation depicted by Figure 2.3(a) or by Figure 2.3(b). In this case we only need to return $v^t x \leq w$ as our tilted inequality, and report $\bar{x}$ as our new affine independent point.

If the problem is unbounded, then we are in the situation depicted by Figure 2.3(c) or by Figure 2.2. In this case, we have in our hands an extreme ray r of P (returned by the optimization oracle). Note that since $a^t x \leq b$ is a valid inequality for P, then $a^t r \leq 0$. If $a^t r = 0$, then it is enough to take any $x \in P_o$ and report the point $x + r$ as our new affine independent point[2], and return our original inequality $a^t x \leq b$ as our tilted inequality.

If not, in the case where the problem is unbounded, define $x' = x + r$, where r is the ray returned by the oracle, and x is some point in P_o. In the case where there is an optimal solution, define x' as the optimal solution returned by the oracle.

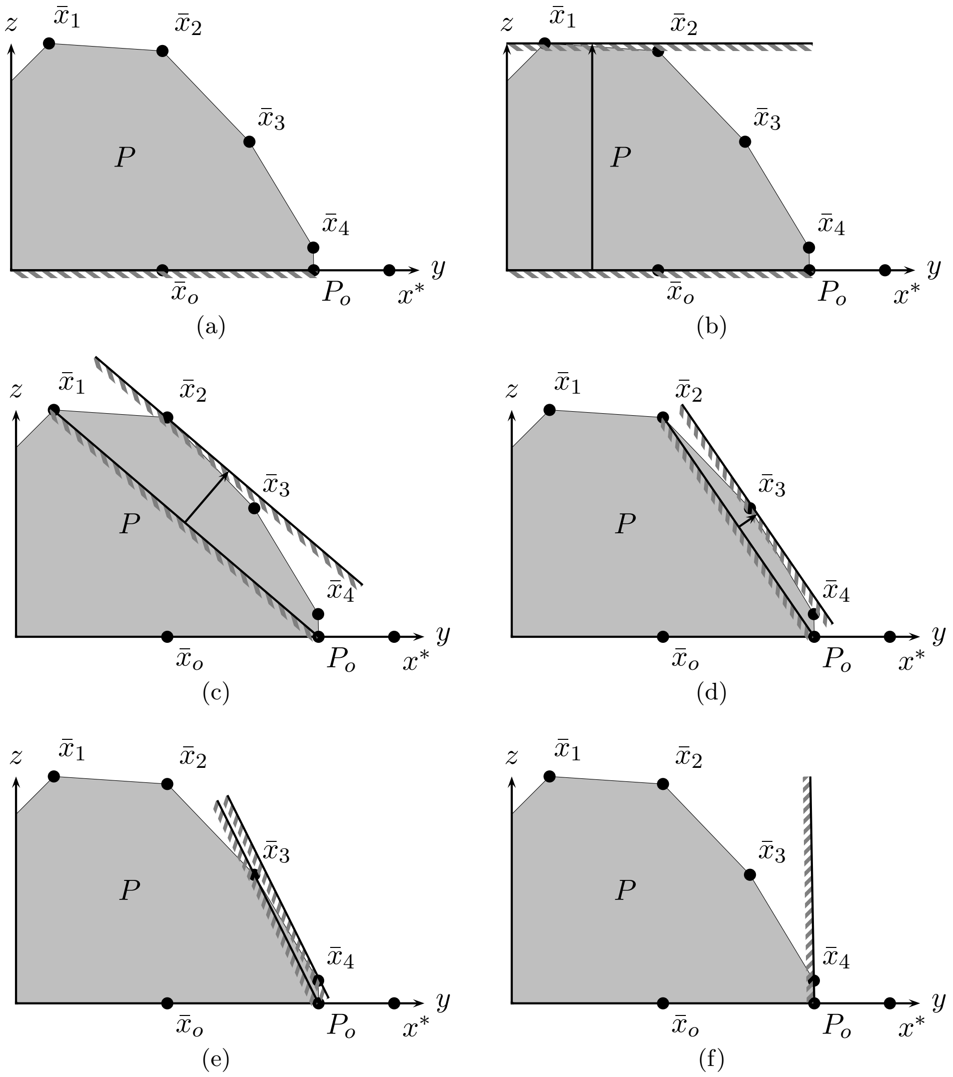


Figure 2.4: Example of a *push* step. Here we show a sequence of push steps that start from some (invalid) inequality $v^t x \leq w$ and rotate it until we obtain a valid inequality for P. The gray area represent P

Note that we are now in the situation depicted in Figure 2.2 or in Figure 2.3(d). Moreover, we have that $x' \in P$, $v^t x' > w$, and thus x' is affine independent from P_o. If $a^t x' = b$,

[2] Note that $x + r$ is affine independent from P_o because $v^t x = w$ for all points in P_o, but $v^t r > 0$.

then we can output our original constraint $a^t x \leq b$ as our resulting constraint, and x' as our new affine independent point. If this is not the case (i.e. $a^t x' < b$), then the trick is to find a positive combination of $v^t x \leq w$ and of $a^t x \leq b$ such that every point in P_o still satisfies it at equality, but such that x' also satisfy it at equality. For that let $\lambda := v^t x' - w$, $\mu := b - a^t x'$, and define the inequality

$$(v', w') = \lambda(a, b) + \mu(v, w). \tag{2.11}$$

Since (2.11) is a positive combination of (a, b) and (v, w), then every point x in P_o satisfies $v'^t x = w'$, but more importantly, x' also satisfies (2.11) at equality. To see this, note that $v'^t x' - w' = \lambda(a^t x' - b) + \mu(v^t x' - w) = -\lambda\mu + \mu\lambda = 0$. Now we replace (v, w) by (v', w') and $\bar{x}$ by x', and we repeat the process of optimizing v over P described above. Note also that

Algorithm 2.3 `TILT`$(a, b, v_o, w_o, \bar{x}_o, x_o, OPT)$

```
Require: $\bar{x}_o \in P$, $a^t \bar{x}_o < b$, $v_o^t \bar{x}_o = w_o$.
    $x_o \in P$, $a^t x_o = b$, $v_o^t x_o = w_o$.
    $OPT$ optimization oracle for $P$.
    $a^t x \leq b$ is a valid inequality for $P$.
    $(a, b)$ and $(v_o, w_o)$ are linearly independent.
$k \leftarrow 0$.
loop
    $(status, \beta, y) \leftarrow OPT(v_k)$
    if $status =$ unbounded and $y \cdot a = 0$ then
        return (unbounded, $a, b, y + x_o$)   /* $y$ is an extreme ray of $P$ */
    end if
    if $status =$ optimal and $\beta = w$ then
        return (optimal, $v_k, w_k, \bar{x}_k$)
    end if
    if $status =$ unbounded then
        $\bar{x}_{k+1} \leftarrow y + x_o$
    else
        $\bar{x}_{k+1} \leftarrow y$
    end if
    $\lambda \leftarrow v_k \cdot \bar{x}_{k+1} - w_k$
    $\mu \leftarrow b - a \cdot \bar{x}_{k+1}$
    if $\mu = 0$ then
        return (optimal, $a, b, \bar{x}_{k+1}$)
    end if
    $v_{k+1} \leftarrow \lambda a + \mu v_k$.
    $w_{k+1} \leftarrow \lambda b + \mu w_k$.
    $k \leftarrow k + 1$.
end loop
```

in the case where the oracle returns unbounded, then new candidate inequality v' satisfies $v' \cdot r = 0$, where r is the ray returned by the oracle.

Every step where we re-define our tentative (v, w) inequality is called a *push* step. Figure 2.4 shows a sequence of push steps that end with the desired inequality. Algorithm 2.3 shows an outline of the tilting algorithm, and also defines the output for it.

Is the Tilting Procedure Correct? Here we show that Algorithm 2.3 satisfies the Condition 2.4. We will only assume that our oracle OPT satisfies the optimization oracle Definition 2.2, $P \neq \emptyset$, $P_o \neq \emptyset$ and that $a^t x \leq b$ is a valid inequality for P.

Claim 2.1. At every step we have that there exists $\lambda_k \geq 0$ and $\mu_k > 0$ such that $(v_k, w_k) = \mu_k(v_o, w_o) + \lambda_k(a, b)$.

Proof. We proceed by induction, the case $k = 0$ being trivially true with $\lambda_o = 0, \mu_o = 1$.

We assume now that the result is true for k, and that the algorithm does not stop during this iteration (otherwise we have finished the proof), then we have that $\bar{x}_{k+1}$ is such that $\lambda = v_k^t \bar{x}_{k+1} - w_k > 0$ and that $\mu = b - a^t \bar{x}_{k+1} > 0$. By definition, $(v_{k+1}, w_{k+1}) = \lambda(a, b) + \mu(v_k, w_k)$, but since $(v_k, w_k) = \lambda_k(a, b) + \mu_k(v_o, w_o)$. Now by defining $\lambda_{k+1} = \lambda + \lambda_k \mu$ and $\mu_{k+1} = \mu\mu_k > 0$ we obtain our result. □

Claim 2.2. At every step we have that:

1. $\bar{x}_k$ and all $x \in P_o$ satisfy $v_k^t x = w_k$.
2. $\bar{x}_k$ is affine independent from P_o.

Proof. We proceed by induction.

Note that for $k = 0$ the result is obvious, since the conditions are assumptions on the input $\bar{x}_o$ and $v_o^t x \leq w_o$.

Now, we may assume that $\bar{x}_k$ and $v_k^t x \leq w_k$ satisfy the conditions of the claim. If the algorithm returns during this iteration with output (`unbounded`, a, b, r), then, by definition, r satisfies $a^t r = 0$ and that $v_k^t r > 0$, but since $v_k = \mu_k v_o + \lambda_k a$, then $0 < v_k^t r = \mu_k v_o^t r + \lambda_k a^t r$ $= \mu_k v_o^t r$. Now since $\mu_k > 0$, then $v_o^t r > 0$.

If the algorithm returns with output (`optimal`, $v_k, w_k, \bar{x}_k$), then by the induction hypothesis $\bar{x}_k$ is affine independent from P_o and $\bar{x}_k \in P$, also $v_k^t x \leq w_k$ is valid for P and all points in P_o satisfy it at equality, and by Claim 2.1, it is a non-negative combination of (v_o, w_o) and (a, b).

Otherwise, since $v_k^t x = w_k$ for all points in P_o and $v_k^t \bar{x}_{k+1} > w_k$, then $\bar{x}_{k+1}$ is affine independent from P_o. Thus, if the algorithm returns (`optimal`, $a, b, \bar{x}_{k+1}$), the output satisfies the Condition 2.4. Otherwise, by definition of v_{k+1}, w_{k+1}, the conditions of Claim 2.2 holds. □

Does the Tilting Procedure Stop? We have proved that if the tilting algorithm stops, the output satisfies the Abstract Tilting Procedure Conditions. It only remains to prove that the algorithm stops after a finite number of steps.

The proof rests on the fact that the set of facets of

$$P' := \{(y, z) \in \mathbb{R}^2 : \exists x \in P, y = a^t x, z = v^t x\}$$

is finite and that the tilting algorithm performs no more than two iterations for each of these facets. We start by proving that if the oracle returns `unbounded`, then it must be during the first iteration of the algorithm, and then we tackle the case of bounded facets of P'.

Claim 2.3. The optimization oracle may return with status `unbounded` only during the first iteration of the tilting procedure.

Proof. By contradiction, assume that for some $k > 0$ the optimization oracle returns with status `unbounded`. Let r be the ray returned by the oracle, then we have that $v_k^t r > 0$ and that $a^t r \leq 0$. By Claim 2.1 we have that $v_k = \lambda_k a + \mu_k v_o$ for some $\lambda_k \geq 0$ and $\mu_k > 0$, then

$$0 \leq -\frac{\lambda_k a^t r}{\mu_k} < v_o^t r.$$

Proving that the problem $\{\max v_o^t x : x \in P\}$ is unbounded. We may thus assume that during our first call to the optimization oracle the output was (`unbounded`,β, r'), where r' is an unbounded ray that maximizes $v_o^t y$ among all rays y of P with $\|y\| = 1$. Note that

$$v_o^t r' \geq v_o^t r \qquad \text{and} \qquad v_k^t r \geq v_k^t r' > 0 \Leftrightarrow \frac{\lambda_k}{\mu_k} a^t r + v_o^t r \geq \frac{\lambda_k}{\mu_k} a^t r' + v_o^t r' > 0.$$

After multiplying the last two inequalities by -1, and adding the term $v_o^t r$, we obtain

$$0 \leq -\frac{\lambda_k}{\mu_k} a^t r' - v_o^t r' + v_o^t r < v_o^t r.$$

By noting that $\lambda_1 = v_o^t r'$ and that $\mu_1 = -a^t r'$, and subtracting the term $v_o^t r$, we obtain

$$\frac{\lambda_k}{\mu_k} < \frac{\lambda_1}{\mu_1}.$$

On the other hand, by Claim 2.1, we have that

$$\frac{\lambda_{k+1}}{\mu_{k+1}} = \frac{\lambda}{\mu \mu_k} + \frac{\lambda_k}{\mu_k} \geq \frac{\lambda_k}{\mu_k} \geq \frac{\lambda_1}{\mu_1}.$$

This contradicts our assumption that the oracle returned `unbounded` for some iteration $k > 0$. □

Now we proceed to prove that for each facet of P', the oracle may return at most two points belonging to the same facet. The idea that we use in the proof is that the quantity λ_k/μ_k can be interpreted as the *slope* of our current inequality $v_k^t x \leq w_k$ when we look at it in the P' space. We link this slope with the slopes of each of the facets of P' in increasing order, so that we can easily identify the possible outputs of the oracle once we look at them in the P' space. We proceed first with some definitions and a claim before proving our main result.

Let $\bar{X} := \{\bar{x}_k\}_{k \in K}$ be the set of points returned by the optimization oracle during the execution of the tilting algorithm (note that we are not assuming this sequence to be finite). We will abuse notation and regard $\bar{x}_k \in \bar{X}$ as an element of P, but also as an element of P', in which case we refer to its projection $(\bar{z}_k, \bar{y}_k) := (v_o^t \bar{x}_k, a^t \bar{x}_k)$.

Let $\mathcal{F}$ be the set of all facets of $P' \cup \{(z, y) : y \geq w\}$, and let $z = \alpha_F(b - y) + z_F$ be the equation defining F in P'. Note that since $(w_o, b) \in P'$ then $z_F \geq w_o$.

We now prove that given $F \in \mathcal{F}$ such that $|F \cap \bar{X}| \geq 3$, then the tilting algorithm stops at one of the corresponding iterations.

Claim 2.4. Given $F \in \mathcal{F}$ with $\bar{x}_{k_1}, \bar{x}_{k_2}, \bar{x}_{k_3} \in F$, then the tilting algorithm stops at or before iteration k_3, where $k_1 < k_2 < k_3$.

Proof. By contradiction, assume that the algorithm does not stop at or before iteration k_3. Then, we have that $0 < b - a^t\bar{x}_{k_i}$ and that $0 < v^t_{k_i-1}\bar{x}_{k_i} - w_{k_i-1}$ for $i = 2, 3$. Thus, by Claim 2.1, we have that

$$\frac{\lambda_{k_1}}{\mu_{k_1}} < \frac{\lambda_{k_2}}{\mu_{k_2}} < \frac{\lambda_{k_3}}{\mu_{k_3}}. \tag{2.12}$$

Note also that from Claim 2.1 we also have that

$$\begin{aligned}\frac{\lambda_{k+1}}{\mu_{k+1}} &= \frac{(v^t_k\bar{x}_{k+1} - w_k) + \lambda_k(b - a^t\bar{x}_{k+1})}{(b - a^t\bar{x}_{k+1})\mu_k} \\ &= \frac{\big(\lambda_k(a^t\bar{x}_{k+1} - b) + \mu_k(v^t_o\bar{x}_{k+1} - w_o)\big) + \lambda_k(b - a^t\bar{x}_{k+1})}{(b - a^t\bar{x}_{k+1})\mu_k} \\ &= \frac{v^t_o\bar{x}_{k+1} - w_o}{b - a^t\bar{x}_{k+1}} = \frac{\bar{z}_{k+1} - w_o}{b - \bar{y}_{k+1}}.\end{aligned} \tag{2.13}$$

Since $\bar{x}_{k_i} \in F$, we have that $v^t_o\bar{x}_{k_i} - z_F = \alpha_F(b - a^t\bar{x}_{k_i})$. Replacing this result in inequality (2.12) we obtain

$$b - a^t\bar{x}_{k_3} < b - a^t\bar{x}_{k_2} < b - a^t\bar{x}_{k_1}. \tag{2.14}$$

Also, from equation (2.13), we have that

$$\frac{\lambda_{k_1}}{\mu_{k_1}} = \alpha_F + \frac{z_F - w_o}{b - a^t\bar{x}_{k_1}} > \alpha_F. \tag{2.15}$$

Note that in (2.15) the strict inequality comes from the fact that $z_F > w_o$; if this is not the case, then it is easy to see that the tilting algorithm would have to stop at iteration $k_1 + 1$, contradicting our hypothesis. On the other hand, the optimality conditions of $\bar{x}_{k_2}$ imply that

$$v^t_{k_2-1}\bar{x}_{k_2} - w_{k_2-1} \geq v^t_{k_2-1}\bar{x}_{k_3} - w_{k_2-1}, \tag{2.16}$$

but for $x \in F$, the function $v^t_{k_2-1}x - w_{k_2-1}$ can be rewritten as

$$(b - a^t x)(\alpha_F\mu_{k_2-1} - \lambda_{k_2-1}) + z_F - w_o. \tag{2.17}$$

Since $k_2 - 1 \geq k_1$, from Claim 2.1 and from (2.15) we have that $\lambda_{k_2-1}/\mu_{k_2-1} > \alpha_F$, then the optimality condition for $\bar{x}_{k_2}$ implies that

$$b - a^t\bar{x}_{k_2} \leq b - a^t\bar{x}_{k_3}, \tag{2.18}$$

contradicting inequality (2.14). This proves our claim. □

From Claim 2.3 and from Claim 2.4, it is easy to see that the number of iterations for the tilting algorithm is bounded by $2|\mathcal{F}| + 2$, thus proving that Algorithm 2.3 is finite.

2.3.3 More on the Facet Procedure

We have shown the correctness of Algorithm 2.2, and we have also shown that if the original inequality separates x^* from P, then our output constraint also separates x^* from P, however, some care must be taken.

On the violation of the output of the facet procedure Since the output of our separation algorithm gives us an inequality that is maximally violated (under L^1 normalization of a), it is clear that the output of our facet procedure can not have a higher violation, but worst, its violation can be arbitrarily small, Figure 2.5 shows an example where no matter how we choose our output constraint, the final violation is very small. However, if instead of limiting ourselves to choose either $a_+^t x \leq b_+$ or $a_-^t x \leq b_-$, we allow us to choose both constraints, then the distance from x^* to the set of points satisfying both constraints is *at least* the distance from x^* to the set of points satisfying our original constraint $a^t x \leq b$, and thus conserving or improving our original violation. This is because the feasible set can only shrink when we consider both tilting results.

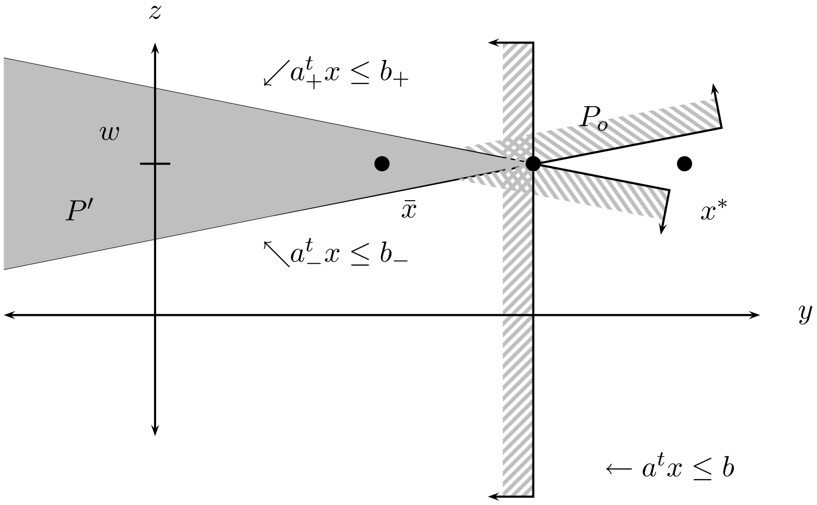

z
$\swarrow a_+^t x \leq b_+$
P_o
w
P'
$\bar{x}$
x^*
$\nwarrow a_-^t x \leq b_-$
y
$\leftarrow a^t x \leq b$

Figure 2.5: Non-Facet Certificates III. Here we show yet another possible outcome for the mapping of P through $a^t x$ and $v^t x$, the points $\bar{x}, P_o$ and x^* refer to their projection into P', as well as all the inequalities. The gray area represent P'.

This hints at a possible modification of our facet al.gorithm, where instead of keeping the most violated inequality, we should save both inequalities and then proceed with the facet procedure on the most violated one. The result is that we report a set of inequalities, which, when considered together, ensure to cut x^* by at least the same amount as the original constraint.

We might take this approach even further, by instead of iterating only in the most violated inequality of the previous iteration, iterate both inequalities through the facet procedure. The advantage of such modification is that we end with a set of facets of P that ensure the same violation as the original inequality, but the drawback is that it may require many more calls to the oracle, slowing down the overall algorithm.

In our actual implementation we choose to keep both inequalities resulting from the tilting procedure (even if they are not violated at all), but we proceed with the facet procedure on the most violated inequality returned by the tilting calls.

High-dimensional faces While the appeal of obtaining facets is clear, it may be that the cost of the overall procedure is too large. In this respect, some rough estimates using Concorde's implementation of the procedure, show that the facet part of the procedure is three to six times more costly than the separation procedure, even when Concorde's implementation heavily exploits the structure of the TSP to speed up calculations. This suggest that, in a more general setting, the procedure to obtain facets can be quite expensive, thus suggesting to stop the facet procedure after a certain number of steps (thus ensuring a minimum dimension for the face).

In this setting, an interesting question is how to choose our non-trivial solution v, w to perform each tilting round?

To answer this question, let us take a closer look into the facet certificate LP. Let us suppose that we have $\bar{x}^* \in P$, a point in the affine space defined by the points in $P_o \cup \{\bar{x}\} = \{x_i : i = 1, \ldots, K\}$. Then, by definition, there exists $\lambda_i \in \mathbb{R}, i = 1, \ldots, K$ such that $\sum(\lambda_i : i = 1, \ldots, K) = 1$ and such that $\sum(\lambda_i x_i : i = 1, \ldots, K) = \bar{x}^*$. This allows us to re-write the facet certificate LP as

$$\begin{aligned} w(y_o^t p_i) + v^t p_i &= 0 && \forall p_i \in P_o^\perp && (2.19a) \\ w - v^t x_i' &= 0 && \forall x_i \in P_o \cup \{\bar{x}\} && (2.19b) \\ w - v^t \bar{x}^* &= 0. && && (2.19c) \end{aligned}$$

Note that Problem (2.19) is equivalent to our original formulation (2.10) but for the normalizing constraints in v, w. Moreover, by replacing w by equation (2.19c) we obtain

$$\begin{aligned} w(y_o^t p_i) + v^t p_i &= 0 && \forall p_i \in P_o^\perp && (2.20a) \\ v^t(x_i - \bar{x}^*) &= 0 && \forall x_i \in P_o \cup \{\bar{x}\} && (2.20b) \\ w - v^t \bar{x}^* &= 0. && && (2.20c) \end{aligned}$$

By choosing $\bar{x}^*$ as the L^2 projection of x^* in the affine space generated by $P_o \cup \{\bar{x}\}$, then the inequality $(x^* - \bar{x}^*)^t x \leq (x^* - \bar{x}^*)^t \bar{x}^*$ is the best inequality separating x^* and the affine space spawned by $P_o \cup \{\bar{x}\}$ in the sense of violation of the inequality, and assuming that its norm is $\|x^* - \bar{x}^*\|_2$. Figure 2.6 shows a possible example of this situation.

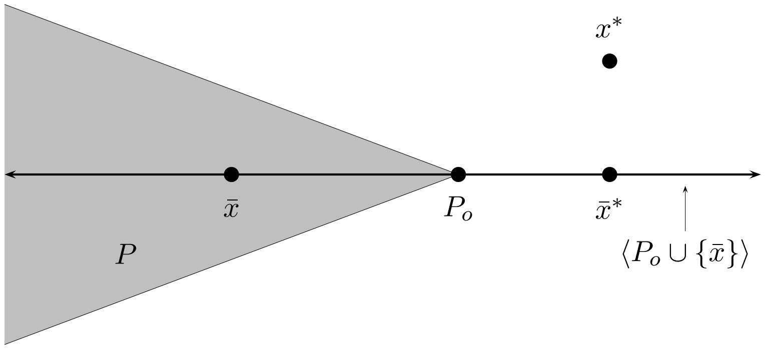


Figure 2.6: Non-Facet Certificates IV. Finding good non-facet certificates. The shaded area represent P, the set $\langle P_o \cup \{\bar{x}\} \rangle$ represent the affine space generated by $P_o \cup \{\bar{x}\}$, and x^* is the point that we are separating from P.

This suggests solving the following problem:

$$\begin{aligned} \max \quad & v^t(x^* - \bar{x}^*) && && (2.21a) \\ w(y_o^t p_i) + v^t p_i &= 0 && \forall p_i \in P_o^\perp && (2.21b) \\ v^t(x_i - \bar{x}^*) &= 0 && \forall x_i \in P_o \cup \{\bar{x}\} && (2.21c) \\ w - v^t \bar{x}^* &= 0 && && (2.21d) \\ \|v\|_2 &\leq 1. && && (2.21e) \end{aligned}$$

Note that the a solution to Problem (2.21) would give us a closest approximation (under L^2 norm) to the ideal solution $x^* - \bar{x}^*$ that also satisfies equation (2.21b).

However, if we are to take this approach, there remains the problem of computing $\bar{x}^*$. Fortunately, that is not needed at all, by back-substituting (2.21d), and then eliminating the linearly dependent equation (2.21d) we end with the following equivalent problem:

$$
\begin{aligned}
\max \quad & v^t x^* - w && && (2.22a)\\
w(y_o^t p_i) + v^t p_i &= 0 && \forall p_i \in P_o^{\perp} && (2.22b)\\
w - v^t x_i &= 0 && \forall x_i \in P_o \cup \{\bar{x}\} && (2.22c)\\
\|v\|_2 &\le 1. && && (2.22d)
\end{aligned}
$$

The only problem with (2.22) is that constraint (2.22d) is not linear, but note that if we replace it with a L^{∞} normalization we should also obtain a reasonably close solution to $x^* - \bar{x}^*$. This is the approach that we took in our implementation, at every step of the facet procedure described in Algorithm 2.2, we choose a non trivial solution (if one exists) of the following problem:

$$
\begin{aligned}
\max \quad & v^t x^* - w && && (2.23a)\\
w(y_o^t p_i) + v^t p_i &= 0 && \forall p_i \in P_o^{\perp} && (2.23b)\\
w - v^t x_i &= 0 && \forall x_i \in P_o \cup \{\bar{x}\} && (2.23c)\\
-1 \le v_i &\le 1. && && (2.23d)
\end{aligned}
$$

Some further questions on the facet procedure Note that the approach that we have described can be seen as a greedy approach for choosing the new tentative $v^t x \le w$ inequality, and although a sensible approach, it is not the only one. Also, we should remember that this approach is very dependent in the choice of $\bar{x}$ (i.e. the affine independent point in P that does not satisfy the current inequality at equality), for which there is a lot of freedom to choose.

It would be interesting to find procedures to choose both $\bar{x}$ and v, w such that our final facet al.so satisfies other properties, for example, that its area is large, or try to find extensions of our oracle model and the facet al.gorithm that would give as an output *all* the facets around a given face of a polyhedron P, thus providing a way to perform IP sensitivity analysis.

2.4 Looking for Easier Problems: The Mapping Problem

Section 2.2 and Section 2.3 provide us with a method to find facets (or high dimensional faces) by using an optimization oracle description of the underlying polyhedron P.

In this section we provide a way to use such a procedure in a cutting plane framework as a source of cuts for our original problem, providing sufficient conditions for the procedure to be successful, and some examples of known inequalities that fit the proposed scheme.

Before going any further, we define some notation that will be used in the following sections. We will call $P_{ip} \subset \mathbb{R}^n$ the linear mixed-integer problem that we want to solve, and we will assume that

$$
\begin{aligned}
(P_{ip}) \quad \max \quad & c^t x && (2.24a)\\
s.t. \quad & Ax \leq b && (2.24b)\\
& l \leq x \leq u && (2.24c)\\
& x_i \in \mathbb{Z} \quad \forall i \in I. && (2.24d)
\end{aligned}
$$

Where $I \subseteq \{1, \dots, n\}$, $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, and $c \in \mathbb{Q}^n$. Furthermore, we will assume that $|I| \geq 1$, i.e. that there is at least one integer requirement for some component of x. We denote by $\overline{P}_{ip}$ as the convex hull of all points in P_{ip}, i.e.

$$\overline{P}_{ip} := \left\{ x \in \mathbb{R}^n : \exists \{x_k, \lambda_k\}_{k \in K} \subset P_{ip} \times \mathbb{R}_+, \sum_{k \in K} \lambda_k = 1, \sum_{k \in K} \lambda_k x_k = x \right\}.$$

Finally, we call P_{lp} as the linear relaxation of P_{ip}, i.e.

$$
\begin{aligned}
(P_{lp}) \quad \max \quad & c^t x && (2.25a)\\
s.t. \quad & Ax \leq b && (2.25b)\\
& A'x \leq b' && (2.25c)\\
& l \leq x \leq u, && (2.25d)
\end{aligned}
$$

where (2.25c) are additional constraints valid for P_{ip}.

2.4.1 Taking Advantage of the so-called Combinatorial Explosion

The typical cutting plane approach to solve P_{ip} starts by solving P_{lp}, obtaining an optimal solution x^* to it, check whether it satisfies the integer requirements (2.24d), if it does, return x^* as our optimal solution, otherwise, find a valid inequality for $\overline{P}_{ip}$ that separates x^* from $\overline{P}_{ip}$, add it to our current relaxation P_{lp}, and repeat the process.

A first naïve approach would be to use our separation procedure over $\overline{P}_{ip}$, of course this implies that in order to solve our separation problem we will need to solve several optimization problems, which may be as hard as solving our original problem, thus rendering the approach useless.

We propose instead to use the so-called *combinatorial explosion* of combinatorial problems to our advantage. Combinatorial explosion, according to Krippendorff "... it occurs when a small increase in the number of elements that can be combined, increase the number of combinations to be computed so fast that it quickly reaches computational limits. E.g., the number of possible coalitions (partitions of unlike individuals into like parts) among 3 individuals is 5, among 5 individuals it is 52, among 10 individuals it is 115,975 and among 20 individuals it is 51,724,156,235,572, etc." [67].

The idea is that instead of working on P_{ip}, we work in a *related* problem P_{ip}' that has a reasonable number of integer variables, that can be solved in a reasonable amount of time, and that hopefully allows us to cut the current fractional point.

