Presolve, crash and software engineering for HiGHS

Ivet Galabova

Doctor of Philosophy

University of Edinburgh

2022

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Ivet Galabova*)

I would first like to express my sincere gratitude to my supervisor, Julian Hall, for his endless patience with me over the past five years. Many thanks to my second supervisor, Jacek Gondzio, for encouraging me to write and for his help with presolve.

To Vera Baturina and Frederic Didier, for introducing me to some better software engineering.

To Thorsten Koch, for seeing the potential of presolve, and for the opportunity to do it on a larger scale. Special thanks go to Ambros Gleixner for the kind support, and to Matthias Miltenberger for his help with the initial setup of HiGHS.

I would also like to thank Nick Gould for his help with Augmented Lagrangians. I would like to express my gratitude to Qi Huangfu for writing the core dual simplex solver, and some valuable advice over the years, and to Lukas Schork for his contributions and style.

My sincere thanks also go to Lucas Dixon for a more creative approach to problem solving, and to Spyros Pougkakiotis for his appreciation of deeper theory and desire to share. My gratitude extends to Petya Todorova and Iain Gordon for some strong fundamentals.

Abstract

The efficient computational solution of linear optimization problems is generally enhanced significantly by using a "presolve" procedure to process the problem logically in order to reduce the dimension of the problem to be solved algorithmically. In the absence of other information about the optimal solution to the problem, it is often worth performing a cheap "crash" procedure to obtain a solution that is near-feasible and, ideally, near-optimal. When a problem has been presolved, it is essential to be able to deduce the original problem's optimal solution from the optimal solution of the presolved problem. This thesis provides an analysis of the Idiot crash algorithm (ICA) for linear programming (LP) problems, and techniques for primal and dual postsolve corresponding to established presolve techniques. It demonstrates that presolve yields significant performance enhancement for standard test problems, and identifies that use of the ICA enhances the solution process for a significant range of test problems. This is particularly so in the case of linearisations of quadratic assignment problems that are, otherwise, very challenging for standard methods of solution. The techniques are implemented in the HiGHS open-source software system. The use of modern techniques to create robust and efficient software system for HiGHS and its interfaces has been a critical feature of its success. Accordingly, this thesis sets out the philosophy and techniques of the software engineering underpinning HiGHS.

Contents

Chapter 1

Introduction

Linear programming problems (LP) are formulated in a wide range of practical applications. LPs can be modelled explicitly in commercially valuable applications, such as resource allocation, logistics, scheduling and human and animal food formulation. LPs are also generated as sub- problems during the solution of mixed-integer programming (MIP) problems, and when solving nonlinear programming problems via sequential linear programming. Hence there is a very strong motivation for improving the efficiency of solution techniques.

This thesis examines crucial aspects of the solution process for large-scale sparse linear programming problems. An LP is considered sparse if zero entries occur in the constraint matrix to such an extent that avoiding addition or multiplication by zero when solving the LP problem improves the solution time meaningfully

Direct solution of LPs is not possible in all but trivial instances or very specific problem classes. For large problems, instead, iterative techniques are used. There are two main classes of solution techniques, most commonly achieving the best performance: The simplex method and interior point methods (IPM). Depending on the structure and coefficients of the problem, a method in one class may perform better than algorithms from the other class.

This thesis examines preprocessing techniques for LP and outlines how preprocessing is integrated within the structure of a larger optimization software system, in particular `HiGHS`. Before the presolve and crash algorithms can be described, some mathematical background is introduced in Chapter 2.

In practice, most LP formulations contain redundancies. Regardless of the solution technique most appropriate for a particular LP, for most problems solved in practice, there exists an equivalent LP of smaller dimensions. This results from redundancies present in the model, often introduced as a result of the modelling process that yields the LP. For example, variables and equations may be introduced to define values used later in the model, or even just for reporting. Alternatively, a modeller may introduce constraints to guarantee a property that is implied by other constraints and bounds on variables.

Identifying a smaller, equivalent problem to solve is called a presolve procedure and is a non-trivial task. It is necessary to recover the primal and dual solution of the original LP using the solution of the smaller LP. This procedure is called postsolve. Presolve and postsolve are examined in Chapter 3.

The simplex algorithm is particularly suitable when solving LP problems for which the optimal solution of a related problem is known. However, in the absence of such information, a heuristic technique that finds a solution that is near-optimal, or at least near-feasible, faster than the simplex method can be extremely valuable.

One such procedure is the “Idiot crash” used in the open-source simplex solver, `Clp` [11]. The

Idiot crash leads to strikingly good performance of `Clp` on particularly challenging LP problems citeMittelmannSimplex so, since there was no published scientific study of it, Google funded a research project to carry this out. Chapter 4 gives the first analysis of the Idiot crash algorithm, and discusses the scope for improving it further.

The presolve techniques developed for this thesis were the catalyst for the creation of the open-source linear optimization software system known as `HiGHS`. Developed over the past five years by combining the presolve with independently-developed linear programming solvers, `HiGHS` is now the world's best open-source linear and mixed-integer optimization software. `HiGHS` is gradually becoming the solver of choice in major application interfaces such as SciPy and JuMP. Critical to its success has been the development of a modern and rigorous software engineering and testing environment. The underlying philosophy and techniques are discussed in Chapter 5.

Chapter 2

Mathematical Background

A linear programming (LP) problem in general bounded form is defined as

$$\text{minimize } f = \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to } \boldsymbol{L} \leq A\boldsymbol{x} \leq \boldsymbol{U} \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}. \tag{2.1}$$

For convenience, discussion and analysis of algorithms in publications are restricted to linear programming (LP) problems in standard form:

$$\text{minimize } f = \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to } A\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0} \tag{2.2}$$

or in general bounded standard form:

$$\text{minimize } f = \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to } A\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \tag{2.3}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{c} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $m < n$. In problems of practical interest, the number of variables and constraints can be large and the matrix A is sparse. It can also be assumed that A has full rank of m. The algorithms, discussion and analysis below extend naturally to more general LP problems, since any problem in general form can be transformed to standard form.

LP duality

In addition to the solution values for the primal variables, for linear programming problems information about the optimal dual solutions must also be calculated. The Lagrangian function for an LP is defined using a dual variable y_i for each row i of the constraint matrix and also a dual variable z_j for to each column j. The primal and dual solutions of an LP must satisfy certain conditions at an optimal point.

2.1 KKT optimality conditions

The Karush-Kuhn-Tucker conditions (KKT) are necessary and sufficient conditions for optimality of an optimization problem. When applied to a general bounded LP ((2.1)), the KKT conditions are as follows:

$$A^T\boldsymbol{y} + \boldsymbol{z} = \boldsymbol{c} \quad \big\} \quad \text{Stationarity of Lagrangian}$$

$$\left.\begin{array}{ccccc} \boldsymbol{L} & \leq & A\boldsymbol{x} & \leq & \boldsymbol{U} \\ \boldsymbol{l} & \leq & \boldsymbol{x} & \leq & \boldsymbol{u} \end{array}\right\} \quad \text{Primal Feasibility}$$

$$\left.\begin{array}{l} y_i \geq 0,\ \text{for each row } i : \boldsymbol{a}_i^T\boldsymbol{x} = L_i \wedge L_i < U_i \\ y_i \leq 0,\ \text{for each row } i : \boldsymbol{a}_i^T\boldsymbol{x} = U_i \wedge U_i > L_i \\ y_i = 0,\ \text{for each row } i : L_i = -\infty \wedge U_i = \infty \\ z_j \geq 0,\ \text{for each column } j : x_j = l_j < u_j \\ z_j \leq 0,\ \text{for each column } j : x_j = u_j > l_j \\ z_j = 0,\ \text{for each column } j : l_j = -\infty \wedge u_j = \infty \end{array}\right\} \text{ Dual Feasibility}$$

$$\left.\begin{array}{l} y_i(\boldsymbol{a}_i^T\boldsymbol{x} - L_i) = 0,\ \text{for each row } i : L_i > -\infty \\ y_i(U_i - \boldsymbol{a}_i^T\boldsymbol{x}) = 0,\ \text{for each row } i : U_i < \infty \\ (x_j - l_j)z_j = 0,\ \text{for each column } j : l_j > -\infty \\ (u_j - x_j)z_j = 0,\ \text{for each column } j : u_j < \infty \end{array}\right\} \text{ Complementary Slackness}$$

For reference, the KKT conditions for problem (2.3) are:

$$A^T\boldsymbol{y} + \boldsymbol{z} = \boldsymbol{c} \quad \} \quad \text{Stationarity of Lagrangian}$$

$$\left.\begin{array}{ccccc} & A\boldsymbol{x} & = & \boldsymbol{b} & \\ \boldsymbol{l} & \leq & \boldsymbol{x} & \leq & \boldsymbol{u} \end{array}\right\} \text{ Primal Feasibility}$$

$$\left.\begin{array}{l} z_j \geq 0,\ \text{for each column } j : x_j = l_j < u_j \\ z_j \leq 0,\ \text{for each column } j : x_j = u_j > l_j \\ z_j = 0,\ \text{for each column } j : l_j = -\infty \wedge u_j = \infty \end{array}\right\} \text{ Dual Feasibility}$$

$$\left.\begin{array}{l} (x_j - l_j)z_j = 0,\ \text{for each column } j : l_j > -\infty \\ (u_j - x_j)z_j = 0,\ \text{for each column } j : u_j < \infty \end{array}\right\} \text{ Complementary Slackness.}$$

2.2 The simplex algorithm and basic solutions

The simplex algorithm builds on the property that any LP that is neither infeasible, nor unbounded, has an optimal solution at a vertex of the feasible region. A basic feasible solution is a useful characterisation of a vertex, since it allows optimality to be determined and, if it is not optimal, allows an improving direction to be identified.

Let m be the number of constraints and n be the number of columns in an LP. Let $\mathcal{B}$ and $\mathcal{N}$ be a partition of the index set $\{1, 2, .., n + m\}$ so that $|\mathcal{B}| = m$ and $|\mathcal{N}| = n$. The variables with indices in $\mathcal{B}$ are referred to as basic and the variables with indices in $\mathcal{N}$ are nonbasic. Permuting the variables according to the index sets yields $B\boldsymbol{x}_B + N\boldsymbol{x}_N = \boldsymbol{b}$, for problem (2.3). A basic solution must have B non-singular. At a vertex, the nonbasic variables are at a bound.

Let $\boldsymbol{x}_N = \boldsymbol{b}_N + \boldsymbol{d}_N$, where $\boldsymbol{b}_N$ is the value of the nonbasic variables and $\boldsymbol{d}_N$ is any deviation from this value. This gives

$$\boldsymbol{x}_B = B^{-1}[\boldsymbol{b} - N(\boldsymbol{b}_N + \boldsymbol{d}_N)] = \hat{\boldsymbol{b}} - B^{-1}N\boldsymbol{d}_N,$$

where $\hat{\boldsymbol{b}} = B^{-1}(\boldsymbol{b} - N\boldsymbol{b}_N)$. Substituting this into

$$f = \boldsymbol{c}_B^T\boldsymbol{x}_B + \boldsymbol{c}_N^T\boldsymbol{x}_N,$$

gives

$$f = \boldsymbol{c}_B^T(\hat{\boldsymbol{b}} - B^{-1}N\boldsymbol{d}_N) + \boldsymbol{c}_N^T(\boldsymbol{b}_N + \boldsymbol{d}_N).$$

After reordering the terms,

$$f = \boldsymbol{c}_B^T\hat{\boldsymbol{b}} + \boldsymbol{c}_N^T\boldsymbol{b}_N - \boldsymbol{c}_B^T(B^{-1}N\boldsymbol{d}_N) + \boldsymbol{c}_N^T\boldsymbol{d}_N.$$

Let $\hat{f} = \boldsymbol{c}_B^T \hat{\boldsymbol{b}} + \boldsymbol{c}_N^T \boldsymbol{b}_N$, so

$$\begin{aligned} f &= \hat{f} - \boldsymbol{c}_B^T(B^{-1}N\boldsymbol{d}_N) + \boldsymbol{c}_N^T\boldsymbol{d}_N \\ &= \hat{f} + (-\boldsymbol{c}_B^T(B^{-1}N) + \boldsymbol{c}_N^T)\boldsymbol{d}_N \\ &= \hat{f} + (\boldsymbol{c}_N^T - \boldsymbol{c}_B^T(B^{-1}N))\boldsymbol{d}_N \\ &= \hat{f} + (\boldsymbol{c}_N - N^T B^- T\boldsymbol{c}_B)\boldsymbol{d}_N \\ &= \hat{f} + \hat{\boldsymbol{c}}_N\boldsymbol{d}_N. \end{aligned}$$

The reduced costs $\hat{\boldsymbol{c}}_N = \boldsymbol{c}_N - N^T B^{-T}\boldsymbol{c}_B$ measure the rate of change in the objective value for a unit change in a nonbasic variable. Since components of $\boldsymbol{d}_N$ have to be non-negative (non-positive) according to whether the nonbasic variable is at its lower (upper) bound (no sign requirement if variable is free at zero), optimality follows from the component of the reduced cost being non-negative (non-positive). At each iteration, simplex chooses one index from the basic set and one index from the nonbasic set and swaps them. The indices are chosen, so that the objective at the new iterate is no larger than the objective at the previous iterate. Partitioning the Lagrangian stationarity equations from Section 2.1 into basic and nonbasic equations,

$$A^T = \begin{bmatrix} B^T \\ N^T \end{bmatrix}, \text{ so}$$

$$\begin{bmatrix} B^T \\ N^T \end{bmatrix} \boldsymbol{y} + \begin{bmatrix} \boldsymbol{z}_B \\ \boldsymbol{z}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_B \\ \boldsymbol{c}_N \end{bmatrix}.$$

Let $\boldsymbol{z}_B = \boldsymbol{0}$. Then, from the first equation, $\boldsymbol{y} = B^{-T}\boldsymbol{c}_B$. From the second equation follows that

$$\boldsymbol{z}_N = \boldsymbol{c}_N - N^T\boldsymbol{y} = \boldsymbol{c}_N - N^T B^{-T}\boldsymbol{c}_B = \hat{\boldsymbol{c}}_N.$$

At a vertex, for $\boldsymbol{z}_B = \boldsymbol{0}$, the complementary slackness condition from (2.1) is satisfied, since every nonbasic variable is at a bound and for all basic columns the dual is zero.

The simplex algorithm requires a basic feasible solution to start from. If no solution information is available, simplex is started from the origin. However, a better starting basis can be greatly beneficial for performance. It is possible to obtain a basic solution from a general solution to an LP using a procedure called crossover. Discussion of crossover techniques is beyond the scope of this thesis.

2.3 LP Solution process

The solution of linear programming problems is most efficiently achieved using an algorithm belonging to one of two main classes of solution techniques: the simplex method and the interior point method (IPM). Depending on the structure and coefficients of the problem, a method in one class may perform better than algorithms from the other class.

In practice, most LP formulations contain redundancies. Regardless of the solution technique most appropriate for a particular LP, for most problems solved in practice, there exists an equivalent LP of smaller dimensions. This results from the redundancies present in the model, often introduced as a result of the LP being formulated via a modelling language.

Furthermore, most often, there are many equivalent LPs of smaller dimensions. Depending on whether simplex or IPM is used for the solution process, one reduced LP formulation may be preferable over another for a particular instance.

Presolve is a procedure which takes an LP and returns an equivalent LP of smaller dimensions. The goal is to obtain a solution of the reduced LP. Then, a postsolve procedure is applied to recover the solution to the original LP from the solution of the reduced LP, most commonly found via simplex or IPM.

Crash start is applied after presolve, once the reduced problem is identified. Not all crashes, however, finish with a basic point. In such cases, or when incomplete information about the solution is available, a crossover is required. A crossover is a procedure, which takes a solution as a starting point and, if successful, returns another solution, which satisfies the conditions for a basic solution. This is then used for simplex to start from. Consequently, the sequence of solution steps is

Presolve LP $\rightarrow$ Crash $\rightarrow$ Crossover $\rightarrow$ Solve reduced LP $\rightarrow$ Recover original LP solution.

2.3.1 Presolve background

In this section are presented previous works on presolve and postsolve for LP. There are not many academic publications regarding presolve due to its complexity. The presolve procedure described in Chapter 3 was originally based on the presolve elimination rules described by Andersen and Andersen [3]. Presolve in an IPM setting was studied by Jacek Gondzio [21]. Meszaros and Suhl give an overview and introduce a new reduction technique in [37]. Presolve techniques for MIP are discussed in [1, 19].

An interesting and beneficial technique for exploiting symmetry is presented by Grohe, Kersting et al. in [23]. The dimensions of an LP are reduced by applying colour refinement, an algorithmic routine for graph isomorphism testing. The number of variables and constraints of the reduced LP correspond to the number of colour classes of the colour refinement algorithm.

2.3.2 Crash background

Efficient crash start techniques are presented by Maros and Mitra in [35, 36]. Robert Bixby outlines an initial basis construction method used in `Cplex` in [8]. The aim of the techniques is to obtain a primal feasible point by removing equations and boxed constraints from the basis, since they are tighter, so less likely to be satisfied if their slacks are basic. Equations and boxed constraints are replaced with free variables or one-sided variables, since in general they are easier to be feasible. The resulting basis matrix is triangular or near-triangular.

The aim of a crash being to find a solution that is near-optimal, or at least near-feasible, faster than the simplex method. Dual simplex has the same aims with crash. The basic dual variables $\boldsymbol{z}_B$ are zero, and nonbasic dual variables are feasible if they correspond to equations or fixed variables, or boxed constraints/variables. So, dual crash has the same motivation: remove equations and boxed constraints from the basis and replace them with free variables or one-sided variables.

Crashes are fast because they do not require matrix factorization, instead, the constraint matrix is examined. Some of the crashes presented by Maros and Mitra, and Bixby, are purely structural. Others make use of numerical properties, however, the objective function is not considered. Hence, they are feasibility crashes, not particularly aiming to achieve optimality. In contrast, the techniques discussed in this thesis aim to reach towards both feasibility and optimality during the crash procedure.

A note on complexity

In some cases, the solution returned from a crash can be a good approximate solution to the problem. This approximate solution is obtained quickly and for some applications less accuracy is sufficient. Examples of such problems are given in Chapter 4, which examines some particular crash procedures.

In that sense, a crash can be seen as a way to obtain an approximate solution to an LP very fast. The reason for that is the computational complexity of the crash procedures we investigate.

Algorithm 1 The augmented Lagrangian algorithm for problem ((2.4)).

```
Initialize $\boldsymbol{x}^0 \geq \mathbf{0}$, $\mu^0$, $\boldsymbol{\lambda}^0$ and a tolerance $\tau^0$
For $k = 0, 1, 2, \ldots$
    Find an approximate minimizer $\boldsymbol{x}^k$ of $\mathcal{L}_A(;\boldsymbol{\lambda}^k, \mu^k)$, starting at $\boldsymbol{x}^k$
        and terminating when $\|\nabla_{\boldsymbol{x}} \mathcal{L}_A(\boldsymbol{x}^k, \boldsymbol{\lambda}^k, \mu^k)\| \leq \tau^k$
    If a convergence test for (2.4) is satisfied
        **stop** with an approximate solution $\boldsymbol{x}^k$
    End if
    Update Lagrange multipliers $\boldsymbol{\lambda}$
        Set $\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \mu^k \boldsymbol{r}(\boldsymbol{x}^k)$
    Choose new penalty parameter $\mu^{k+1}$ so that $0 \leq \mu^{k+1} \leq \mu^k$
    Choose new tolerance $\tau^{k+1}$
End
```

Generally, each iteration of the minimization of the subproblem requires a standard vector-matrix multiplication. Each iteration is so fast because the minimization is done for a single component at a time. Hence, the function being minimized becomes a quadratic function in one variable and has a fixed form solution. There are similarities between the crash procedures in Chapter 4 and two well known classes of algorithms, the quadratic penalty methods and augmented Lagrangian methods.

Quadratic penalty methods

For the nonlinear equality problem

$$\text{minimize } f(\boldsymbol{x}) \quad \text{subject to } \boldsymbol{r}(\boldsymbol{x}) = \mathbf{0}, \tag{2.4}$$

the quadratic penalty method minimizes

$$\phi(\boldsymbol{x}, \mu) = f(\boldsymbol{x}) + \frac{1}{2\mu} \boldsymbol{r}(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x}), \tag{2.5}$$

for a decreasing sequence of positive values of μ. If $\boldsymbol{x}^k$ is the global minimizer of $\phi(\boldsymbol{x}, \mu^k)$ and $\mu^k \to 0$, Nocedal and Wright [32] show that every limit point $\boldsymbol{x}^*$ of the sequence $\{\boldsymbol{x}^k\}$ is a global solution of ((2.4)). The subproblem of minimizing $\phi(\boldsymbol{x}, \mu^k)$ is known to be increasingly ill-conditioned as smaller values of μ^k are used [32] and this is one motivation for the use of the augmented Lagrangian method, for which μ would not need to be as small as machine precision.

Augmented Lagrangian methods

The augmented Lagrangian method, outlined in Algorithm 1, was originally presented as an approach to solving nonlinear programming problems like ((2.4)). It was first proposed by Hestenes in his survey of multiplier and gradient methods [29] and then fully interpreted and analysed, first by Powell [44] and then by Rockafellar [47]. The augmented Lagrangian function (2.6) is a combination of the Lagrangian function and the quadratic penalty function [32]. It is the quadratic penalty function with an explicit estimate of the Lagrange multipliers $\boldsymbol{\lambda}$:

$$\mathcal{L}_A(\boldsymbol{x}, \boldsymbol{\lambda}, \mu) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{r}(\boldsymbol{x}) + \frac{1}{2\mu} \boldsymbol{r}(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x}). \tag{2.6}$$

Although originally intended for nonlinear programming problems, the augmented Lagrangian method has also been applied to linear programming problems [16, 24]. However, neither article assesses its performance on large-scale practical LP problems.

2.4 Further solution methods for LP

The simplex method and the interior point methods are widely used and very efficient on many problems. Not many algorithms could demonstrate such competitive performance on general test LP sets. Davis and Hager suggest a strong alternative in [14, 25, 26]. Yen et al. present another method in [51].

2.4.1 Graph Partitioning

For some larger LP instances, it is beneficial to exploit the underlying pattern of the nonzeros in the constraint matrix. Decomposition methods make use of block structure on the diagonal with linking columns and rows. Identifying block structure and corresponding permutations of the columns was investigated by Aykanat et al. in [6]. Ferris and Horn present heuristics in [17]. Rosen and Meier suggest a solution method in [48]. Graph partitioning has been used for parallel solution of linear programs in [30, 49].

2.4.2 First order methods

The crash presented in this thesis has the complexity of a first order method. Recent work on applying first order methods to LP problems was presented by Applegate et al. in [4, 5]. The techniques can be related to the crash technique in Chapter 4, particularly due to performance on certain classes of hard problems.

Chapter 3

Presolve

As stated in Section 2.3, significant redundancies in an LP model may make the solution process very inefficient. Consequently, redundancies must be handled separately, prior to applying the simplex method. Presolving for LP problems seeks to identify redundant variables and constraints as well as exploring logical implications of the bounds on variables and constraints. This identifies a smaller LP problem with a relatively smaller feasible region which is often solved much more efficiently. A postsolve procedure deduces the solution of the original problem from the optimal solution of the presolved problem.

Ideally, the solution to the original LP that is recovered from the solution of the reduced LP will be optimal. However, numerical calculations during postsolve may mean that tolerated primal and dual infeasibilities in the solution of the reduced LP are scaled up to the extent that the recovered solution is not optimal with respect to the required tolerances. In the case when the recovered solution does not fully satisfy the LP's feasibility and optimality conditions, additional simplex iterations may be required to reach optimality for the original problem after the postsolve procedure.

My work was initially based on the presolve elimination rules described by Andersen and Andersen [3]. The initial implementation of the rules was done first for equality constrained LPs. Later, the corresponding rules for an LP problem in general bounded standard form were derived and implemented in the context of a high-performance simplex solver. The resulting presolve procedure already reduced the dimensions of many test problems significantly. Despite the good performance on a subset of the test problems, the presolve collection of rules needed to be expanded to ensure the presolve implementation performance is comparable to good open source LP solvers. In addition to the rules presented in the publication by the Andersens, several other rules were implemented.

Presolve for linear programming problems recovers primal as well as dual solutions for the LP column and row values. It is a non-trivial task to implement dual postsolve correctly due to the degeneracy of the solution, often observed in practical applications.

It is possible to detect unboundedness or infeasibility of the original problem during presolve as well. In that case, further presolve or calling the solver are not required or necessary, and the information is propagated back to the user. It is important to note that dual infeasibility does not imply primal unboundedness, since the problem could also be primal infeasible. Hence, there is only one status for infeasible or unbounded, rather than two separate ones.

3.1 A sequence of rules

The presolve procedure itself consists of applying elimination rules to the current LP to eliminate one or multiple rows or columns of the problem. Each presolve rule has a corresponding postsolve procedure. Once the reduced LP solution is obtained from the solver, the postsolve steps are applied in the corresponding order.

Removing rows

Trivial examples of elimination rules are the empty row and redundant row rules. Empty rows can be removed from the problem as long as their bounds are consistent. Otherwise, the problem may be detected as primal infeasible. Redundant rows can be removed from the LP. The dual row values can be set to zero in both cases, for the eliminated row. Detailed descriptions of trivial presolve rules are presented in [3].