We now make our notion of *related* problem precise.

Definition 2.4 (Valid Mapping). We say that $P'_{ip} \subset \mathbb{R}^{n'}$ is a *valid mapping* for $P_{ip} \subset \mathbb{R}^n$, if there exists a function $\pi : \mathbb{R}^n \to \mathbb{R}^{n'}$ such that for all points $x \in P_{ip}$ we have that $\pi(x) \in P'_{ip}$. The function π is called the mapping function.

Note that we may have $\pi(P_{ip}) \subsetneq P'_{ip}$, and moreover, it could be that $\dim(P'_{ip}) > \dim(P_{ip})$. Note also that by the definition, any valid linear mapping will also satisfy $\pi(\overline{P}_{ip}) \subset \overline{P}'_{ip}$.

Although we might want to use mappings that are obtained from general functions π (for example, in a 0-1 IP problem, we might use the function $\pi(x) = x^2$ as our mapping function), linear affine mappings are of special interest for us. The main reason for this is that if we find a valid cutting plane $a^t y \leq b$ in the mapped space $\overline{P}'_{ip}$, then we also have found a valid cutting plane for the original problem, namely $a^t \pi(x) \leq b$, and if we write $\pi(x) = Mx - m_o$, then the cut in the original space can be written as $a^t Mx \leq b + a^t m_o$, and thus providing a cut that preserves the linearity of the original problem once we add the cut to the current LP relaxation P_{lp}. We will consider from now on only linear affine mappings.

Note that the mappings used by Applegate et al. [5, 6, 7] are in fact linear mappings, and moreover, their choice was geared towards obtaining spaces $\overline{P}'_{ip}$ where they could provide efficient optimization oracles, (In fact, they use mappings that map a TSP problem into a GTSP problem on k nodes where at least $k-1$ nodes satisfy the constraint $x(\delta(n_i)) = 2$, and where k is in the range $5 - 48$). Of course this choice comes at a price, while they are able to have very fast oracles[3], only about 1% of the trials are successful[5]. Despite this low success rate, they make heavy use of the special structure of the TSP which allows them to have good overall results.

2.4.2 Mappings with Guarantees

Since one of our objectives is to extend the idea of local cuts to general MIP, we take a slightly different approach than that of Applegate et al. [7]. We would like to find conditions that ensure that our mapping will separate our current fractional point.

In order to do that, we make some further assumptions, namely, we will assume that our current fractional optimal solution is a basic optimal solution for our current LP relaxation P_{lp}, which is a natural assumption in the branch-and-cut framework.

Definition 2.5 (Separating Mapping). Given a MIP P_{ip}, a linear programming relaxation for it P_{lp}, an optimal basic fractional solution x^* of P_{lp}, a mapping function π and an image space P'_{ip}. We say that the mapping is separating if $\pi(x^*) \notin \overline{P}'_{ip}$.

The relevance of separating mappings is that they ensure that if $x^* \notin \overline{P}_{ip}$, then $\pi(x^*) \notin \overline{P}'_{ip}$, and thus ensuring success for the separation algorithm presented in Section 2.2. A necessary condition that a separating mapping has to satisfy is the following:

Condition 2.5 (S1). Given π a mapping for P_{ip}, an image space P'_{ip}, a current LP relaxation P_{lp}, and x^* an optimal basic solution of P_{lp}. If there exists $\mu \in \mathbb{Q}^{n'}$ an objective function for P'_{ip} such that

$$\max\left\{\mu y : y \in \overline{P}'_{ip}\right\} < \mu\pi(x^*),$$

Then we say that the mapping satisfy condition **S1**.

[3] The average running time for their oracle on a couple of TSPLIB problems is $158\mu s^4$, running on a Linux workstation with an Intel P4 with 2.4GHz.

[5] These figures were taken from sample runs of `Concorde` on TSPLIB instances, the actual numbers are 7662 successful separations out of a total of 788089 trials.

Note that condition **S1** is a rephrasing of the definition of separating mappings. To see this it is enough to note that $\overline{P}'_{ip}$ is a convex set and μ serves as a separating linear constraint for $\overline{P}'_{ip}$ and $\pi(x^*)$.

To give further conditions we must know some details about our mapped space. To fix these ideas we need some definitions.

Definition 2.6 (Simple Mapping). Given a MIP problem P_{ip}, a mapping function π, and a mapped space P'_{ip} with representation

$$P'_{ip} = \left\{ y \in \mathbb{R}^{n'} : \begin{array}{c} A'y \leq b' \\ l' \leq y \leq u' \\ y_i \in \mathbb{Z} \quad \forall i \in I' \end{array} \right\},$$

where $I' \subseteq \{1, \ldots, n'\}$, $A' \in \mathbb{Q}^{n' \times m'}$, $b' \in \mathbb{Q}^{m'}$ and $l', u' \in \mathbb{Q}^{n'}$. We say that the mapping π is *simple* if $\pi(P_{lp}) \subseteq P'_{lp}$, where P'_{lp} is defined as

$$P'_{lp} = \left\{ y \in \mathbb{R}^{n'} : \begin{array}{c} A'y \leq b' \\ l' \leq y \leq u' \end{array} \right\},$$

and P_{lp} is the current linear relaxation of P_{ip}.

Note that the notion of a simple mapping is tied with the actual representation of P'_{ip} and of P_{lp}, not just to the mapping function π and the original space and mapped space. Although considering simple mappings seems to be too restrictive, in practice, we always know a linear description of P'_{ip} as shown in Definition 2.6, and a linear description of P'_{lp}. Even better, working with such a fixed description of P'_{ip} and P'_{lp} allows us to give more meaningful conditions on separating mappings.

Condition 2.6 (S2). Given π a simple mapping for P_{ip}, an image space P'_{ip}, a current LP relaxation P_{lp}, and x^* an optimal basic solution of P_{lp}. If there exists $\mu \in \mathbb{Q}^{n'} \setminus \{0\}$ an objective function for P'_{ip} such that

$$\max \{\mu y : y \in \pi(P_{lp})\} \leq \max \left\{\mu y : y \in P'_{lp}\right\} \leq \mu\pi(x^*),$$

then we say that the simple mapping satisfies condition **S2**.

Note that condition **S2** is a necessary condition for separating simple mappings, but not all simple mappings satisfying condition **S2** are separating. Also note that Condition 2.6 can be interpreted as asking $\pi(x^*)$ to be a point on the boundary of P'_{lp}. We will assume from now one that all mappings are simple, unless otherwise stated.

The fact that x^* is a basic solution to P_{lp} allows us to go further. Let us assume that

$$P_{lp} = \left\{ x \in \mathbb{R}^n : \begin{array}{c} Ax = b \\ l \leq x \leq u \end{array} \right\},$$

where we may have added slack variables in the description of P_{lp}; also assume that B is the basis defining x^*, and that $\pi(x) = Mx + m_o$. From Condition 2.6 we have that

$$\max \{\mu M x : Ax = b, l \leq x \leq u\} = \mu M x^*. \tag{2.26}$$

However, our assumptions on x^*, and on the representation of P_{lp}, imply that

$$\begin{array}{rc} \max & \mu Mx \\ s.t. & Ax = b \\ & l \leq x \leq u \end{array} \iff \begin{array}{rc} \max & \mu(M^N - M^B A^{B^{-1}} A^N) x_N \\ s.t. & x_B = A^{B^{-1}}(b - A^N x_N) \\ & l \leq x \leq u, \end{array} \tag{2.27}$$

where A^B is the square submatrix of A with basic columns, M^B is the submatrix of columns of M of all nonbasic variables in x^*, and A^N, M^N are the corresponding nonbasic submatrices of A and M respectively. Note also that $A^{B^{-1}}A^N$ is just the nonbasic part of the tableau rows of A for basis B, i.e. $A^{B^{-1}}A^N = \overline{A}^N$.

Thus, the optimality conditions for equation (2.26), can be written as:

$$\sum_{i=1}^{n'} \mu_i \left(M^N_{ij} - \sum_{k \in B} M^B_{ik} \overline{A}_{kj} \right) \geq 0 \qquad \forall j \in N, x^*_j = u_j \tag{2.28a}$$

$$\sum_{i=1}^{n'} \mu_i \left(M^N_{ij} - \sum_{k \in B} M^B_{ik} \overline{A}_{kj} \right) \leq 0 \qquad \forall j \in N, x^*_j = l_j. \tag{2.28b}$$

equations (2.28) seem to give little intuition, but in fact, they can be of great help in the design of our space P'_{ip} and its linear representation P'_{lp}.

To see this, let us consider the case of MIR cuts. Figure 2.7 shows a typical configuration of a MIR cut. Note that in the case of an MIR inequality we have that

$$P'_{ip} : \begin{array}{c} y_1 - y_2 + y_3 = \hat{b} \\ y_2, y_3 \geq 0 \\ y_1 \in \mathbb{Z}, \end{array} \quad \text{and that} \quad P'_{lp} : \begin{array}{c} y_1 - y_2 + y_3 = \hat{b} \\ y_2, y_3 \geq 0, \end{array}$$

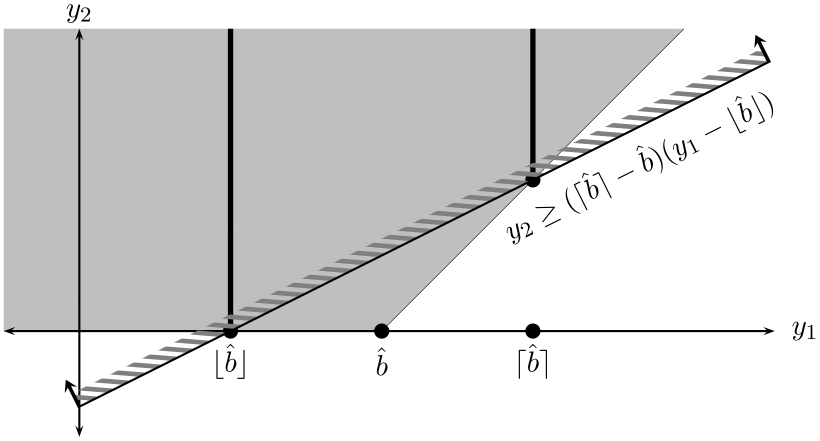

y_2
y_1
$\lfloor\hat{b}\rfloor$
$\hat{b}$
$\lceil\hat{b}\rceil$
$y_2 \geq (\lceil\hat{b}\rceil - \hat{b})(y_1 - \lfloor\hat{b}\rfloor)$

Figure 2.7: The MIR Mapping. Here we show a projection into y_1 and y_2 of the MIR mapped space P'_{ip}, and the resulting MIR cut. The gray area represent P'_{lp}, while the dark thick lines represent P'_{ip}

where $\hat{b}$ is a fractional value. Note also that in the setting of MIR inequalities, we want that $\pi(x^*) = (\hat{b}, 0, 0)$, which is a vertex of P'_{lp} that does not belong to $\overline{P}'_{ip}$. This in turn, by the optimality conditions of **S2**, implies that $\mu_1 + \mu_2 \leq 0$ and that $\mu_1 - \mu_3 \leq 0$. We can choose $\mu = (1, -1, 1)$. Note that this choice of μ does not depend on the value of $\hat{b}$, but rather on the structure of P'_{lp}.

This leaves the question of how to choose M and m_o. In the usual setting of MIR cuts, it is common to derive the MIR cut such that it depends on only one basic fractional variable, this implies that

$$M^B\overline{A} = \left(\lambda_1\bar{a}^k, \lambda_2\bar{a}^k, \lambda_3\bar{a}^k\right)^t,$$

where $\bar{a}^k$ is the tableau row of the basic variable that we have chosen to generate our cut. Moreover, since $\bar{a}^k_k = 1$, and the variable x_k is chosen such that it is integer, this leads to the common choice of $\lambda = (1, 0, 0)$, this decision completely defines M^B as $M^B_{\cdot,j} = 0$ for $j \neq k$ and $M^B_{\cdot,k} = e_1$.

With these choices, and calling $M = (m_{ij})$, we can rewrite equations (2.28) as follows:

$$m_{1j} - m_{2j} + m_{3j} - \bar{a}^k_j \geq 0 \qquad \forall j \in N, x^*_j = u_j \tag{2.29a}$$

$$m_{1j} - m_{2j} + m_{3j} - \bar{a}^k_j \leq 0 \qquad \forall j \in N, x^*_j = l_j. \tag{2.29b}$$

Note however, that the requirement $\pi(P_{lp}) \subseteq P'_{lp}$ can only be enforced by the constraint $m_{1\cdot} - m_{2\cdot} + m_{3\cdot} = \bar{a}^k$, which trivially implies equations (2.29).

An important point to note from the MIR mapping example is the mechanism that is used to ensure that there always is a violated inequality, namely, the choice of $\pi(x^*)$ to be an extreme point of P'_{lp} but such that it does not belong to P'_{ip}, which also implies that $\pi(x^*) \notin \overline{P}'_{ip}$, and then ensuring the existence of a cut separating $\pi(x^*)$ and $\overline{P}'_{ip}$.

We formalize this notion, and end this section with a sufficient condition for simple mappings to be also separating mappings.

Definition 2.7 (Pointed Mappings). We say that a simple mapping is a *pointed mapping* if $\pi(x^*)$ is an extreme point of P'_{lp} and if $\pi(x^*) \notin P'_{ip}$.

We can now enunciate the main theorem of this section:

Theorem 2.1. All pointed mappings are separating mappings.

Proof. It follows from the discussion above. □

2.4.3 Final notes on mappings

Some final notes are necessary at this point. While Theorem 2.1 gives us sufficient conditions for a mapping to be separating, this is not the only way to achieve success while implementing a local cut approach. A clear example of this is the original implementation of Applegate et al. [7].

What seems to be an important component of the conditions defined above is the use of linear simple mappings in general. Those conditions are satisfied by the mappings used by Applegate et al. [7], as well as by the MIR mapping and the more general Fenchel cuts as implemented by Boyd [17], where the mapping reduces to looking at one constraint of the original problem at a time. In Section 2.6 we show how to exploit Theorem 2.1 in order to build more general mappings.

2.5 Putting it all Together

We are now in position to describe the full scheme for local cuts on general MIP. We start by refreshing our memory regarding the building blocks that we have described in the previous sections, we then give a unifying scheme to generate cuts for general MIP problems, and then we move on to discuss some implementation details and modifications that we use in our actual implementation.

2.5.1 The Melting Pot

Section 2.2 describes a general mechanism to separate a point from a polyhedron given in oracle form. The mechanism will either prove that the given point belongs to the convex hull of the given polyhedron, or find a separating inequality which maximizes the violation of the inequality under L^1 normalization.

Section 2.3 provides us with an algorithm that given a face of a polyhedron, and an oracle description of it, it returns a facet of the polyhedron (or a high dimensional face of it), or, if we choose to use the modified version of the algorithm, a set of faces that provide a joint violation as large as the violation of the original inequality.

Section 2.4 gives conditions under which we can map our original problem into a *smaller* or simpler space such that we can recover a violated inequality for the original problem whenever the mapped problem (and point) are separable, and also it provide conditions under which we can guarantee that the mapped problem does not contain the mapped fractional solution.

Note however that Section 2.4 does not provides us with oracles for the mapped problems P'_{ip}, but assuming that the mapped problems are simple enough, it should always be possible to give such description (as a last resort we can always use a branch and bound algorithm as our oracle description for the mapped problem).

2.5.2 The Local Cut Procedure (v0.1)

The procedure for local cuts now can easily be explained. We start with a given MIP problem P_{ip}, a linear relaxation P_{lp}, and a current fractional optimal basic solution x^* to P_{lp}. We move to chose some linear mapping π and an image space P'_{ip}, for which we provide an oracle description $\mathcal{O}_{P'_{ip}}$. We then call Algorithm 2.1 as $\texttt{SEP}(\mathcal{O}_{P'_{ip}}, \pi(x^*), \emptyset, \emptyset)$. If the algorithm finds a separating inequality, we proceed to call Algorithm 2.2 using as input the separating inequality found by Algorithm 2.1, and using as P_o the set of points satisfying the separating inequality at equality, then we can take the resulting inequalities and map them back to the original space and add them to our current relaxation. An outline of the implementation can be seen in Algorithm 2.4.

Algorithm 2.4 LOCAL_CUT(P_{ip}, P_{lp}, c, x^*)

```
Require: $P_{ip}$ original MIP problem being solved,
    $P_{lp}$ current linear relaxation of $P_{ip}$,
    $c$ objective function that we are maximizing/minimizing over $P_{ip}$,
    $x^*$ an optimal basic feasible and fractional solution to $P_{lp}$.
$\mathcal{C} \leftarrow \emptyset$
while some condition do
    $\pi \leftarrow$ some linear mapping of the form $\pi(x) = Mx + m_o$.
    $P'_{ip} \leftarrow$ a related mapped space
    $\mathcal{O} \leftarrow$ oracle description of $P'_{ip}$.
    $I_c \leftarrow \emptyset$, $I_r \leftarrow \emptyset$.
    $(status, a, b) \leftarrow \texttt{SEP}(\mathcal{O}, \pi(x^*), I_c, I_r)$.
    if $status = \pi(x^*) \notin P'_{ip}$ then
        $P_o \leftarrow \{v \in I_c : a^t v = b\}$, $P_o^{\perp} = \emptyset$.
        $\mathcal{C}' \leftarrow \texttt{FACET}(a, b, \pi(x^*), P_o, P_o^{\perp}, \mathcal{O})$.
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{(a^t M, b - a^t m_o) : (a, b) \in \mathcal{C}'\}$.
    end if
end while
return $\mathcal{C}$
```

Note however that there are several possible variations. First we can try several mappings before reporting back our set of cutting planes, and then report a larger set of violated inequalities. Second, note that the call to Algorithm 2.2 is not really mandatory, we could map back the inequality returned by the separation algorithm right away and obtain a separating inequality in our original space without any problem.

We choose to try several mappings at every round of the algorithm (between 5-10), and moreover, instead of asking for real facets of the projected problem, we ask for faces with dimension at least 10, and instead of keeping just the final inequality produced by algorithm FACET, we keep all intermediate constraints. This set of choices are based on experimentation and common sense, however, it may well be that a more in depth study of these parameters may yield different choices.

2.5.3 The Dark Corners of Local Cuts

Although we have discussed some issues relevant to the algorithm, there are some more details that we should take care of.

How much precision is precise? The first problem is related to the accuracy needed whenever we perform optimization calls and optimization of LP problems. These problems are of great importance for both the separation and the facet al.gorithms, which rely on LP formulations to find a separating inequality (in the separation algorithm) and to find new search direction (in the facet al.gorithm). Any error in the solution of the separation LP may easily yield wrong inequalities, and also any error in the solutions obtained from the oracle, will also result in wrong inequalities.

This problem was also observed by Applegate et al. [7]; they deal with it by using integer representation of the inequalities and of solutions, and by using an oracle whose solutions are always integer. The problem with their approach is that it is specially tailored to the TSP. We follow their main ideas by using rational representation of both solutions from oracles and of cuts, and using as a linear solver the code developed in Chapter 1.

However, this still leaves the problem of solving MIPs in rational arithmetic. To fill in this void, we also implemented a branch and bound and cut code that uses our exact LP solver as its core element, details about the features and design of this MIP solver can be found in Section 2.7.

A side effect of working with rational inequalities is that their representation may grow as we go along in the algorithm, and in turn, the addition of inequalities with long representations can generate extreme points that need a larger representation, and thus entering in a vicious cycle of longer and longer representations of optimal points and inequalities. Moreover, it might be the case that we start with an LP relaxation with solutions that require long encoding; Figure 2.8 shows that such cases represent about the 20% of the LPs studied in Chapter 1.

It is hard to give a reason for this behavior. From our computer experiments we know

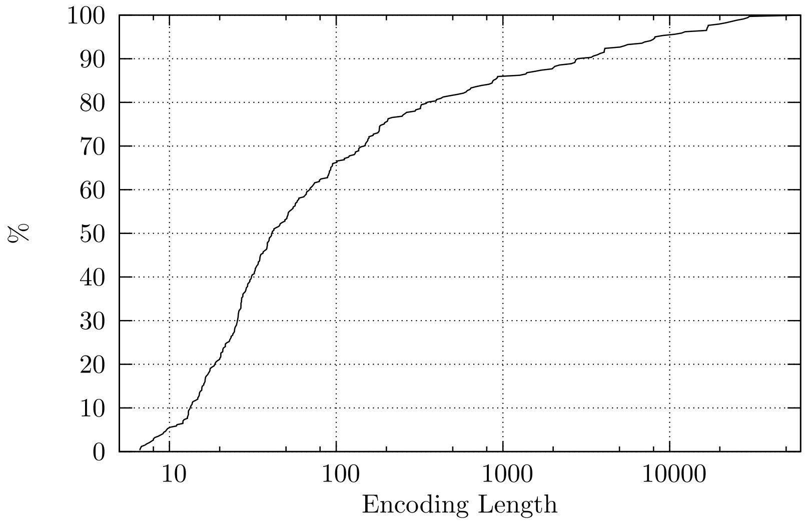


Figure 2.8: Encoding length distribution. Here we show the experimental distribution of the average encoding length for nonzero coefficients in optimal solutions. The data consists of 341 LP problems from MIPLIB, NETLIB and other sources. Note that the x axis is in logarithmic scale.

that even problems with simple coefficients can generate solutions that need a long encoding to be represented, and it seems to be the case that as we allow more and more complicated inequalities, the situation only worsens, while winning very little in the quality of the bounds encountered by the algorithm.

In order to settle this point, we choose to accept cuts such that the denominator and numerator of each coefficient can be expressed using at most some fixed number of bits. Usual values for this limit are between 32 and 256 bits. Note that this is an arbitrary decision made in the light of some experiments.

Are our oracles fortunate? A second issue is related with our definition of optimization oracles. While Definition 2.2 provides us with a suitable (and simple) framework to work with, it does not take into account some practical considerations. For example, we would like to allow our oracle to give-up on hard instances, where the time to solve the given problem is too long to be tolerated, or where the actual computer implementation of the oracle can not deal with the given instance because of memory constraints or because of design problems.

It is also known that, given an MIP and a linear objective function, it is easier to find a solution above (or equal to) some pre-established value (or to prove that no solution above the given threshold exists), than to find an optimal solution for the given objective value. So we would like to consider an oracle that, given an objective and a lower bound, finds a better solution or finds that the given bound can not be improved.

The first point is relevant if we want to have a *fault tolerant* implementation of the local cut procedure, and the changes required in the algorithm are of minor importance. The second point is specially relevant when we look at the tilting procedure as described in Algorithm 2.3, where at every iteration we have a candidate inequality, and we ask whether or not the current inequality is valid for the polyhedron; if not, then is enough to have a point in P such that it violates the current candidate inequality to define a new candidate inequality; if no such point exists, then we know that we are done with the tilting procedure. While using such an *improving oracle* model would invalidate our proof of finiteness for the tilting and separation algorithm, we have seen than in practice working with such an oracle speeds-up the overall performance of the local cut procedure. Note also that this same behavior can be exploited in the separation procedure described in Algorithm 2.1.

These considerations lead us to work with the following oracle model:

Definition 2.8 (Improving Oracle). We say that OPT is an improving oracle for a rational polyhedron $P \subseteq \mathbb{Q}^n$, if for any $c \in \mathbb{Q}^n$ and for any $b \in \mathbb{Q}$ it returns ($\texttt{empty}$,$-,-$) or ($\texttt{fail}$,$-,-$) if $P \cap \{x : c \cdot x \geq b\}$ is the empty set, or it returns ($\texttt{unbounded}$,$r,-$) or ($\texttt{fail}$,$-,-$) if the problem is unbounded, where $r^* \in \mathbb{Q}^n$ is a ray in P such that $c \cdot r > 0$ and $c \cdot r^* \geq c \cdot r$ for all r ray of P with $\|r\| \leq 1$, or it returns ($\texttt{feasible}$,$x^*, x^* \cdot c$) or ($\texttt{fail}$,$-,-$) if the problem is bounded and feasible where $x^* \in P$ and such that $c \cdot x^* > b$, or it returns ($\texttt{optimal}$,x^*, b) or ($\texttt{fail}$,$-,-$) if $c \cdot x^* = \max\{c \cdot x : x \in P\} \geq b$.

As we have already mention before, the use of this optimization oracle model invalidates some of our proofs regarding finite termination and correctness of our algorithms. However, it is possible to define conditions under which this type of oracle would also guarantee correctness and finiteness. One such condition is to assume that the set of possible outputs for the oracle is finite.

Definition 2.9 (Finite Oracle). An oracle $\mathcal{O}$, is said to be finite if there exists a finite set I such that $\mathcal{O}(x) \in I$ for all valid inputs x for $\mathcal{O}$.

Note that neither Definition 2.9 nor Definition 2.8 assume that we have a deterministic oracle. Moreover, for any MIP problem, any simplex-based branch and bound oracle can be seen as a finite and improving oracle, and in the case of pure IP problems, Lenstra's algorithm [61] provides us with a polynomial-time finite improving oracle for fixed dimension.

2.6 Choosing Relevant Mappings

In Section 2.4 we define the notion of general mappings, and also provided conditions under which the class of simple mappings always yield separating mappings. In this section we provide several examples of simple and pointed mappings. In each case we provide some intuition as to why they seem to be reasonable choices, but by no means do they constitute an exhaustive set of complete choices. Indeed, the examples shown here only scratch the surface of the possibilities.

The mappings that we show are presented in the order in which they came to be in our code, rather than in order of importance. The idea behind this decision is to provide a *natural* path of the ideas that lead to them.

2.6.1 Problem Notation

Here we present the notation, assumptions, and representations that we will use throughout this section. We start by defining our original MIP problem P_{ip}, and its current LP relaxation P_{lp}:

$$\begin{aligned} P_{ip}: \qquad Ax &= b && \text{(2.30a)}\\ l \le x &\le u && \text{(2.30b)}\\ x_i &\in \mathbb{Z}\ \forall i \in I, && \text{(2.30c)} \end{aligned}$$

and

$$\begin{aligned} P_{lp}: \qquad Ax &= b && \text{(2.31a)}\\ Ex &= f && \text{(2.31b)}\\ l \le x &\le u. && \text{(2.31c)} \end{aligned}$$

Here we assume that all data is rational, $u \in (\mathbb{Q} \cup \{\infty\})^n$, $l \in (\mathbb{Q} \cup \{-\infty\})^n$, and the constraints (2.31b) are valid constraints for P_{ip}, but note that this set of constraints might be empty; these constraints can be interpreted as the current set of cuts added to our LP relaxation.

Note also that the equality sign in equation (2.30a), (2.31a) and (2.31b) are not restrictive in the sense that any inequality can be converted into this format by adding a suitable slack variable. The choice of this representation was made in the light of how actual LP solvers work on problems.

Now we define our mapped integer problem P'_{ip} and its linear relaxation P'_{lp} as follows:

$$P'_{ip}: \qquad A'y = b' \tag{2.32a}$$
$$l' \leq y \leq u' \tag{2.32b}$$
$$y_i \in \mathbb{Z}\ \forall i \in I', \tag{2.32c}$$

and

$$P'_{lp}: \qquad A'y = b' \tag{2.33a}$$
$$l' \leq y \leq u'. \tag{2.33b}$$

2.6.2 Simple Local Cuts

It is well known that although MIP is $\mathcal{NP}$-complete, if we fix the number of integer constraints (i.e. the number of variables that must be integer is fixed), then the problem can be solved in polynomial time[6], even when the number of continuous variables is not fixed.

This suggests the following simple mapping. Set π as the identity function in $\mathbb{R}^n$, define $A' = \binom{A}{B}$, $u' = u$, $l' = l$, and $I' \subseteq I$ such that $|I'| \leq k$ for some fixed k. Note that this mapping is a simple mapping as in Definition 2.6, and also a pointed mapping as in Definition 2.7, thus ensuring that the separation procedure over P'_{ip} will succeed, and we will obtain a valid cut for x^* and P_{ip}.

The oracle that we used was a branch and bound solver which uses some simple cuts like Gomory cuts and cover cuts.

Although this particular mapping satisfies all of the theoretical requirements to be successful (and in practice it does generate quite good bounds), the problem is that the time to solve each oracle call can be quite expensive as we move to larger instances. This is due in

[6] See Lenstra [61] for more details.

part because the number of oracle calls grows also with the dimension of P'_{ip}, and because each oracle call starts to be more expensive.

2.6.3 Integer Local Cuts

One way to address the problem of the increasing number of calls to the oracle due to the dimension of the sub-problem is to *forget* about all the continuous variables and work only in the (projected) space of integer variables selected. More precisely, we select a set $I' = \{i_1, \ldots, i_k\} \subseteq I$ where k is some small number, and define P'_{ip} as follows:

$$P'_{ip} := \left\{ y \in \mathbb{Z}^k : \ \exists x \in P_{ip}, \ x_{I'} = y \right\}. \tag{2.34}$$

In this case our mapping function π is just the (linear) projection operator from $\mathbb{R}^n$ into $\mathbb{R}^{I'}$. However, to make this a simple mapping, we would need to consider P'_{lp} as

$$\begin{aligned} P'_{lp}: \qquad (b - A_{I'}y)\, z + l_{I''}v - u_{I''}w \le 0 \qquad & \forall (z, v, w) \in Q && \text{(2.35a)} \\ l_{I'} \le y \le u_{I'}, \qquad & && \text{(2.35b)} \end{aligned}$$

where $Q = \{(z, v, w) : A_{I''}z + v - w = 0, v, w \ge 0\}$ and where $I'' = \{1, \ldots, n\} \setminus I'$. Even worse, to get a representation for P'_{lp} of the form of (2.33), we would need to compute all extreme rays T of Q, with which we can rewrite (2.35) as

$$\begin{aligned} P'_{lp}: \qquad (b - A_{I'}y)\, z + l_{I''}v - u_{I''}w \le 0 \qquad & \forall (z, v, w) \in T && \text{(2.36a)} \\ l_{I'} \le y \le u_{I'}. \qquad & && \text{(2.36b)} \end{aligned}$$

Fortunately, this is not necessary, we instead use an oracle definition of (2.34) that internally works in the full formulation of the problem, with a cost coefficient of zero for all variables outside I', but that only reports the solution on the selected set of variables I'.

While this approach succeeded in reducing the number of calls to the oracle due to large dimensional problems, it fails in two other important aspects. First, the mappings are usually not separating, and second, the time spent in each oracle call grows with the number of inequalities in our current LP relaxation and with the dimension of the original problem. Note that while the problem of expensive oracle calls might be overcome by computing (2.36), the first point seems to be more difficult to overcome.

Our experience with this approach seem not to be promising. On the other hand, the local cut implementation of Applegate et al. [7] fits precisely this scheme. The difference is that they usually work on spaces P'_{ip} with dimension ranging between 136 and 1176, while our experiments worked on the range of 3 to 6 for the dimension of P'_{ip}. Nevertheless, the success rate that Applegate et al. achieve is about 1%, but they are able to offset this by having a very fast oracle implementation for their problem, while we rely on a general implementation of branch and bound.