Removing columns: an example

A trivial example for a column elimination rule is the fixed column. If a column has equal lower and upper bounds then its primal value can be set to the corresponding bound value and the column can be eliminated from the problem. The values of all row bounds corresponding to rows with a non-zero coefficient of the fixed column would need to be updated. A constant term is added to keep track of the changes to the objective function of the LP. Dual postsolve then sets the dual value for this column to be zero, which is sufficient in this case. For a fixed column j,

$$l_j = u_j \quad \rightarrow \quad \begin{cases} x_j = l_j \\ z_j = 0. \end{cases}$$

Consequently, such a column j can be substituted out of the problem.

3.2 Elimination rules in `HiGHS`

Below is presented a list of the presolve elimination rules implemented in `HiGHS`. The rules belong to one of several subsets of rules, grouped by characteristics of the elimination rules. The subsets are as follows:

- Trivial rules
- Singleton column rules
- Implied bounds rules
- Matrix modification rules

3.2.1 Rules based on available publications

As mentioned above, the presolve implementation of `HiGHS` was originally based on the work by Andersen and Andersen [3]. Several remarks on the implementation are presented below.

The rules from [3] which are in `HiGHS`:

Trivial rules

- Empty row
- Empty column
- Fixed column
- Singleton row
- Forcing row
- Redundant row

Singleton column rules

- Free column singleton
- Column singleton in a doubleton equality: a special case of the column singleton inequality rule implemented in `HiGHS`. The latter was an extended version of the rule in [3], modified, so it can handle general bounded LP formulations.

- Implied free column singleton

Implied bounds rules

- Dominated row
- Dominated column
- Weakly dominated column

A modification of the rules presented in [3] was required for their application to LPs in general bounded form (2.1). Section 3.2.3 explores such modifications, implemented in `HiGHS`.

3.2.2 Extensions

As mentioned above, additional rules were implemented in order to improve the efficiency of the overall solution time. Such rules were selected based on the performance of `HiGHS` and other open source solvers, such as Google's `glop` [22] and `Clp`, on test problems as well as LP models arising from industrial applications.

Zero cost column singleton

A column singleton x_j, for a column j with zero cost $c_j = 0$, can be eliminated from the problem. Since the column is not present elsewhere in the matrix, it can be set to any feasible value within the column bounds, without a change in the obective. In that case, the bounds of the column j can be transferred to the row bounds.

This rule was observed to perform well in `glop`, Google's OR Tools simplex solver.

Doubleton equation

The doubleton equation presolve rule handles constraints of the type

$$a_{ij}x_j + a_{ik}x_k = b_i,$$

for a given row i. This is a well known rule, implemented in open source solvers such as `glop` and `Clp`.

The doubleton equation belongs to the subset of rules which may lead to changes in the non-zero pattern of the matrix.

3.2.3 Modifications

This section explores modifications of the rules presented in [3] for LPs in general bounded form (2.1).

Trivial rules

For some rules no modifications were necessary. For instance, the presolve and postsolve of a fixed column are not affected by the values of the row bounds in which the fixed column is present. Some rules, which utilize the row bound value, were modified. For the calculation of implied bounds the algorithm was modified to only use the finite ones for calculation of implied primal and dual values. The postsolve had to be adjusted accordingly in the presence of distinct bounds.

Column singleton in a doubleton inequality

The doubleton equality rule was generalized to handle inequality constraints as well. In practice, many LPs would have at least one constraint with a distinct upper and lower bound, possibly infinite. It is possible to detect that a column singleton can be removed if the doubleton inequality leads to tighter bounds on the column. An example is

$$0 \leq x + y \leq 1, \quad x \in [0, 1], \quad y \in [-5, 5].$$

We know that $x \in [0, 1]$ so adding that to y we obtain $x + y \in [-5, 6]$. The row is limited between 0 and 1 so it can be deduced that when the constraint is satisfied, $y \in [0, 1]$. Hence, the original bounds on y of -5 and 5 would never be reached at a feasible solution and can be removed from the problem. Then the column would be eliminated along with the row with the free column singleton rule.

While the rule to eliminate a column singleton in a doubleton equation always removes the row and column singleton, the generalized version of the rule would only remove the column singleton if it is found to be implied free.

Parallel rows and columns

Note, that the implementation of `HiGHS` presolve does not contain all of the rules presented in [3]. In particular, the parallel row and parallel column elimination rules were omitted. It was observed that on LP test set instances the performance of `HiGHS` presolve was satisfactory even without the parallel row or column elimination rules. Due to the multiple primal and dual implied bounds' calculations, many of the parallel rows and columns were eliminated due to a combination of other elimination rules, applied in a suitable order.

3.3 Postsolve

During postsolve, the primal and dual values of the original problem are recovered using the solution values to the reduced LP, as well as information about the reduction process, stored during presolve. For some presolve rules, applied to an equality form LP, primal and dual postsolve are given in [3]. The values should be chosen so that the KKT conditions in section 2.1 are satisfied, and the basis remains valid and non-singular.

Some examples of the postsolve of trivial rules are given below, to illustrate the idea:

- Empty row
  The row dual y_i can be set to any value, so set $y_i = 0$,
- Singleton row with $a_{ij} \neq 0$

$$y_i = \frac{c_j - \sum_{k \neq i} y_k a_{kj}}{a_{ij}}, z_j = 0,$$

- Empty or fixed column

$$z_j = c_j - \sum_i y_i a_{ij}.$$

For more sophisticated rules, at each step of postsolve, it is possible to derive bounds on the row or column duals using information from the current solution. The values should be chosen so that the KKT conditions are satisfied, and the solution remains basic.

Doubleton equality

Below are presented the presolve and postsolve steps of the doubleton equation rule. It handles constraints of the type

$$a_{ij} x_j + a_{ik} x_k = b_i,$$

for a given row i. Suppose, without loss of generality, that column k has nonzeros fewer than or equal to the number of nonzeros of column j in the constraint matrix.

The column value of x_k must satisfy the constraint, hence

$$x_k = \frac{b - a_{ij}x_j}{a_{ik}}.$$

This is a well known rule, implemented in open source solvers such as `glop` and `Clp`.

Presolve

The variable is substituted out of the problem, the presolve procedure performs the following steps:

- Update bounds on the remaining variable. Suppose, that $a_{ij} > 0, a_{ik} > 0$ and both l_k and u_k are finite. Column j_k must satisfy its bounds so

  $$l_j \leq x_j \leq u_j.$$

  Multiplying by the coefficient yields

  $$a_{ij}x_j \in [b - a_{ik}u_k, +b - a_{ik}l_k].$$

  The column bounds of x_j are updated correspondingly in the reduced problem, if necessary.

- Update the cost of the remaining variable.
- Update the bounds and coefficient of x_j at each row where x_k is nonzero.

Basis

In addition to satisfying the KKT conditions in Section 2.1, a point returned by postsolve must also be a basic feasible solution (BFS) in order to hot-start the simplex algorithm. At each step of postsolve where a new row is introduced, a variable must be identified as basic. Basic variables can have primal values between their lower and upper bounds but must have a zero dual value. Nonbasic variables must be at a bound but can have nonzero duals. The sign of the dual value must correspond to the bound at which the primal value is fixed.

Postsolve

The primal values are postsolved according to the reductions in the presolve step. For dual postsolve there are several cases, depending on the bound and solution values for the column remaining in the matrix.

There are several possibilities for the ordering of the reduced solution for the remaining column and its bounds. Suppose, that l_j and r_j in the reduced problem are updated to l_j^r and u_j^r, respectively, and the solution for the reduced problem for column j is x_j^r.

- If $l_j^r < x_j^r < u_j^r$, then the reduced x_j is basic. In the postsolved problem it remains basic, since the column bounds of x_j may become tighter after applying the presolve rule, but would not be looser than the original bounds of column j.
- If $l_j^r = x_j^r < u_j^r$, or similarly $l_j^r < x_j^r = u_j^r$ there are two cases:
  - If $l_j^r = l_j$, x_j can be either basic or nonbasic.
  - Otherwise, $l_j^r > l_j$ and the postsolve depends on the basis status of x_j in the reduced problem.
    * If x_j^r is nonbasic it must become basic in the postsolved solution. During postsolve an attempt is made to do that. If the dual value of the eliminated variable x_k is calculated to be infeasible, the dual value is transferred to the row by making the row basic.

    * If x_j^r is basic it must remain basic, since the reduced bounds are tighter than the original bounds.

- If $l_j^r = x_j^r = u_j^r$ there is no restriction on the dual value of column j in the reduced problem but it may become infeasible in the original one.

If the eliminated variable is strictly between bounds it must be ensured that it is basic in the postsolved problem. If it is at a bound, the dual value are either non-positive or non-negative, unless it is an equality. Note that the equality case, when $l_j = x_j$ would be handled by the fixed column presolver.

In the cases when x_k can be both basic and nonbasic and the current values do not fall into one of the special cases above, an attempt is made to set it to basic. If that assignment of values is infeasible, the remaining column x_j is selected for the basis.

Effect of order of elimination rules on performance

The experiments following the implementation of the zero cost column singleton presolve elimination rule confirmed a high sensitivity of the method to the order in which the elimination rules are applied. In fact, the `HiGHS` presolve implementation performed faster than Google's LP solver `glop` on many LP test set instances, as a result of modifying the default presolver order in `HiGHS`. At the time `HiGHS` had fewer rules implemented so to ensure the consistency of the experiment results, `glop` was used with a corresponding subset of presolve rules.

3.4 Implementation

Despite the fact that many of the elimination rules are not based on challenging or elaborate mathematical concepts, the efficient implementation of presolve is far from simple. One reason there are so few publications on the topic and even fewer implementation details available is the nature of the algorithm: a subset and order of elimination rules that is beneficial for a particular model may not be an appropriate subset or order for another LP model. Additionally, the data structures most appropriate for presolve may make the corresponding postsolve impossible to implement efficiently.

Another challenging aspect in the development is the tracking and removing of bugs in presolve. An incorrect application of a single presolve rule to an edge case model may lead to a reduced LP which is no longer equivalent to the original one. Recovering the solution to the original LP from the calculated solution to the reduced LP will not, in general, result in an optimal and feasible solution to the original model. However, in practice, it may happen that such an issue is not triggered on standard LP test sets. Issues naturally may go unnoticed for a long time, until an attempt is made at solving a new model with numerical properties not previously tested against.

Assuming that there are no issues in the implementation and mathematical justification and edge cases for the presolve rules, the reduced LP is equivalent to the original LP. Recovering the primal values is a relatively straightforward task in contrast to recovering the dual values. Due to degeneracy and not only, at many steps of postsolve there may be multiple correct alternatives for primal-dual values which satisfy the optimality conditions. A poor selection may lead to infeasibilities on subsequent steps of postsolve. Such issues are observed often during the implementation.

Implied bounds

During presolve, it is essential to keep track of which column was used to deduce an implied bound on a row and which row was used for an implied bound of a column. This is necessary for presolve, to ensure the elimination steps are correct and also for postsolve, to deduce accurate dual values during the postsolve of each rule.

KKT conditions check

A single primal or dual infeasibility at a particular step of postsolve would get propagated all the way through to the recovered solution of the original LP. Hence, the implementation of postsolve rules would not be possible without a way to ensure correctness at each step. This is achieved by checking that the KKT conditions are satisfied following each individual postsolve rule applied to the current problem.

Performance

The improvement of presolve can lead to significant improvement in the overall solution time. In-depth analysis of particular LP models and classes of problems can be highly beneficial for the performance of a solver on the corresponding models or other LPs belonging to a similar class.

The presolve procedure in `HiGHS` has been developed and tested on standard LP test problems and industrial models, results are given in Section 3.7. For all these LP problems, its computational overhead is not significant relative to the time required to solve the LP.

3.5 Further analysis

After the initial implementation of presolve and postsolve, followed a selection of presolve rules and an order for the default presolver list. In addition, further analysis of the residuals after postsolve was required.

Zero cost column singleton

The zero cost column singleton rule was added to `HiGHS`, however, enabling it led to a reduction in elimination counts for some test problems, and it was disabled by default. Analysis of the performance of other presolve implementations on the test instances revealed that a special case of the zero cost column rule may be particularly beneficial.

The special case is a zero cost column singleton in a doubleton equality. A necessary presolve step is to update the row bounds according to the column bounds and matrix coefficient of the column singleton. During postsolve, the primal value of the eliminated column singleton is calculated so the KKT conditions are satisfied.

Note: Generalizing the zero cost column singleton

Additional experiments were performed with a generalized version of the rule, which handles general bounded inequality constraints as well as equality constraints. This generalized version is likely to be part of any presolve implementation since it is beneficial for many common LPs. However, allowing the generalized version of the rule may lead to incorrect presolve elimination steps for some models.