2.6.4 Gomory-Projected Local Cuts

An alternative to fix the problem of non-separating mappings is to consider Gomory-like projections, more precisely, given an integer variable x_k with fractional optimal solution x^*_k, we know that it must be a basic variable, then, from the tableau row, we know that x_k satisfies

$$x_k + \sum_{i\in N} \bar{a}_i x_i = \hat{b},$$

where N is the set of nonbasic variables, and $\bar{a}$ is the associated tableau row for variable x_k in the current LP relaxation. We then define three aggregated variables y^k_1, y^k_2 and y^k_3 such that $y^k_2, y^k_3 \geq 0$ and y^k_1 is integer that satisfy $y^k_1 + y^k_2 - y^k_3 = x_k + \sum(\bar{a}_i x_i : i \in N)$, i.e.

$$\begin{aligned} y_1 &= x_k + \sum(m^1_i x_i : i \in N) && (2.37a)\\ y_2 &= \sum(m^2_i x_i : i \in N) && (2.37b)\\ y_3 &= \sum(m^3_i x_i : i \in N) && (2.37c)\\ \bar{a}_i &= m^1_i + m^2_i - m^3_i \qquad \forall i \in N. && (2.37d) \end{aligned}$$

We also have the conditions $m^1_i \in \mathbb{Z}$ and $m^1_i = 0$ for all $i \notin I$, and we choose $m^2_i, m^3_i < 1$ for all $i \in I$ and satisfying $m^2_i \cdot m^3_i = 0$ for all $i \in N$. Note that these conditions do not completely define m^j_i, in fact, there are at least two well known choices for these coefficients, one is to use Gomory rounding to define m^1_i as the round down of $\bar{a}_i$ for integer variables, and set m^2_i, m^3_i as the remaining part of $\bar{a}_i$, and the second choice is to use mixed integer rounding coefficients, where m^1_i is either the ceiling or the floor of $\bar{a}_i$ depending on the fractional value of x^*_k and of $\bar{a}_i$. We call the first approach Gomory-projection, and the second MIR-projection. Note also that we can write $y^k = M^k x$ where M^k is the matrix with coefficients m_{ij} as defined above.

The procedure starts by selecting some small set I' of integer variables with fractional coefficients, and for each of them define the aggregated variables y^k_i for $i = 1, 2, 3$ and $k \in I'$, using one of the two approaches defined above, with that, we define P'_{ip} as follows:

$$P'_{ip} = \left\{ y = (y^k)_{k\in I'} : \ \exists x \in P_{lp},\ y = Mx,\ y^k_1 \in \mathbb{Z}\ \forall k \in I' \right\},$$

where M is the matrix obtained by appending all M^k matrices. Note however that we can discard the variable y^k_3 because it satisfies $y^k_1 + y^k_2 - y^k_3 = \hat{b}$, and then the dimension of P'_{ip} is no more than $2|I'|$. Note also that since each y^k is a simple pointed mapping, and P'_{ip} can be seen as the product of y^k spaces, then P'_{ip} is also a simple pointed mapping.

Unfortunately, this mapping suffers the same representability problems that the previous mapping has. We again avoid this problem by providing an oracle that works on the following problem.

$$\begin{aligned}
P_{lp}: \qquad & \max\ c^t y && \text{(2.38a)}\\
& \quad Ax = b && \text{(2.38b)}\\
& \quad Ex = f && \text{(2.38c)}\\
& \quad y = Mx && \text{(2.38d)}\\
& \quad l \le x \le u && \text{(2.38e)}\\
& \quad y^k_1 \in \mathbb{Z} \qquad \forall k \in I'. && \text{(2.38f)}
\end{aligned}$$

While this form of mapping ensures that we can always find a violated cut, some unexpected issues start to play a role, namely, the length of the encoding of the obtained inequality. To our surprise, the inequalities obtained from our algorithm require, on many problems, above 512 bits to represent some of their coefficients. This forced us (as already mentioned earlier) to forbid constraints with coefficients that needed more than 256 bits to be represented, which in practice meant that we discarded most of the generated cuts.

2.6.5 Minimal Projection Local Cuts

The Gomory-projected and MIR-projected mappings introduced earlier succeed in giving us separating mappings, but they fail in the sense that the coefficients with which we end up can be very large, which in turn make the LP solution process more difficult and lengthy. Although we tried to get around this problem by approximating the cuts with other cuts with nicer coefficients, it turned out that adding approximate cuts made matters worse. We will see this effect in Section 2.7.

By doing some wishful thinking, if we could choose a small set of basic variables and look for cuts involving just that small set of variables. We should get *nice* cuts for the original problem, however, as we have shown, usually this kind of mapping does not give us separating mappings (at least for low dimensional spaces P'_{ip}).

This takes us to the following question, could we modify (in a slight manner) this mapping in such a way as to always have a separating mapping? Fortunately, condition (2.28) ensures that by adding one extra aggregated variable we can *always* have a separating mapping.

To see this we need to fix some ideas. We have a small set of variables $I' \subset B \subset \{1,\dots,n\}$, where B is the set of basic variables and at least one of the elements in I' is integer constrained and has a fractional value in the optimal solution to the current LP relaxation of P_{ip}. Moreover, we want to work on the space

$$P'_{ip} := \left\{ y \in \mathbb{R}^{|I'|+1} : \ \exists x \in P_{lp}, y = Mx, \ y_i \in \mathbb{Z} \forall i \in I' \cap I \right\}.$$

Assuming that $I' = \{i_1,\dots,i_k\}$, then $M_{\cdot j} = e_{i_j}$ for $j = 1,\dots,k$, we may also assume that $m_{k+1,j} = 0$ for all $j \in B$. With this, equations (2.28) reduce to

$$\mu_{k+1} m_{k+1,j} - \sum \left(\mu_i \overline{A}_{ij} : i = 1,\dots,k \right) \geq 0 \qquad \forall j \in N, x^*_j = u_j \tag{2.39a}$$
$$\mu_{k+1} m_{k+1,j} - \sum \left(\mu_i \overline{A}_{ij} : i = 1,\dots,k \right) \leq 0 \qquad \forall j \in N, x^*_j = l_j, \tag{2.39b}$$

where $\bar{A}_{j\cdot}$ is the tableau row associated with the basic variable x_{i_j}. The disadvantage of equation (2.39) is that they provide only necessary conditions for separating mappings. A simple modification, however, gives us a sufficient condition to obtain simple pointed mappings:

$$\mu_{k+1} m_{k+1,j} - \sum \left(\mu_i \overline{A}_{ij} : i = 1,\dots,k \right) \geq \varepsilon \qquad \forall j \in N, x^*_j = u_j \tag{2.40a}$$
$$\mu_{k+1} m_{k+1,j} - \sum \left(\mu_i \overline{A}_{ij} : i = 1,\dots,k \right) \leq -\varepsilon \qquad \forall j \in N, x^*_j = l_j, \tag{2.40b}$$

where ε is any positive (rational) value. The reason equations (2.40) give us sufficient conditions is because if they hold, it implies that $\pi(x^*)$ is the only optimal solution in P'_{lp} to the objective function μ. Note also that there is no feasible solution $(\mu, m_{k+1,\cdot})$ with $\mu = 0$. Moreover, if there is a solution to (2.40) with $\mu_{k+1} = 0$, then there is an equivalent solution with $m_{k+1,\cdot} = 0$. Thus we can assume that $\mu_{k+1} = 1$, and then the problem of finding a feasible solution for equations (2.40) is a linear one. Moreover, given μ, we can always choose $m_{k+1,j} = \left(\varepsilon + \sum(\mu_i \bar{A}_{ij} : i = 1, \ldots, k)\right)^+$ for $j \in N, x^*_j = u_j$ and $m_{k+1,j} = \left(-\varepsilon + \sum(\mu_i \bar{A}_{ij} : i = 1, \ldots, k)\right)^-$ for $j \in N, x^*_j = l_j$ and obtain a feasible solution of (2.40).

The coefficients $m_{k+1,j}$ for $j \in N$ can be seen as the amount of perturbation from the *ideal* mapping into I' to obtain a separating mapping, so our ideal solution would be one that satisfy $m_{k+1,j} = 0$. Thus, the problem of finding a *minimal perturbation* can be stated as follows:

$$\min \sum \left(\alpha_j |m_{k+1}, j| : j \in N\right) \tag{2.41a}$$
$$\mu_{k+1} m_{k+1,j} - \sum \left(\mu_i \overline{A}_{ij} : i = 1, \ldots, k\right) \geq \varepsilon \qquad \forall j \in N, x^*_j = u_j \tag{2.41b}$$
$$\mu_{k+1} m_{k+1,j} - \sum \left(\mu_i \overline{A}_{ij} : i = 1, \ldots, k\right) \leq -\varepsilon \qquad \forall j \in N, x^*_j = l_j. \tag{2.41c}$$

The parameter ε can be seen as a measure of how pointed we want the projected vertex $\pi(x^*)$ to be in P'_{lp}, and the objective coefficients α_j should reflect the relative importance of each of these variables.

In our experiments ε, was chosen in the range of 2^{-20}, while α_j was chosen as 1 for integer *structural* variables, 10 for continuous *structural* variables[7], $|a_{j\cdot}|_1$ for integer *logical* variables and $10|a_{j\cdot}|_1$ for continuous *logical* variables, where $a_{j\cdot}$ represent the row of the constraint matrix A defining the logical variable.

In practice, we do not select beforehand the set I'. Instead, we let the solution of (2.41) define the set I'. In order to do this, we try for some of the integer constrained variables with the most fractional value to have $\mu_j = 1$ or $\mu_j = -1$ and solve the system. We then pick I' as those basic variables with high μ solutions, and aggregate everything else into the remaining variable y_{k+1}. Typically we do this for the five or six most fractional variables.

2.6.6 2-Tableau and 4-Tableau Local Cuts

While the minimum projected mappings improve somewhat the situation from the Gomory and MIR mappings described above, they all share one common drawback, the oracle that

[7] As CPLEX, QSopt defines for every inequality (and equality) a slack or logical variable, the original variables are referred as structural variables.

we use for them effectively works on a space with dimension as large as the original problem (although the number of integer variables is small). This implies that despite the fact that we have separating mappings, the procedure as a whole tends to be slow.

These considerations force us to look into mapped problems P'_{ip} for which we can provide *fast* oracle implementations, and that also ensure separability.

We choose the following problems:

$$\begin{aligned} 2-\text{Tableau}: \qquad & y_1 + y_3 + y_4 - y_5 - y_6 = b_1 && \text{(2.42a)}\\ & y_2 + y_3 - y_4 - y_5 + y_6 = b_2 && \text{(2.42b)}\\ & y_3, y_4, y_5, y_6 \geq 0 && \text{(2.42c)}\\ & y_1, y_2 \in \mathbb{Z}, && \text{(2.42d)} \end{aligned}$$

and

$$\begin{aligned} 4-\text{Tableau}: \qquad & y_1 + y_5 + y_6 + y_7 + y_8 - y_9 - y_{10} - y_{11} - y_{12} = b_1 && \text{(2.43a)}\\ & y_2 + y_5 - y_6 + y_7 - y_8 - y_9 + y_{10} - y_{11} + y_{12} = b_2 && \text{(2.43b)}\\ & y_3 + y_5 + y_6 - y_7 - y_8 - y_9 - y_{10} + y_{11} + y_{12} = b_3 && \text{(2.43c)}\\ & y_4 + y_5 - y_6 - y_7 + y_8 - y_9 + y_{10} + y_{11} - y_{12} = b_4 && \text{(2.43d)}\\ & y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12} \geq 0 && \text{(2.43e)}\\ & y_i \in \mathbb{Z},\ i = 1, \ldots, 4, && \text{(2.43f)} \end{aligned}$$

where at least one b_i is fractional. Note that $(b, 0)$ is a basic solution to the LP relaxation of both (2.42) and (2.43), and under our assumption that at least one b_i is fractional, it does not belong to the convex hull of integer solutions, thus any linear mapping π that maps x^* to $(b, 0)$ will be a pointed mapping, giving us a separating mapping.

Moreover, after some work, is easy to see that the possible number of outputs for an optimization oracle is bounded by 4 rays and 16 points for Problem (2.42) and 8 rays and 256 points for Problem (2.43). This allows us to define an oracle that checks the list of possible outputs and returns the best unbounded ray or the best solution if the problem is bounded.

Note also that both problems can be represented in the form

$$\begin{aligned} P: \qquad & (\ I \mid B \mid -B)y = b && \text{(2.44a)}\\ & y_i \in \mathbb{Z},\ i = 1, \ldots, m && \text{(2.44b)}\\ & y_i \geq 0,\ i = m+1, \ldots, 3m, && \text{(2.44c)} \end{aligned}$$

where B is a non-singular matrix and m is the number of constraints for the problem.

In a sense, this is a natural extension to the MIR mapping, which can be seen as fitting the description given in Problem (2.44), for the case of $m = 1$. Also, note that for Problem (2.42) and Problem (2.43), we have that $B^{-1} = \frac{1}{m}B$ and that $B^t = B$.

With this knowledge, we turn our attention to building the mapping $\pi(x) = Mx + m_o$. A first observation is that since we have a fixed mapped space P'_{ip}, a desired extreme point $(b, 0)$ to separate in P'_{lp}, and we are looking for a simple mapping, Condition 2.6 implies that we need to consider $\mu \neq 0$, ensuring that

$$\max\left\{\mu^t y : y \in P'_{lp}\right\} = \mu^t(b, 0),$$

which is equivalent to

$$\mu_N - \mu_M(\ B \mid -B) \leq 0,$$

where $N = \{m{+}1, \dots, 3m\}$, $M = \{1, \dots, m\}$ and B is as in (2.44). To satisfy this condition, it is enough to set $\mu_M = (-1, \dots, -1)$ and $\mu_N = (-m, \dots, -m)$, where m is the number of constraints (other than bounds) in (2.44).

Now, we restrict ourselves to consider mappings derived from m tableau rows associated with integer-constrained variables, where at least one of them has a fractional value in the optimal solution to the current LP relaxation. Assume that the integer variables are x_i, $i = 1, \dots, m$, and choose M^B such that $M^B_{i\cdot} = e_i$ for $i = 1, \dots, m$ and zero otherwise. Now that we have chosen values for μ, and M^B, we can re-write equations (2.28) as follows:

$$\sum_{i=1}^{m}\left(\bar{A}_{ij} - M_{ij}\right) - m\sum_{i=m+1}^{3m} M_{ij} \geq 0 \qquad \forall j \in N, x^*_j = u_j \tag{2.45a}$$

$$\sum_{i=1}^{m}\left(\bar{A}_{ij} - M_{ij}\right) - m\sum_{i=m+1}^{3m} M_{ij} \leq 0 \qquad \forall j \in N, x^*_j = l_j. \tag{2.45b}$$

Since $y_i \in \mathbb{Z}$ for $i = 1, \dots, m$, then M_{ij} must be zero for continuous variables for $i = 1, \dots, m$, and folk wisdom dictates that we should choose $M_{ij} = \left\lfloor \bar{A}_{ij} \right\rceil$ for integer nonbasic variables for $i = 1, \dots, m$. If we also define $b_i = x^*_i$, then we must define m_o as $(b, 0) - Mx^*$, thus leaving as unknowns the coefficients M_{ij} for $j \in N$, $i = m + 1, \dots, 3m$.

However, the condition of having a simple mapping, implies that $\pi(P_{lp})$ should be contained in P'_{lp}. To fulfill this condition, note that

$$\begin{aligned}
(\ I \mid B \mid -B \ )y = b & & \Leftrightarrow \\
(\ I \mid B \mid -B \ )Mx = b - (\ I \mid B \mid -B \ )m_o & & \Leftrightarrow \\
(\ I \mid B \mid -B \ )Mx = b - (\ I \mid B \mid -B \ ) \left(\begin{pmatrix} b \\ 0 \\ 0 \end{pmatrix} - Mx^* \right) & & \Leftrightarrow \\
(\ I \mid B \mid -B \ )Mx = (\ I \mid B \mid -B \ )Mx^*. & &
\end{aligned}$$

Thus, it is enough to ask that

$$(\ I \mid B \mid -B \ )M = \left(\bar{A}^t_{1\cdot}, \ldots, \bar{A}^t_{m\cdot}\right)^t,$$

or equivalently

$$\begin{pmatrix} M_{m+1,j} - M_{2m+1} \\ \vdots \\ M_{2m,j} - M_{3m,j} \end{pmatrix} = \frac{1}{m} B \begin{pmatrix} \bar{A}_{1j} - M_{1j} \\ \vdots \\ \bar{A}_{mj} - M_{mj} \end{pmatrix} \qquad \forall j = 1, \ldots, n. \tag{2.46}$$

Note that the choices that we have made up to now, ensure that equation (2.46) holds for all j in the current set of basic variables of the optimal solution of P_{lp}. With this, we are ready to define M_{ij} for $i = m+1, \ldots, 3m$ and $j \in N$ as follows:

$$(M_{m+i,j}, M_{2m+i,j}) = \frac{1}{m} \begin{cases} \left((\phi_i)^+, -(\phi_i)^-\right) & j : x^*_j = l_j, \\ \left((\phi_i)^-, -(\phi_i)^+\right) & j : x^*_j = u_j, \end{cases} \qquad \forall i = 1, \ldots, m, \tag{2.47}$$

where $\phi_i = B_{i\cdot} \left(\bar{A}_{ij} - M_{1j}, \ldots, \bar{A}_{mj} - M_{mj}\right)^t$. Note that with this definition we ensure the requirement of $y_i \geq 0$ for $i = m+1, \ldots, 3m$. Moreover, if we call $\hat{A}_{ij} = \bar{A}_{ij} - M_{ij}$ for $i = 1, \ldots, m$ and $j \in N$, then we can rewrite equations (2.45) for Problem (2.42) and for Problem (2.43) as

$$\sum_{i=1}^{m} \hat{A}_{ij} + \sum_{i=1}^{m} |\phi_i| \geq 0 \qquad \forall j \in N, x^*_j = u_j \tag{2.48a}$$

$$\sum_{i=1}^{m} \hat{A}_{ij} - \sum_{i=1}^{m} |\phi_i| \leq 0 \qquad \forall j \in N, x^*_j = l_j, \tag{2.48b}$$

which trivially holds since $\phi_1 = \sum(\hat{A}_{ij} : i = 1, \ldots, m)$, thus proving that we have defined a simple pointed mapping for both problems.

The experience with this kind of mapping functions is encouraging. The average time spend in each oracle call is about $16.07\mu s$ for Problem (2.42), and $172.36\mu s$ for Problem (2.43) on a 3GHz Pentium 4 CPU. Note also that we could compute all facets for both problems and then scan the list of facets to return the most violated one, or all violated facets, as the result of our separation routine. The oracle description of the problem allows us to add some extra constraints, for example, we could take into account that $y_i, \in F_i \; i = 1, \ldots, m$, where $F_i \subset \mathbb{Z}$ is determined by the actual possible values for y_i from our original problem P_{ip}.

2.7 Computational Experience

In this section we show our computational experience and explain the framework under which the experiments where done. We start by describing our branch and bound and cut implementation, we then move on to decide the actual *default* configuration against which the local cuts runs are compared, and then we present our results.

2.7.1 The Branch and Cut Solver

Some design decisions The first question that arises when programming a branch and bound and cut program is how independent it should be from the underling solver. Since we had already made the choice to work with exact (rational) solvers, the only choice available is our QSopt_ex solver introduced in Chapter 1, thus, in order to achieve more efficiency we decided to work with the low-level interface of QSopt, taking advantage of all structures and information already stored inside QSopt.

The second question relates on how to store information for every node in the branch and bound tree. We choose the common strategy of storing *differences*. Our implementation store the *difference* between the parent and child node in every node, thus, to get the actual LP relaxation at any node, we must traverse the path from the root node to the corresponding node and apply all recorded changes relevant to the LP formulation. These changes include any variable bound change, added or deleted cuts, and changes in the optimal basis. A possible advantage of this approach is that the memory required to store a given branch and bound tree should be smaller than storing all LP relaxations in every leaf of the branch and bound tree. Moreover, in a binary tree, if we combine parent nodes with only one children into one node, then the number of leaves equal the number of internal nodes in the tree, thus showing that the overhead in number of nodes stored in memory is no more than twice as the number of active nodes. Another possible advantage is that this scheme allows for pools of cuts that are only feasible in a sub-tree of the branch and bound tree, to be stored in a central place, and mix it with cuts that are globally valid (by storing the globally valid cuts in the pool of cuts of the root LP). A disadvantage is that as the tree gets larger, the time spend in re-computing the actual LP relaxation at a given node grows.

Branching Rules Branching rules are an essential part of modern branch and bound MIP solvers (see Achterberg et al. [1] for a review of common branching rules). We implemented four branching rules. Branch on most fractional variable. Branch in the first fractional variable. A form of strong branching (see Applegate et al. [4] and CPLEX 7.5 [58]). And a form of pseudocost branching (this rule was introduced by Benichou et al. [14], and has been further explored by J. T. Linderoth and M. W. P. Savelsbergh [72] and by A. Martin [76]), which combine strong branching for initialization of pseudocosts with the approach described in [72]. Versions of this mixed strategy for branching have been shown to perform very well on a range of problems (see [72, 76, 1]), and our experiments with branching rules confirmed this.

Cutting Planes As we have said before, cutting planes are the heart of any modern branch and bound framework. Based in the Results of Bixby et al. [16], we decided to implement MIR cuts and sequence independent lifted cover inequalities, as defined by Gu et al. [51, 53, 52].

For our MIR cuts we try two different implementations. The first one compute the MIR cut derived from each tableau row associated to a basic integer constrained variable with fractional value, and moreover, it also derives cuts from multiples of the tableau row, usually we try multiples from 1 to 15. Moreover, we only consider tableau for variables whose fractional part is between $\frac{1}{2000}$ and $\frac{1999}{2000}$.

The second approach applies the MIR technique to linear combinations of pairs of tableau, we consider the seven most fractional integer variables, and generate MIR cuts from all combinations of pairs of tableau combined with coefficients ranging from -3 to 3.

Since our procedures can generate large numbers of cuts, we add cuts to the LP in groups, usually 200 of them at a time, and, by default, discard any cut that is not violated.

Furthermore, we provide a callable interface to add cuts to a MIP, which allows to call several cutting planes routines in a given order. We also have two modes of operations inside the cutting loop for a given LP relaxation. In the first approach we call each and all of the cutting routines and then add the resulting cuts in batches, this mode is called full cutting. In the second approach we call the cutting routines in a given order and stop the cutting phase as soon as we find more than 200 cuts, this mode is called partial cutting.

A typical problem in cutting plane experiments is that while cutting planes help to improve the quality of the LP relaxation, they also increase the number of constraints in the LP, making it slower to solve as we add more and more cuts. Moreover, it is usual to see that older cuts tend not to be tight as we go along in the cutting procedure, and in fact, they become dominated by newer cuts (this situation is very common with MIR cuts). To keep this problem at bay, we implement a procedure that automatically eliminates cuts from the current LP relaxation. The procedure checks the slack of all cuts after each LP resolve, and discard all cuts with a strictly positive slack. Note that this choice is *safe* in the sense that it ensures that we will not cycle in the cutting phase, although it might happen that a previously discarded cut becomes active in a different child of the node from where it was deleted. The price that we pay for this safe approach is that we may be keeping cuts that are dominated or just *slightly* important for the LP relaxation (a more aggressive scheme is used by Applegate et al. [7] in their `Concorde` branch and cut code for the TSP).

2.7.2 Choosing The Default Settings

The basic options: In the previous sections we have described several variations for performing the cutting plane procedure. In this section, we perform numerical tests to establish a default configuration against which we will compare our local cut procedures.

We evaluate several possibilities, all of them have been introduced earlier, but for completeness and clarity we re-define them, and give different names for each of them.

Gomory Cuts: We consider three alternatives.

G1: For each fractional integer variable[8] we generate the MIR cut for multiples of the tableau ranging between 1 and 15. Variables are complemented to their bound in the basis.

G2: For all pairs among the 7 most fractional integer variables, and for all integer combinations with multipliers ranging from -3 to 3, we generate the MIR cut for the aggregated equality

$$\alpha_1\bar{a}_1x + \alpha_2\bar{a}_2x = \alpha_1\bar{b}_1 + \alpha_2\bar{b}_2,$$

where $\alpha_i,\ i = 1,2$ are the integer multipliers, and $\bar{a}_ix = \bar{b}_i,\ i = 1,2$ are the pairs of tables that we are looking at.

G3: Use both strategies described above.

C1: Use sequence independent lifting for cover inequalities as described in Gu et al. [51, 53, 52], the generation of the original cover inequalities follows the same heuristic ideas described in those papers.

i1: Detection of integer slacks, when using this option we check whether or not the implicit slack for an inequality can be assumed to be integer, the conditions are that all variables present in the inequality should be integer constrained, and that (using a multiplier of up to $2^{16} = 65536$) all coefficients of the inequality should be integer. For the case of original inequalities, the scaling factor must be one.

R1: Rounding cuts. As we have described earlier, all our implementation works on rational representation of inequalities, this is done in order to ensure correctness of our cutting planes and procedures, but, as a side effect, many cuts require extremely long rational representations for its coefficients. As a way to manage this problem, we only accept cuts whose rational coefficients can be represented in 64 bits for the numerator and denominator (i.e. a maximum of 128 bits of representation). This implies that some cuts are discarded. The rounding cut procedure takes those discarded cuts and approximate the coefficients of the rational inequality by the continued fractions method, ensuring that the resulting inequality is still valid for the original problem

[8] Note that we are assuming the bounds on integer variables to be integers or infinity, thus ensuring that integer variables with fractional optimal values are basic.

and that the coefficient can be represented in the accepted format. If the cut can not be approximated such that it satisfies those requirements, we discard it.

The selection procedure Note that all these configurations are not exclusive, but can be mixed, thus generating a large set of possibilities to test. In order to reduce our running times, we choose our configuration in two steps.

First we run all configurations without rounding cuts, and choosing among them the best one, and then, compare that configuration with and without rounding cuts and picking up the best as our default.

This is done because running times for configurations with rounding of cuts enabled typically require much more running time.

Table 2.5: Gap closed I. Here we show the geometric average gap closed for each configuration in a test set of 51 problems from MIPLIB

Configuration	GAP Closed	Time (s)
C1	2.0596383	9.6782810
G2	7.0048098	111.9273010
G2i1	7.1320951	108.1570604
C1G2	10.5433004	135.6745354
C1G2i1	11.0725560	124.1899176
G1i1	12.7157958	80.2739946
G1	12.9378161	82.9078098
G3i1	13.0540541	119.0279938
G3	13.1831940	125.8740389
C1G1	14.3647000	93.5269510
C1G1i1	14.3853854	87.0436880
C1G3	14.6564994	151.6379040
C1G3i1	14.7206547	148.1970612

Measuring effectiveness As a measure of the effectiveness we take the percentage of the gap closed by each configuration, this is measured as

$$100 \cdot \frac{Z_{conf} - Z_{LP}}{Z_{IP}^{+} - Z_{LP}},$$

where Z_{conf} is the best bound obtained by the given configuration, Z_{LP} is the value of the LP relaxation, and Z_{IP}^{+} is either the optimal value of the problem, or an upper/lower bound for it if the problem is a minimization/maximization problem. Note that we stop at the root node and do not perform branching while computing Z_{conf}. Taking a subset of the MIPLIB problems, we obtain the averages shown in Table 2.5. Using the set of mixed integer problems (both bounded and unbounded) presented by Atamtürk [8], we obtain the averages shown in Table 2.6. The combined averages can be seen in Table 2.7. Note that

Table 2.6: Gap closed II. Here we show the geometric average gap closed for each configuration in a test set of 119 mixed integer problems from Atamtürk.

Configuration	GAP Closed	Time (s)
C1	1.0000000	3.9322205
C1G2	42.7267361	162.1870281
C1G2i1	42.7267361	161.6868330
G2	42.7267361	158.9433407
G2i1	42.7267361	159.3574375
C1G3	76.9814349	150.5784483
C1G3i1	76.9814349	150.0529741
G3	76.9814349	149.2547119
G3i1	76.9814349	149.3140403
C1G1	77.0778314	141.1773095
C1G1i1	77.0778314	140.5166063
G1	77.0778314	139.5766899
G1i1	77.0778314	139.8467765

Table 2.7: Gap closed III. Here we show the geometric average gap closed for each configuration over 170 problems coming from both MIPLIB and mixed integer problems from Atamtürk.

Configuration	GAP Closed	Time (s)
C1	1.2420449	5.1521201
G2	24.8375472	143.0706154
G2i1	24.9720928	141.8655999
C1G2	28.0790737	153.7307635
C1G2i1	28.4947052	149.3819622
G1i1	44.8904237	118.3938612
G1	45.1241395	119.3844214
G3i1	45.2057651	139.4971026
G3	45.3394653	141.8177157
C1G1	46.5628559	124.7718067
C1G1i1	46.5829611	121.7111993
C1G3	46.8036020	150.8955054
C1G3i1	46.8649695	149.4937730

while for the problems in MIPLIB all parameters and configuration have an impact, for the mixed integer problems from Atamtürk, the only relevant settings are those related to Gomory cuts, and in fact, the configurations with Gomory cuts of type G1 and G3 are really close. When we look at both sets of problems at the same time, we can see that, on average, using Gomory cuts of both types, with lifted cover inequalities and detection of integer slack variables provides the best configuration while maintaining a reasonable average running time. Figure 2.9 shows the distribution of the gap closed for several configurations over a set of 170 problems from both MIPLIB and from Atamtürk. Note that this graph confirm that the geometric mean presented in the previous tables are good estimator of the behavior of the settings.

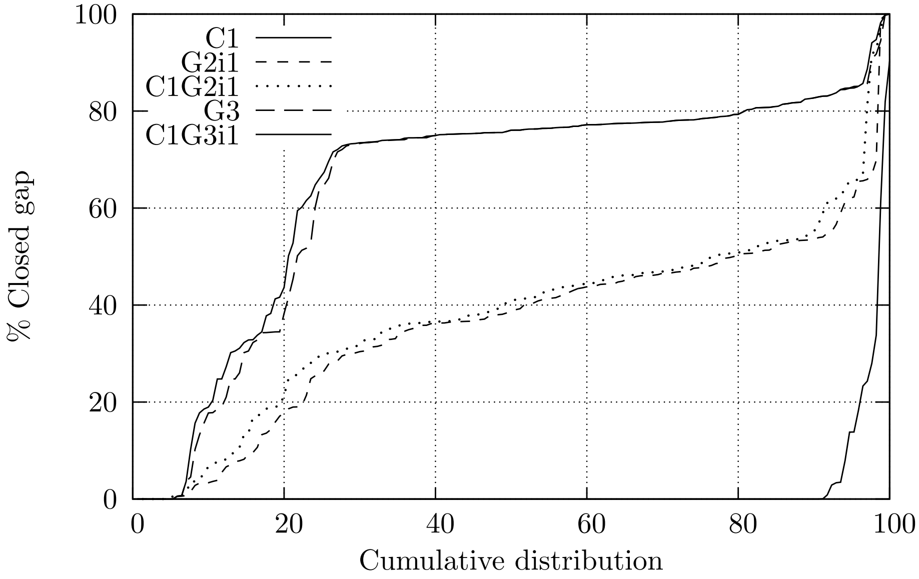


Figure 2.9: Closed gap distribution. This figure shows the closed gap distribution for several configurations over 170 problems from MIPLIB and the mixed integer problems of Atamtürk.