Fixed column update

Analysis of presolve on the test problems revealed that fixed and empty column detection may be modified to increase the row and column reductions. The code of `HiGHS` was updated to identify and remove fixed and empty columns more often during the presolve procedure.

The new rules combined

Experiments were performed combining the fixed column update with the zero cost column singleton special case. This led to no significant changes when compared with only the fixed column update. Thus confirming that the zero cost column singleton rule should not be included in the default presolver list.

Order of presolve rules

For some problems the presolve performance is very sensitive to the order in which the rules are applied. Hence, an analysis of presolve would be incomplete without an investigation of this effect for particular LP instances. Alternative orderings may be more suitable for a problem, depending on the problem structure.

Modifications to HiGHS presolve

The fixed column update has been added to the HiGHS presolve, but the zero cost column singleton rule and the order modifications were not considered beneficial for general LPs, so these will not be added to the default settings of HiGHS.

Residual values

The primal and dual values recovered in the HiGHS implementation of postsolve are, in most cases, accurate within the feasibility tolerance, as demonstrated in Tables 3.2, 3.3 and 3.4. This can lead to a great performance improvement, for if the values are not accurate, additional simplex iterations are required to "clean up" the solution. This process can take a significant amount of time.

3.6 Further notes on primal and dual postsolve

For primal and dual postsolve steps, elimination data from the corresponding presolve steps must be stored efficiently and applied correctly.

The dual postsolve is more challenging when compared to the primal postsolve, since, for some rules, there are multiple feasible solutions. Care must be taken with the choices of a solution, because a poor choice at a step may lead to an infeasibility later in postsolve.

Additionally, a solution may be feasible within the tolerance at a postsolve step, but may violate feasibility later in postsolve due to the scaling of the residuals.

Some very high performing commercial solvers, such as Gurobi, for some problems, return infeasible solutions after postsolve. The addition of more elaborate presolve and postsolve rules makes that difficult to avoid at all times. Besides complex presolve rules, the propagation of residual errors during postsolve, mentioned above, is another cause of violation of the feasibility of the original problem. Additional simplex iterations after postsolve ensure that the solution returned to the user is feasible within the desired tolerances.

In Tables 3.2, 3.3 and 3.4 are presented the residual values for primal and dual infeasibility after HiGHS 's postsolve.

3.7 Results

The effectiveness of the presolve and accuracy of postsolve were assessed by running HiGHS with and without presolve on a large number of LP test problems. Most problems in the classic Netlib test set [20] are too small to be of interest and, for the problems where presolve reductions were possible, the geometric mean of the speed-up in total solution time achieved by presolve was 1.10. For only one problem (PILOT87) was it necessary to perform iterations after postsolve. A more challenging test set of 74 LPs was compiled from the problems used by Mittelmann in his industry standard benchmarks [40] over recent years, and four industrial LP problems. Of the latter, DCP1 and DCP2 come from Format International (see Chapter 5), and the other two (DETEQ8 and DETEQ27) are deterministic equivalents of stochastic LPs. In the experiments, HiGHS was given a time limit of 3600 seconds. Of these 74 problems, for the 40 listed in Table 3.1, HiGHS achieved some presolve reductions and solved the problem to optimality both with and without presolve. Of the remaining 34, HiGHS failed to solve 17, either with or without presolve (generally both), and no presolve reductions were achieved for the other 17. It was necessary to perform iterations after postsolve for only two (RAIL02 and WATSON_1) of the 42 problems solved by HiGHS.

Focusing on the results given in Table 3.1, the first three columns after the problem name give the solution time without presolve, the time taken by presolve, and the total solution time for the problem. Also given are the percentage of the total solution time accounted for by presolve, and the speed-up in (total) solution time as a consequence of performing presolve. The advantage of performing presolve is not universal. For three problems, it is significantly slower to solve the problem by performing presolve. However, for many more the advantage is significant, and the geometric mean of the speed-up is 1.67. Mittelmann uses a shifted geometric mean solution time to de-emphasise problems solved faster. Using this measure, the mean solution time is 111 seconds without presolve, and 82 with presolve, a relative improvement of 1.35. Remarkable amongst the test problems is SGPF5Y6, where presolve accounts for 38% of the solution time. Rather than indicating inefficiency of the presolve, this large figure reflects its success in reducing the solution time by a factor of 45!

The results illustrate, similarly to the results in other publications on presolve such as [3], that presolve in most cases leads to a significant reduction in the overall solution time.

Accuracy

Tables 3.2, 3.3 and 3.4 give the sums of primal infeasibilities, dual infeasibilities and dual residuals when presolve is not used, and after postsolve when presolve is used. The primal residuals after postsolve are zero. Considering the sums when presolve is used, relative to the sums when presolve is not used (and is positive), the (geometric) mean values indicate a modest increase due to accumulation of numerical error.

Model	Original	Presolve	Total	Presolve (%)	Speed-up
NEOS5251015	40.40	0.26	337.00	0.08	0.12
CONT4	913.00	0.05	3440.00	0.00	0.27
S250R10	218.00	0.21	317.00	0.07	0.69
RAIL4284	1690.00	1.21	1820.00	0.07	0.93
SAVSCHED1	748.00	0.31	796.00	0.04	0.94
FOME12	55.30	0.02	56.60	0.03	0.98
PDS-40	13.80	0.07	13.00	0.55	1.06
FOME13	143.00	0.04	132.00	0.03	1.08
RAIL02	1180.00	0.12	1080.00	0.01	1.09
SHS1023	92.40	0.32	79.40	0.41	1.16
NEOS5052403	202.00	0.36	172.00	0.21	1.17
STORM-125	2.77	0.07	2.34	3.20	1.18
LP22	9.18	0.00	7.70	0.06	1.19
STAT96V4	98.00	0.04	82.20	0.05	1.19
BUILDINGEN	49.50	0.08	38.90	0.20	1.27
SUPPORT10	225.00	0.17	176.00	0.10	1.28
MOD2	34.80	0.02	27.10	0.09	1.28
WORLD	46.10	0.02	35.80	0.07	1.29
STORM_1000	97.50	1.10	74.00	1.49	1.32
PDS-80	50.60	0.18	37.90	0.48	1.34
LINF_520C	1740.00	0.06	1290.00	0.00	1.35
FHNW-BIN0	2730.00	0.20	1830.00	0.01	1.49
STP3D	191.00	0.20	127.00	0.16	1.50
PDS-100	74.60	0.21	49.50	0.42	1.51
PSCHED3-3	1860.00	0.16	1230.00	0.01	1.51
WATSON_2	135.00	0.24	87.60	0.27	1.54
DETEQ8	0.41	0.02	0.24	7.99	1.67
WATSON_1	43.20	0.15	23.70	0.62	1.82
DETEQ27	2.28	0.07	1.25	5.50	1.82
EX10	265.00	0.10	143.00	0.07	1.85
NS1687037	1190.00	0.08	608.00	0.01	1.96
DCP1	0.62	0.01	0.22	2.44	2.88
KEN-18	11.20	0.10	3.17	3.02	3.53
DCP2	8.90	0.03	2.41	1.44	3.69
CRE-B	4.89	0.02	1.31	1.56	3.73
BRAZIL3	95.40	0.02	17.00	0.09	5.61
DBIC1	689.00	0.08	120.00	0.06	5.74
NS1644855	1080.00	0.11	145.00	0.07	7.45
SQUARE41	572.00	0.76	51.50	1.48	11.11
SGPF5Y6	133.00	1.12	2.97	37.71	44.78

Table 3.1: Effect of presolve reductions on solution time for 40 test LP problems.

Model	Without presolve	With presolve	Relative
BRAZIL3	4.1E-11	2.5E-11	5.9E-01
BUILDINGEN	7.9E-14	2.0E-12	2.5E+01
CONT4	8.9E-16	2.9E-04	3.3E+11
CRE-B	5.4E-13	6.8E-13	1.3E+00
DBIC1	3.1E-11	8.8E-11	2.9E+00
DCP1	0	2.1E-12	
DCP2	1.8E-15	2.0E-11	1.1E+04
DETEQ27	0	2.2E-13	
DETEQ8	3.3E-16	5.9E-14	1.8E+02
EX10	1.1E-12	1.6E-11	1.4E+01
FHNW-BIN0	0	4.5E-12	
FOME12	1.4E-10	2.9E-10	2.0E+00
FOME13	1.0E-09	1.3E-09	1.2E+00
KEN-18	0	0	
LINF_520C	5.8E-05	1.2E-05	2.1E-01
LP22	1.8E-13	1.8E-11	9.6E+01
MOD2	3.7E-05	2.3E+01	6.3E+05
NEOS5052403	3.5E-11	2.7E-11	7.5E-01
NEOS5251015	1.1E-09	4.3E-10	4.0E-01
NS1644855	5.2E-12	2.5E-11	4.9E+00
NS1687037	0	9.9E-04	
PDS-100	0	0	
PDS-40	0	0	
PDS-80	0	1.8E-15	
PSCHED3-3	2.1E-11	7.4E-09	3.5E+02
RAIL02	0	2.6E-13	
RAIL4284	4.4E-14	9.8E-13	2.2E+01
S250R10	4.0E-14	7.0E-15	1.8E-01
SAVSCHED1	1.4E-14	9.4E-13	6.9E+01
SGPF5Y6	2.9E-10	0	0
SHS1023	1.5E-10	1.1E-10	7.2E-01
SQUARE41	2.7E-11	4.6E-11	1.7E+00
STAT96V4	1.3E-05	2.0E-05	1.5E+00
STORM_1000	0	1.4E-10	
STORM-125	0	1.5E-11	
STP3D	6.7E-16	1.0E-13	1.5E+02
SUPPORT10	0	1.7E-13	
WATSON_1	0	4.5E-10	
WATSON_2	0	4.3E-09	
WORLD	3.2E-05	2.3E+01	7.4E+05
Mean			4.6E+01

Table 3.2: Sum of primal infeasibilities without and with presolve.

Model	Without presolve	With presolve	Relative
BRAZIL3	7.7E-10	5.3E-10	6.9E-01
BUILDINGEN	4.9E-17	0	0
CONT4	0	0	
CRE-B	1.0E-10	7.0E-11	6.7E-01
DBIC1	0	0	
DCP1	4.0E-11	0	0
DCP2	8.3E-11	1.0E-10	1.2E+00
DETEQ27	7.0E-09	7.0E-09	1.0E+00
DETEQ8	1.3E-12	1.6E-12	1.2E+00
EX10	1.2E-08	6.0E-09	4.9E-01
FHNW-BIN0	8.6E-08	9.3E-08	1.1E+00
FOME12	7.1E-09	8.0E-09	1.1E+00
FOME13	1.9E-08	1.2E-08	6.7E-01
KEN-18	0	0	
LINF_520C	0	0	
LP22	2.0E-14	1.1E-13	5.6E+00
MOD2	0	0	
NEOS5052403	9.9E-12	1.9E-12	1.9E-01
NEOS5251015	0	0	
NS1644855	1.5E-10	1.3E-07	8.8E+02
NS1687037	0	0	
PDS-100	0	0	
PDS-40	0	0	
PDS-80	1.3E-11	1.6E-11	1.2E+00
PSCHED3-3	3.0E-07	5.5E-07	1.8E+00
RAIL02	2.2E-12	2.7E-05	1.2E+07
RAIL4284	1.5E-12	1.3E-11	8.7E+00
S250R10	3.3E-14	9.2E-15	2.8E-01
SAVSCHED1	0	0	
SGPF5Y6	2.5E-04	2.5E-04	1.0E+00
SHS1023	2.6E-11	2.1E-11	8.2E-01
SQUARE41	2.9E-12	7.7E-13	2.7E-01
STAT96V4	4.2E-14	0	0
STORM_1000	6.9E-13	1.1E-12	1.6E+00
STORM-125	1.1E-12	7.7E-13	7.1E-01
STP3D	1.2E-14	1.2E-14	1.0E+00
SUPPORT10	0	0	
WATSON_1	1.8E-05	1.4E-05	7.5E-01
WATSON_2	1.2E-06	6.4E-07	5.2E-01
WORLD	0	4.5E-13	
Mean			2.3E+00

Table 3.3: Sum of dual infeasibilities without and with presolve.