To round or not to round Now that we have chosen a basic configuration, we evaluate the use of rounding cuts. From both Table 2.8 and from Figure 2.10 is clear that the default setting with rounding is able to close several extra points of the overall gap, but at the expense of a tremendous running time.

This bad behavior in running time is what forced us to choose the configuration C1G3i1

Table 2.8: Gap closed IV. Here we show the geometric average gap closed by our default configuration with and without rounding cuts over 32 problems coming from MIPLIB.

Configuration	GAP Closed	Time (s)
C1G3i1	17.8327738	31.5883756
C1G3i1R1	25.3462967	13183.4560473

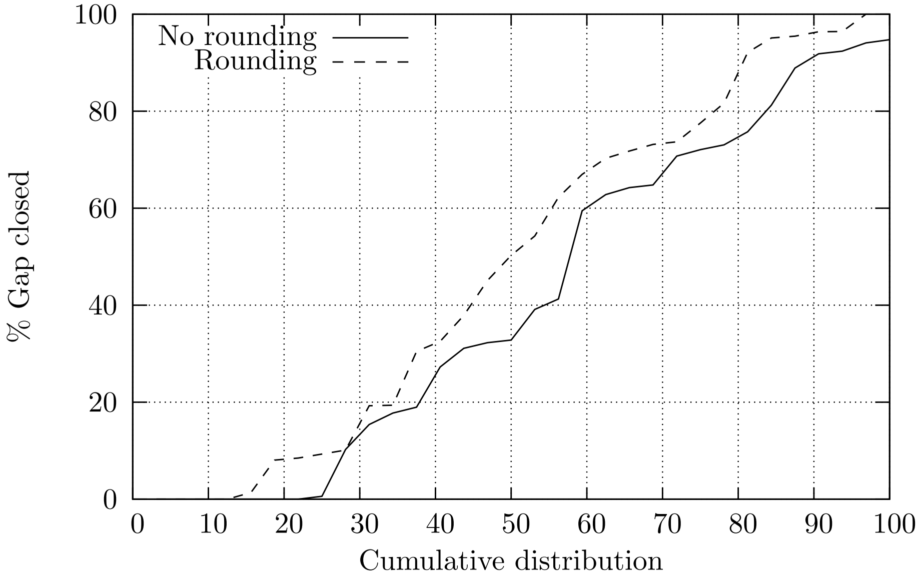


Figure 2.10: Closed gap distribution with rounding. This figure shows the closed gap distribution for our default configuration C1G3i1 with and without rounding over 32 problems from MIPLIB.

without rounding as our default setting. Note however that the experiment suggest that the idea of approximate cuts may be good, and it may be that different choices in the rounding procedure could yield better running times without loosing the extra gap closed.

2.7.3 Comparing the default settings against Local Cuts

Now that we have explained the default settings for our solver, we re-state the different implementations for local cuts:

LS: In this configuration our mapping function is the identity, and the only modification to the problem is that now at most five variables remain integer constrained. At every iteration we try up to ten sets of five integer-constrained variables.

L1: This is the minimum projection local cuts, where the mapping preserves some of the original variables and aggregates the remaining variables in such a way as to obtain a pointed mapping that minimizes a weighted sum of the coefficient of the aggregated variables.

L2: Local cuts obtained by using a MIR projection using up to four tableau.

L3: Local cuts obtained by using a Gomory projection using up to four tableau.

L4: In this configuration we use a mapping that is a projection into a sub-set of at most five integer variables.

L5: This configuration is the 2-Tableau mapping described earlier.

L6: This configuration is the 4-Tableau mapping described earlier.

Note that whenever we use a particular local cut configuration we are also using our default setting, i.e. Lx = C1G3i1Lx. To simplify notation, we call L0 our default seating C1G3i1.

One of the difficulties of comparing cuts is that as we add several rounds of cuts, the problems tend to diverge markedly. In order to minimize those effects, we compare the two first full rounds of cutting planes. We start with a small subset of seventeen problems from MIPLIB where we compare all our configurations. Table 2.9 show the average gap closed for each configuration after the first and second round of cuts, as well as the average running time.

Table 2.9: Local Cuts performance I. Here we compare all our local cuts configurations on a set of seventeen problems from MIPLIB. The results show average running time and average gap closed after the first and second round of cuts. Time is measured in seconds.

Configuration	First Round		Second Round	
	%GAP	Time (s)	%GAP	Time (s)
L3	33.00	169.71	37.93	1623.74
L2	33.48	70.89	39.21	1302.89
L0	32.70	1.06	39.38	3.03
L6	32.25	10.76	39.38	26.98
L1	36.45	254.88	39.41	1650.13
L5	32.72	3.47	39.65	9.72
L4	32.80	6.56	39.74	24.91
LS	32.75	111.27	42.61	594.75

It is not surprising to see that configuration LS outperforms all other configurations, because the number of integer variables that it considers is the same as all other configurations, but the mapping is the identity and thus as close as possible from the original problem at hand. However the running time is, as we discussed earlier, prohibitive.

If we consider also running times, then the best configurations are L4, L5, L6 and L0 (in that order). Now we move on to a more detailed analysis of those four configurations. Table 2.10 show the average gap closed for the configurations L0, L4, L5 and L6 over 64 problems from MIPLIB after the first and second round of cuts. Table 2.11 show the average gap closed for the configurations L0 and L5 over 70 problems from MIPLIB after the first, second and last round of cuts. Note that the improvement in GAP closed is slightly above half percent after the second round of cuts. Figure 2.11 shows the cumulative distribution of closed gap for both L0 and L5 configurations over a set of 70 instances from MIPLIB. Note that this figure also shows that the difference between both configurations is very small.

Table 2.10: Local Cuts performance II. Here we compare configurations L0, L4, L5 and L6 over 64 problems from MIPLIB. The results show average running time and average gap closed after the first and second round of cuts. Time is measured in seconds.

Configuration	First Round		Second Round	
	%GAP	Time (s)	%GAP	Time (s)
L6	17.94	23.20	23.28	55.86
L0	19.34	6.34	24.58	14.96
L4	19.35	22.49	24.67	60.51
L5	19.35	11.49	25.15	28.33

Table 2.11: Local Cuts performance III. Here we compare configurations L0 and L5 on a set of 70 problems from MIPLIB. The results show average running time and average gap closed after the first, second and last round of cuts. Time is measured in seconds.

Configuration	First Round		Second Round		Last Round		
	%GAP	Time (s)	%GAP	Time (s)	%GAP	Time (s)	Rounds
L0	20.24	10.46	25.24	25.05	33.03	184.71	23.49
L5	20.24	17.79	25.80	43.28	33.31	416.33	17.46

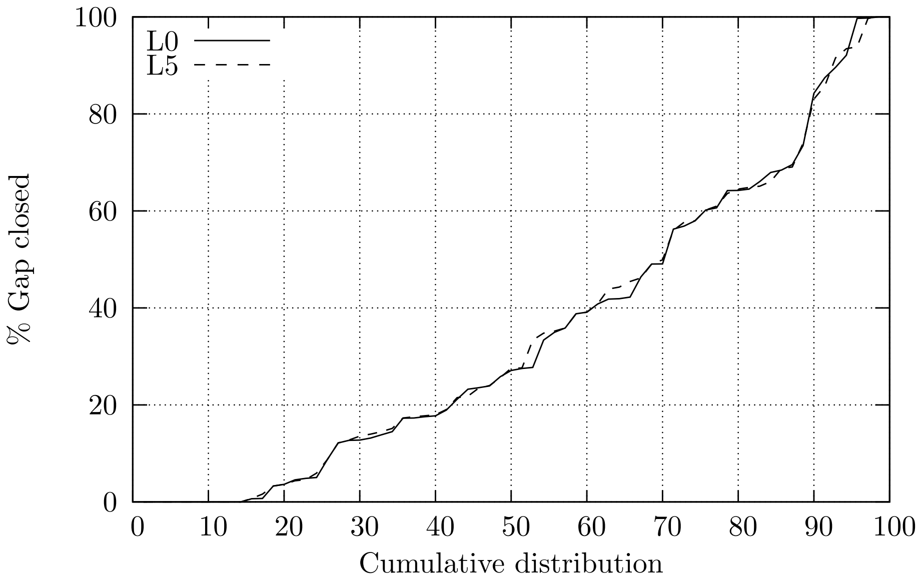


Figure 2.11: Closed gap distribution for Local Cuts. Here we show the cumulative distribution of closed gap for configurations L0 and L5 over a set of 70 problems from MIPLIB.

2.8 Final Thoughts on Local Cuts

In previous sections we have developed a general technique to generate cuts for general MIPs using an oracle description for a relaxation of the original problem. We also provided conditions under which the separation problem over the relaxation can be guaranteed to generate valid cuts for the original problem. We then proposed some possible simple mappings to construct possible relaxations and compare their performance against our default settings at the root node. Those experiments show that there is potential for this kind of mechanism to work in practice for general MIPs.

Note however that the set of mappings presented here are just scratching the possibilities for this kind of approach, and moreover, the scheme could be used in structured problems where small instances are easily solvable.

A surprising outcome for us was that in several instances we were unable to add cuts to our LP relaxation because the encoding of the obtained cuts exceeded our auto imposed limits. This raises the question of whether the problem of long encodings are common and unavoidable, or if there are techniques (or special relaxations) that naturally yield inequalities with short descriptions. The question is not just a rhetorical one, note that in the case of the TSP, the local cuts procedure usually finds inequalities with short descriptions.

Some implementation issues also aroused naturally once we start to look at the profile of our program. From these analysis we can see that on average over 87% for the L0 configuration and 86% of the time is spend computing maximum common denominator to mantain our fractions in normal (irreducible) form and perform rational arithmetic. This hints that our rational representation is not the best for our purposes, or at least shows that to maintain irreducibility at every step is not necessarily a good idea.

CHAPTER 3

The TSP Problem and the Domino Parity Inequalities

3.1 Introduction

The *Traveling Salesman Problem* or *Travelling Salesman Problem* (TSP), also known as the *Traveling salesperson problem*, is the problem of given a number of *cities* and the cost of traveling from any city to any other city, find the cheapest round-trip route that visits each city exactly once and then returns to the starting point.

An equivalent formulation in terms of graph theory is: Given a complete graph on n points, and weights on the edges, find a *Hamiltonian cycle* with minimum weight.

This problem is NP-hard , even in the special case when all distances are either 1 or 2. It remains NP-hard even if the distances are euclidean. The problem also remains NP-hard when we remove the condition of visiting each city *exactly once*. The decision version is NP-complete (see [43] for more complexity results).

The problem is of considerable practical importance, apart from the evident transportation and logistics areas. A classic example is in printed circuit manufacturing: scheduling of a route of the drill machine to drill holes in a board. In this case the *cities* are the holes that the robot must drill, and the *cost between cities* is the time needed to move the robot arm and retooling it.

Although the most direct solution method would be to try all the permutations of cities and then see which one is the cheapest, the number of combinations to try is $n!$, which translates into the following values for different numbers of cities:

- 10 cities $\approx 10^{5.5}$ possibilities.
- 100 cities $\approx 10^{156}$ possibilities.
- 1,000 cities $\approx 10^{2,565}$ possibilities.
- 33,810 cities $\approx 10^{138,441}$ possibilities.

To put those numbers in perspective, let us record some physically large numbers:

- Age of the universe $\approx 10^{18}$ seconds.
- Number of atoms in the universe $< 10^{100}$.

These numbers show that the brute force approach (of enumerating all possible combinations and pick the best) is absolutely hopeless for problems with more than a handful of cities.

Using techniques of dynamic programming one can solve the problem exactly in $\mathcal{O}(n^2 2^n)$ time. Although this is exponential, it is still much better than the $\mathcal{O}(n!)$ running time for the brute force approach.

The best approach to exactly solve TSP problems today is the Branch and Bound and Cut algorithm proposed by Dantzig, Fulkerson and Johnson [33]. In this method we start with a linear programming relaxation description of all tours, and we iteratively add inequalities satisfied by all tour vectors. Once no more inequalities can be found, we resort to branching and keep cutting in every node in the branch-and-bound tree. However, the cutting phase is crucial to find optimal solutions for TSP problems, without them, the scheme reduces to a clever enumeration of (almost) all possible tours, which, as we have seen, can be a quite time consuming endeavor.

The literature of valid and facet-defining inequalities for the TSP is vast and with a long history, they include the subtour elimination constraints, the blossom constraints introduced by Edmonds [36] in 1965, the comb inequalities introduced by Chvátal [26] in 1973 and generalized by Grötschel and Padberg [49] in 1979, the path, wheelbarrow and bicycle inequalities introduced by Cornuéjols et al. [29] in 1985, the clique-tree inequalities introduced by Grötschel and Pulleyblank [50] in 1986, the star inequalities introduced by Fleischmann [39] in 1988, the bipartition inequalities introduced by Boyd and Cunningham [19] in 1991, the 2-brushes inequalities introduced by Naddef and Rinaldi [80] in 1991, the binested inequalities introduced by Naddef [78] in 1992, the crown inequalities introduced by Naddef and Rinaldi [81], the ladder inequalities introduced by Boyd et al. [20] in 1993, and the domino-parity inequality introduced by Letchford [70] in 2000.

Many of these classes of inequalities have been proven to be facet defining under some mild conditions, however, polynomial time separation algorithms are lacking for most of them, exceptions are the subtour inequalities which can be separated in $\mathcal{O}(n^3)$ time, where n is the number of nodes or cities in the problem; the blossom inequalities can be separated exactly in $\mathcal{O}(m(m+n)^3)$ time [88, 90], where m is the number of edges in the graph, and n is the number of nodes or cities in the problem; Carr [24] also proposed a polynomial time separation algorithm for clique trees and bipartition inequalities for fixed number of handles and teeth whose complexity is $\mathcal{O}(n^{h+t+3})$, where n is the number of nodes or cities in the problem, h the number of handles and t the number of teeth that we are considering.

Most implementations of the branch and bound and cut procedure for the TSP include exact separation routines for subtour inequalities and some form of the separation of blossom inequalities, but they have to rely on heuristic separation routines for other classes of inequalities. Although this mixture of exact methods and heuristic algorithms for the separation of inequalities have proven to be effective in many cases (see Padberg and Rinaldi [91], Jünger, Reinelt and Thienel [63], Applegate et al. [5], and Naddef and Thienel [82, 83]), finding exact separation methods could allow to solve larger problems and also solve problems faster than what is nowadays possible.

An interesting new approach to TSP separation problems was adopted by Letchford [70], building on an earlier work of Fleischer and Tardos [38]. Letchford [70] introduces a new class of TSP inequalities, called domino-parity constraints, and provides a separation algorithm under some sparsity conditions, moreover, the separation algorithm runs in $\mathcal{O}(n^3)$ time, where n is the number of cities or nodes in the problem. This theoretical complexity hints that the algorithm might be used in practice. An initial computational study of this algorithm by Boyd et al. [18], combining a computer implementation with by-hand computations, showed that the method can produce strong cutting planes for instances with up to 1,000 nodes. Further studies by Naddef and Wild [84] showed that *most* of the domino parity constraints are facet defining for the TSP.

In this chapter we present a further study of Letchford's algorithm, reporting computational results on a range of TSPLIB test instances. The rest of this chapter is organized as follows: In Section 3.2 we define our notation. The domino parity constraints are described in Section 3.3, together with a review of results from Letchford [70] and a short description of the steps adopted in our implementation to improve the practical efficiency of the separation algorithm. In Section 3.4 we describe shrinking techniques that allow us to handle large instances and also to handle the common case where the original sparsity conditions do not hold. A local-search procedure for improving domino-parity constraints is described in Section 3.6 and computational results are presented in Section 3.7. These results include the optimal solution of a 33,810-city instance from the TSPLIB.

3.2 Problem Description

We consider the traveling salesman problem (TSP) with symmetric costs, that is, the cost to travel from city a to city b is the same as traveling from b to a. The input to the problem can be described as a complete graph $G = (V, E)$ with nodes V, edges E, and edge costs $(c_e : e \in E)$. Here V represents the cities and the problem is to find a tour, of minimum total edge cost, where a tour is a cycle that visits each node exactly once (also known as a Hamiltonian cycle).

Tours are represented as a 0-1 vectors $x = (x_e : e \in E)$, where $x_e = 1$ if edge e is used in the tour and $x_e = 0$ otherwise. For any $S \subseteq V$ let $\delta(S)$ denote the set of edges with exactly one end in S and let $E(S)$ denote the set of edges having both ends in S. For disjoint sets $S, T \subseteq V$ let $E(S : T)$ denote the set of edges having one end in S and one end in T. For any set $F \subseteq E$ define $x(F) = \sum(x_e : e \in F)$.

Using this notation the *subtour-elimination constraints* can be defined as

$$x(\delta(S)) \geq 2 \qquad \forall\, \emptyset \neq S \subsetneq V. \tag{3.1}$$

Using network-flow methods, the separation problem for these inequalities can be solved efficiently, that is, given a non-negative vector x^* a violated subtour-elimination constraint can be found in polynomial time, provided one exists. The subtour-elimination constraints were employed by Dantzig et al. (1954) and they are a basic ingredient of modern implementations of the cutting-plane method. The solution set of the TSP relaxation

$$\begin{aligned} x(\delta(\{v\})) &= 2 \qquad && \forall\, v \in V \\ x(\delta(S)) &\geq 2 \qquad && \forall\, \emptyset \neq S \subsetneq V \\ 0 \leq x_e &\leq 1 \qquad && \forall\, e \in E \end{aligned}$$

is known as the *subtour-elimination polytope* and is denoted by $SEP(\text{n})$, where $n = |V|$.

3.3 DP Inequalities and Letchford's Algorithm

We begin by introducing the domino-parity constraints and giving an overview of Letchford's separation algorithm. We highlight some important algorithmic steps, placing emphasis on our implementation. All lemmas and theorems not proved in this section can be found in Letchford [70].

Define $I(p) = \{1, \ldots, p\}$ for any $p \in \mathbb{N}$. Let $\Lambda = \{E_i\}_{i \in I(k)}$, where $E_i \subseteq E$, $\forall i \in I(k)$. Define $\mu_e = |\{F \in \Lambda : e \in F\}|$ The family Λ is said to *support* the cut $\delta(H)$ if $\delta(H) = \{e \in E : \mu_e \text{ is odd}\}$. Define a *domino* as a pair $\{A, B\}$ satisfying $\emptyset \neq A, B \subseteq V$, $A \cap B = \emptyset$, and $A \cup B \neq V$.

Theorem 3.1. Let p be a positive odd integer, let $\{A_j, B_j\}$ be dominoes for $j \in I(p)$, and let $H \subsetneq V$. Suppose that $F \subseteq E$ is such that $\{E(A_j : B_j), j \in I(p); F\}$ supports the cut $\delta(H)$ and define μ_e accordingly. Then, the domino-parity (DP) constraint,

$$\sum_{e \in E} \mu_e x_e + \sum_{j \in I(p)} x(\delta(A_j \cup B_j)) \geq 3p + 1 \tag{3.2}$$

is valid for all tours.

The set H is called the handle of the constraint.

Letchford [70] proposes a two-stage algorithm which exactly separates this class of constraints, provided that the support graph G^* is planar and all subtour-elimination constraints are satisfied. In the first stage, a set of candidate dominoes is constructed. In the second stage, a handle and an odd number of dominoes are selected in such a way as to define a maximally violated constraint, provided one exists.

For the remainder of this section, assume that all of the subtour-elimination constraints are satisfied. Also, assume that G^* is a planar graph and let $\bar{G^*}$ be the planar dual of G^*. For any subset $F \subseteq E(G^*)$, denote by $\overline{F}$ the corresponding edges in $\bar{G^*}$.

3.3.1 Building Dominoes

Lemma 3.2. Consider $s, t \in V(\bar{G^*})$ and three node-disjoint s-t paths P_1, P_2, P_3 in $\bar{G^*}$. There exists a domino $\{A, B\}$ such that $\left(\overline{\delta(A \cup B)} \cap E(\bar{G^*})\right) \cup \left(\overline{E(A:B)} \cap E(\bar{G^*})\right) = E(P_1) \cup E(P_2) \cup E(P_3)$.

Algorithm 3.1 Primalizing Dual Dominoes (`prim_dom(`p_1, p_2, p_3`)`)

```
Require: $p_1, p_2$ and $p_3$ be three simple $s - t$ paths in $\bar{G^*}$.
1: Compute $\hat{p}_1, \hat{p}_2$ and $\hat{p}_3$ three non-crossing s-t paths in $\bar{G^*}$ such that $\bigcup_{i \in I(3)} p_i = \bigcup_{i \in I(3)} \hat{p}_i$.
2: Compute $S_1, S_2, S_3$ a partition of $V$ such that $\delta(S_i) = \hat{p}_{i+1} \cup \hat{p}_{i+2}$.
3: Let $A$ be the smallest $\{S_i\}_{i \in I(3)}$ and $B$ be the second smallest $\{S_i\}_{i \in I(3)}$.
4: return $(A, B)$
```

Given three paths in $\bar{G^*}$ as described above, it is easy to construct a domino in G^*. Algorithm 3.1 describes the procedure by which we build them even in the case when $x^* \notin SEP(\text{n})$. Take note that, as shown in step 3 of Algorithm 3.1, there is no a unique primal domino whose cut-edges correspond in the dual to the three node-disjoint s-t paths. This allows us some freedom at the moment of choosing the primal representation of the domino, such as to choose A and B in such a way as to force $x(E(A:B))$ or $|E(A:B)|$ to be minimum or maximum. From the point of view of the exact separation algorithm this does not affect the outcome, but when doing heuristics to generate more cuts, this can have a great impact. A proof that we can indeed choose A and B in any way is in section 3.4.

A surprising fact is that it suffices to use only these dominoes (generated from three dual $s - t$ paths) as candidates. Algorithm 3.2 describes the basic procedure by which to obtain them.

Algorithm 3.2 although correct is far too inefficient to be used directly. The first observation to be made is that by defining the weight of a domino $\{A, B\}$ as $w(\{A, B\}) = x(\delta(A \cup B)) + x(E(A:B)) - 3$, then a DP inequality with domino-set $\{A_j, B_j\}_{j \in I(p)}$ can

Algorithm 3.2 Generating Candidate Dominoes (basic)

```
1: $\mathcal{L} \leftarrow \emptyset$.
2: Compute $\bar{G^*}$ , the planar dual of $G^*$.
3: for all $s, t \in V(\bar{G^*}\ )$ do
4:    Find three edge disjoint $s - t$ paths $p_1, p_2, p_3$ of minimum joint weight.
5:    $\mathcal{L} \leftarrow \mathcal{L} \cup$ prim_dom($p_1, p_2, p_3$)
6: end for
7: return $\mathcal{L}$
```

be re-written as

$$x(F) + \sum_{j \in I(p)} w(\{A_j, B_j\}) \geq 1. \tag{3.3}$$

Thus, if we are only interested in violated inequalities, we should consider dominoes $\{A, B\}$ with weight $w(\{A, B\}) < 1$, i.e. we can add the constraint $x(p_1 \cup p_2 \cup p_3) < 4$ while we are generating the dominoes between steps 4 and 5 of Algorithm 3.2.

More can be done with this bound if we assume that $x \in SEP(\text{n})$. First of all, note that no node at distance 2 or more from either s or t can be present in any of the three paths, otherwise, if such a node is in path p_3, then we know that p_1, p_2 form a cycle in the dual, and thus correspond to a cut in the primal, from that we have that $x(p_1 \cup p_2) \geq 2$ and that $x(p_3) \geq 2$, thus we would violate the bound of 4. This allow us to run Dijkstra's algorithm on a much smaller graph than the original one. Moreover, since we use a successive shortest path algorithm as described in [2], we can use the weight of the previously computed paths as a bound for the weight of the remaining paths, this is because $x(p_1) \leq x(p_2) \leq x(p_3)$, and then sufficient conditions for the bound to be violated are $3x(p_1) \geq 4$ or $x(p_1) + 2x(p_2) \geq 4$. Finally, note that $x \in SEP(\text{n})$ also implies that for any domino satisfies $w(\{A, B\}) \geq 0$.

To take advantage of these bounds we implemented Dijkstra's algorithm with heaps and with the capacity of whenever the latest labeled node has a value over a given bound, stop the algorithm. We call this function `dijkstra`(G,s,w,*bound*), where G is a graph, s is a node in $V(G)$, w is a weight vector on the edges of G and *bound* is the stopping bound, this function returns a vector of distances from s to all nodes in G (with distance less than *bound*, and infinite otherwise).

Another small speed-up is possible by realizing that while computing all $s - t$ paths for $t \in V(\bar{G^*}\ ) \setminus \{s\}$, the solution to the first path can be computed with only one run of dijkstra. A more detailed version of the actual implementation is in Algorithm 3.3.

Algorithm 3.3 Generating Candidate Dominoes (get_all_dominoes(G^*,α))

```
 1: $\mathcal{L} \leftarrow \emptyset$.
 2: Compute $\bar{G^*}$ , the planar dual of $G^*$.
 3: $w(e) \leftarrow x_e^*, \quad \forall e \in E(\bar{G^*})$    /* weight definition */
 4: $c(e) \leftarrow 1, \quad \forall e \in E(\bar{G^*})$    /* capacity of edges */
 5: for all $s \in V(\bar{G^*})$ do
 6:    $d_o \leftarrow$dijkstra($\bar{G^*}$ ,$s$,$w$,2)
 7:    for all $t \in V(\bar{G^*})$ and $t > s$ and $d_o(t) < (3+\alpha)/3$ do
 8:       $d \leftarrow d_o$
 9:       Send unit flow in shortest $s - t$ path according to $d$
10:       Update residual graph $\bar{G^*}$ , residual costs $w$, and capacity $c$
11:       $val \leftarrow d(t)$
12:       $bound \leftarrow \min\left(\frac{3+\alpha-val}{2}, 2\right)$
13:       $d \leftarrow$dijkstra($\bar{G^*}$ ,$s$,$w$,$bound$).
14:       if $val + 2d(t) < 3 + \alpha$ then
15:          Send unit flow in shortest $s - t$ path according to $d$
16:          Update residual graph $\bar{G^*}$ , residual costs $w$, and capacity $c$
17:          $val \leftarrow val + d(t)$
18:          $bound \leftarrow \min(bound, 3 + \alpha - val)$
19:          $d \leftarrow$dijkstra($\bar{G^*}$ ,$s$,$w$,$bound$).
20:          if $val + d(t) < 3 + \alpha$ then
21:             Send unit flow in shortest $s - t$ path according to $d$
22:             Compute the three unit flows paths $p_1, p_2, p_3$ from $s$ to $t$.
23:             $D_{st} \leftarrow$ prim_dom($p_1, p_2, p_3$), $w(D_{st}) \leftarrow val + d(t)$.
24:             $\mathcal{L} \leftarrow \mathcal{L} \cup D_{st}$.
25:          end if
26:       end if
27:    end for
28: end for
29: return $\mathcal{L}$
```

Note that Algorithm 3.3 has a parameter α as input. To actually get al.l dominoes that might be used in a violated constraint, suffices to choose α as one. In practice however, we have seen that choosing α to be .55 greatly reduces the computation time to get the dominoes, and at the same time it seems that it does not hurt the quality of the inequalities that we actually get from this code. In all the test presented in section 3.7, that is the value used for α.

Another interesting possibility to speed-up the domino generation step, is to do steps 6-27 of Algorithm 3.3 in parallel, this is possible because to compute all dominoes originating at a node s is independent of the computations to get al.l dominoes originating from any other node t. The only synchronization needed is to collect all resulting partial lists of dominoes, which is done by a *master* program, and we need to send a copy of the working graph $\bar{G^*}$ to each of the sub-programs generating the dominoes, this is done only once during each separation call. Although this is not a theoretical speed-up, it allows us to use a cluster of 50-100 machines to generate all dominoes, greatly reducing the (actual) time to do the testing of the code.

3.3.2 The Odd-Cycle Problem

Now we will focus in the problem of how to build an inequality from this set of dominoes, and the edges in the dual graph. For that, define an auxiliary multigraph M^* with node set $V(M^*) = V(\bar{G^*})$. For each edge $e = \{u, v\} \in E(\bar{G^*})$ define an *even* edge $e = \{u, v\} \in E(M^*)$ with weight $w_e = x^*_e$, and for each $D_{uv} \in \mathcal{L}$ define an *odd* edge $e = \{u, v\} \in E(M^*)$ with weight $w_e = w(D_{uv})$. An *odd cycle* in M^* is a cycle with an odd number of odd edges.

Lemma 3.3. Given an odd cycle $C \subseteq E(M^*)$ with weight $w(C) < 1$ it is possible to construct a DP-inequality with violation $1 - w(C)$.

In fact, to construct the DP-inequality from the cycle it suffices to define the set F as the even edges in C, and choose the set of dominoes used by the inequality $\mathcal{T}$ as those dominoes corresponding to odd edges in C.

Theorem 3.4. There exists a violated DP-inequality in G^* if and only if there exists an odd cycle in M^* with weight less than one. Furthermore, if such a cycle exists, a minimum weight odd cycle in M^* corresponds to a maximally violated DP-inequality.

From these results Algorithm 3.4 directly follows.

This algorithm can be improved by some simple observations, first we can omit all edges $e \in E(\bar{G^*})$ such that $x^*_e \geq 1 - \varepsilon$, where ε is the minimum violation that we would like to have, and can be set to zero if we want an exact separation version. Note that even in the case when we set ε to zero, the number of edges that can be discarded is usually very large, specially when the current LP relaxation is very good, because the number of edges having values of one (or close) is very large.

Algorithm 3.4 DP-inequality separation (basic)

```
$max\_violation \leftarrow 0$
$\mathcal{L} \leftarrow$get_all_dominoes($G^*$,1).
build graph $M^*$
for all $v \in V(M^*)$ do
    Compute min odd cycle $C$ passing through $v$
    if $1 - w(C) > max\_violation$ then
        $max\_violation \leftarrow 1 - w(C)$
        $F \leftarrow$ even edges in $C$
        $\mathcal{T} \leftarrow \{D_{uv} \in \mathcal{L} : e_{uv}$ odd $, e_{uv} \in C\}$
    end if
end for
return $F, \mathcal{T}$
```

3.3.3 Generating More Inequalities

One drawback of Algorithm 3.4 is that it only provides one maximally violated domino parity inequality. But the amount of work involved to get it is really huge, and in practice we want to find several violated inequalities for the current fractional LP solution x^*. A first approach to get several inequalities is to keep all violated inequalities found during the algorithm. Although this approach works, it has several pitfalls, first of all, many of the violated inequalities generated in this form are the same, and from our computational experience we know that there is a large set of inequalities with high violation that do not correspond to a minimum odd cycle with a given node in it. This lead us to implement (on top of the previous improvement) an heuristic search procedure to attempt to find additional inequalities. The technique we use is to sample the odd cycles by performing random walks starting at each node. At each step of the walk we select a new edge to extend the current path that does not create an even cycle, nor create an odd cycle with only one odd edge in it (with any previously visited node), with a probability proportional to the weight of the edge. If the resulting path has total weight more than one, we re-start the process, if the resulting path creates an odd cycle within the path, we keep the constraint and re-start the process, otherwise, we keep iterating. In our tests we spend 10-30 seconds in this step, evenly distributed among all nodes in the network. Incredibly, this simple procedure is able to find several hundred of thousands of different inequalities within a couple of seconds, which in turn lead us to the following problem: Which of those inequalities should we choose to report?. Clearly we can not return them all (just storing all those inequalities is impossible within our computer memory of 4GB of RAM), and keeping the set of most violated inequalities leads to storing several inequalities that are *almost* identical and that use roughly the same set of edges in the graph M^*.

Algorithm 3.5 Selection of Cuts (`add_if_new`($\mathcal{C}$,$\{F,\mathcal{T}\}$))

```
$\mathcal{C}$: The set of cuts that we have selected up to now.
$\{F,\mathcal{T}\}$: The cut that we are consider to add to the set of cuts.
$N_{\max}$: Maximum number of cuts to return.
$\varepsilon$: Minimum violation required.
$\alpha_{\min}$: Closeness parameter.
$\beta_o$: Spread parameter.
$w_o$: Quality parameter.
$\beta_{\bar{F},\bar{\mathcal{T}}} = 1 + \sum_{\{F',\mathcal{T}'\}\in\mathcal{C}\setminus\{\bar{F},\bar{\mathcal{T}}\}} \frac{\langle(\bar{F},\bar{\mathcal{T}}),(F',\mathcal{T}')\rangle}{\|(\bar{F},\bar{\mathcal{T}})\|\|(F',\mathcal{T}')\|}$
$\beta_{\max} = \max\left\{\beta_{F',\mathcal{T}'} : \{F',\mathcal{T}'\}\in\mathcal{C}\right\}$
$\{F_\beta,\mathcal{T}_\beta\} = \operatorname{argmax}\left\{\beta_{F',\mathcal{T}'} : \{F',\mathcal{T}'\}\in\mathcal{C}\right\}$
$w_{\min} = \min\{1 - w(F') - w(\mathcal{T}') : \{F',\mathcal{T}'\}\in\mathcal{C}\}$
$\{F_w,\mathcal{T}_w\} = \operatorname{argmin}\{1 - w(F') - w(\mathcal{T}') : \{F',\mathcal{T}'\}\in\mathcal{C}\}$
$\alpha_o = \max\left\{\frac{\langle(F,\mathcal{T}),(F',\mathcal{T}')\rangle}{\|(F,\mathcal{T})\|\|(F',\mathcal{T}')\|} : \{F',\mathcal{T}'\}\in\mathcal{C}\right\}$
$\{F_o,\mathcal{T}_o\} = \operatorname{argmax}\left\{\frac{\langle(F,\mathcal{T}),(F',\mathcal{T}')\rangle}{\|(F,\mathcal{T})\|\|(F',\mathcal{T}')\|} : \{F',\mathcal{T}'\}\in\mathcal{C}\right\}$
if ($|\mathcal{T}| < 3$) or ($|\mathcal{T}|$ even) or ($1-w(F)-w(\mathcal{T}) < \varepsilon$) or ($\mathcal{T}$ has repetitions) or ($\{F,\mathcal{T}\}\in\mathcal{C}$) then
  return $\mathcal{C}$
else if ($N_{\max} > |\mathcal{C}|$) then
  return $\mathcal{C} \leftarrow \mathcal{C}\cup\{F,\mathcal{T}\}$.
else if ($\alpha_o > \alpha_{\min}$) and ($1 - w(F) - w(\mathcal{T}) > 1 - w(F_o) - w(\mathcal{T}_o)$) then
  return $\mathcal{C} \leftarrow \mathcal{C} - \{F_o,\mathcal{T}_o\} + \{F,\mathcal{T}\}$
else if ($1 - w(F) - w(\mathcal{T}) > w_{\min}$) then
  return $\mathcal{C} \leftarrow \mathcal{C} - \{F_w,\mathcal{T}_w\} + \{F,\mathcal{T}\}$
else if ($\beta_{\max} > \beta_o$) and ($\beta_{\max} > \beta_{F,\mathcal{T}}$) and ($1 - w(F) - w(\mathcal{T}) > w_{\min}\cdot w_o$) then
  return $\mathcal{C} \leftarrow \mathcal{C} - \{F_\beta,\mathcal{T}_\beta\} + \{F,\mathcal{T}\}$
end if
return $\mathcal{C}$
```

After a lot of testing, we choose a balance between keeping highly violated inequalities, and inequalities that cover *different* parts of the graph M^*. The basic ideas of our selection procedure are described in Algorithm 3.5. An algorithmic description of the actual separation procedure used in our implementation can be seen in Algorithm 3.6.

In Table 3.1 we show the impact of different variations on the selection rules outlined in

Algorithm 3.5; strategy 4 refers to the full Algorithm 3.5; strategy 3 is the same as strategy 4 but it does not perform steps 23-24 of the algorithm; strategy 2 is the same as strategy 3 but it does not perform steps 21-22; and strategy 1 is the same as strategy 2 but it does not perform steps 19-20. The numbers represent the aggregated fraction of the optimal value obtained over all TSPLIB instances with less than 2.000 cities that do not solve to optimality with CONCORDE, using DP-inequalities, and without branching (actually, since there is a random component in the overall CONCORDE separation routines, we perform four runs for each instance). This criteria left us with 10 instances, namely att532, d657, dsj1000, pcb1173, rat575, rl1323 rl1889, u1060, u1817 and u724. Note that the results suggest that the selection strategy outlined in Algorithm 3.5 has some advantage, but the results are not conclusive.

Table 3.1: Selecting DP Cuts

Strategy	Average	Geometric Mean
4	0.999806124	0.99981348
3	0.999805975	0.99979177
2	0.999795751	0.99979177
1	0.999796262	0.99979148

The chosen parameters for Algorithm 3.5 in our test-runs were $\varepsilon = 10^{-3}$, $N_{\max} = 500$, $\alpha_{\min} = 0.55$, $\beta_o = 1.5$ and $w_o = \frac{1}{2}$. Note that in Algorithm 3.5, the condition on line 15 just enforce that we do not add trivial (or dominated) DP-inequalities to our current set. The condition on line 17 just says to fill-up the set of inequalities up to the maximum number of desired inequalities. Condition on line 19 says that if we pick the *closest* inequality in our pool to the new inequality, and this inequality is close enough (defined by the parameter $\alpha_{\min}$), we replace the old inequality by the new one as long as the new inequality has better violation. Condition in line 21 says that if the new inequality has better violation than the worst violation that we have, we replace that inequality by our newly found inequality. Finally, condition in line 23 says that if we improve the *spread* of our set of inequalities, and the new inequality do not degrade too much our worst violation, we replace the inequality with the worst spread by our new inequality. This scheme assure that we keep the most violated inequality in our set, while trying to get inequalities that differ significantly from each other.

3.3.4 Putting it All Together

After all the previous discussion, we are ready to present the separation algorithm that we implemented:

Algorithm 3.6 DP-inequality separation (get_cuts($G^*, \alpha, \varepsilon, t$))

```
$\mathcal{C} \leftarrow \emptyset$
$\mathcal{L} \leftarrow$get_all_dominoes($G^*, \alpha$)
Compute dual graph $\bar{G^*}$ .
$V(M^*) = V(\bar{G^*})$, $E_{even} = \{e \in E(\bar{G^*}) : x_e < 1 - \varepsilon\}$, $E_{odd} = \{uv : D_{uv} \in \mathcal{L}\}$
for all $v \in V(M^*)$ do
  Compute min odd cycle $\{F, \mathcal{T}\}$ passing through $v$
  $\mathcal{C} \leftarrow$ add_if_new($\mathcal{C}, \{F, \mathcal{T}\}$)
  find_heuristic_cuts($v, \mathcal{C}, t$)
end for
return $\mathcal{C}$
```

Note that in Algorithm 3.6 we have some number of extra parameters, t is the time to spend in the heuristic random walk step at every node in $V(M^*)$ (set to 10 to 30 seconds), α is the bound parameter used to separate all dominoes (set to .55 in our tests), and ε is the minimum violation required (set to 10^{-6} in our tests) to consider an inequality as violated.

3.4 Shrinking and Non-Planar Graphs

Although the Domino Parity separation algorithm for planar support graphs is a major breakthrough in terms of exact separation algorithms, in practice, the cases where the support graph is planar are very rare. Another problem of the separation algorithm is that despise all speed-ups, it is still a very time consuming algorithm. A common approach to tackle this kind of problems is to use some kind of preprocessing on the input that somehow simplifies the problem to a more manageable size. In the context of the TSP there exists the concept of *safe shrinking* introduced by Padberg and Rinaldi [91] which is extensively used in modern TSP codes. They provide conditions for shrinking a pair of nodes that guarantee that if there was a violated cut for the TSP, then the shrunken graph will also have a violated inequality. In section 3.4.1 we provide a proof that some of these conditions also hold for DP-Inequalities.

Unfortunately, safe shrinking for DP-inequalities does not guarantee that the final shrunken graph is planar, but a natural thing to do is to keep doing unsafe shrinking until a planar graph is obtained, this approach is explained in section 3.5. But shrinking is not the only alternative to obtain planar graphs. A second alternative is to simply eliminate edges from the support graph until a planar graph is obtained, this approach is explained in section 3.5.1. Note that many more schemes are possible to generate planar graphs from a graph, and the two methods that we explain here are only heuristics, and in fact, neither dominates the other (in all of our tests, our code separate the DP-inequalities in the graphs obtained from both heuristics). We think that this problem admits much more study, but for the purposes of our code, the two heuristics presented proved more than enough in practice.

3.4.1 Safe Shrinking

In applying the DP-separation algorithm it is crucial to preprocess G^* to reduce the size of the graph that must be handled. Given a graph G, and two nodes $u, v \in V(G)$, let $G/\{u, v\}$ denote the graph obtained by collapsing the nodes u, v into a single node y and eliminating any resulting self-loop edges. This operation is called *shrinking* u, v in G. Padberg and Rinaldi [91] proved that under some simple conditions, if there exists a violated inequality for the TSP (that is, x^* is not in the convex hull of tours), then the shrunken graph will also have a violated inequality. Their result however does not imply that if there is a violated inequality from a particular class (like the Domino Parity inequalities) then there remains a violated inequality from the same class. The following result provides conditions that allow us to shrink pairs of nodes (u, v), guaranteeing that violated DP constraints will be available in the shrunken graph $G^*/\{u, v\}$ if violated DP constraints where present in G^*. These conditions are a sub-set of the original conditions described in Padberg and Rinaldi paper.

Theorem 3.5. Let $x^* \in SEP(\text{n})$ and let $u, v, t \in V(G^*)$ be such that $x^*_{uv} = 1$ and $x^*_{ut} + x_{tv} = 1$. If there exists a violated DP inequality in G^*, then there exists a DP inequality in $G^*/\{u,v\}$ with violation no less than the violation of the original inequality.

Although the conditions for safe shrinking for DP-inequalities are not as general as those for TSP cuts, the reader should bear in mind that the DP safe shrinking conditions account (on average) for above 95% of all safe shrinking for TSP cuts, thus giving it a significant impact in practice. Our implementation uses DP safe-shrinking before working on the actual separation step and then converts back the shrunken inequality to the original problem.

The remaining of this section is devoted to a proof of Theorem 3.5.

3.4.2 Domino Transformations

Let us start by refreshing the notation and hypothesis for Domino Parity inequalities. Let x be a fractional point for the Subtour Elimination Polyhedron (SEP) of the TSP, and let $\mu, \mathcal{T}$ be a DP-inequality, where $\mathcal{T}$ is the set of dominoes associated with the inequality, and $\mu \in \mathbb{Z}^E$ is defined as $\mu_e = |\{S \in \{F, E(A_T : B_T) : T \in \mathcal{T}\} : e \in S\}|$. There are some observations that are of interest about valid DP-Inequalities:

1. μ supports a cut[1] on G^*, but that cut may be empty. This is due to the fact that we only need μx to be even for all valid tours, and if the cut is empty the condition is trivially true.
2. $|\mathcal{T}|$ must be odd, but we do not require $|\mathcal{T}| \geq 3$; a DP-Inequality with one domino is still valid.
3. $\mathcal{T}$ may have repeated elements in it, and they are counted as many times as they appear.
4. $w(T) = x(\delta(T)) + x(E(A_T : B_T)) \geq 3$ for any domino and $x \in \text{SEP}(n)$.

In terms of notation, we will sometimes interpret $D \subseteq E(G^*)$ as a set of edges, and at other times as a vector in $\mathbb{Z}_+^{|E(G^*)|}$ with 1 for each edge in the set and 0 for all other edges. For a domino $T = \{A_T, B_T\} \in \mathcal{T}$ we use the short-hand $\delta(T)$ to mean $\delta(A_T \cup B_T)$ and we denote by C_T the set $V \setminus \{A_T \cup B_T\}$. Given a set S, we define the function $\mathbb{1}_S(a) = 1$ if

[1] A set of coefficients $\mu \in \mathbb{Z}_+^{|E(G^*)|}$ supports a cut in G^* if the set of edges with odd coefficients correspond to $\delta(H)$ for some $H \subseteq V(G)$

$a \in S$, zero otherwise.

Now we will present transformations that given a DP-Inequality $\mu, \mathcal{T}$, and x in SEP(n) will give us another DP-Inequality $\mu', \mathcal{T}'$ with slack less than or equal to the slack of the original constraint.

Duplicate Domino Elimination Let $T_1, T_2 \in \mathcal{T}$ such that $T_1 = T_2$, define $\mathcal{T}' = \mathcal{T} \setminus \{T_1, T_2\}$, and $\mu' = \mu - E(A_{T_1} : B_{T_1}) - E(A_{T_2} : B_{T_2})$. Note that the cut defined by μ' is the same cut defined by μ given that we subtract an even number for all entries. Note also that the slack of the new inequality is less than (or equal) to the slack of the original inequality.

$$
\begin{aligned}
&\mu x + \sum_{T \in \mathcal{T}} x(\delta(T)) - 3|\mathcal{T}| - 1 \\
=&\mu' x + \sum_{T \in \mathcal{T}'} x(\delta(T)) - 3|\mathcal{T}'| - 1 \\
&+ \underbrace{x(\delta(T_1)) + x(E(A_{T_1} : B_{T_1})) - 3}_{\geq 0} + \underbrace{x(\delta(T_2)) + x(E(A_{T_2} : B_{T_2})) - 3}_{\geq 0} \\
\geq&\mu' x + \sum_{T \in \mathcal{T}'} x(\delta(T)) - 3|\mathcal{T}'| - 1
\end{aligned}
$$

And thus obtaining a new DP-inequality $\mu', \mathcal{T}'$ with slack less than or equal to the slack of the original constraint, but with two less copies of $T_1 \in \mathcal{T}$. This allow us to assume that $\mathcal{T}$ does not contain multiple copies of the same domino in it.

Domino Reduction Let $T_o \in \mathcal{T}$, and let $A'_{T_o} \subseteq A_{T_o}$, $B'_{T_o} \subseteq B_{T_o}$ be such that $x(\delta(A'_{T_o} \cup B'_{T_o})) \leq x(\delta(T_o))$. Define:

- T'_o with $A-$partition A'_{T_o} and $B-$partition B'_{T_o}.
- $\mathcal{T}' = (\mathcal{T} \setminus \{T_o\}) \cup \{T'_o\}$.
- $\mu' = F\Delta\left(E(A_{T_o} : B_{T_o}) \setminus E(A'_{T_o} : B'_{T_o})\right) + \sum_{T \in \mathcal{T}'} E(A_T : B_T)$

To simplify notation, call $S = E(A_{T_o} : B_{T_o})$ and $S' = E(A' : B')$. Then, we have that

$$\begin{aligned}
\mu &= F + \sum_{T \in \mathcal{T}} E(A_T : B_T) \\
&= \underbrace{F \setminus \bigl(S \setminus S')\bigr) + F \cap \bigl(S \setminus S')\bigr)}_{F} + \underbrace{\bigl(S \setminus S'\bigr) + S'}_{S} + \sum_{T \in \mathcal{T} \setminus \{T_o\}} E(A_T : B_T) \\
&= \underbrace{F \setminus \bigl(S \setminus S'\bigr) + \bigl(S \setminus S'\bigr) \setminus F}_{F'} + 2F \cap \bigl(S \setminus S'\bigr) + \sum_{T \in \mathcal{T}'} E(A_T : B_T) \\
&= \mu' + 2F \cap \bigl(S \setminus S'\bigr).
\end{aligned}$$

And then the parity of all edges does not change from μ to μ', so the implied cut of μ and μ' are the same. Finally, note that the left-hand-side of the new inequality is less than or equal to that of the original constraint

$$\begin{aligned}
\mu x + \sum_{T \in \mathcal{T}} x(\delta(T)) &= \mu' x + \underbrace{2x(F \cap \bigl(S \setminus S'\bigr))}_{\geq 0} + \sum_{T \in \mathcal{T} \setminus \{T_o\}} x(\delta(T)) + \underbrace{x(\delta(T_o))}_{\geq x(\delta(T_o'))} \\
&\geq \mu' x + \sum \bigl(x(\delta(T)) : T \in \mathcal{T}'\bigr).
\end{aligned}$$

We thus obtaining a new DP-inequality $\mu', \mathcal{T}'$ with slack less than or equal to the slack of the original constraint, with T_o replaced by T_o'.

Domino Rotation Boyd et al. [18] have shown that in a DP inequality a domino $\{A_T, B_T\}$ in $\mathcal{T}$ can be *rotated* to $\{A_T, C_T\}$ without altering the inequality. To see this let $T_o \in \mathcal{T}$, and define:

- $T_o' : A' = A_{T_o}, B' = T_o^c$.
- $\mathcal{T}' = (\mathcal{T} \setminus \{T_o\}) \cup \{T_o'\}$
- $\mu' = \mu - E(A_{T_o} : B_{T_o}) + E(A'_{T_o} : B'_{T_o})$.

To simplify the notation, call $S_1 = E(A_{T_o} : B_{T_o})$, $S_2 = E(C_{T_o} : A_{T_o})$ and $S_3 = E(C_{T_o} : B_{T_o})$. Note that μ' supports the cut $\delta(H \Delta A_{T_o})$, that is,

$$Odd(\mu') = Odd(\mu) \Delta \delta(A_{T_o}) = \delta(H) \Delta \delta(A_{T_o}) = \delta(H \Delta A_{T_o})$$

where $Odd(\mu)$ denotes the edges having odd value μ_e. Note also that the left-hand-side of the inequality does not change:

$$\begin{aligned}
\mu x + \sum_{T \in \mathcal{T}} x(\delta(T)) &= (\mu - S_1)\, x + x(S_1) + \sum_{T \in \mathcal{T} \setminus \{T_o\}} x(\delta(T)) + \underbrace{x(S_2) + x(S_3)}_{x(\delta(T_o))} \\
&= \underbrace{(\mu - S_1 + S_2)\, x}_{\mu' x} + \sum_{T \in \mathcal{T}' \setminus \{T_o'\}} x(\delta(T)) + \underbrace{x(S_1) + x(S_3)}_{x(\delta(T_o'))} \\
&= \mu' x + \sum \left(x(\delta(T)) : T \in \mathcal{T}' \right).
\end{aligned}$$

We thus obtaining a new DP-inequality $\mu', \mathcal{T}'$ with slack equal to the slack of the original constraint, with T_o replaced by T_o' and H replaced by $H \Delta A_{T_o}$.

3.4.3 Proof of Safe-Shrinking Conditions for DP-inequalities

Let $(\mu, \mathcal{T})$ be a violated DP inequality for x^* and let u, v, t satisfy the conditions in Theorem 3.5, namely $x^*_{uv} = 1$ and $x^*_{ut} + x^*_{vt} = 1$. Let w denote the aggregated node for $V \setminus \{u, v, t\}$. This graph configuration is shown in Figure 3.1. We prove Theorem 3.5 by showing that there is another DP inequality $(\mu', \mathcal{T}')$, with violation at least that of the original, such that the coefficient of uv in the new inequality is zero. This proves that we can shrink uv into a single node and keep a violated DP inequality.

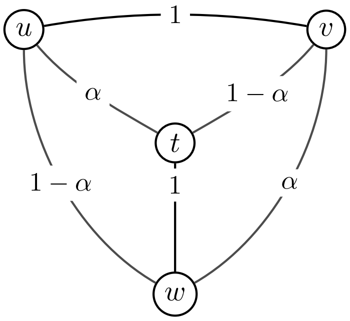


Figure 3.1: Safe-Shrinking configuration, here nodes u, v are the candidates to be shrunk into a single node. Values on the edges show the fractional value of x^* in the reduced graph.

The general coefficient for any edge in a DP-inequality can be written as

$$\text{coeff}(uv) = |\{T \in \mathcal{T} : uv \in \delta(T)\}| + |\{T : uv \in E(A_T : B_T)\}| + F_{uv}$$

Our proof will transform the inequality to reduce each component of the above expression to zero.

Claim 1: We may assume that for all $T \in \mathcal{T}$ we have $uv \notin \delta(T)$.

Proof. Let $T_o \in \mathcal{T}$ s.t. $uv \in \delta(T_o)$, and let assume that $u \in A_{T_o}, v \in T_o^c$, by rotating T_o and redefining $B_o' = T_o^c$ as in Section 3.4.2 we obtain an equivalent constraint $\mu', \mathcal{T}'$ but with $|\{T \in \mathcal{T} : uv \in \delta(T)\}| > |\{T \in \mathcal{T}' : uv \in \delta(T)\}|$. □

Now we may assume that:

$$\text{coeff}(uv) = |\{T : uv \in E(A_T : B_T)\}| + F_{uv}.$$

Claim 2: We may assume that for all $T \in \mathcal{T}$ such that $uv \in E(A_T : B_T)$ we have that $T = \{\{u\}, \{v\}\}$.

Proof. Let $T_o \in \mathcal{T}$ s.t. $uv \in E(A_{T_o} : B_{T_o})$, we may assume that $u \in A_{T_o}, v \in B_{T_o}$. Then, by applying Domino Reduction as in Section 3.4.2 with $B'_{T_o} = \{v\}$ and $A'_{T_o} = \{u\}$, and noting that $x(\delta(\{u,v\})) = 2 \leq x(\delta(S)), \forall S \subsetneq V$ we obtain our result. □

Combining Claim 2 with Duplicate Domino Elimination, and by using Claim 1, we may assume that there is at most one domino $T_I = \{\{u\}, \{v\}\}$ in $\mathcal{T}$ containing uv in $E(T_A : T_B)$. With this we may assume that

$$\text{coeff}(uv) = 1\!\!1_{\{T_I \in \mathcal{T}\}} + F_{uv}$$

Claim 3: We may assume that $\text{coeff}(uv) \leq 1$.

Proof. If $\text{coeff}(uv) = 2$ then $T_I \in \mathcal{T}$ and $F_{uv} = 1$. In this case, we will replace T_I by a new domino $T_{II} = \{\{u,v\}, \{t\}\}$ and change F by $F\Delta\{uv, ut, vt\}$ as in Figure 3.2.

The detailed definition of the new inequality is as follows:

- $T_{II} : A_{T_{II}} = \{u, v\}, B_{T_{II}} = \{t\}$.
- $\mathcal{T}' = (\mathcal{T} \setminus \{T_I\}) \cup \{T_{II}\}$.
- $\mu' = (F\Delta\{uv, ut, vt\}) + \sum\limits_{T \in \mathcal{T}'} E(A_T : B_T)$.

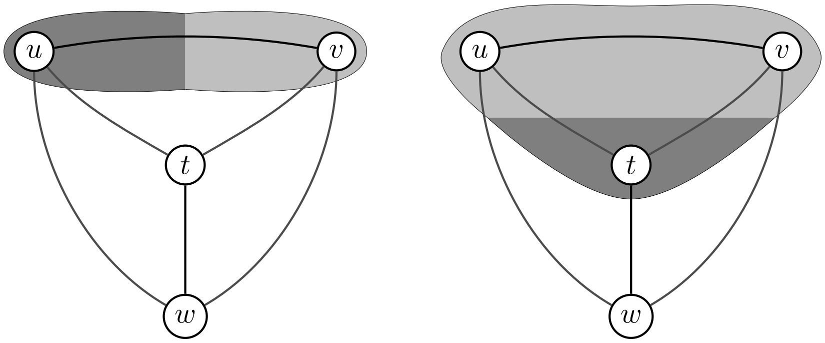


Figure 3.2: Domino Transformation I, left before, right after. Shaded areas represent the halves of the dominoes T_I and T_{II}.

To simplify notation call $S_1 = \{uv, vt, ut\}$, $S_2 = E(A_{T_I} : B_{T_I}) = \{uv\}$ and $S_3 = E(A_{T_{II}} : B_{T_{II}}) = \{vt, ut\}$. Note that μ' support the same cut as μ:

$$\begin{aligned}
\mu &= F + \sum \left(E(A_T : B_T) : T \in \mathcal{T}\right) \\
&= \underbrace{F \setminus S_1 + F \cap S_1}_{F} + \underbrace{S_1 - S_3}_{S_2} + \sum_{T \in \mathcal{T} \setminus \{T_I\}} E(A_T : B_T) \\
&= \underbrace{F \setminus S_1 + S_1 \setminus F}_{F'} + 2(S_1 \cap F - S_3) + S_3 + \sum_{T \in \mathcal{T}'} E(A_T : B_T) \\
&= \mu' + 2(S_1 \cap F - S_3).
\end{aligned}$$

Thus the difference between μ and μ' is an even vector. This proves that the implied cut remains unchanged. Note also that the left-hand-side value of the new inequality is less than or equal of the original constraint:

$$\begin{aligned}
\mu x + \sum_{T \in \mathcal{T}} x(\delta(T)) &= \mu' x + \underbrace{2x(S_1 \cap F)}_{\geq 2} - \underbrace{2x(S_3)}_{=2} + \sum_{T \in \mathcal{T} \setminus \{T_I\}} x(\delta(T)) + \underbrace{x(\delta(T_I))}_{=x(\delta(T_{II}))} \\
&\geq \mu' x + \sum \left(x(\delta(T)) : T \in \mathcal{T}'\right).
\end{aligned}$$

Thus we have obtained a DP-inequality with coeff$(uv) = 0$ and with violation at least as great as that of the original one. □

Claim 4: We may assume that $uv \notin F$.

Proof. If $F_{uv} = 1$, then $T_I \notin \mathcal{T}$ and we may assume that $u \in H, v \in V \setminus H$. We will redefine μ in such a way that $H' = H \cup \{v\}$. For that we redefine F as $F\Delta\{uv, vt, vw\}$ by defining a new inequality as

- $\mathcal{T}' = \mathcal{T}$.
- $F' = F\Delta\{uv, vt, vw\}$.
- $\mu' = F' + \sum\limits_{T \in \mathcal{T}'} E(A_T : B_T)$.

Note that μ' supports the cut $H \cup \{v\}$:

$$Odd(\mu') = Odd(\mu)\Delta(uv, vt, vw) = \delta(H)\Delta\delta(v) = \delta(H\Delta\{v\}) = \delta(H \cup \{v\}).$$

Also note that the left-hand-side of the new inequality is less than or equal to the left-hand-side of the original inequality:

$$\begin{aligned} \mu x + \sum_{T \in \mathcal{T}} x(\delta(T)) - \mu' x - \sum_{T \in \mathcal{T}'} x(\delta(T)) =& \, x(F) - x(F\Delta\{uv, vt, vw\}) \\ =& \, \underbrace{x(F \cap \delta(v))}_{\geq x(uv)} - \underbrace{x((F\Delta\delta(v)) \cap \delta(v))}_{\leq 2 - x(uv)} \\ \geq& \, 2x(uv) - 2 = 0. \end{aligned}$$

Thus the new inequality $(\mu', \mathcal{T}')$ has better violation than the original constraint and also has coeff$(uv) = 0$. □

Using Claim 4, if $T_I \notin \mathcal{T}$ then we have coeff$(uv) = 0$ and then proving our result. We may therefore assume that $T_I \in \mathcal{T}$ and that $u, t \in H$ and $v \in V \setminus H$ (the alternative case, when v and t are on the same side of the handle, is analogous to this one). This implies that μ_{vt} is odd.

Claim 5: We may assume that $vt \notin F$.

Proof. If $vt \in F$ then we can replace T_I by a new domino T_II, replace F by $F\Delta\{vt, uw\}$ and add v to the handle as shown in Figure 3.3, where $w = V \setminus \{u, v, t\}$.

Formally speaking, define:

- $T_{II} : A_{T_{II}} = \{w\}, B_{T_{II}} = \{u, v\}$.
- $\mathcal{T}' = (\mathcal{T} \setminus \{T_I\}) \cup \{T_{II}\}$.

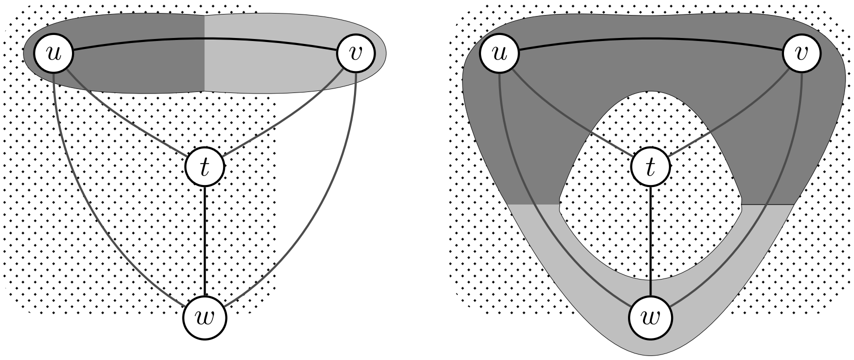


Figure 3.3: Domino Transformation II, left before, right after. Shaded areas represent halves of dominoes, the dotted area represent the handle.

- $\mu' = \underbrace{F\Delta\{vt, uw\}}_{F'} + \sum_{T\in\mathcal{T}'} E(A_T : B_T)$.

Clearly $w(T_I) = w(T_{II}) = 3$. Furthermore the left-hand-side of the new DP-inequality $\mu', \mathcal{T}'$ is less than or equal to the left-hand-side of the original DP-inequality $\mu, \mathcal{T}$:

$$\begin{aligned} x(F) - x(F') + \sum_{T\in\mathcal{T}} w(T) - \sum_{T\in\mathcal{T}'} w(T) &= x(F \cap \{vt, uw\}) - x((F\Delta\{vt, uw\}) \cap \{vt, uw\}) \\ &= \underbrace{x(F \cap \{vt, uw\})}_{\geq x(vt)} - \underbrace{x(\{vt, uw\} \setminus F)}_{\leq x(uw)} \\ &\geq (1-\alpha) - (1-\alpha) = 0. \end{aligned}$$

Note also that the cut supported by μ' is $\delta(H \cup \{v\})$:

$$Odd(\mu') = Odd(\mu)\Delta\{uv, vt, vw\} = Odd(\mu)\Delta\delta(v) = \delta(H)\Delta\delta(v) = \delta(H\Delta\{v\}) = \delta(H \cup \{v\}).$$

Thus we have a new DP inequality with $vt \notin F$. □

Since we may now assume that $vt \notin F$, and μ_{vt} is odd, there must exist a domino $T_{II} \in \mathcal{T}$ such that $v \in A_{T_{II}}, t \in B_{T_{II}}$. Moreover, since $\text{coeff}(uv) = \mathbb{1}_{\{T_I\in\mathcal{T}\}}$ then $u \in A_{T_{II}}$. Now, by Domino Reduction we may assume that $A_{T_{II}} = \{u, v\}$ and $B_{T_{II}} = \{t\}$. Note that this (plus parity, since $ut \notin \delta(H)$) implies that $\mu_{ut} \geq 2$.