Model	Without presolve	With presolve	Relative
BRAZIL3	1.0E-11	1.3E-11	1.3E+00
BUILDINGEN	1.1E-10	1.1E-10	1.0E+00
CONT4	1.1E-09	1.3E-07	1.2E+02
CRE-B	3.9E-09	3.7E-09	9.4E-01
DBIC1	0	0	
DCP1	2.4E-09	2.8E-09	1.2E+00
DCP2	1.5E-08	1.8E-08	1.2E+00
DETEQ27	4.3E-11	1.2E-10	2.8E+00
DETEQ8	2.6E-11	9.8E-11	3.8E+00
EX10	2.6E-10	1.1E-10	4.3E-01
FHNW-BIN0	1.5E-09	2.3E-09	1.6E+00
FOME12	6.9E-08	7.5E-08	1.1E+00
FOME13	1.9E-07	1.1E-07	5.8E-01
KEN-18	3.5E-08	4.8E-08	1.4E+00
LINF_520C	0	0	
LP22	8.9E-12	1.8E-11	2.0E+00
MOD2	1.0E-09	1.1E-09	1.0E+00
NEOS5052403	5.3E-12	3.8E-12	7.1E-01
NEOS5251015	0	0	
NS1644855	2.0E-10	4.1E-10	2.0E+00
NS1687037	2.8E-10	3.1E-10	1.1E+00
PDS-100	0	0	
PDS-40	0	0	
PDS-80	4.4E-12	5.7E-13	1.3E-01
PSCHED3-3	7.3E-09	1.3E-08	1.7E+00
RAIL02	5.1E-12	9.6E-12	1.9E+00
RAIL4284	8.9E-11	8.6E-11	9.7E-01
S250R10	1.2E-13	1.1E-13	9.1E-01
SAVSCHED1	2.8E-14	2.7E-14	9.8E-01
SGPF5Y6	1.8E-16	3.3E-16	1.8E+00
SHS1023	1.4E-11	2.0E-11	1.5E+00
SQUARE41	2.3E-10	8.4E-11	3.6E-01
STAT96V4	6.4E-14	5.3E-14	8.3E-01
STORM_1000	2.1E-07	2.0E-07	9.5E-01
STORM-125	2.9E-08	2.9E-08	9.9E-01
STP3D	9.9E-13	9.2E-13	9.3E-01
SUPPORT10	4.5E-14	2.4E-14	5.2E-01
WATSON_1	3.1E-14	3.1E-14	9.9E-01
WATSON_2	2.5E-14	3.0E-14	1.2E+00
WORLD	1.7E-09	1.8E-09	1.0E+00
Mean			1.2E+00

Table 3.4: Sum of dual residuals without and with presolve.

Chapter 4

Crash

A crash is a heuristic technique that finds a solution that is near-optimal, or at least near-feasible, fast. As mentioned in Chapter 2, such a procedure can be extremely valuable, if it can find a solution faster than the simplex method. The Idiot crash algorithm used in the `Clp` open-source simplex solver leads to strikingly good performance on particularly challenging LP problems [40]. Since there is no published scientific study of it, the first part of this chapter gives an analysis of the Idiot crash algorithm. In the second part of this chapter is presented additional work on crash techniques and crash starting in `HiGHS`.

4.1 The Idiot crash

This section presents the Idiot crash algorithm (ICA) in `Clp`, followed by some practical and mathematical analysis of its behaviour. Experiments assess the extent to which ICA accelerates the solution of representative LP test problems using the primal simplex method, and can be used to find a feasible and near-optimal solution of the problems. Theoretical analysis of the limiting behaviour of the algorithm shows that it will solve any LP problem that has an optimal solution. As such, this section follows closely the work published by Galabova and Hall in [18].

4.1.1 The algorithm

The general structure of the ICA algorithm is set out in Algorithm 2. The function, being minimized in each iteration, is

$$h(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{\lambda}^T\boldsymbol{r}(\boldsymbol{x}) + \frac{1}{2\mu}\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}), \quad \text{where} \quad \boldsymbol{r}(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}, \tag{4.1}$$

subject to the bounds $\boldsymbol{x} \geq \boldsymbol{0}$ for sequences of values of parameters $\boldsymbol{\lambda}$ and μ. The minimization function (4.1) corresponds to (2.5) of the Quadratic Penalty algorithm and to (2.6) of the Augmented Lagrangian algorithm. However, the update of the λ parameters and the criterion for parameter updates differ between the ICA and Augmented Lagrangian. Another significant difference is the minimization of the subproblem. In classical Augmented Lagrangian methods the minimization is exact, while in the ICA, the function at each iteration is minimized approximately.

The minimization is performed with respect to each component of $\boldsymbol{x}$ in turn, with the starting index of this loop over all components being chosen randomly. Except for the alternative expression for updating $\boldsymbol{\lambda}^k$ and the component-wise minimization of $h(\boldsymbol{x})$, this algorithm is very close to that of `LANCELOT` [13] applied to problem ((2.2)). The number of ICA iterations is determined heuristically according to the size of the LP and progress of the algorithm. Unless ICA is abandoned after around 20 "sample" iterations (see below), the number of iterations performed ranges between 30 and 200. The value of μ^0 ranges between 0.001 and

Algorithm 2 The Idiot crash algorithm for problem ((2.2)), with $\boldsymbol{r}(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}$.

```
Initialize $\boldsymbol{x}^0 = \boldsymbol{0}$, $\mu^0$, $\boldsymbol{\lambda}^0 = \boldsymbol{0}$
Set $\mu^1 = \mu^0$ and $\boldsymbol{\lambda}^1 = \boldsymbol{\lambda}^0$
For $k = 1, 2, 3, \ldots$
    $\boldsymbol{x}^k \approx \arg\min_{\boldsymbol{x}\geq\boldsymbol{0}} h(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{\lambda}^{k^T}\boldsymbol{r}(\boldsymbol{x}) + \frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x})$
    If a criterion is satisfied (see 4.1.3) update $\mu^k$:
        $\mu^{k+1} = \mu^k/\omega$, for some factor $\omega > 1$
        $\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k$
    Else update $\boldsymbol{\lambda}^k$:
        $\mu^{k+1} = \mu^k$
        $\boldsymbol{\lambda}^{k+1} = \mu^k\boldsymbol{r}(\boldsymbol{x}^k)$
End
```

1000, again according to the LP dimensions. When μ is changed, the factor by which it is reduced is typically $\omega = 0.333$. The final value of μ is typically a little less than machine precision.

The version of ICA implemented in `Clp` has several additional features. If $\boldsymbol{x}^0$ is feasible then the algorithm is not performed and the value $\boldsymbol{x} = \boldsymbol{x}^0$ is returned. Otherwise, initially, the approximate component-wise minimization is performed twice. If a 10% decrease in primal infeasibility is not observed in about 30 iterations, it is considered that ICA would not be beneficial and the value $\boldsymbol{x} = \boldsymbol{x}^0$ is returned. Otherwise, ICA continues but the mechanism for approximate minimization is adjusted. During each subsequent iteration, the function $h(\boldsymbol{x})$ is minimized componentwise 105 times. There is no indication why this particular value was chosen. However, one of the features is the option to decrease this number. From the 50th minimization onward, a check is performed after the function is minimized componentwise 10 times. Progress is measured with a moving average of expected progress. If it is considered that not enough progress is being made, the function is not minimized any longer for the same values of the parameters. Instead, either μ or $\boldsymbol{\lambda}$ is updated and the next iteration is performed. Thus, in the cases when it is likely that the iteration would not be beneficial, not much unnecessary time is spent. Another feature is that in some cases there is a limit on the step size for the update of each x_j. Additionally, there is a statistical adjustment of the values of $\boldsymbol{x}$ at the end of each iteration. These features are omitted from this paper because experiments showed that they have little effect on performance. Depending on the problem size and structure, the weight parameter (μ) is updated either every 3 iterations or every 6. Again, there is no indication why these values are chosen. To a large extent it must be assumed that the algorithm has been tuned to achieve a worthwhile outcome when possible, and terminate promptly when not. The dominant computational cost for each component-wise minimization of $h(\boldsymbol{x})$ is about the same as a matrix-vector product $A\boldsymbol{v}$.

Relation to augmented Lagrangian and quadratic penalty function methods

In form, the augmented Lagrangian function ((2.6)) and the ICA function ((4.1)) are identical for LP problems and in both methods the penalty parameter μ is reduced over a sequence of iterations. However, they differ fundamentally in the update of $\boldsymbol{\lambda}$. For ICA, new values of $\boldsymbol{\lambda}$ are given by $\boldsymbol{\lambda}^{k+1} = \mu^k\boldsymbol{r}(\boldsymbol{x}^k)$. Since μ is reduced to around machine precision and the aim is to reduce $\boldsymbol{r}(\boldsymbol{x})$ to zero, the components of $\boldsymbol{\lambda}$ become small. Contrast this with the values of $\boldsymbol{\lambda}$ in the augmented Lagrangian method, as set out in Algorithm 1. These are updated by the value $\mu^k\boldsymbol{r}(\boldsymbol{x}^k)$ and converge to the (generally non-zero) Lagrange multipliers for the equations.

In ICA, when the values of $\boldsymbol{\lambda}$ are updated, the linear and quadratic functions of the residual $\boldsymbol{r}(\boldsymbol{x})$ in the ICA function ((2.6)) are respectively $\mu^k\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x})$ and $(2\mu^k)^{-1}\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x})$. Thus, since the values of μ^k are soon significantly less than unity, the linear term becomes relatively negligible. In this way the ICA function reduces to the quadratic penalty function ((2.5)) and the later behaviour of ICA is akin to that of a simple quadratic penalty method.

Table 4.1: Test problems, the speed-up of the Clp primal simplex solver when ICA is used, and the percentage of solution time accounted for by ICA. Only for the problem names in bold does default Clp use ICA and the primal simplex solver.

Model	Speed-up	Idiot (%)	Model	Speed-up	Idiot (%)
CRE-B	2.6	28.9	PDS-40	1.3	5.0
DANO3MIP	1.4	3.6	PDS-80	1.0	0.1
DBIC1	1.5	40.6	PILOT87	1.3	2.5
DFL001	1.0	0.1	**QAP12**	2.5	0.6
FOME12	1.1	0.1	**QAP15**	4.0	0.1
FOME13	1.9	3.3	**SELF**	6.1	22.7
KEN-18	1.0	0.7	SGPF5Y6	1.4	4.8
L30	1.9	1.4	**STAT96V4**	1.7	1.2
LINF_520C	9.4	8.2	STORM_1000	4.5	0.8
LP22	1.4	1.9	STORM-125	4.1	10.1
MAROS-R7	0.9	7.8	STP3D	6.5	0.9
MOD2	1.4	2.7	TRUSS	0.8	17.1
NS1688926	1.4	1.0	WATSON_1	1.8	8.9
NUG15	4.2	0.1	WATSON_2	1.1	4.4
PDS-100	2.5	5.4	WORLD	1.3	2.0

4.1.2 Preliminary experiments

The effectiveness of ICA is assessed via experiments with Clp (Version 1.16.10), using a set of 30 representative LP test problems in Table 4.1. This is the set used by Huangfu and Hall in [31], with QAP15 replacing DCP2 because QAP problems are of particular interest and DCP2 is not a public test problem, and NUG15 replacing NUG12 for consistency with the choice of QAP problems used by Mittelmann [38]. The three problems NUG15, QAP12 and QAP15 are linearizations of quadratic assignment problems, where NUG15 and QAP15 differ only by row and column permutations. The experiments are carried out on an Intel i7-6700T processor rated at 2.80GHz with 16GB of available memory. In all cases the Clp presolve routine is run first, and is included in the total solution times.

To assess the effectiveness of ICA in speeding up the Clp primal simplex solver over all the test problems, total solution times were first recorded for Clp with the -primals option. This forces Clp to use the primal simplex solver but makes no use of ICA. To compare these with total solution time when Clp uses the primal simplex solver following ICA, it was necessary to edit the source code so that Clp is forced to use ICA and the primal simplex solver. Otherwise, Clp ran in its default state. The relative total solution times are given in the columns in Table 4.1 headed “Speed-up”. The geometric mean speed-up is 1.9, demonstrating clearly the general value of ICA for the Clp primal simplex solver. Although ICA is of little value (speed-up below 1.25) for seven of the 30 problems, for only two of these problems does it lead to a small slow-down. However, for ten of the 30 problems the speed-up is at least 2.5, a huge improvement. The columns headed “Idiot (%)” give the percentage of the total solution time accounted for by ICA, the mean value being 6.2%. For five problems the percentage is ten or more, and this achieves a handsome speed-up in three cases. However, it does include TRUSS, for which ICA takes up 17% of an overall solution time that is 20% more than with the vanilla primal simplex solver. For only this problem can ICA be considered a significant and unwise investment. Of the ten problems where ICA results in a speed-up of at least 2.5, for only three does it account for at least ten percent of the total solution time. Indeed, for five of these problems ICA is no more than one percent of the total solution time.

This remarkably cheap way to improve the performance of the primal simplex solver is not always of value to Clp because, when it is run without command line options (other than the model file name), it decides whether to use its primal or dual simplex solver. When the former is used, Clp uses problem characteristics to decide whether to use ICA and, if used, to set parameter values for the algorithm. Default Clp chooses the primal simplex solver (and

Table 4.2: Solution times for `Cplex`-12.8.0, `Gurobi`-7.5.0, `Xpress`-8.4.0 and `Clp`-1.16.10 on five notable problem instances from the Mittelmann benchmarks (29/12/17) [38].

Model	`Cplex`	`Gurobi`	`Xpress`	`Clp`
LINF_520C	495	1057	255	35
NUG15	338	14	7	14
QAP12	26	1	1	5
QAP15	365	14	6	13
SELF	18	12	15	5

Table 4.3: Residual and relative objective error following ICA in `Clp`.

Model	Residual	Objective	Model	Residual	Objective
CRE-B	1.3×10^{-9}	6.1×10^{-2}	PDS-40	7.0×10^{-8}	3.0×10^{-2}
DANO3MIP	6.1×10^{-10}	2.0×10^{-2}	PDS-80	2.2×10^{-7}	3.4×10^{-1}
DBIC1	3.8×10^{-1}	8.5×10^{-2}	PILOT87	2.1×10^{0}	6.8×10^{-1}
DFL001	1.1×10^{-9}	3.7×10^{-3}	QAP12	3.6×10^{-10}	1.7×10^{-1}
FOME12	6.4×10^{-9}	4.3×10^{-3}	QAP15	2.1×10^{-10}	2.8×10^{-3}
FOME13	1.2×10^{-8}	5.2×10^{-3}	SELF	5.7×10^{-5}	2.4×10^{-3}
KEN-18	5.4×10^{-8}	7.1×10^{-2}	SGPF5Y6	4.0×10^{-10}	2.1×10^{-1}
L30	1.1×10^{-9}	3.9×10^{0}	STAT96V4	3.0×10^{-3}	1.0×10^{0}
LINF_520C	1.1×10^{-1}	9.1×10^{-3}	STORM_1000	5.9×10^{-6}	5.9×10^{-2}
LP22	1.1×10^{-9}	1.3×10^{-3}	STORM-125	1.4×10^{0}	1.2×10^{-1}
MAROS-R7	4.0×10^{-9}	2.3×10^{-5}	STP3D	7.0×10^{-5}	2.7×10^{-2}
MOD2	3.9×10^{0}	2.1×10^{-1}	TRUSS	7.1×10^{-1}	3.2×10^{-1}
NS1688926	2.5×10^{-9}	4.8×10^{5}	WATSON_1	7.7×10^{-6}	8.7×10^{-1}
NUG15	2.1×10^{-10}	3.7×10^{-4}	WATSON_2	1.4×10^{-10}	9.7×10^{-1}
PDS-100	7.6×10^{-10}	3.7×10^{-4}	WORLD	4.3×10^{0}	5.5×10^{-1}

always performs ICA) for just the ten LP problems whose name is given in bold type. For half of these problems there is a speed-up of at least 2.5, so ICA contributes significantly to the ultimate performance of `Clp`. However, for five problems (CRE-B, PDS-100, STORM-125, STORM_1000 and STP3D), ICA yields a primal simplex speed-up of at least 2.5 but, when free to choose, `Clp` uses its dual simplex solver. In each case the dual simplex solver is at least as fast as using the primal simplex solver following ICA, the geometric mean superiority being a factor of 4.0, so the choice to use the dual simplex solver is justified.

Further evidence of the importance of ICA to the performance of `Clp` is given in Table 4.2, which gives the solution times from the Mittelmann benchmarks [38] for the three major commercial simplex solvers and `Clp` when applied to five notable problem instances. When solving LINF_520C, `Clp` is vastly faster than the three commercial solvers. For the three QAP linearizations (NUG15, QAP12 and QAP15), `Clp` is very much faster than `Cplex`. Finally, for SELF, `Clp` is significantly faster than the commercial solvers.

To assess the limiting behaviour of ICA as a means of finding a point that is both feasible and optimal, `Clp` was run with the `-idiot 200` option using the modified code that forces ICA to be used on all problems. The results are given in Table 4.3, where the columns headed "Residual" contain the final values of $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$. The columns headed "Objective" contain values of $(f - f^*)/\max\{1, |f^*|\}$ as a measure of how relatively close the final value of f is to the known optimal value f^*, referred to below as the objective error. This measure of optimality is clearly of no practical value because f^* is not known. However, it is instructive empirically, and motivates later theoretical analysis. The geometric mean of the residuals is 1.2×10^{-6} and the geometric mean of the objective error measures is 6.1×10^{-2}.

For 17 of the 30 problems in Table 4.3, the norm of the final residual is less than 10^{-7}. Since this is the default primal feasibility tolerance for the `Clp` simplex solver, ICA can be considered to have obtained an acceptably feasible point. Among these problems, the objective

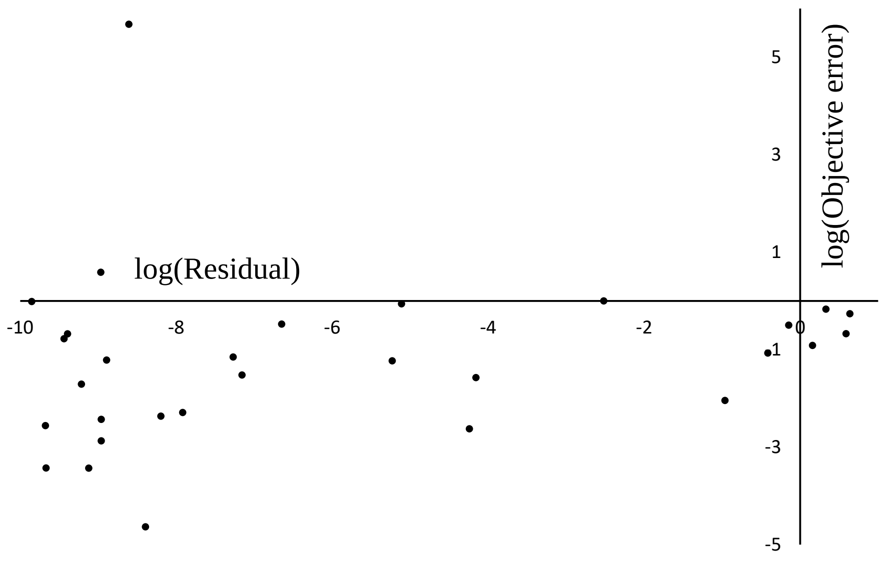

Figure 4.1: Distribution of residual and objective errors.

error ranges between 4.8×10^5 for NS1688926 and 2.3×10^{-5} for MAROS-R7, with only eight problems having a value less than 10^{-2}. Thus, even if ICA yields a feasible point, it may be far from being optimal. A single quality measure for the point returned by ICA is convenient, and this is provided by the product of the residual and objective error, conveniently referred to as the "solution error". As illustrated by the distribution of the objective errors and residual in Figure 4.1, it is unsurprising that there are no problems for which a low value of this product corresponds to an accurate optimal objective function value but large residual.

The remainder of this chapter addresses the question to what extent does the ICA possess theoretical optimality and convergence properties. Additionally, it is investigated which problem features may influence the performance of ICA.

4.1.3 Analysis

In analysing ICA, the initial focus is the function ((4.1)). Fully expanded, this is the quadratic function

$$h(\boldsymbol{x}) = \frac{1}{2\mu}\boldsymbol{x}^T A^T A\boldsymbol{x} + (\boldsymbol{c}^T + \boldsymbol{\lambda}^T A - \frac{1}{\mu}\boldsymbol{b}^T A)\boldsymbol{x} - \boldsymbol{\lambda}^T\boldsymbol{b} + \frac{1}{2\mu}\boldsymbol{b}^T\boldsymbol{b}.$$

Although convexity of the function follows from the Hessian matrix A^TA being positive semi-definite, the Hessian has rank $m < n$. However, the possibility of unboundedness of $h(\boldsymbol{x})$ on $\boldsymbol{x} \geq \boldsymbol{0}$ can be discounted as follows. First, observe that unboundedness could only occur in non-negative directions of zero curvature, so they must satisfy $A\boldsymbol{d} = \boldsymbol{0}$. Hence $h(\boldsymbol{x} + \alpha\boldsymbol{d}) = h(\boldsymbol{x}) + \alpha\boldsymbol{c}^T\boldsymbol{d}$, which, if unbounded below for increasing α, implies unboundedness of the LP along the ray $\boldsymbol{x} + \alpha\boldsymbol{d}$ from any point $\boldsymbol{x} \geq \boldsymbol{0}$ satisfying $A\boldsymbol{x} = \boldsymbol{b}$. Thus, as long as the LP is neither infeasible nor unbounded, $h(\boldsymbol{x})$ is bounded below on $\boldsymbol{x} \geq \boldsymbol{0}$. For some problems, the size of the residual and objective measures in Table 4.3 indicate that ICA has found a point that is close to being optimal. It is therefore of interest to know whether ICA possesses theoretical optimality and convergence properties. With approximate minimization of the ICA function ((4.1)), it is not conducive to detailed mathematical analysis. However, Theorem 1 shows that if the ICA function is minimized exactly and an optimal solution to the LP exists, every limit point of the sequence $\{\boldsymbol{x}^k\}$ is a solution to the problem.

Notes

- During each iteration, at most one of the parameters μ and $\boldsymbol{\lambda}$ is updated: in `Clp`, μ is updated once every few (e.g. 3 or 6) iterations. How often μ is updated does not affect the validity of the proof as long as $\{\mu^k\} \to 0$ as $k \to \infty$ and $\boldsymbol{\lambda}$ is updated at least once every W iterations for some constant $W \geq 1$.
- In the statement of Algorithm 2 it is said that ω is larger than 1. This is not required for the proof, which would still hold in the case of non-monotonicity of $\{\mu^k\}$ as long as $\{\mu^k\} \to 0$ as $k \to \infty$.

Theorem 1. *Suppose that $\boldsymbol{x}^k$ is the exact global minimizer of $h^k(\boldsymbol{x})$ for each $k = 1, 2, \ldots$ and that $\{\mu^k\} \to 0$ as $k \to \infty$. Then every limit point of the sequence $\{\boldsymbol{x}^k\}$ is a solution to problem* (2.2).

Proof. Let $\bar{\boldsymbol{x}}$ be a solution of (2.2) so that, for all feasible $\boldsymbol{x}$, $\boldsymbol{c}^T\bar{\boldsymbol{x}} \leq \boldsymbol{c}^T\boldsymbol{x}$. For each k, $\boldsymbol{x}^k$ is the exact global minimizer for

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}} & h^k(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}) + \dfrac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}) \\ \text{s.t.} & \boldsymbol{x} \geq \boldsymbol{0}, \end{array} \tag{4.2}$$

and, since $\bar{\boldsymbol{x}}$ is feasible for (2.2), it is also feasible for (4.2). Thus, since $h^k(\boldsymbol{x}^k) \leq h^k(\bar{\boldsymbol{x}})$ for each k, it follows that

$$\boldsymbol{c}^T\boldsymbol{x}^k + {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k) + \frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) \leq \boldsymbol{c}^T\bar{\boldsymbol{x}} + {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\bar{\boldsymbol{x}}) + \frac{1}{2\mu^k}\boldsymbol{r}(\bar{\boldsymbol{x}})^T\boldsymbol{r}(\bar{\boldsymbol{x}}). \tag{4.3}$$

Since $\bar{\boldsymbol{x}}$ is a solution of (2.2), $\boldsymbol{r}(\bar{\boldsymbol{x}}) = 0$ and (4.3) simplifies to

$$\begin{aligned} & \boldsymbol{c}^T\boldsymbol{x}^k + {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k) + \frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) \leq \boldsymbol{c}^T\bar{\boldsymbol{x}} \\ \Longrightarrow \quad & \frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) \leq \boldsymbol{c}^T\bar{\boldsymbol{x}} - \boldsymbol{c}^T\boldsymbol{x}^k - {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k) \\ \Longrightarrow \quad & \boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) \leq 2\mu^k\big(\boldsymbol{c}^T\bar{\boldsymbol{x}} - \boldsymbol{c}^T\boldsymbol{x}^k - {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k)\big). \end{aligned} \tag{4.4}$$