Claim 6: We may assume that $ut \notin F$.

Proof. If $ut \in F$ then we will eliminate T_I, T_{II} from $\mathcal{T}$ and redefine H as $H \cup \{v\}$. Formally speaking, define:

- $\mathcal{T}' = \mathcal{T} \setminus \{T_I, T_{II}\}$.
- $\mu' = F\Delta\{ut, wv\} + \sum\limits_{T \in \mathcal{T}'} E(A_T : B_T)$.

Note that the cut defined by μ' is $H \cup \{v\}$ and that the new slack is less than or equal to the slack of the original inequality (details are analogous as those before). □

It follows that there exists a domino T_{III} such that $u \in A_{T_{III}}$ and $t \in B_{T_{III}}$. Using the same arguments as before we can transform T_{III} into T_{II} and then we have that $\mu_{ut} = |\{T_{II} \in \mathcal{T}\}| = \mu_{tv}$. But this contradicts the fact that μ_{vt} is odd and μ_{ut} is even. Thus completing the proof of theorem 3.5.

3.5 Finding a Planar Graph

In practice large TSP instances rarely produce LP solutions with planar support. In some cases an application of the safe shrinking rules can succeed in obtaining a final shrunken graph that is planar, but again this is quite rare. To succeed in practice it is necessary to modify G^* and the edge weights given by x^* in order to obtain a planar graph that can be used as a substitute in the DP separation algorithm. Unfortunately such a process means that we may lose some violated inequalities, but the hope is that if the new planar graph is close (under some measure) to G^* then we can still produce a good selection of cutting planes.

In the pioneering study by Boyd et al. [18], the researchers encountered non-planar graphs in a small number of test cases. In these instances their approach was to perform general (possibly unsafe) shrinking steps by hand, using a visual inspection of a drawing of G^* to guide the process to produce a planar graph. We also adopt such an approach, but we use planarity testing algorithms to automate the process as suggested in Vella [18].

Testing planarity is very efficient and algorithms are available that either return a planar embedding of a graph or a $K_{3,3}$ or K_5 minor. A straightforward way to use such an algorithm to obtain a planar graph is given in Algorithm 3.7.

This simple algorithm is often good enough for our purposes, and it is one of the implementations used in our tests. If $x^* \in SEP(\text{n})$ then the new shrunken fractional solution also satisfies the subtour-elimination constraints. A drawback is that the resulting fractional

Algorithm 3.7 Planarization by Shrinking (srk_planarize(G^*))

```
while $G^*$ is not planar do
    Let $K$ be a $K_{3,3}$ or $K_5$ minor of $G^*$
    Let $G^*[K]$ be the graph induced by $V(K)$ in $G^*$
    Let $u, v \in V(G^*[K])$ such that degree($v$),degree($u$)$\geq 3$
    $G^* \leftarrow G^*/\{u, v\}$, where $G^*/\{u, v\}$ is the graph resulting from shrinking $v, t$ in $G^*$
end while
return $G^*$
```

solution does not necessarily satisfy the *degree constraints*

$$x(\delta(\{v\})) = 2 \qquad \forall\, v \in V.$$

This detail is not crucial, however, since the DP inequalities and the separation algorithm are valid also for the *graphical traveling salesman problem*, where nodes and edges can be used more than once in the tour. A discussion of this point is given in Section 4 of Letchford [70].

3.5.1 Edge-Elimination Planarization

An alternative way to obtain planar graphs is to simply eliminate edges found in a $K_{3,3}$ or K_5 minor. Unfortunately, just deleting random edges from a minor returned by the planarity testing code performs very poorly in practice. An intuitive reason for this behavior is that there might be many forbidden minors sharing edges in the graph, so the edge deletion may not make substantial progress toward obtaining a planar graph. A second problem is that it does not take into account the weight of the edge to eliminate; since we want to modify as little as possible the original graph and x^*, we would prefer to remove light edges (that is, edges having small value x^*_e). With this in mind, we chose the edge to eliminate with a routine `find_bad_edge`(G^*). This routine uses binary search to identify the minimum-weight edge such that the subgraph containing all edges of greater weight is planar (a description of the procedure can be found in Algorithm 3.8).

Thus the minimum weight to eliminate from the graph to make it planar is at least the x^*_e value of the selected edge. Clearly this edge selection rule does not eliminate the problem of minors sharing edges, but during our tests it proved to be effective. Since the complexity of planarity testing is $\mathcal{O}(|V(G^*)|)$, the total complexity for `find_bad_edge`(G^*) is

Algorithm 3.8 Choosing edge to eliminate (`find_bad_edge`(G^*))

```
Sort edges by fractional value, such that $x_{e_i} \geq x_{e_{i+1}}, \forall i = 1, \ldots, |V(G^*)| - 1$
$lo \leftarrow 5$
$up \leftarrow |V(G^*)|$
while $lo \neq up$ do
    $md \leftarrow lo + \frac{up-lo}{2}$
    Let $G_{md} \leftarrow (V(G^*), \{e_i : i = 1, \ldots, md + 1\})$
    if $G_{md}$ is planar then
        $lo \leftarrow md + 1$
    else
        $up \leftarrow md$
    end if
end while
return $e_{lo}$
```

$\mathcal{O}(\log(|E(G^*)|)|E(G^*)|)$. A complete description of how this is used in our edge-elimination planarization heuristic is given in Algorithm 3.9.

Algorithm 3.9 Planarization by Edge Elimination (`elim_planarize`(G^*))

```
while $G^*$ is not planar do
    Let $e =$find_bad_edge$(G^*)$
    $G^* \leftarrow G^*/\{u, v\}$, where $e = \{u, v\}$
end while
return $G^*$
```

In our tests, we have found that the total weight of eliminated edges is usually quite low. A problem with the method, however, is that it produces a vector x^* that does not satisfy the degree constraints or the subtour-elimination constraints. This implies that the weight of the dominoes found during the domino-generation step may be negative, which may create negative weight cycles in M^*. To avoid this issue we simply set the weight of all negative dominoes to zero. Now, before returning the cuts found during the second phase of the algorithm, we re-compute exactly the actual violation for the inequality in the original graph.

Note that many more schemes are possible to generate planar graphs from a graph, and the two simple methods we presented here are only heuristics. Neither method dominates the other and in our computational tests we separate the DP inequalities in the graphs obtained from both methods. The problem of obtaining a planar graph that represents well an LP solution deserves further study, but for the purposes of our code the two heuristics proved to work well in practice.

3.6 Tightening DP Inequalities

After adding a cutting plane to an LP and re-solving, it is possible that we may obtain another fractional solution that differs very little from the one just separated. In this case, rather than generating new cuts it may be desirable to attempt to “fix up” some tight constraints (or those with small slack) currently in the LP or stored in a pool, by slightly modifying them in such a way as to make the new fractional point infeasible. This is certainly much faster than separating from scratch, and it does not require G^* to be planar. In fact, these tighten procedures are called on each DP-inequality found by our separation algorithm before adding it to the LP relaxation. This is done in the hope that the procedure can fix some of the error incurred in the separation algorithm due to the planarization phase of G^*, and in practice it helps to get better cuts. This type of approach has been very successful on other handle-tooth type inequalities in the work of Applegate et al. [7] and it had a great impact in our computational tests.

To formalize this notion of *simple modifications* for DP inequalities, note that every DP inequality is completely defined by a family of dominoes $\{A_i, B_i\}_{i=1}^{p}$ and a handle H. In our implementation, the simple modifications we consider are:

(1) Add a node to a domino.

(2) Add a node to the handle.

(3) Change sides of a node in a domino.

(4) Change sides of a node in a domino and the handle.

(5) Remove a node from a domino.

(6) Remove a node from the handle.

A node u is *relevant* in our heuristic if there exists $e \in \delta(u)$ such that it has a non-zero coefficient in the DP inequality. We begin by computing the move that improve the violation of the inequality the most from among all feasible moves in all relevant nodes, call such move the *best move*. While the best move reduces the slack of the constraint by at least some suitable factor ε, perform the move, and recompute the best move. If the best move has a value between $(-\varepsilon, \varepsilon)$ (that is, they change the slack, up or down, but very slightly), we first do moves that enlarge either the handle or a domino, then do moves that flip elements within a domino and then do moves that shrink a domino or a handle. We repeat this process until some improving move is found (i.e. the best move improve the violation by at least ε), and then go back to our greedy approach, or until we can not make any further move. Note that the algorithm will never cycle, since each move can only be performed once within each ε-improvement phase. If at the end of this process we have found a constraint with better violation, we have succeeded and return the resulting cut. In our tests ε was chosen as 10^{-6}.

It turns out that the second phase of the algorithm (while performing ε-moves), is the most crucial for the algorithm to work well in practice. Without it we were able to get some improved inequalities, but using the second phase where we first grow the handle and teeth of the inequality and then shrink them back, enabled us to generate many more violated inequalities. This effect has also been seen in the tightening of other TSP-inequalities by Applegate et al. [7].

3.7 Computational Results

In this section we present our computational experience with DP inequalities, we tested our routines on problems from the TSPLIB collection of Reinelt [93] having at least 3,000 cities; this size is to ensure that the problems are not *easy* to solve. This leaves us with 13 problems, which we split into two groups, one with problems having at most 7,000 cities, this is called the medium size instance set, and has 5 problems. The other set contains the problems having at least 7,000 cities, which leaves us with 8 problems, this set is called the large size instance set.

The routines described in the previous sections where implemented in the C programming language, and are available on-line at `http://www.isye.gatech.edu/~despinoz`. The planarity-testing functionality is provided by an implementation of J. Boyer of the Boyer and Myrvold [22] planarity-testing algorithm.

These routines where incorporated into the Concorde TSP code of Applegate et al. [7], which manage the cut pools as well as add a large set of heuristic cuts as well as subtours inequalities. We use ILOG CPLEX 6.5 as our LP solver of choice.

The computations where carried on a 2.66 GHz Xeon CPU with 2GB of RAM. The operating system is GNU/Linux.

Table 3.2: Domino Parity vs Local Cuts I. Here we present a comparison of domino-parity inequalities against local cuts in Concorde in the medium size instance set, the GAP is the percentage of gap closed by the routines above the bound obtained by Concorde without local cuts (mC0Z0 configuration). Running times are in hours.

Problem	Optimal	mC0Z0	mC48Z0			mC0Z3		
Name	Value	Bound	Bound	GAP	Time	Bound	GAP	Time
pcb3038	137694	137589	137658	65.714	41.30	137666	73.333	1.65
fl3795	28772	28700	28769	95.833	7.50	28761	84.722	8.61
fnl4461	182566	182471	182551	84.211	7.50	182529	61.053	0.86
rl5915	565530	565172	565377	57.263	98.80	565352	50.279	5.37
rl5934	556045	555725	555922	61.562	21.80	555810	26.562	4.87

Our first test was to compare domino parity cuts against the local cut procedure of Applegate et al. [6] as implemented in Concorde. We choose to use multiple passes through ours (and Concorde's) cutting plane routines. The base comparison is Concorde's default setting but without local cuts enabled (we call this setting mC0Z0). The local-cut runs are done with local-cuts of size up to 48, which is the largest size that Concorde can effectively

Table 3.3: Domino Parity vs Local Cuts II. Here we present a comparison of domino-parity inequalities against local cuts in Concorde in the large size instance set, the GAP is the percentage of gap closed by the routines above the bound obtained by Concorde without local cuts (mC0Z0 configuration). Running times are in hours.

Problem	Optimal	mC0Z0	mC48Z0			mC0Z3		
Name	Value	Bound	Bound	GAP	Time	Bound	GAP	Time
pla7397	23260728	23205647	23255280	90.109	70.50	23252107	84.349	5.31
rl11849	923288	922116	923053	79.949	118.00	922967	72.611	48.34
usa13509	19982859	19966278	19979209	77.987	81.20	19977859	69.845	28.53
brd14051	469385	469085	469321	78.667	53.20	469264	59.667	21.83
d15112	1573084	1572175	1572863	75.688	124.00	1572756	63.916	30.21
d18512	645238	644880	645166	79.888	73.90	645093	59.497	57.04
pla33810	66048945	65960860	65972887	13.654	19.00	65980036	21.770	15.54
pla85900	142382641	142252677	142265646	9.979	224.20	142271357	14.373	77.30

manage (we call this setting mC48Z0). The domino-parity runs are done without local cuts and with default settings for other parameters (we call this setting mC0Z3). The runs start with the degree constraints on the initial LP and edges as variables (which dynamically get in and out of the current LP formulation depending on their reduced cost). Table 3.2 shows our results on the medium size instances, while Table 3.3 shows our results on the large size instances.

Note that while the domino parity inequalities do not improve the lower bound as much as local cuts, they tend to run in a noticeable shorter time than local cuts, moreover, the bounds still improve noticeable with respect to the basic Concorde configuration without local cuts.

Table 3.4: Domino Parity with Local Cuts I. Here we present a comparison of domino-parity inequalities with local cuts in Concorde in the medium size instance set, the GAP is the percentage of gap closed by the routines above the bound obtained by Concorde with local cuts (mC48Z0 configuration). Running times are in hours.

Problem	Optimal	mC48Z0		mC48Z3		
Name	Value	Bound	Time	Bound	GAP	Time
pcb3038	137694	137658	65.714	137684	72.222	9.20
fl3795	28772	28769	95.833	28771	66.667	9.80
fnl4461	182566	182551	84.211	182558	46.667	9.80
rl5915	565530	565377	57.263	565479	66.667	28.10
rl5934	556045	555922	61.562	555994	58.537	21.50

Our second test consist on using Concorde's local cuts in conjunction with domino parity inequalities to see how much of the remaining gap we can close (we call this configuration mC48Z3). Since there is some randomness in our results, we did ten runs for each configuration for each problem, and then, compute geometric means of the lower bounds obtained and of the total running time. For the case of our test set of medium size instances, we run both configurations from scratch, however, due to the long running times for the larger instances, we choose to run domino parity configuration *after* the run with local cuts was finish. Table 3.4 presents the results in the medium size instance set, and Table 3.5 presents the results in the large size instance set. Note again that for the medium size instance set, the total running time for Concorde with domino parities and local cuts is faster than just

Table 3.5: Domino Parity with Local Cuts II. Here we present a comparison of domino-parity inequalities with local cuts in Concorde in the large size instance set, the GAP is the percentage of gap closed by the routines above the bound obtained by Concorde with local cuts (mC48Z0 configuration). Running times are in hours.

Problem	Optimal	mC48Z0		mC48Z3		
Name	Value	Bound	Time	Bound	GAP	Time
pla7397	23260728	23255280	90.109	23258947	67.309	268.20
rl11849	923288	923053	79.949	923209	66.383	104.40
usa13509	19982859	19979209	77.987	19981200	54.548	109.90
brd14051	469385	469321	78.667	469354	51.562	159.50
d15112	1573084	1572863	75.688	1572967	47.059	152.70
d18512	645238	645166	79.888	645195	40.278	186.50
pla33810	66048945	65972887	13.654	66001234	37.270	231.00
pla85900	142382641	142265646	9.979	142296660	26.509	174.10

doing local cuts, the speed up factor ranges between 9.77 and 2.03. Unfortunately, since we run the domino parity separation routines after we finish the local cut cutting loop for the larger instances, we can not make the same comparisons in this test set.

It is clear from these results that the domino parity inequalities greatly help to provide improved lower bounds from what was possible before, although, the improvements, decrease as the problems grow larger.

3.7.1 Solution of D18512 and PLA33810

Given the large improvements in the LP bounds obtained with domino parity inequalities and Concorde, we took these routines and tried to solve to optimality d18512 (which is a collection of cities in Germany) and pla33810 (which is a VLSI application at AT&T), these problems are two of the three unsolved problems in the TSPLIB.

For d18512, we use as starting point an LP relaxation obtained by Applegate et al. [7]. Using as upper bound the value of a tour found by Tamaki [96] plus one, we did three rounds of cutting planes and branching up to 1,000 nodes, (note that we keep adding cuts as we go along in the branch and bound tree, but the aggressiveness of the cutting phase decreases as we go deeper into the tree). From these runs we obtain a larger pool of valid cuts for the problem, which build on top of each other. This iterated procedure produced an LP relaxation with a value of 645,209. Using again the same upper bound as before of 645,239, we let the branch and bound procedure go from this starting point. This run required 424,241 nodes to prove the optimality of the tour found by Tamaki, with value 645,238. The total running time was approximately 57.5 years of CPU time. The runs were made in a cluster of 2.66GHz Intel Xeon processors.

Table 3.6: LP Bounds for d18512 and pla33810

Name	Optimal	Concorde (with pool)	Concorde+DP (with pool)	Gap Δ
d18512	645238	645202	645209	19.4%
pla33810	66048945	66018619	66037858	63.4%

For pla33810, we also use as starting point an LP relaxation obtained by Applegate et al. [7]. Using as upper bound one plus the value of a tour found by Helsgaun [56] of 66,050,499, we did five rounds of cutting planes with domino parity and branching. To our surprise, at the sixth iteration of this procedure, the branch and bound run stoped after exploring 577 nodes with an optimal solution of value 66,048,945. The total running time for this experiment was 15.7 years of CPU time. In order to double-verify our procedure, we took the LP relaxation obtained after this branch and bound run with all cuts, this gave us a bound of 66,037,858, and using as upper bound a value of 1 plus our improved solution, we run a new branch and bound test. This second test took only 86.6 days to finish, and explored 135 nodes and found again the optimal tour of value 66,037,858. Note that this is a slight improvement upon the best known tour reported by Helsgaun.

3.7.2 Final Comments on our Computational Tests

Our solutions of d18512 and pla33810 should be viewed only as evidence of the potential strength of the new procedures; the computational studies were made as we were developing our code and the runs were subject to arbitrary decisions to terminate tests as the code improved. The 33,810-city TSP is currently the largest test instance that has been solved, improving on the 24,978-city tour of Sweden computed by Applegate et al. The relatively small search tree for pla33810 may be due in part to the natural structure in the VLSI-derived data set that is not present in d18512.

APPENDIX A

RUNNING TIMES FOR QSOPT_EX COMPARED AGAINST QSOPT

A.1 Primal Simplex

Table A.1: Comparison for primal simplex on instances with hot-start. Here we show the total running time for the exact LP solver and the original QSopt code, we also show the percentage of time spend on solving the double LP approximation, the extended float approximation (if any), and the checking process in rational arithmetic for the exact LP solver code, the last column shows the ratio of the running time of the exact code versus the running time of the original QSopt code. All running times are in seconds. The runs where made using a Linux workstation with 4Gb of RAM, and with an AMD Opteron 250 CPU.

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
10teams	2255	230	0	73.11	0.00	19.94	0	2.06
25fv47	2392	821	3	71.13	0.00	19.51	1	3.98
80bau3b	12061	2262	2	82.14	0.00	7.41	1	1.57
a1c1s1	6960	3312	1	80.32	0.00	8.81	0	1.45
aa01	9727	823	21	97.80	0.00	1.35	14	1.52
aa03	9452	825	14	97.82	0.00	1.09	16	0.85
aa3	9452	825	13	97.40	0.00	1.35	15	0.86
aa4	7621	426	3	91.84	0.00	4.39	3	1.20
aa5	9109	801	11	97.24	0.00	1.44	10	1.14
aa6	7938	646	7	96.49	0.00	1.76	5	1.22
aflow40b	4170	1442	0	69.23	0.00	16.29	0	1.46
air03	10881	124	1	76.72	0.00	9.29	1	1.51
air04	9727	823	21	97.86	0.00	1.33	14	1.54
air05	7621	426	4	93.11	0.00	3.63	3	1.34
air06	9453	825	26	98.42	0.00	0.96	18	1.49
aircraft	11271	3754	2	87.97	0.00	5.31	1	1.58
arki001	2436	1048	1	12.57	42.70	26.54	0	9.78
bas1lp	9872	5411	4	63.61	0.00	17.02	2	1.91
baxter	42569	27441	13	92.66	0.00	3.35	11	1.18
baxter.pre	29709	18917	22	83.81	9.69	3.04	16	1.38

Continued on Next Page...

Table A.1 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
bnl2	5813	2324	2	88.78	0.00	5.35	1	1.55
car4	49436	16384	14	17.77	0.00	75.75	2	7.12
cari	1600	400	1	37.73	0.00	27.18	0	3.26
ch	8762	3700	4	70.82	16.39	7.27	3	1.68
co5	13767	5774	41	77.98	13.70	5.77	16	2.50
complex	2431	1023	20	81.43	0.00	17.82	11	1.82
cq5	12578	5048	20	62.48	26.49	6.88	9	2.16
cq9	23056	9278	203	73.95	16.13	8.27	42	4.86
crew1	6604	135	0	72.43	0.00	0.00	0	1.70
cycle	4760	1903	1	17.89	55.10	14.49	0	9.02
czprob	4452	929	0	65.11	0.00	17.13	0	1.60
d6cube	6599	415	3	88.86	0.00	7.14	1	1.84
danoint	1185	664	0	80.34	0.00	12.25	0	1.66
dbic1	226435	43200	726	99.22	0.00	0.43	691	1.05
dbir1	46159	18804	7	66.06	0.00	13.16	5	1.47
dbir2	46261	18906	8	67.72	0.00	12.96	6	1.19
dbir2.pre	32091	7228	11	80.63	0.00	7.47	8	1.48
de080285	2424	936	1	14.74	31.16	36.10	0	9.11
degen3	3321	1503	2	89.00	0.00	7.75	2	1.40
delf000	8592	3128	9	9.11	41.74	43.53	1	13.25
delf001	8560	3098	6	16.76	49.48	24.87	1	7.21
delf002	8595	3135	6	15.75	50.31	24.54	1	6.41
delf003	8525	3065	9	15.72	46.75	25.92	1	6.88
delf004	8606	3142	9	16.41	42.40	29.20	1	8.41
delf005	8567	3103	9	13.34	44.87	30.58	1	8.49
delf006	8616	3147	15	10.00	33.43	48.76	2	9.78
delf007	8608	3137	18	8.38	52.21	31.43	1	12.62
delf008	8620	3148	17	8.04	45.48	38.75	1	13.56
delf009	8607	3135	20	5.95	39.31	48.14	1	13.98
delf010	8619	3147	15	8.96	49.01	34.06	1	13.11
delf011	8605	3134	13	10.57	51.95	29.17	1	11.01
delf012	8622	3151	15	12.87	45.99	33.43	2	9.46
delf014	8642	3170	9	18.94	43.48	27.16	1	7.08
delf015	8632	3161	14	14.26	44.53	33.52	2	9.10
delf017	8647	3176	10	13.97	41.60	34.46	1	7.45
delf018	8667	3196	7	23.31	39.31	27.37	1	5.68
delf019	8656	3185	6	23.58	42.80	22.76	1	5.40
delf020	8685	3213	11	15.83	50.80	24.62	1	8.20
delf021	8679	3208	11	17.88	50.06	22.78	2	6.96
delf022	8686	3214	10	18.48	48.98	22.93	1	7.61
delf023	8686	3214	12	25.46	45.48	19.63	1	10.35

Continued on Next Page…

Table A.1 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
delf024	8673	3207	19	8.49	53.10	29.45	1	15.34
delf025	8661	3197	13	9.42	54.03	26.75	1	11.85
delf026	8652	3190	13	10.62	55.03	24.47	1	12.35
delf027	8644	3187	10	12.75	51.80	24.21	1	9.42
delf028	8629	3177	11	12.53	52.41	24.38	1	10.50
delf029	8633	3179	11	16.59	49.52	23.60	1	7.47
delf030	8668	3199	12	12.89	51.31	26.31	1	8.43
delf031	8631	3176	11	14.98	50.68	24.07	1	7.59
delf032	8663	3196	13	13.31	54.34	23.21	2	8.45
delf033	8629	3173	11	13.02	55.28	21.48	1	9.35
delf034	8630	3175	13	14.99	49.21	26.82	1	15.34
delf035	8661	3193	12	13.72	52.70	24.27	1	11.40
delf036	8629	3170	12	13.43	55.00	22.12	1	10.13
deter0	7391	1923	1	83.75	0.00	6.38	1	1.46
deter1	21264	5527	7	94.37	0.00	2.25	6	1.26
deter2	23408	6095	10	89.16	2.83	3.57	10	1.05
deter3	29424	7647	13	96.03	0.00	1.64	12	1.15
deter4	12368	3235	3	83.70	3.78	5.78	2	1.87
deter5	19632	5103	6	93.91	0.00	2.40	5	1.19
deter6	16368	4255	4	92.77	0.00	3.03	3	1.43
deter7	24528	6375	11	89.02	3.01	3.52	7	1.49
deter8	14736	3831	3	91.33	0.00	3.60	2	1.32
df2177	10358	630	3	57.28	0.00	37.09	1	2.23
dfl001	18301	6071	321	74.91	22.55	2.40	212	1.51
dfl001.pre	12953	3881	234	58.74	39.19	1.93	100	2.34
disctom	10399	399	13	98.16	0.00	1.10	10	1.30
ex3sta1	25599	17443	823	4.85	62.45	31.97	45	18.31
fast0507	63516	507	30	95.03	0.00	1.94	24	1.27
fit2d	10525	25	24	97.62	0.00	0.91	21	1.15
fit2p	16525	3000	18	97.45	0.00	1.09	16	1.12
fome20	139602	33874	264	98.95	0.00	0.61	214	1.23
fome21	279204	67748	745	99.29	0.00	0.41	650	1.15
fxm2-16	9502	3900	2	79.82	0.00	9.28	1	1.50
fxm2-6	3692	1520	0	63.05	0.00	18.47	0	1.70
fxm3_6	15692	6200	3	80.59	0.00	8.45	2	1.47
fxm4_6	53132	22400	16	87.29	0.00	5.62	3576	0.00
ge	21197	10099	35	68.30	19.58	9.59	18	1.93
gen4.pre	5648	1475	121239	0.00	1.60	98.32	77	1574.72
greenbea	7797	2392	16	93.33	0.00	4.05	9	1.67
greenbeb	7797	2392	9	62.48	0.00	29.80	5	1.79
grow15	945	300	1	10.85	0.00	70.68	0	12.98

Continued on Next Page. . .

Table A.1 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
grow22	1386	440	2	29.34	0.00	57.13	0	11.76
grow7	441	140	0	5.39	0.00	77.08	0	17.67
ken-11	36043	14694	11	93.05	0.00	2.72	8	1.32
ken-18	259826	105127	1606	99.56	0.00	0.23	1544	1.04
ken-18.pre	129203	39856	524	99.38	0.00	0.30	478	1.09
kent	47920	31300	4	65.04	0.00	14.48	2	1.74
kl02	36770	71	1	68.86	0.00	0.00	1	1.52
large000	11072	4239	20	8.29	56.47	29.88	1	14.12
large001	10996	4162	9	20.64	32.48	31.24	2	5.63
large002	11084	4249	31	10.91	52.73	30.66	3	11.27
large003	11035	4200	16	18.80	48.12	22.98	2	7.04
large004	11086	4250	19	13.47	38.85	37.25	2	9.89
large005	11074	4237	14	20.27	48.18	21.74	3	5.53
large006	11086	4249	22	11.75	51.68	28.84	2	10.22
large007	11072	4236	23	13.27	49.03	30.23	3	8.99
large008	11085	4248	24	11.81	50.73	30.06	2	11.19
large009	11074	4237	28	11.05	43.00	39.43	3	10.83
large010	11084	4247	23	11.53	51.15	29.41	2	9.88
large011	11073	4236	22	14.36	45.05	33.24	2	9.23
large012	11091	4253	23	13.10	51.85	27.33	2	9.60
large013	11086	4248	21	16.56	50.54	25.59	3	7.84
large014	11109	4271	19	14.22	53.99	23.64	3	7.50
large015	11103	4265	20	14.50	48.92	28.93	2	8.20
large016	11125	4287	22	13.23	54.38	24.99	2	9.35
large017	11114	4277	14	22.84	39.04	28.88	3	5.17
large018	11134	4297	12	20.34	48.56	22.45	2	5.75
large019	11136	4300	11	23.09	45.28	22.31	2	5.31
large020	11152	4315	17	16.42	54.76	20.58	3	5.94
large021	11149	4311	18	19.57	53.18	19.44	3	6.19
large022	11146	4312	19	18.33	57.37	17.13	3	7.17
large023	11137	4302	18	21.37	49.30	20.55	3	5.79
large024	11123	4292	25	12.93	54.73	25.06	3	9.57
large025	11129	4297	32	9.23	47.91	35.84	3	11.99
large026	11108	4284	26	10.86	52.73	28.48	2	11.07
large027	11096	4275	17	15.34	55.37	20.35	2	7.70
large028	11135	4302	22	15.04	55.16	21.65	3	7.44
large029	11133	4301	25	15.29	52.20	25.06	3	9.60
large030	11108	4285	21	16.13	56.09	19.76	3	6.93
large031	11120	4294	26	12.42	47.00	33.94	3	9.95
large032	11119	4292	39	8.87	36.87	49.42	3	12.06
large033	11090	4273	19	14.63	57.79	19.38	2	8.11

Continued on Next Page…

Table A.1 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
large034	11125	4294	37	12.85	39.07	43.00	4	9.07
large035	11122	4293	46	7.55	46.95	39.56	3	13.55
large036	11104	4282	33	10.82	39.04	45.04	3	12.22
lpl3	44366	10828	4	90.47	0.00	0.00	3	1.29
maros	2289	846	0	55.72	0.00	27.73	0	1.52
mitre	12778	2054	0	57.64	0.00	17.92	0	2.04
mkc	8736	3411	0	27.62	0.00	34.80	0	2.55
mod011	15438	4480	1	65.50	0.00	13.75	0	1.75
model11	25344	7056	11	93.68	0.00	3.20	9	1.20
model2	1591	379	0	45.01	0.00	32.47	0	2.49
model9	13136	2879	19	33.74	39.71	19.82	7	2.90
momentum1	47854	42680	36	91.93	0.00	4.46	52	0.68
momentum2	27969	24237	240	57.43	37.75	2.70	93	2.57
mzzv42z	22177	10460	106	99.46	0.00	0.25	5	21.23
nemsemm1	75358	3945	8	47.57	0.00	21.25	4	2.24
nemsemm2	49076	6943	6	73.83	0.00	11.03	3	1.65
neos	515905	479119	3693	99.65	0.00	0.17	13516	0.27
neos1	133473	131581	650	99.37	0.00	0.32	688	0.95
neos2	134128	132568	1566	99.71	0.00	0.17	1149	1.36
neos3	518833	512209	84821	90.37	9.58	0.03	32243	2.63
nesm	3585	662	1	24.61	50.55	13.88	0	3.77
net12	28136	14021	5	89.91	0.00	4.54	3	1.57
nsct1	37882	22901	5	62.33	0.00	14.89	3	1.69
nsct2	37984	23003	5	65.83	0.00	13.84	2	2.30
nsct2.pre	19094	7797	5	72.45	0.00	10.76	4	1.26
nsir1	10124	4407	1	49.73	0.00	20.41	0	2.44
nsrand-ipx	7356	735	1	30.92	0.00	26.57	0	3.25
nug07	1533	602	1	91.90	0.00	7.04	1	1.86
nug08	2544	912	4	88.80	0.00	10.19	3	1.42
nug12	12048	3192	1990	96.72	0.00	3.26	1073	1.85
nug15	28605	6330	73010	96.10	2.11	1.79	47550	1.54
nw04	87518	36	4	49.53	0.00	16.15	2	1.77
nw14	123482	73	7	74.42	0.00	0.00	6	1.13
orna1	1764	882	4	3.20	0.00	46.87	0	38.57
orna2	1764	882	4	3.07	0.00	46.79	0	39.29
orna3	1764	882	5	3.83	0.00	48.80	0	30.80
orna4	1764	882	8	9.20	0.00	66.50	0	47.32
orna7	1764	882	5	7.72	0.00	45.40	0	15.67
osa-07	25067	1118	2	70.50	0.00	11.01	1	1.63
osa-60	243246	10280	74	92.79	0.00	2.55	65	1.15
osa-60.pre	234334	10209	133	97.16	0.00	1.00	107	1.24