At the end of the previous iteration of the loop in Algorithm 2, one of the two parameters μ and $\boldsymbol{\lambda}$ was updated. If during the previous iteration the update was of $\boldsymbol{\lambda}$, then

$$\boldsymbol{\lambda}^k = \mu^{k-1}\boldsymbol{r}(\boldsymbol{x}^{k-1}).$$

Alternatively, during the previous iteration, μ was updated and $\boldsymbol{\lambda}$ remained unchanged, so $\boldsymbol{\lambda}^k = \boldsymbol{\lambda}^{k-1}$. Consider iterations $k - W, \ldots, k - 1$ of the loop and let p be the index of the latest iteration when $\boldsymbol{\lambda}$ was changed. Then for some p satisfying $k - W \leq p < k$,

$$\boldsymbol{\lambda}^k = \mu^p\boldsymbol{r}(\boldsymbol{x}^p).$$

Suppose $\boldsymbol{x}^*$ is a limit point of $\{\boldsymbol{x}^k\}$, so that there is an infinite subsequence $\mathcal{K}$ such that

$$\lim_{k\in\mathcal{K}} \boldsymbol{x}^k = \boldsymbol{x}^*.$$

Taking the limit in inequality (4.4) gives

$$\lim_{k\in\mathcal{K}} \boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) \leq \lim_{k\in\mathcal{K}} 2\mu^k\big(\boldsymbol{c}^T\bar{\boldsymbol{x}} - \boldsymbol{c}^T\boldsymbol{x}^k - {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k)\big). \tag{4.5}$$

For all $k > W$ there is an index p with $k - W \leq p < k$ and $\boldsymbol{\lambda}^k = \mu^p\boldsymbol{r}(\boldsymbol{x}^p)$, so the value of $\boldsymbol{\lambda}^k$

can be substituted in (4.5) to give

$$\begin{aligned}
\lim_{k\in\mathcal{K}} \boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) &\le \lim_{k\in\mathcal{K}} 2\mu^k\big(\boldsymbol{c}^T\bar{\boldsymbol{x}} - \boldsymbol{c}^T\boldsymbol{x}^k - {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k)\big)\\
\implies \qquad \boldsymbol{r}(\boldsymbol{x}^*)^T\boldsymbol{r}(\boldsymbol{x}^*) &= \lim_{k\in\mathcal{K}} 2\mu^k\big(\boldsymbol{c}^T\bar{\boldsymbol{x}} - \boldsymbol{c}^T\boldsymbol{x}^k - \mu^p\boldsymbol{r}(\boldsymbol{x}^p)^T\boldsymbol{r}(\boldsymbol{x}^k)\big)\\
\implies \qquad \|\boldsymbol{r}(\boldsymbol{x}^*)\|^2 &= \lim_{k\in\mathcal{K}} 2\mu^k(\boldsymbol{c}^T\bar{\boldsymbol{x}} - \boldsymbol{c}^T\boldsymbol{x}^k) - \lim_{k\in\mathcal{K}} 2\mu^k\mu^p\boldsymbol{r}(\boldsymbol{x}^p)^T\boldsymbol{r}(\boldsymbol{x}^k) = 0,
\end{aligned}$$

since $\{\mu^k\} \to 0$ for $k \in \mathcal{K}$, so $A\boldsymbol{x}^* = \boldsymbol{b}$. For each $\boldsymbol{x}^k$, $\boldsymbol{x}^k \ge \boldsymbol{0}$, so after taking the limit, $\boldsymbol{x}^* \ge \boldsymbol{0}$. Thus, $\boldsymbol{x}^*$ is feasible for (2.2). To show optimality of $\boldsymbol{r}(\boldsymbol{x}^*)$, from (4.3)

$$\begin{aligned}
\boldsymbol{c}^T\boldsymbol{x}^k + {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k) + \frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) &\le \boldsymbol{c}^T\bar{\boldsymbol{x}}\\
\implies \qquad \lim_{k\in\mathcal{K}}\big(\boldsymbol{c}^T\boldsymbol{x}^k + {\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k) + \frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k)\big) &\le \lim_{k\in\mathcal{K}}\boldsymbol{c}^T\bar{\boldsymbol{x}}\\
\implies \qquad \boldsymbol{c}^T\boldsymbol{x}^* + \lim_{k\in\mathcal{K}}{\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k) + \lim_{k\in\mathcal{K}}\frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) &\le \boldsymbol{c}^T\bar{\boldsymbol{x}}. \qquad (4.6)
\end{aligned}$$

For all $k > W$ there is an index p with $k - W \le p < k$ and $\boldsymbol{\lambda}^k = \mu^p\boldsymbol{r}(\boldsymbol{x}^p)$ such that

$$\lim_{k\in\mathcal{K}}{\boldsymbol{\lambda}^k}^T\boldsymbol{r}(\boldsymbol{x}^k) = \lim_{k\in\mathcal{K}}\mu^p\boldsymbol{r}(\boldsymbol{x}^p)^T\boldsymbol{r}(\boldsymbol{x}^k) = 0,$$

since $\{\mu^k\} \to 0$ for $k = 1, 2, \ldots$ and $p \to \infty$ as $k \to \infty$. This value can be substituted in (4.6) to give

$$\boldsymbol{c}^T\boldsymbol{x}^* + \lim_{k\in\mathcal{K}}\frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) \le \boldsymbol{c}^T\bar{\boldsymbol{x}}.$$

For each k, $\mu^k > 0$ and $\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}) \ge 0$ for each $\boldsymbol{x}$, so that

$$\frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) \ge 0 \quad \forall k \implies \boldsymbol{c}^T\boldsymbol{x}^* \le \boldsymbol{c}^T\boldsymbol{x}^* + \lim_{k\in\mathcal{K}}\frac{1}{2\mu^k}\boldsymbol{r}(\boldsymbol{x}^k)^T\boldsymbol{r}(\boldsymbol{x}^k) \le \boldsymbol{c}^T\bar{\boldsymbol{x}}.$$

Consequently, $\boldsymbol{x}^*$ is feasible for (2.2) and has an objective value less than or equal to the optimal value $\boldsymbol{c}^T\bar{\boldsymbol{x}}$, so $\boldsymbol{x}^*$ is a solution of (2.2). □

4.2 Fast approximate solution of LP problems

Although Theorem 1 establishes an important "best case" result for the behaviour of ICA, the results in Table 4.3 show that this is far from being representative of its practical performance. For some problems ICA yields a near-optimal point; for others it terminates at a point that is far from being feasible. Which problem characteristics might explain this behaviour and, if it is seen to perform well for a whole class of problems, to what extent is this of further value?

4.2.1 Problem characteristics affecting the performance of the Idiot crash

There is a clear relation between the condition number of the matrix A and the solution error of the point returned by ICA. The condition number of a rectangular matrix is defined as the ration of the largest and smallest singular value. Of the problems in Table 4.3, all but STORM_1000 are sufficiently small for the condition of A (after the `Clp` presolve) to be computed with the resources available to the authors. These values are plotted against the solution error in Figure 4.2, where the solution error is the product of the residual and (relative) objective error introduced in Section 4.1.2. Figure 4.2 clearly shows that the problems solved accurately have low condition number. Notable amongst these are the QAPs which, with the exception of MAROS-R7, have very much the smallest condition numbers of the 29 problems in Table 4.3 for which condition numbers could be computed.

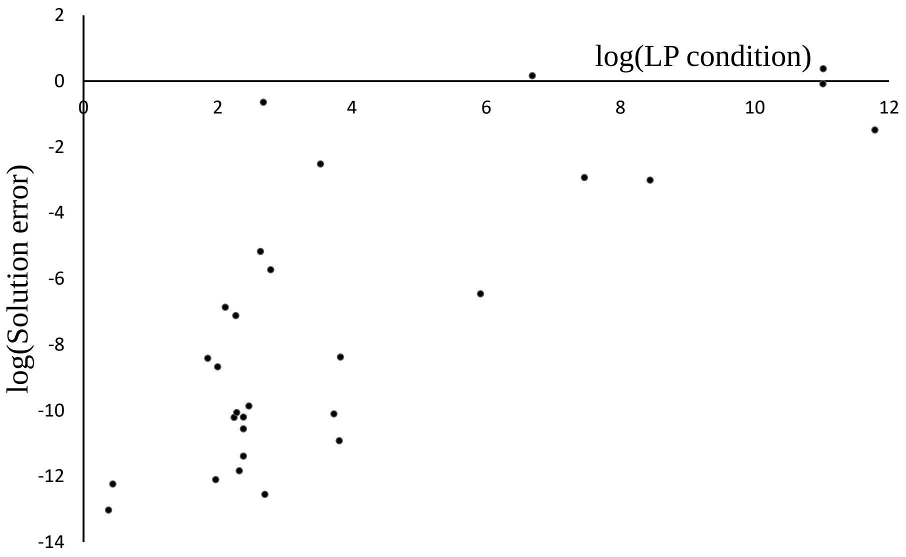

Figure 4.2: Solution error and LP condition.

Nocedal and Wright [32, p.512] observe that “there has been a resurgence of interest in penalty methods, in part because of their ability to handle degenerate problems”. However, analysis of optimal basic solutions of the problems in Table 4.3 showed no meaningful correlation between their primal or dual degeneracy and accuracy of the point returned by ICA.

4.2.2 The Idiot crash on QAPs

Since ICA yields a near-optimal point for the three QAPs in Table 4.3, it is of interest to know the extent to which this behaviour is typical of the whole class of such problems, and its practical value. Both of these issues are explored in this section.

Quadratic assignment problems

The quadratic assignment problem (QAP) is a combinatorial optimization problem, being a special case of the facility location problem. It concerns a set of facilities and a set of locations. For each pair of locations there is a distance, and for each pair of facilities there is a weight or flow specified, for instance the number of items transported between the two facilities. The problem is to assign all facilities to different locations so that the sum of the distances multiplied by the corresponding flows is minimized. QAPs are well known for being very difficult to solve, even for small instances. They are NP-hard and the travelling salesman problem can be seen as a special case. Often, rather than the quadratic problem itself, an equivalent linearization is solved. A comprehensive survey of QAP problems and their solution is given by Loiola et al. [33].

The test problems NUG15, QAP12 and QAP15 referred to above are examples of the Adams and Johnson linearization [2]. Although there are many specialized techniques for solving QAP problems, and alternative linearizations, the popular Adams and Johnson linearization is known to be hard to solve using the simplex method or interior point methods [46]. Table 4.4 gives various performance measures for ICA when applied to the Nugent [43] problems, using the default iteration limit of `Clp`. The first of these is the value of the residual $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$ at the point obtained by ICA, which is clearly feasible to within the `Clp` simplex tolerance. The objective function value and relative error are also given, and the latter is well within 1%. Finally, the time for ICA is given. Whilst this is growing, ICA clearly obtains a near-optimal solution for QAP instances NUG20 and NUG30, which cannot be solved with commercial

Table 4.4: Performance of the Idiot crash on QAP linearizations.

Model	Rows	Columns	Optimum	Residual	Objective	Error	Time
NUG05	210	225	50.00	9.4×10^{-9}	50.01	1.5×10^{-4}	0.04
NUG06	372	486	86.00	7.8×10^{-9}	86.01	1.2×10^{-4}	0.11
NUG07	602	931	148.00	7.9×10^{-9}	148.64	4.3×10^{-3}	0.25
NUG08	912	1613	203.50	7.0×10^{-9}	204.41	4.5×10^{-3}	0.47
NUG12	3192	8856	522.89	8.8×10^{-9}	523.86	1.8×10^{-3}	2.58
NUG15	6330	22275	1041.00	8.9×10^{-9}	1041.38	3.7×10^{-4}	5.13
NUG20	15240	72600	2182.00	7.5×10^{-9}	2183.03	4.7×10^{-4}	14.94
NUG30	52260	379350	4805.00	1.1×10^{-8}	4811.41	1.3×10^{-3}	82.28

simplex or interior point implementations on the machine used for ICA experiments because of excessive time or memory requirements.

There is currently no practical measure of the point obtained by ICA that gives any guarantee it can be taken as a near-optimal solution of the problem. The result of Theorem 1 cannot be used because the major iteration minimization is approximate, and the major iterations are terminated rather than being performed to the limit. Clearly the measure of objective error in Table 4.4 requires knowledge of the optimal objective function value. What can be guaranteed, however, is that since the point returned is feasible, the corresponding objective value is an upper bound on the optimal objective function value. With the aim of identifying an interval containing the optimal objective function value, ICA was applied to the dual of the linearization. Although it obtained points that were feasible for the dual problems to within the `Clp` simplex tolerance, the objective values were far from being optimal, so the lower bounds thus obtained were too weak to be of value.

4.3 ICA on the dual problem

If the ICA terminates with a primal feasible solution, an upper bound f_u on the optimal objective is obtained. If ICA is applied to the dual problem, and it terminates with a feasible solution, a lower bound f_l on the optimal objective is acquired. Hence, an interval $[f_l, f_u]$ is obtained, and the optimal objective value can be proved to belong to that interval.

4.3.1 Crash as an approximate solver

Experiments were performed, with the aim of obtaining a tight interval $[f_l, f_u]$ as a measure of the optimality of the primal solution obtained by ICA. Further work could be promising as the initial experiments successfully yielded a lower bound. However, the bound was not tight enough for practical applications of the idea outlined above.