Continued on Next Page…

Table A.1 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
p010	29090	10090	12	87.94	0.00	7.72	8	1.46
p05	14590	5090	3	80.17	0.00	10.75	2	1.50
pcb1000	3993	1565	1	65.26	0.00	24.75	0	1.91
pcb3000	10770	3960	6	71.14	0.00	23.48	3	1.61
pds-06	38536	9881	4	83.89	0.00	7.37	2	1.47
pds-100	661603	156243	32125	99.95	0.00	0.04	34898	0.92
pds-20.pre	89523	10240	21	93.00	0.00	3.07	17	1.23
pds-30	204942	49944	1017	99.60	0.00	0.24	872	1.17
pds-40	279703	66844	3097	99.79	0.00	0.13	2195	1.41
pds-50	353155	83060	5689	99.87	0.00	0.08	6087	0.93
pds-60	429074	99431	14586	99.94	0.00	0.03	12250	1.19
pds-70	497255	114944	22021	99.95	0.00	0.03	19191	1.15
pds-80	555459	129181	24105	99.95	0.00	0.03	21683	1.11
pds-90	609494	142823	29525	99.96	0.00	0.02	29515	1.00
perold	2001	625	10	13.20	0.00	80.80	0	20.29
pf2177	10628	9728	4	78.55	0.00	17.39	2	1.58
pgp2	13254	4034	2	50.16	18.22	13.91	1	2.59
pilot.we	3511	722	8	10.50	33.30	40.58	1	12.29
pilot4	1410	410	4	5.90	19.09	61.33	0	21.99
pilotnov	3147	975	13	36.70	0.00	61.20	1	17.32
pldd000b	6336	3069	4	32.40	30.13	27.40	1	3.53
pldd001b	6336	3069	5	36.81	26.33	26.97	1	3.24
pldd002b	6336	3069	5	37.00	27.37	25.71	1	3.14
pldd003b	6336	3069	4	35.30	28.13	26.21	1	3.34
pldd004b	6336	3069	5	39.72	27.22	22.95	2	2.98
pldd005b	6336	3069	4	37.24	28.49	23.72	2	2.82
pldd006b	6336	3069	5	38.70	27.77	23.37	1	3.08
pldd007b	6336	3069	5	38.77	27.92	23.50	2	2.96
pldd008b	6336	3069	5	37.32	29.46	23.18	1	3.31
pldd009b	6336	3069	5	37.02	29.67	23.22	1	3.15
pldd010b	6336	3069	5	38.77	28.36	23.04	2	2.94
pldd011b	6336	3069	7	34.06	33.60	24.43	1	4.61
pldd012b	6336	3069	5	39.91	27.56	22.72	2	3.04
pltexpa3_16	102522	28350	8	27.84	34.75	16.15	2	3.74
pltexpa3_6	16042	4430	1	12.80	22.02	29.06	0	7.73
pltexpa4_6	97258	26894	8	26.59	36.20	15.93	2	4.05
primagaz	12390	1554	2	87.31	0.00	5.26	1	1.37
progas	3075	1650	40	1.95	0.00	74.38	1	55.85
protfold	3947	2112	2	83.23	0.00	12.36	1	1.08
qap12	12048	3192	484	87.75	0.00	12.20	454	1.07
qiu	2032	1192	2	19.26	71.08	6.09	0	6.02

Continued on Next Page…

Table A.1 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
r05	14690	5190	5	83.73	0.00	8.33	3	1.67
rail2586	923269	2586	7088	99.32	0.00	0.31	7638	0.93
rail4284	1096894	4284	28088	99.80	0.00	0.09	25266	1.11
rail507	63516	507	29	94.80	0.00	2.03	25	1.18
rail516	47827	516	11	91.26	0.00	3.29	8	1.46
rail582	56097	582	24	94.17	0.00	2.15	21	1.14
rd-rplusc-21	126521	125899	69	69.56	0.00	26.32	41	1.70
rentacar	16360	6803	2	84.18	0.00	7.77	1	1.88
rlfddd	61521	4050	1	46.71	0.00	0.00	1	1.56
rlfdual	74970	8052	6	88.16	0.00	0.00	5	1.20
rlfprim	66918	58866	85	99.12	0.00	0.00	42	2.04
roll3000	3461	2295	1	77.45	0.00	11.01	0	1.49
rosen10	6152	2056	4	87.80	0.00	7.18	3	1.31
rosen2	3080	1032	2	78.35	0.00	11.52	1	1.53
route	44817	20894	21	95.02	0.00	2.23	15	1.35
sc205-2r-1600	70427	35213	9	89.21	0.00	4.90	42613	0.00
sc205-2r-200	8827	4413	1	86.87	0.00	6.03	1	1.33
sc205-2r-400	17627	8813	7	95.54	0.00	1.95	6	1.14
sc205-2r-800	35227	17613	25	97.56	0.00	1.08	5670	0.00
scagr7-2b-64	20003	9743	9	94.27	0.00	2.42	7	1.33
scagr7-2c-64	5027	2447	0	71.02	0.00	11.87	0	1.61
scagr7-2r-108	8459	4119	1	81.65	0.00	7.84	1	2.00
scagr7-2r-216	16883	8223	5	92.39	0.00	3.15	4	1.25
scagr7-2r-432	33731	16431	23	90.14	2.78	3.19	23	0.99
scagr7-2r-54	4247	2067	0	68.88	0.00	14.20	0	1.57
scagr7-2r-64	5027	2447	0	72.61	0.00	12.31	0	1.80
scagr7-2r-864	67427	32847	119	94.91	2.36	1.27	127	0.94
scfxm1-2b-16	6174	2460	1	70.18	0.00	14.77	0	1.59
scfxm1-2b-64	47950	19036	54	79.65	14.93	2.92	28	1.95
scfxm1-2r-128	47950	19036	57	77.19	17.72	2.73	41	1.41
scfxm1-2r-16	6174	2460	1	71.12	0.00	15.11	0	1.61
scfxm1-2r-256	95694	37980	272	89.98	7.90	1.15	184	1.48
scfxm1-2r-27	10277	4088	2	77.98	0.00	10.88	1	1.44
scfxm1-2r-32	12142	4828	2	83.58	0.00	8.11	2	1.46
scfxm1-2r-64	24078	9564	15	67.63	22.77	5.23	9	1.74
scfxm1-2r-96	36014	14300	30	68.55	23.98	4.01	18	1.64
scrs8-2r-256	16961	7196	1	62.42	0.00	16.82	0	1.90
scrs8-2r-512	33857	14364	4	84.95	0.00	6.65	3	1.41
scsd8-2b-64	41040	5130	29	90.86	3.90	2.30	26	1.11
scsd8-2c-64	41040	5130	8	70.02	11.70	8.11	4	1.94
scsd8-2r-108	17360	2170	3	61.22	16.97	10.06	1	2.03

Continued on Next Page…

Table A.1 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
scsd8-2r-216	34640	4330	18	86.97	6.13	3.04	10	1.86
scsd8-2r-432	69200	8650	79	93.66	3.17	1.40	56	1.40
scsd8-2r-54	8720	1090	1	48.16	16.40	16.51	0	2.63
scsd8-2r-64	10320	1290	1	51.22	19.54	13.40	1	2.04
scsd8	3147	397	1	19.71	59.79	12.35	0	5.04
sctap1-2b-64	40014	15390	3	85.07	0.00	0.00	2	1.49
sctap1-2r-108	16926	6510	0	59.08	0.00	0.00	0	1.95
sctap1-2r-216	33774	12990	1	78.39	0.00	0.00	1	1.42
sctap1-2r-480	74958	28830	4	83.04	0.00	0.00	4	1.17
seymour	6316	4944	4	93.86	0.00	2.86	3	1.26
seymourl	6316	4944	4	93.92	0.00	2.91	3	1.26
sgpf5y6	554711	246077	7188	78.93	20.78	0.13	1316	5.46
sgpf5y6.pre	58519	19499	28	5.70	86.56	3.36	1	18.98
slptsk	6208	2861	33	23.75	0.00	63.34	7	4.60
small002	1853	713	1	5.19	45.45	34.18	0	20.02
small006	1848	710	1	10.14	43.12	28.44	0	12.96
small007	1848	711	1	10.11	46.65	27.18	0	11.14
small008	1846	712	1	8.96	46.99	27.40	0	12.80
small009	1845	710	1	11.60	43.02	27.60	0	9.59
small010	1849	711	1	10.97	40.43	28.96	0	10.02
small015	1813	683	1	9.98	39.38	29.94	0	10.80
south31	53846	18425	137	97.29	0.00	1.11	108	1.27
sp97ar	15862	1761	9	90.35	0.00	3.92	5	1.68
stair	823	356	7	1.68	0.00	93.79	0	72.29
stocfor2	4188	2157	1	73.08	0.00	11.76	0	1.60
stocfor3	32370	16675	45	97.54	0.00	1.01	41	1.11
stormG2_1000	1787306	528185	36698	99.92	0.00	0.03	29504	1.24
stormG2_1000.pre	1410155	377036	9145	99.73	0.00	0.12	8922	1.02
stormG2-125	223681	66185	325	98.80	0.00	0.53	193	1.68
stormG2-125.pre	176405	47161	51	94.35	0.00	2.43	52	0.98
stormg2-27	48555	14441	8	89.83	0.00	4.29	6	1.31
stormg2-8	14602	4409	1	70.29	0.00	12.86	0	1.82
sws	26775	14310	1	46.99	0.00	20.82	1	2.62
t0331-4l	47579	664	124	84.11	0.00	13.52	99	1.25
t1717	74436	551	75	93.83	0.00	4.05	70	1.08
testbig	48836	17613	24	97.11	0.00	1.22	22	1.09
ulevimin	51195	6590	70	96.52	0.00	1.84	318	0.22
us04	28179	163	4	80.39	0.00	7.18	3	1.55
watson_1	585082	201155	742	76.73	18.42	2.27	919	0.81
watson_1.pre	239575	65266	317	81.18	12.77	3.05	247	1.28
wood1p	2838	244	2	9.04	0.00	55.05	0	14.15

Continued on Next Page…

Table A.1 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
woodw	9503	1098	1	81.21	0.00	10.84	1	1.82

Table A.2: Comparison for primal simplex on instances without hot-start. Here we show the total running time for the exact LP solver and the original QSopt code, we also show the percentage of time spend on solving the double LP approximation, the extended float approximation (if any), and the checking process in rational arithmetic for the exact LP solver code, the last column shows the ratio of the running time of the exact code versus the running time of the original QSopt code. All running times are in seconds. The runs where made using a Linux workstation with 4Gb of RAM, and with an AMD Opteron 250 CPU.

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
atlanta-ip	70470	21732	6288	34.23	65.60	0.13	525	11.98
co9	25640	10789	1458	5.16	94.09	0.48	99	14.75
d2q06c	7338	2171	280	5.27	71.55	21.83	12	23.97
dano3mip	17075	3202	2268	5.16	94.22	0.56	127	17.92
dano3mip.pre	16988	3151	8261	2.85	96.75	0.38	123	67.06
de063155	2340	852	4	12.51	71.86	7.18	0	27.64
de063157	2424	936	9	2.57	90.11	5.03	0	87.44
delf013	8588	3116	33	9.52	72.11	14.28	1	30.15
ds	68388	656	185612	98.96	1.03	0.00	82	2262.15
fome11	36602	12142	37831	3.23	96.76	0.01	432	87.54
fome12	73204	24284	18127	11.11	88.83	0.05	1040	17.42
fome13	146408	48568	31938	10.10	89.84	0.05	3384	9.44
fxm3_16	105502	41340	2100	6.76	92.89	0.07	103	20.31
gen	3329	769	4383	4.84	71.66	23.09	29	151.28
gen1	3329	769	2885	7.79	69.11	22.69	29	100.85
gen2	4385	1121	29028	0.34	7.62	91.91	71	408.09
gen4	5834	1537	133088	0.28	63.13	36.52	106	1257.43
iprob	6002	3001	1	83.72	0.00	5.29	1	1.49
jendrec1	6337	2109	23	86.04	0.00	6.74	13	1.81
l30	18081	2701	7371	0.16	99.83	0.00	150	49.17
lp22	16392	2958	3566	10.37	88.89	0.71	161	22.22
lp22.pre	11565	2872	3057	12.16	87.15	0.67	148	20.60
maros-r7	12544	3136	1367	0.96	88.81	5.49	14	100.84
mod2	66502	34774	10375	3.22	96.48	0.20	1031	10.06

Continued on Next Page...

Table A.2 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
mod2.pre	54286	27186	6796	7.53	92.06	0.30	976	6.97
model10	19847	4400	2390	2.70	90.05	6.76	101	23.72
model3	5449	1609	86	6.72	90.58	2.02	4	24.63
model4	5886	1337	116	6.68	90.69	1.65	5	24.08
model5	13248	1888	114	20.60	76.87	1.26	5	23.32
model6	7097	2096	157	15.86	69.46	13.50	6	24.32
model7	11365	3358	403	10.50	86.90	2.12	20	20.55
momentum3	70354	56822	37243	2.04	45.11	52.19	1121	33.22
msc98-ip	36993	15850	3443	20.79	79.17	0.01	103	33.56
mzzv11	19739	9499	618	24.93	74.89	0.04	59	10.49
nemspmm1	10994	2372	423	15.47	83.28	0.99	19	22.07
nemspmm2	10714	2301	575	4.66	91.31	3.55	28	20.40
nemswrld	34312	7138	9893	2.77	93.79	3.28	586	16.88
neos.pre	476610	440494	3227	99.89	0.00	0.00	2923	1.10
nl	16757	7039	437	9.19	90.37	0.22	29	15.01
nsir2	10170	4453	11	14.41	74.38	2.00	0	23.91
pilot.ja	2928	940	59	12.44	65.10	19.57	1	41.89
pilot	5093	1441	618	2.11	38.23	57.71	13	47.74
pilot87	6913	2030	9493	0.64	25.86	72.69	31	301.75
pilot87.pre	6375	1885	7518	0.72	9.14	89.13	34	224.28
rat1	12544	3136	1988	0.58	98.80	0.38	16	123.99
rat5	12544	3136	2018	2.34	16.21	79.37	12	169.72
rat7a	12544	3136	5988	2.66	97.32	0.00	56	106.20
self	8324	960	53900	0.09	73.26	26.26	502	107.28
stat96v1	203467	5995	11054	2.93	97.02	0.00	565	19.56
stat96v4	65385	3173	71393	0.46	85.62	8.33	2897	24.64
stat96v5	78086	2307	13770	0.68	42.34	44.62	78	176.90
stp3d	364368	159488	102211	45.55	54.43	0.01	6795	15.04
watson_2	1023874	352013	86852	8.94	90.99	0.02	8091	10.73
world	67240	34506	32604	0.98	98.97	0.03	1034	31.52
world.pre	55916	27057	9320	10.11	89.79	0.04	872	10.69

A.2 Dual Simplex

Table A.3: Comparison for dual simplex on instances with hot-start. Here we show the total running time for the exact LP solver and the original QSopt code, we also show the percentage of time spend on solving the double LP approximation, the extended float approximation (if any), and the checking process in rational arithmetic for the exact LP solver code, the last column shows the ratio of the running time of the exact code versus the running time of the original QSopt code. All running times are in seconds. The runs where made using a Linux workstation with 4Gb of RAM, and with an AMD Opteron 250 CPU.

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
10teams	2255	230	1	87.90	0.00	7.87	0	1.37
25fv47	2392	821	4	73.96	0.00	17.92	1	3.23
80bau3b	12061	2262	2	82.65	0.00	7.58	1	2.07
a1c1s1	6960	3312	0	23.39	0.00	33.33	0	4.17
aa01	9727	823	8	94.83	0.00	2.99	6	1.20
aa03	9452	825	4	91.76	0.00	3.82	3	1.10
aa3	9452	825	3	91.75	0.00	3.93	3	1.21
aa4	7621	426	2	83.78	0.00	8.56	1	1.34
aa5	9109	801	3	91.74	0.00	4.06	3	1.14
aa6	7938	646	2	86.62	0.00	6.36	1	1.39
aflow40b	4170	1442	1	93.19	0.00	3.09	1	1.45
air03	10881	124	1	57.64	0.00	16.94	0	1.73
air04	9727	823	8	94.90	0.00	2.91	6	1.21
air05	7621	426	2	87.89	0.00	5.43	1	1.34
air06	9453	825	4	91.42	0.00	4.18	3	1.09
aircraft	11271	3754	1	81.60	0.00	7.71	1	1.39
arki001	2436	1048	11	1.75	93.21	2.94	0	55.48
bas1lp	9872	5411	51	97.56	0.00	1.09	17	3.04
baxter	42569	27441	13	92.25	0.00	3.48	11	1.18
baxter.pre	29709	18917	8	72.84	8.54	8.79	4	1.79
bnl2	5813	2324	2	89.32	0.00	4.97	0	5.73
car4	49436	16384	13	14.56	0.00	78.82	2	7.78
cari	1600	400	1	18.54	0.00	35.11	0	6.16
ch	8762	3700	3	71.69	9.87	10.41	2	1.76
complex	2431	1023	16	76.60	0.00	22.48	8	2.14
cq5	12578	5048	180	17.56	81.20	0.80	9	20.32
crew1	6604	135	1	83.41	0.00	7.47	1	1.69
cycle	4760	1903	7	4.46	92.07	1.86	0	61.66
czprob	4452	929	0	67.66	0.00	17.52	0	1.76
d6cube	6599	415	1	59.63	0.00	25.53	0	2.12
danoint	1185	664	0	79.50	0.00	12.74	0	1.44
dbic1	226435	43200	447	98.64	0.00	0.79	7836	0.06
dbir1	46159	18804	16	84.36	0.00	6.36	16	1.01

Continued on Next Page...

Table A.3 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
dbir2	46261	18906	10	75.86	0.00	9.40	6	1.61
dbir2.pre	32091	7228	5	52.38	0.00	18.70	2	2.43
de063155	2340	852	1	8.85	9.13	53.75	0	2.44
de080285	2424	936	1	10.37	18.69	48.01	0	11.00
degen3	3321	1503	2	89.08	0.00	7.99	1	2.65
delf000	8592	3128	3	12.47	33.30	35.18	0	10.15
delf001	8560	3098	3	10.13	33.65	37.60	0	11.88
delf002	8595	3135	3	12.77	32.23	36.15	0	10.67
delf003	8525	3065	4	12.39	17.79	46.39	0	10.63
delf004	8606	3142	9	8.26	34.65	42.93	1	16.08
delf005	8567	3103	6	12.71	18.77	49.79	1	8.41
delf006	8616	3147	9	7.56	14.89	64.83	1	15.52
delf007	8608	3137	11	8.00	28.75	50.21	1	17.38
delf008	8620	3148	12	7.00	24.12	58.11	1	16.84
delf009	8607	3135	16	4.48	20.70	66.45	1	19.57
delf010	8619	3147	11	7.43	31.25	50.55	1	14.82
delf011	8605	3134	7	12.74	34.43	37.22	1	9.54
delf012	8622	3151	10	8.48	29.11	50.42	1	17.27
delf013	8588	3116	13	6.64	31.09	51.84	1	17.60
delf014	8642	3170	6	14.16	29.85	38.81	1	10.21
delf015	8632	3161	7	10.42	26.34	50.00	1	10.25
delf017	8647	3176	6	10.98	28.79	44.60	1	12.17
delf018	8667	3196	6	11.34	36.76	39.08	1	11.39
delf019	8656	3185	3	15.88	23.82	35.30	0	6.55
delf020	8685	3213	6	15.66	35.94	31.25	1	7.82
delf021	8679	3208	6	12.17	42.85	29.04	1	9.53
delf022	8686	3214	6	11.02	41.21	31.36	1	11.08
delf023	8686	3214	7	11.25	38.27	34.02	1	9.53
delf024	8673	3207	12	8.76	29.72	47.08	1	13.85
delf025	8661	3197	9	13.73	35.87	36.24	1	10.73
delf026	8652	3190	9	12.36	37.63	35.52	1	11.62
delf027	8644	3187	7	10.82	38.84	34.39	1	8.89
delf028	8629	3177	8	14.36	34.54	35.68	1	10.12
delf029	8633	3179	7	11.16	36.36	36.64	1	10.47
delf030	8668	3199	8	11.03	29.65	43.40	1	10.93
delf031	8631	3176	8	10.39	39.64	34.86	1	9.71
delf032	8663	3196	9	9.26	39.97	36.98	1	13.31
delf033	8629	3173	8	11.49	39.84	34.10	1	14.27
delf034	8630	3175	9	7.38	39.31	39.47	1	13.03
delf035	8661	3193	9	14.93	37.52	33.63	1	14.52
delf036	8629	3170	8	10.46	41.78	33.03	1	12.71

Continued on Next Page…

Table A.3 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
deter0	7391	1923	0	58.01	0.00	18.73	0	2.19
deter1	21264	5527	1	71.03	0.00	11.57	1	1.74
deter2	23408	6095	2	50.33	7.66	18.46	1	2.37
deter3	29424	7647	2	74.48	0.00	10.74	1	1.75
deter4	12368	3235	1	34.77	9.25	26.14	0	3.51
deter5	19632	5103	1	68.13	0.00	13.26	1	1.73
deter6	16368	4255	1	45.41	8.65	20.73	0	2.73
deter7	24528	6375	2	52.11	8.01	17.91	1	2.37
deter8	14736	3831	1	67.13	0.00	13.59	0	1.87
df2177	10358	630	3	57.18	0.00	37.15	1	2.51
dfl001	18301	6071	231	71.79	25.65	2.37	157	1.47
dfl001.pre	12953	3881	124	73.91	22.60	3.23	86	1.45
disctom	10399	399	8	97.13	0.00	1.42	10	0.74
ds	68388	656	164	95.94	0.00	2.13	66	2.50
ex3sta1	25599	17443	2654	2.14	77.71	19.77	127	20.87
fast0507	63516	507	39	96.18	0.00	1.50	41	0.96
fit2d	10525	25	1	32.55	0.00	25.97	1	1.21
fit2p	16525	3000	14	96.64	0.00	1.52	11	1.24
fome20	139602	33874	140	98.02	0.00	1.15	106	1.32
fome21	279204	67748	334	98.59	0.00	0.75	276	1.21
fxm2-16	9502	3900	2	81.50	0.00	8.39	1	2.07
fxm2-6	3692	1520	0	62.76	0.00	17.30	0	1.62
fxm3_6	15692	6200	3	80.50	0.00	8.52	2	1.43
fxm4_6	53132	22400	16	87.20	0.00	5.73	3540	0.00
ge	21197	10099	17	46.05	29.97	18.58	6	2.65
greenbea	7797	2392	29	96.53	0.00	2.07	8	3.58
greenbeb	7797	2392	9	63.86	0.00	28.76	15	0.62
grow15	945	300	2	22.90	0.00	63.66	0	6.84
grow22	1386	440	5	11.18	0.00	79.24	1	8.48
grow7	441	140	1	9.68	0.00	77.75	0	12.96
jendrec1	6337	2109	12	73.33	0.00	12.97	5	2.40
ken-11	36043	14694	3	74.03	0.00	10.47	2	1.57
ken-18	259826	105127	266	97.55	0.00	1.22	242	1.10
ken-18.pre	129203	39856	84	96.11	0.00	1.85	71	1.18
kent	47920	31300	2	39.47	0.00	24.44	1	2.76
kl02	36770	71	4	80.11	0.00	7.15	4	1.00
large000	11072	4239	8	7.70	44.01	34.27	0	15.77
large001	10996	4162	14	32.48	27.99	28.08	4	3.61
large002	11084	4249	19	10.60	39.55	40.40	2	11.21
large003	11035	4200	10	16.78	31.23	35.48	1	7.64
large004	11086	4250	24	7.62	34.64	48.69	1	18.15

Continued on Next Page…

Table A.3 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
large005	11074	4237	8	15.44	32.22	35.94	1	8.66
large006	11086	4249	16	7.01	36.93	45.02	1	17.85
large007	11072	4236	17	9.37	38.22	42.13	1	12.55
large008	11085	4248	18	8.30	38.28	43.40	1	14.45
large009	11074	4237	24	8.22	29.92	54.21	1	21.26
large010	11084	4247	16	7.73	37.43	43.84	1	16.28
large011	11073	4236	16	10.65	35.02	44.13	1	11.02
large012	11091	4253	16	9.01	38.32	41.73	1	12.49
large013	11086	4248	13	10.40	40.15	37.55	1	11.68
large014	11109	4271	12	10.30	38.60	38.59	1	12.45
large015	11103	4265	14	8.84	37.62	42.26	1	14.17
large016	11125	4287	14	8.43	40.57	39.29	1	16.74
large017	11114	4277	8	12.73	31.31	40.14	1	10.70
large018	11134	4297	7	13.81	33.44	37.61	1	10.28
large019	11136	4300	6	16.78	34.45	29.94	1	7.25
large020	11152	4315	10	17.75	42.31	26.30	1	10.65
large021	11149	4311	10	14.25	46.22	25.58	1	7.06
large022	11146	4312	9	12.64	46.99	25.98	1	11.63
large023	11137	4302	12	17.08	40.48	29.33	1	10.63
large024	11123	4292	18	14.94	40.15	34.41	2	8.12
large025	11129	4297	24	11.88	31.31	47.66	2	9.98
large026	11108	4284	19	15.55	33.44	40.23	3	6.53
large027	11096	4275	13	18.63	41.38	27.57	3	5.04
large028	11135	4302	17	20.59	39.85	28.71	2	7.15
large029	11133	4301	18	13.22	40.89	35.44	2	9.35
large030	11108	4285	15	14.81	44.87	28.86	2	8.27
large031	11120	4294	20	11.99	32.89	46.64	2	8.87
large032	11119	4292	32	7.43	23.27	63.34	2	16.52
large033	11090	4273	15	18.39	45.00	25.69	3	5.84
large034	11125	4294	29	8.73	25.76	58.99	2	14.50
large035	11122	4293	31	7.16	31.41	52.62	2	17.76
large036	11104	4282	27	7.58	25.72	60.38	2	14.47
lpl3	44366	10828	4	91.75	0.00	0.00	4	1.20
maros	2289	846	0	57.24	0.00	26.60	0	2.20
mitre	12778	2054	1	71.75	0.00	11.43	0	1.73
mkc	8736	3411	20	99.28	0.00	0.34	14	1.47
mod011	15438	4480	1	72.35	0.00	11.60	1	0.71
model11	25344	7056	10	90.31	0.00	5.37	9	1.18
model2	1591	379	0	52.69	0.00	29.41	0	1.00
model6	7097	2096	65	21.88	8.51	66.83	9	6.95
momentum1	47854	42680	24	86.93	0.00	7.65	16	1.52

Continued on Next Page…

Table A.3 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
momentum2	27969	24237	1673	9.74	89.57	0.38	75	22.41
mzzv42z	22177	10460	120	99.50	0.00	0.23	151	0.80
nemsemm1	75358	3945	9	54.05	0.00	18.79	5	2.05
nemsemm2	49076	6943	5	69.99	0.00	12.67	3	1.56
neos	515905	479119	3720	99.63	0.00	0.18	11674	0.32
neos1	133473	131581	603	99.30	0.00	0.36	5571	0.11
neos2	134128	132568	1598	99.75	0.00	0.14	1176	1.36
neos3	518833	512209	62840	99.97	0.00	0.02	60070	1.05
nesm	3585	662	1	65.85	6.55	15.64	1	2.07
net12	28136	14021	10	94.72	0.00	2.34	10	1.08
nsct1	37882	22901	9	79.44	0.00	8.46	6	1.55
nsct2	37984	23003	6	73.85	0.00	10.55	10	0.64
nsct2.pre	19094	7797	3	55.99	0.00	17.15	2	2.00
nsir1	10124	4407	2	70.14	0.00	12.44	0	3.32
nsir2	10170	4453	2	63.92	0.00	14.40	1	2.70
nsrand-ipx	7356	735	1	29.45	0.00	27.45	0	6.60
nug07	1533	602	1	81.64	0.00	15.30	0	1.55
nug08	2544	912	5	95.36	0.00	3.99	3	1.88
nug12	12048	3192	10118	99.34	0.00	0.66	1046	9.67
nug15	28605	6330	52036	98.78	0.00	1.22	48229	1.08
nw04	87518	36	5	58.81	0.00	13.26	2	2.09
nw14	123482	73	7	60.49	0.00	13.36	4	1.71
orna1	1764	882	4	7.13	0.00	45.02	0	13.51
orna2	1764	882	5	8.43	0.00	44.22	0	13.84
orna3	1764	882	5	6.34	0.00	47.64	0	14.44
orna4	1764	882	8	4.05	0.00	70.19	0	30.48
orna7	1764	882	5	10.21	0.00	44.24	0	11.90
osa-07	25067	1118	3	78.82	0.00	8.03	2	1.47
osa-60	243246	10280	408	98.66	0.00	0.46	360	1.14
osa-60.pre	234334	10209	361	98.95	0.00	0.36	318	1.14
p010	29090	10090	12	87.71	0.00	7.76	10	1.19
p05	14590	5090	3	80.35	0.00	10.99	2	1.43
pcb1000	3993	1565	1	54.99	0.00	32.48	1	1.39
pcb3000	10770	3960	4	59.45	0.00	33.17	2	2.22
pds-06	38536	9881	3	82.26	0.00	7.56	2	1.58
pds-100	661603	156243	3730	99.62	0.00	0.24	3821	0.98
pds-20.pre	89523	10240	20	92.37	0.00	3.42	80	0.24
pds-30	204942	49944	460	99.16	0.00	0.47	396	1.16
pds-40	279703	66844	1456	99.59	0.00	0.25	984	1.48
pds-50	353155	83060	1955	99.64	0.00	0.21	1727	1.13
pds-60	429074	99431	2423	99.63	0.00	0.22	2054	1.18

Continued on Next Page...

Table A.3 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
pds-70	497255	114944	19885	99.94	0.00	0.04	2318	8.58
pds-80	555459	129181	3773	99.68	0.00	0.20	3104	1.22
pds-90	609494	142823	4328	99.70	0.00	0.18	3373	1.28
pf2177	10628	9728	17	94.93	0.00	4.19	18	0.97
pgp2	13254	4034	5	22.32	66.84	4.67	0	19.53
pilot.ja	2928	940	18	5.20	0.00	85.70	1	23.09
pilot.we	3511	722	4	23.37	0.00	44.22	1	5.56
pilot4	1410	410	3	7.04	7.41	70.00	0	20.36
pilotnov	3147	975	7	7.27	0.00	88.66	0	15.20
pldd000b	6336	3069	2	32.03	0.00	42.48	0	4.07
pldd001b	6336	3069	1	31.52	0.00	42.41	0	3.76
pldd002b	6336	3069	1	31.74	0.00	41.94	0	3.73
pldd003b	6336	3069	1	32.15	0.00	41.50	0	3.86
pldd004b	6336	3069	1	32.09	0.00	39.67	0	3.80
pldd005b	6336	3069	1	32.75	0.00	39.82	0	3.80
pldd006b	6336	3069	2	17.78	18.22	45.42	0	6.87
pldd007b	6336	3069	3	17.69	18.25	45.81	0	6.98
pldd008b	6336	3069	3	19.76	17.81	44.59	0	5.94
pldd009b	6336	3069	1	34.74	0.00	38.34	0	3.34
pldd010b	6336	3069	3	18.46	17.76	45.35	0	6.21
pldd011b	6336	3069	3	20.80	17.54	44.04	0	5.32
pldd012b	6336	3069	3	19.16	17.70	45.55	0	6.82
pltexpa3_16	102522	28350	301	14.50	84.50	0.43	12	26.14
pltexpa3_6	16042	4430	11	13.52	81.71	2.14	0	29.19
pltexpa4_6	97258	26894	453	16.89	82.47	0.28	13	33.72
primagaz	12390	1554	1	84.05	0.00	6.77	1	1.42
progas	3075	1650	39	1.16	0.00	74.99	0	118.89
protfold	3947	2112	7	93.01	0.00	6.02	3	2.36
qap12	12048	3192	989	95.06	0.00	4.92	4860	0.20
qiu	2032	1192	1	29.10	57.41	8.50	0	12.54
r05	14690	5190	4	79.40	0.00	10.82	3	1.40
rail2586	923269	2586	5449	99.14	0.00	0.40	4901	1.11
rail4284	1096894	4284	15457	99.65	0.00	0.15	14748	1.05
rail507	63516	507	24	93.66	0.00	2.47	23	1.04
rail516	47827	516	13	91.96	0.00	3.00	12	1.06
rail582	56097	582	21	93.27	0.00	2.53	21	0.99
rat1	12544	3136	35	66.66	0.00	21.49	77	0.46
rd-rplusc-21	126521	125899	83	78.94	0.00	16.82	47	1.76
rentacar	16360	6803	2	84.92	0.00	7.43	4	0.63
rlfddd	61521	4050	1	49.96	0.00	0.00	1	1.82
rlfdual	74970	8052	2	64.37	0.00	0.00	47	0.04

Continued on Next Page…

Table A.3 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
rlfprim	66918	58866	29	97.29	0.00	0.00	24	1.19
roll3000	3461	2295	1	80.98	0.00	9.44	0	1.70
rosen10	6152	2056	4	88.12	0.00	6.85	1	4.27
rosen2	3080	1032	2	78.18	0.00	11.76	0	4.02
route	44817	20894	2	58.50	0.00	17.80	1	2.02
sc205-2r-200	8827	4413	1	87.86	0.00	5.94	1	1.27
sc205-2r-400	17627	8813	7	95.44	0.00	1.98	6	1.15
scagr7-2b-64	20003	9743	9	94.27	0.00	2.47	5	1.64
scagr7-2c-64	5027	2447	0	72.06	0.00	13.30	0	1.15
scagr7-2r-108	8459	4119	1	81.95	0.00	8.06	1	1.23
scagr7-2r-216	16883	8223	6	92.79	0.00	2.99	5	1.20
scagr7-2r-432	33731	16431	24	96.45	0.00	1.48	21	1.16
scagr7-2r-54	4247	2067	0	71.19	0.00	13.02	0	1.24
scagr7-2r-64	5027	2447	1	75.25	0.00	11.77	0	1.28
scagr7-2r-864	67427	32847	139	94.86	2.77	1.11	154	0.90
scfxm1-2b-16	6174	2460	1	71.09	0.00	14.98	0	1.90
scfxm1-2b-64	47950	19036	32	95.12	0.00	2.44	31	1.02
scfxm1-2r-16	6174	2460	1	71.96	0.00	14.34	0	1.95
scfxm1-2r-27	10277	4088	2	81.48	0.00	9.07	1	1.61
scfxm1-2r-32	12142	4828	3	85.74	0.00	7.11	2	1.89
scfxm1-2r-64	24078	9564	9	90.97	0.00	4.38	6	1.41
scfxm1-2r-96	36014	14300	67	91.70	4.92	1.81	17	3.86
scrs8-2r-256	16961	7196	1	66.85	0.00	15.04	1	1.72
scrs8-2r-512	33857	14364	5	86.94	0.00	5.92	3	1.42
scsd8-2b-64	41040	5130	1	33.33	0.00	25.55	0	3.15
scsd8-2c-64	41040	5130	1	32.42	0.00	25.82	0	3.11
scsd8-2r-108	17360	2170	1	9.52	33.74	24.60	0	10.50
scsd8-2r-216	34640	4330	2	17.16	19.66	27.05	0	6.55
scsd8-2r-432	69200	8650	5	20.39	33.79	18.64	1	6.52
scsd8-2r-54	8720	1090	0	9.74	13.06	36.57	0	13.58
scsd8-2r-64	10320	1290	0	10.76	0.00	31.47	0	8.10
scsd8	3147	397	0	45.53	19.75	19.54	0	2.81
sctap1-2b-64	40014	15390	2	78.35	0.00	0.00	1	1.46
sctap1-2r-108	16926	6510	0	46.51	0.00	0.00	0	2.13
sctap1-2r-216	33774	12990	1	64.25	0.00	0.00	1	1.72
sctap1-2r-480	74958	28830	4	79.98	0.00	0.00	3	1.24
self	8324	960	51601	0.09	63.85	35.65	190	272.00
seymour	6316	4944	7	96.56	0.00	1.60	6	1.24
seymourl	6316	4944	8	96.46	0.00	1.73	6	1.24
sgpf5y6.pre	58519	19499	88	3.68	93.79	1.12	2	49.22
slptsk	6208	2861	36	28.96	0.00	59.00	3	10.59

Continued on Next Page…

Table A.3 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
small002	1853	713	1	5.43	26.97	46.93	0	18.42
small006	1848	710	1	7.10	31.48	38.38	0	16.81
small007	1848	711	1	6.48	35.03	36.77	0	18.42
small008	1846	712	1	4.63	35.12	37.43	0	26.71
small009	1845	710	0	4.48	30.96	38.89	0	23.38
small010	1849	711	0	4.62	23.84	43.55	0	19.57
small015	1813	683	0	2.68	27.25	41.84	0	37.36
south31	53846	18425	148	97.48	0.00	1.02	125	1.18
sp97ar	15862	1761	3	72.95	0.00	11.04	3	1.30
stair	823	356	7	1.59	0.00	93.87	0	119.36
stocfor2	4188	2157	1	73.80	0.00	11.76	0	2.54
stocfor3	32370	16675	47	97.55	0.00	1.01	21	2.20
stormG2_1000	1787306	528185	6251	99.51	0.00	0.21	5767	1.08
stormG2_1000.pre	1410155	377036	4336	99.43	0.00	0.25	3642	1.19
stormG2-125	223681	66185	86	95.66	0.00	1.90	76	1.13
stormG2-125.pre	176405	47161	56	94.70	0.00	2.29	50	1.12
stormg2-27	48555	14441	4	80.75	0.00	8.37	3	1.44
stormg2-8	14602	4409	1	56.68	0.00	18.18	0	2.00
sws	26775	14310	1	35.29	0.00	25.57	0	3.40
t0331-4l	47579	664	122	85.01	0.00	12.70	113	1.08
t1717	74436	551	116	95.96	0.00	2.65	108	1.08
testbig	48836	17613	27	97.33	0.00	1.13	10	2.64
ulevimin	51195	6590	69	96.52	0.00	1.81	56	1.23
us04	28179	163	2	48.73	0.00	18.81	1	3.22
watson_1.pre	239575	65266	5180	13.08	86.41	0.26	250	20.70
wood1p	2838	244	2	28.40	0.00	43.19	0	5.30
woodw	9503	1098	3	89.74	0.00	5.57	2	1.32

Table A.4: Comparison for dual simplex on instances without hot-start. Here we show the total running time for the exact LP solver and the original QSopt code, we also show the percentage of time spend on solving the double LP approximation, the extended float approximation (if any), and the checking process in rational arithmetic for the exact LP solver code, the last column shows the ratio of the running time of the exact code versus the running time of the original QSopt code. All running times are in seconds. The runs where made using a Linux workstation with 4Gb of RAM, and with an AMD Opteron 250 CPU.

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
atlanta-ip	70470	21732	4665	57.16	42.70	0.07	280	16.65
co5	13767	5774	260	11.04	87.88	0.64	13	19.54
co9	25640	10789	1418	2.01	97.21	0.50	70	20.13
cq9	23056	9278	747	19.17	78.90	1.45	47	15.98
d2q06c	7338	2171	277	3.73	73.23	21.67	10	28.04
dano3mip	17075	3202	2583	7.40	92.17	0.38	213	12.12
dano3mip.pre	16988	3151	3065	7.39	92.23	0.34	229	13.40
de063157	2424	936	9	1.65	90.91	5.09	1	12.62
fome11	36602	12142	6141	21.42	78.49	0.07	330	18.61
fome12	73204	24284	11381	13.19	86.71	0.08	915	12.44
fome13	146408	48568	28294	9.67	90.26	0.06	2373	11.92
fxm3_16	105502	41340	1693	5.76	93.79	0.09	122	13.88
gen	3329	769	17867	1.38	80.70	17.82	13	1412.32
gen1	3329	769	17602	1.42	80.40	18.08	13	1384.79
gen2	4385	1121	45421	0.37	3.38	96.16	59	775.09
gen4	5834	1537	53448	1.18	6.98	91.67	15	3648.04
gen4.pre	5648	1475	53630	0.11	3.16	96.57	3	16445.97
iprob	6002	3001	9	97.76	0.00	0.82	4	2.24
l30	18081	2701	10427	0.67	98.79	0.50	44	239.70
lp22	16392	2958	1932	13.59	85.46	0.90	89	21.66
lp22.pre	11565	2872	1963	21.11	78.00	0.85	86	22.84
maros-r7	12544	3136	2284	0.55	93.33	3.28	31	74.66
mod2	66502	34774	6827	5.10	94.43	0.32	671	10.17
mod2.pre	54286	27186	5276	16.94	82.55	0.35	413	12.76
model10	19847	4400	2361	1.70	90.94	6.87	0	2361070.00
model3	5449	1609	91	14.81	82.58	1.95	4	23.66
model4	5886	1337	118	7.12	90.28	1.63	14	8.16
model5	13248	1888	148	18.02	79.94	1.07	6	24.31
model7	11365	3358	388	8.03	84.14	7.34	18	21.08
model9	13136	2879	103	15.01	81.68	1.91	12	8.64
momentum3	70354	56822	34138	4.72	50.75	43.79	1703	20.04
msc98-ip	36993	15850	2140	23.28	76.65	0.02	187	11.45
mzzv11	19739	9499	5090	3.83	96.15	0.01	314	16.22

Continued on Next Page...

Table A.4 (continued)

Problem name	Number of columns	Number of rows	QSopt_ex time	% Double precision	% Extended precision	% Exact check	QSopt time	Time ratio
nemspmm1	10994	2372	413	12.87	85.85	1.01	30	13.64
nemspmm2	10714	2301	571	2.97	92.97	3.57	39	14.64
nemswrld	34312	7138	10009	2.77	93.84	3.24	402	24.92
neos.pre	476610	440494	56336	13.01	86.97	0.00	5623	10.02
nl	16757	7039	77	97.76	0.00	1.40	13	5.92
perold	2001	625	31	9.63	55.10	33.14	0	90.47
pilot	5093	1441	623	1.70	41.05	55.31	7	83.83
pilot87	6913	2030	7724	0.38	9.01	89.63	59	129.92
pilot87.pre	6375	1885	7463	0.45	9.29	89.24	57	131.38
rat5	12544	3136	8787	0.24	81.05	18.22	46	192.53
rat7a	12544	3136	70876	0.19	14.75	84.89	281	252.49
sc205-2r-1600	70427	35213	7621	0.15	98.70	0.00	45504	0.17
sc205-2r-800	35227	17613	2518	0.09	97.71	0.00	11643	0.22
scfxm1-2r-128	47950	19036	446	23.43	76.04	0.17	38	11.77
scfxm1-2r-256	95694	37980	2174	11.40	88.39	0.07	196	11.11
sgpf5y6	554711	246077	31050	14.93	85.01	0.01	788	39.43
stat96v1	203467	5995	22920	1.12	97.33	1.00	277	82.85
stat96v4	65385	3173	39202	0.42	69.77	19.65	452	86.71
stat96v5	78086	2307	7503	1.22	40.98	34.60	28	266.17
stp3d	364368	159488	111680	56.49	43.50	0.00	6340	17.61
watson_1	585082	201155	29168	21.64	78.25	0.03	2758	10.58
watson_2	1023874	352013	196670	4.70	95.27	0.01	8424	23.35
world	67240	34506	11138	3.16	96.47	0.26	1102	10.11
world.pre	55916	27057	6759	7.40	92.13	0.34	610	11.07

REFERENCES

[1] Achterberg, T., Koch, T., and Martin, A., "Branching rules revisited," *Operations Research Letters*, vol. 33, pp. 42–54, 2005.

[2] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network flows: Theory, algorithms, and applications.* Prentice Hall, 1993.

[3] Amaral, A. and Letchford, A. N., "An improved upper bound for the two-dimensional non-guillotine cutting problem." Working paper, 2003.

[4] Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W., "Finding cuts in the tsp," Tech. Rep. 95-05, DIMACS, 1995.

[5] Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W., "On the solution of traveling salesman problems," *Documenta Mathematica*, vol. Extra Volume Proceedings ICM III (1998), pp. 645–656, 1998.

[6] Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W., "Tsp cuts which do not conform to the template paradigm," in *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School]*, (London, UK), pp. 261–304, Springer-Verlag GmbH, 2001.

[7] Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W., "Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems," *Mathematical Programming*, vol. 97, pp. 91–153, 2003.

[8] Atamtürk, A., "On the facets of the mixed-integer knapsack polyhedron," *Mathematical Programming*, vol. 98, pp. 145–175, 2003.

[9] Balas, E., "Disjunctive programming," *Annals of Discrete Mathematics*, vol. 5, pp. 3–51, 1979.

[10] Balas, E., "Disjunctive programming: Properties of the convex hull of feasible points," *Discrete Applied Mathematics*, vol. 89, pp. 3–44, 1998.

[11] Balas, E., Ceria, S., and Cornuéjols, G., "A lift-and-project cutting plane algorithm for mixed 0-1 programs," *Mathematical Programming*, vol. 58, pp. 295–324, 1993.

[12] Balas, E. and Perregaard, M., "A prcise correspondence between lift-and-project cuts, simple disjuntive cuts, and mixed integer gomory cuts for 0-1 programming," *Mathematical Programming*, vol. 94, pp. 221–245, 2003.

[13] Bárány, I. and Pór, A., "On 0-1 polytopes with many facets," *Advances in Mathematics*, vol. 161, pp. 209–228, 2001.

[14] Benichou, M., Gauthier, J. M., Girodet, P., Hentges, G., Ribiere, G., and Vincent, O., "Experiments in mixed-integer linear programming," *Mathematical Programming*, vol. 1, pp. 76–94, 1971.

[15] Bixby, R. E., "Solving real-world linear programs: A decade and more of progress," *Operations Research*, vol. 50, pp. 3–15, 2002.

[16] Bixby, R. E., Fenelon, M., Gu, Z., Rothberg, E., and Wunderling, R., "Mip: Theory and practice - closing the gap," in *Proceedings of the 19th IFIP TC7 Conference on System Modelling and Optimization*, (Deventer, The Netherlands, The Netherlands), pp. 19–50, Kluwer, B.V., 2000.

[17] Boyd, A. E., "Fenchel cutting planes for integer programs," *Operations Research*, vol. 42, pp. 53–64, 1992.

[18] Boyd, S. C., Cockburn, S., and Vella, D., "On the domino-parity inequalities for the stsp," Tech. Rep. TR-2001-10, University of Ottawa, Ottawa, Canada, 2001.

[19] Boyd, S. C. and Cunningham, W. H., "Small travelling salesman polytopes," *Mathematics of Operations Research*, vol. 16, no. 2, pp. 259–271, 1991.

[20] Boyd, S. C., Cunningham, W. H., Queyranne, M., and Wang, Y., "Ladders for travelling salesmen," *SIAM Journal on Optimization*, vol. 5, pp. 408–420, 1995.

[21] Boyd, S. C. and Labonté, G., "Finding the exact integrality gap for small traveling salesman problems," in Cook and Schulz [28], pp. 83–92.

[22] Boyer, J. M. and Myrvold, W. J., "On the cutting edge: Simplified $\mathcal{O}(n)$ planarity by edge addition," *Journal of Graph Algorithms and Applications*, vol. 8, pp. 241–273, 2004.

[23] Buchta, C., Müller, J., and Tichy, R. F., "Stochastical approximation of convex bodies," *Mathematische Annalen*, vol. 271, pp. 225–235, 1985.

[24] Carr, R., "Separating clique trees and bipartition inequalities having a fixed number of handles and teeth in polynomial time," *Mathematics of Operations Research*, vol. 22, no. 2, pp. 257–265, 1997.

[25] Christof, T. and Reinelt, G., "Decomposition and parallelization techniques for enumerating the facets of combinatorial polytopes.," *Int. J. Comput. Geometry Appl.*, vol. 11, no. 4, pp. 423–437, 2001.

[26] Chvátal, V., "Edmonds polytopes and weakly hamiltonian graphs," *Mathematical Programming*, vol. 5, pp. 29–40, 1973.

[27] Cook, W., Kannan, R., and Schrijver, A., "Chvátal colsured for mixed integer programming problems," *Mahtematical Programming*, vol. 47, pp. 155–174, 1990.

[28] Cook, W. and Schulz, A. S., eds., *Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings*, vol. 2337 of *Lecture Notes in Computer Science*, Springer, 2002.

[29] Cornuéjols, G., Fonlupt, J., and Naddef, D., "The traveling salesman problem on a graph and some related integer polyhedra," *Mathematical Programming*, vol. 33, pp. 1–27, 1985.

[30] Dakin, R. J., "A tree-search algorithm for mixed integer programming problems," *The Computer Journal*, vol. 8, no. 3, pp. 250–255, 1965.

[31] Dantzig, G. B., "Programming in a linear structure." Comptroller, USAF Washington D.C., 1948.

[32] Dantzig, G. B., "The story about how it began: Some legends, a little about its historical significance, and comments about where its many mathematical programming extensions may be headed," in Lenstra *et al.* [69], pp. 19–31.

[33] Dantzig, G. B., Fulkerson, D. R., and Johnson, S., "Solution of a large-scale traveling salesman problem," *Operations Research*, vol. 2, pp. 393–410, 1954.

[34] de la Vallée, P., "Sur la méthode de l'approximation minimum," *Société Scientifique de Bruxelles, Annales, Seconde Partie, Mémories*, vol. 35, pp. 1–16, 1911.

[35] Dhiflaoui, M., Funke, S., Kwappik, C., Mehlhorn, K., Seel, M., Schömer, E., Schulte, R., and Weber, D., "Certifying and repairing solutions to large lps how good are lp-solvers?," in *SODA 2003, Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 255–256, 2003.

[36] Edmonds, J., "Maximum matching and a polyhedron with 0-1 vertices," *Journal of Research of the National Bureau of Standards*, vol. 65B, pp. 125–130, 1965.

[37] Fleiner, T., Kaibel, V., and Rote, G., "Upper bounds on the maximal number of facets of 0/1-polytopes," *European Journal of Combinatorics*, vol. 21, pp. 121–130, 2000.

[38] Fleischer, L. and Tardos, E., "Separating maximally violated comb inequalities in planar graphs," *Mathematics of Operations Research*, vol. 24, pp. 130–148, 1999.

[39] Fleischmann, B., "A new class of cutting plpanes for the symmetric traveling salesman problem," *Mathematical Programming*, vol. 40, pp. 225–246, 1988.

[40] Forrest, J. J. and Goldfarb, D., "Steepest-edge simplpex algorithms for linear programming," *Mathematical Programming*, vol. 57, pp. 341–374, 1992.

[41] Fourier, J. B. J., "Analyse des travaux de l'académie royale des sciences, pendant l'anné 1823, partie mathématique," *Historie de l'Académie Royale des Sciences de l'Institut de France*, vol. 6, pp. xxix–xli, 1826.

[42] Fredman, M. L., Johnson, D. S., McGeoch, L. A., and Ostheimer, G., "Data structures for the traveling salesmen," *Journal of Algorithms*, vol. 18, pp. 432–479, 1995.

[43] Garey, M. R. and Johnson, D. S., *Computers and Intractability, A guide to the Theory of NP-Completeness*, ch. A2, pp. 211–212. W. H. Freeman and Company, 1978.

[44] Gärtner, B., "Exact arithmetic at low cost - a case study in linear programming," *Computatioinal Geometry*, vol. 13, pp. 121–139, 1999.

[45] Gatzouras, D., Giannopoulos, A., and Markoulakis, N., "Lower bounds for the maximal number of facets of a 0/1 polytope," *Discrete & Computational Geometry*, vol. 34, pp. 331–349, 2005.

[46] Gomory, R. E., "Outline of an algorithm for integer solutions too linear programs," *Bulletin of the American Mathematical Society*, vol. 64, pp. 275–278, 1958.

[47] Gomory, R. E., "Solving linear pprogramming in integers," in *Combinatorial Analysis, Proceedinigs of the Symposia in Applied Mathematics*, American Mathematical Society, 1960.

[48] Gomory, R. E., *Recent Advances in Mathematical Programming*, ch. An Algorithm for Integer Solutions to Linear Programs, pp. 269–302. McGray-Hill, New York, 1963.

[49] Grötschel, M. and Padberg, M. W., "On the symmetric traveling salesman problem ii: Lifting theorems and facets," *Mathematical Programming*, vol. 16, pp. 281–302, 1979.

[50] Grötschel, M. and Pulleyblank, W. R., "Clique tree inequalities and the symmetric traveling salesman problem," *Mathematics of Operations Research*, vol. 11, pp. 537–569, 1986.

[51] Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. P., "Lifted cover inequalities for 0-1 integer programs: Computation," *INFORMS Journal on Computing*, vol. 10, pp. 427–437, 1998.

[52] Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. P., "Lifted cover inequalities for 0-1 integer programs," *Mahtematical Programming*, vol. 85, pp. 437–467, 1999.

[53] Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. P., "Lifted cover inequalities for 0-1 integer programs: Complpexity," *INFORMS Journal on Computing*, vol. 11, pp. 117–123, 1999.

[54] Harris, P. M. J., "Pivot selection methods of the devex lp code," *Mathematical Programming*, vol. 5, pp. 1–28, 1973.

[55] Held, M. and Karp, R. R., "The traveling salesman problem and minimum spanning trees," *Operations Research*, vol. 18, pp. 1138–1162, 1970.

[56] Helsgaun, K., "An effective implementation of the lin-kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, pp. 106–130, 2000.

[57] IEEE Computer Society, *IEEE Standard for Binary Floating-Point Arithmetic*, ieee std 754-1985 ed., 1985.

[58] ILOG Inc., *CPLEX 7.5 Reference Manual*, 2003.

[59] Jansson, C., "Rigorous lower and upper bounds in linear programming," *SIAM Journal on Optimization*, vol. 14, pp. 914–935, 2004.

[60] Jewell, W. S., "Optimal flow through networks," Tech. Rep. 8, Massachusetts Institute of Technology (MIT), 1958.

[61] Jr., H. W. L., "Integer programming with a fixed number of variables," *Mahtematics of Operations Research*, vol. 8, pp. 538–548, 1983.

[62] Jr., L. R. F. and Fulkerson, D. R., "A suggested computation for maximal multicocmodity network flows," *Managment Science*, vol. 5, pp. 97–101, 1958. Reprinted in Managment Science, Vol. 50, 2004, pages 1778–1780.

[63] Jünger, M., Reinelt, G., and Thienel, S., "Provably good solutions for the traveling salesman problem," *Mathematical Methods of Operations Research (ZOR)*, vol. 40, pp. 183–217, 1994.

[64] Karmakar, N., "A new polynomial-timie algorithm for linear programming," *Combinatorica*, vol. 4, pp. 373–395, 1984.

[65] Khachiyan, L. G., "Polynomial algorithms in linear programming," *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki (Journal of Computational Mathematics and Mathematical Physics)*, vol. 20, pp. 51–68, 1980.

[66] Koch, T., "The final netlib-lp results," *Operations Research Letters*, vol. 32, pp. 138–142, 2003.

[67] Krippendorff, K., "A dictionary of cybernetics." An 80 p. unpublished report dated Feb. 2, 1986, available on-line at `http://pespmc1.vub.ac.be/ASC/Combin_explo.html`, 1986.

[68] Land, A. H. and Doig, A. G., "An automatic method for solving discrete programming problems," *Econometrica*, vol. 28, pp. 497–520, 1960.

[69] Lenstra, J. K., Kan, A. H. G. R., and Schrijver, A., eds., *History of Mathematical Programming: A Collection of Personal Reminiscences.* Elsiever Science Publishers B.V., 1991.

[70] Letchford, A. N., "Separating a superclass of comb inequalities in planar graphs," *Mathematics of Operations Research*, vol. 25, pp. 443–454, 2000.

[71] Levi, R., Shmoys, D. B., and Swamy, C., "Lp-based approximation algorithms for capacited facility location," *Lecture Notes in Computer Science*, vol. 3064/2004, pp. 206–218, 2004.

[72] Linderoth, J. T. and Savelsbergh, M. W. P., "A computational study of search strategies for mixed integer programming," *INFORMS Journal on Computing*, vol. 11, pp. 173–187, 1999.

[73] Little, J. D. C., Murtky, K. G., Sweeney, D. W., and Karel, C., "An algorithm for the travelling salesman problem," *Operations Research*, vol. 11, p. 972, 1963.

[74] Lovász, L. and Schrijver, A., "Cones of matrices and setfunctions, and 0-1 optimization," Tech. Rep. BS-R8925, Centrum voor Wiskunde en Informatica (CWI), 1989.

[75] Lovász, L. and Schrijver, A., "Cones of matrices and setfunctions, and 0-1 optimization," *SIAM Journal on Optimization*, vol. 1, pp. 166–190, 1991.

[76] Martin, A., "Integer programs with block structure," Tech. Rep. SC 99-03, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), 1999.

[77] Mittlemann, H., "Benchmarks for optimization software." Available on-line at `http://plato.asu.edo/ftp/lpfree.html`, 2005.

[78] Naddef, D., "The binested inequalities for the symmetric traveling salesman polytope," *Mathematics of Operations Research*, vol. 17, pp. 882–900, 1992.

[79] Naddef, D. and Pochet, Y., "The symmetric traveling salesman polytope revisited," *Mathematics of Operations Research*, vol. 26, pp. 700–722, 2001.

[80] Naddef, D. and Rinaldi, G., "The symmetric traveling salesman polytope and its graphical relaxation: Composition of valid inequalities," *Mathematical Programming*, vol. 51, pp. 359–400, 1991.

[81] Naddef, D. and Rinaldi, G., "The crown inequalities for the symmetric traveling salesman polytope," *Mathematics of Operations Research*, vol. 17, pp. 308–326, 1992.

[82] Naddef, D. and Thienel, S., "Efficient separation routines for the symmetric traveling salesman problem i: General tools and comb separation," *Mathematical Programming*, vol. 92, pp. 237–255, 2002.

[83] Naddef, D. and Thienel, S., "Efficient separation routines for the symmetric traveling salesman problem ii: Separating multi handle inequalities," *Mathematical Programming*, vol. 92, pp. 257–283, 2002.

[84] Naddef, D. and Wild, E., "The domino inequalities: Facets for the symmetric traveling salesman polytope," *Mathematical Programming*, vol. 98, pp. 223–251, 2003.

[85] Nemhauser, G. L. and Wolsey, L. A., "A recursive procedure for generating all cuts for 0-1 mixed integer programs," *Mathematical Programming*, vol. 46, pp. 379–390, 1990.

[86] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization*. Discrete Mathematics and Optimization, Wiley-Interscience, 1999.

[87] Neumaier, A. and Shcherbina, O., "Safe bounds in linear and mixed-integer lineear programming," *Mathematical Programming*, vol. 99, pp. 283–296, 2004.

[88] Padberg, M. W. and Rao, M. R., "Odd minimum cut-sets and b-matchings," *Mathematics of Operations Research*, vol. 7, pp. 67–80, 1982.

[89] Padberg, M. W. and Rinaldi, G., "An efficient algorithm for the minimum capacity cut problem," *Mathematical Programming*, vol. 47, pp. 19–36, 1990.

[90] Padberg, M. W. and Rinaldi, G., "Facet identification for the symmetric traveling salesman polytope," *Mathematical Programming*, vol. 47, pp. 219–257, 1990.

[91] Padberg, M. W. and Rinaldi, G., "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM Review*, vol. 33, pp. 60–100, 1991.

[92] Ramakrishnam, K. G., Resende, M. G. C., Ramachandran, B., and Penky, J. F., "Tight qap bounds via linear programming," *Combinatorial and Global Optimization*, pp. 297–303, 2002.

[93] Reinelt, G., "Tsplib - a traveling salesman library," *ORSA Journal on Computing*, vol. 3, pp. 376–384, 1991.

[94] Schrijver, A., *Theory of Linear and Integer Programming.* Discrete Mathematics and Optimization, Wiley-Interscience, 1986.

[95] Sloane, N. J. A. and Stufken, J., "A linear programming bound for orthogonal arrays with mixed levels," *Journal of Statistical Planning and Inference*, vol. 56, pp. 295–306, 1996.

[96] Tamaki, H., "Alternating cycle contribution: a tour-merging strategy for the travelling salesman problem," Tech. Rep. MPI-I-2003-1-007, Max-Planck Institute, Saarbrücken, Germany, 2003.

[97] Wiedemann, D. H., "Solving sparse linear equations over finite fields," *IEEE Transactions on Information Theory*, vol. 32, pp. 54–62, 1986.

VITA

Daniel G. Espinoza was born in Santiago, Chile, on 7 September 1976. After graduating from Instituto Nacional, in Santiago, 1993, he attended Universidad de Chile in Santiago, from wich he received a Bachelor in Engineering Sciences, with a major in Mathematics in June 2000. After receiving his Mathematical Engineering Title from Universidad de Chile in August 2001, he enrolled in the School of Industrial and Systems Engineering at Georgia Institute of technology in August of 2001, where he completed his doctoral research on methods to find cuts for general Mixed Integer Problems.