4.4 Levels of approximation

One of the main limitations of the ICA algorithm is the accuracy of the component-wise minimization. While it performs well on some problems, performance is not always so satisfactory on general LP test instances. One way to improve the accuracy of the minimization is to select a subset of components at each iteration to minimize over. We can consider this a generalization of the ICA. As presented in the previous section, ICA can be seen as a special case, where the size of the subset over which we minimize at each iteration is of size 1. Another special case is when the size of the subset is equal to n, the number of columns in the problem. This case is equivalent to solving the subproblem exactly.

Experiments were performed with exact solution of the subproblem. Not surprisingly, performance was prohibitively slow even with a dedicated bound constrained QP solver.

Minimizing over a smaller subset of size k may prove to be beneficial, especially if block

Table 4.5: Solution times for QAPs.

	Time	iCrash &	
		Crossover &	
Model	iCrash	Primal Simplex	Dual Simplex
QAP08	0.0587	0.17	0.48
QAP09	0.0778	0.41	4.41
QAP10	0.124	1.85	11.62
QAP11	0.187	4.87	35.28
QAP12	0.257	9.40	123.28
QAP13	0.371	39.60	1055.72
QAP14	0.545	89.41	267.17
QAP15	0.662	150.72	566.69
QAP16	0.887	435.05	940.35

structure in the constraint matrix could be exploited. Several algorithms exist, to identify underlying block structure in LPs and permute the columns correspondingly, some given in Section 2.4.1.

4.5 iCrash and crossover

Implemented in `HiGHS`, iCrash is a sequential quadratic minimization algorithm with approximate subproblem solution. It aims to reach a feasible point, close to optimality, very fast. It does not provide dual values or a basic solution, so a crossover is required to start simplex. There is a crossover in `HiGHS`, originating from the interior point solver, and a linking of the LP solution steps allowed for a significant improvement in performance on some problems. Results for QAPs are given in Table 4.5.

Chapter 5

HiGHS

The development of HiGHS can be traced back to the autumn of 2016, when I combined Huangfu's serial dual simplex solver [31] with my prototype presolve implementation. This also enabled Hall to reimplement the Maros and Mitra [36] crash used in his legacy FORTRAN primal simplex solver EMSOL, which had no presolve. This prototype solver was benchmarked against EMSOL on Format International's "Multimix" feed formulation test LP problem DCP2. The effect of presolve is seen in Table 5.1, and the influence of presolve on the performance of hsol relative to EMSOL is seen in Table 5.2.

Model	Rows	Columns	Nonzeros
Raw	32388	21087	559390
Presolved	13944	19371	276288

Table 5.1: Effect of presolve on DCP2.

			Speedup	
Solver	Presolve	Solution time	Due to presolve	Relative to EMSOL
hsol	Off	11.13		2.31
	On	3.06	3.64	8.83
crash+hsol	Off	5.28		4.62
	On	2.68	1.97	9.10

Table 5.2: Influence of presolve on the performance of hsol relative to EMSOL.

Format International Ltd. had been using EMSOL as the LP solver in their software for almost twenty years, and were very impressed by the prospect of a new LP solver offering an order of magnitude improvement in performance. A three-year £46k consultancy contract was signed to fund the development of an industrial strength presolve and rudimentary MIP solver. This work reinvigorated the relationship between the University and Format, leading to one of five Impact Case Studies submitted by the School of Mathematics to the 2021 Research Excellence Framework (REF). This cited the value to Cargill (who had bought Format in the interim) of the optimization software developed for them. This software blends half the world's manufactured animal food, worth approximately $50Bn per annum.

The transformation of a computer program, written and used by a single person, into a well maintained product, used by multiple people, is a non-trivial task. Avoiding mistakes early on can be of great significance for the future success of a project.

5.1 Design and architecture

Frederic Brooks, who is best known for managing the development of IBM's System/360 family of computers and the OS/360 software support package, has authored a book called The Mythical Man Month [9], where many important concepts are noted. To a large extent, `HiGHS` was engineered respecting these guidelines.

Given a program, several further components are needed for a scalable, usable product. A Program here is considered to be the code with some desired functionality. Brooks defines a Programming Product as the code, along with any tools for testing the code, user and development documentation and build infrastructure. A Systems Program is a program along with any interfaces and links to integrate the program within other programs and systems.

Combining these two concepts, a Systems Programming Product is the code, along with any tools for testing the code, documentation and build infrastructure as well as any interfaces and links to integrate the program within other programs and systems.

In particular, for `HiGHS`, the transformation of what was previously known as `hsol` into a maintainable user product could be defined by the following components:

- Program
- Product
  - Generalization, Documentation, Testing
- System
  - Build, Platforms, Linking, Interfaces

Brooks estimates that transforming a Program into a Product takes 3 times the amount of resources, in terms of man-hours. A System Program also takes around 3 times the amount of resources. A Programming System Product takes 9 times the amount of resources, when compared to the original Program.

A Note on Design

One aspect emphasized in the book is the time spent on different components of the development. Rather than code close to all of the time, a different approach is suggested:

$$1/3 \text{ design}, 1/6 \text{ code}, 1/4 \text{ component test}, 1/4 \text{ system test}.$$

The reasoning is that the best way to fix issues is to avoid as many as possible from happening in the first place. The second fraction can be seen as getting the parts of the code to work, and the final fraction can be seen as getting the parts to work together.

Conceptual Integrity

One of the main ideas in the book is that of Conceptual Integrity. As new features are added to the code, it is key to maintain a unified structure in order to avoid future inconsistencies. The execution of the program must be well-defined. It is essential to avoid duplication of functionality or broken or outdated code facing the user, or in the code repository in the first place. The conceptual integrity idea is what inspired the design decisions in `HiGHS`.

Project Specification

The specification should be what is documented, what is presented to the user and what is being tested. The specification includes purpose, environment, domain, I/O, instructions, options, runtime options, accuracy and checking. Ideally, the project specification would be the documentation and would serve as a single point of reference.

Consequently, the preparation and design of `HiGHS` involved a process of defining and redefining the core functionality we would wish to offer to our users.

5.2 The `HiGHS` Design Process

In this section are given some details about the design process of `HiGHS`. The task was to make `HiGHS` reliable and maintainable as a software product. What was available was a C++ parallel simplex code.

What we wanted to have eventually:

- interfaces
- light weight, well documented, highly efficient solver available to a wide range of users.

The problem:

- options / information extraction, functionality modifications required by external APIs
- a significant refactoring was required.

Multiple modifications to the code were necessary, some of the details are split across the categories outlined in Section 5.1.

Product: Generalization

Starting with the product features, the generalization includes handling more problem types and incorporating multiple solvers, such as a MIP solver, QP solver and an interior point solver as an alternative to simplex for LPs.

Product: Documentation

Prior to the release, `HiGHS` documentation could be build with `Doxygen`, there was the README in the repository and also instructions on the `HiGHS` website. This setup allowed for inconsistencies between the instructions on the website and the latest version of `HiGHS`.

Robert Ramey, one of the `boost` library authors, shares great insight in his CppCon talk [45]. According to Ramey, the problem with “automated” tools such as Doxygen is that they produce the illusion of documentation rather than documentation itself. In particular, they paraphrase the code, which should be readable in the first place. Instead, documentation should aim to guide the user through the setup and usage of a software product, rather than duplicate what is already in the code base, which, in the open source case, is freely available to the general public.

Documentation should certainly include a description of what the program is supposed to do and what is it expected to be used for. What is the user API, how does one use the program, ideally with examples. Implementation details should not be included, rather found in code itself. Brooks reasonably states, that documentation preparation should be seen as an aid to building a coherent design rather than some afterthought to try and fix something that has been made overcomplicated in the first place.

The code itself constitutes documentation - which is an extremely important note, especially when determining what documentation already exists.

The public documentation for the official release of `HiGHS` was created using `Clang` and `Julia` and will be hosted at GitHub pages, so it is easily accessible for users. It is automatically generated from `.md` files, allowing developers quick edits and updates to the documentation page.

Product: Testing

Testing is necessary, in order to be able to track bugs more easily and to have reliable information about performance. As the size of the project grows, compile time increases, as well as the complexity of calls to internal data structures and the linking between them. Previously, the state of some of the interfaces and platform builds was unclear or untested.

A better design of the testing infrastructure may be time-consuming but can greatly improve the focus of developers on the algorithmic part, with fewer uncaught bugs and, consequently, fewer unpleasant surprises in the future.

Scaffolding

The concept of "scaffolding" aims to add structure to testing of software. The goal is to maintain the conceptual integrity of the code structure. Ideally, one should be able to run test and development code without modifying the current source code. Thus, it is ensured that behaviour corresponds to requirements and the documentation is consistent and reflects that.

A scaffold would contain:

- all programs and data build for debugging purposes but never intended to be in the final product,
- generators for test data,
- special analysis printouts.

In addition to unit tests, more problem instance tests were added to the `CMake` tests. Continuous integration is essential for any piece of software, and we developed several scripts automated by GitHub Actions.

System: Build, Platforms

The `CMake` code used to build the `HiGHS` C++ code had to be updated in order to ensure the success of building and installing the `HiGHS` library and executable on different platforms. Additionally, bits of system dependent code had to be removed from `HiGHS` or handled properly. Windows, Linux and macOS are all supported now.

Dependencies

Previous solvers have relied on extensive external dependencies to allow generating executable and library code on a particular platform. This motivated the decision to use `CMake` initially. Apart from `CMake` and a standard threads library for the parallel code, `HiGHS` is entirely independent of external libraries. This is very beneficial for a wide range of users and practical applications.

System: Interfaces

Currently, there are multiple interfaces allowing `HiGHS` to be used from different programming languages such as `Python` and Julia. At the moment not all interfaces cover all features of `HiGHS`. The goal is to develop them further as the user base continues to grow.

Programming languages differ in the extent to which they can communicate with other programming languages. For some it is possible to call C++ directly, others require to call a C wrapper instead. Multiple interfaces introduce a possibility for inconsistencies. For the design of the `Highs` class and the C API it was crucial to observe and remove such possibilities from the beginning, in order to avoid the necessity for a large restructuring of the code later on. A decision was made on a single external API corresponding to the C++ Highs class and C API.

Extended work on the solver was required, in order to support increased functionality, required by some interfaces.

5.2.1 The Product & Resources

As a result of the development process, for the official release of HiGHS, the user would have access to the following resources, and would be able to build the targets using CMake.

- website [highs.dev] [27],
- github repository [28] | master | latest version | tests passing,
- documentation site (GH Pages); README.md.

Targets:

- executable,
- shared library (optional static),
- interfaces.

A thorough and disciplined approach to software engineering pays off for both developers and users alike. Unit tests and continuous integration mean that developers know the consequences of code changes, and can be confident that the software will run on architectures other than the one used to develop it. It is also clear that, for users large and small, the value of having open-source software build easily, reliably, and without external dependencies is very much appreciated. It builds confidence in the software, and has been instrumental in the uptake of HiGHS by users. For SciPy [50] and JuMP [15], the two most notable applications that have embraced HiGHS, the professionalism of the build and support were very welcome. Indeed, ease of build is a major factor in the successful take-up of linear optimization software such as GLPK [34] and lp_solve [7], despite their poor performance [41]. Solvers with vastly better performance such as Clp [11] and Cbc [10] have, historically, been hampered by the cumbersome build systems resulting from their integration within COIN-OR [12].

Five years after the prototype presolve and Huangfu's dual simplex solver were combined for an industrial application in relation with Format International, HiGHS now exists as open-source linear optimization software with the best overall performance in the world [41]. Developing the industrial strength implementation of presolve and postsolve described in Chapter 3 was a key technical contribution to HiGHS, and led to consultancy work that was critical to funding the development of the MIP solver.

HiGHS is the only open-source software to offer simplex and barrier for LP, a MIP solver and a QP solver, and in most of these categories, HiGHS is the best amongst all open-source software in Mittelmann's benchmarks [39, 40, 42]. However, performance without users is not sufficient for a software project to be successful. The thorough and disciplined approach to software engineering in HiGHS, as described in Chapter 5, has been instrumental in attracting users. These range from the authors of high-profile interfaces such SciPy [50] and JuMP [15], where the presence of HiGHS represents a vote of confidence in it as software, to individual developers who welcome the ease with which they have been able to switch to HiGHS from alternative solvers such as GLPK [34], lp_solve [7] and Cbc [10].

Looking forward, the future looks very bright for HiGHS. The three key members of the team are employed by the School of Mathematics and there is an ever-growing base of users. With the expansion of the code base, scaling becomes crucial to the success of most software products and computational mathematical software is no exception to this rule. Nevertheless, with advanced software engineering and efficient implementation of algorithms, many practical problems can be solved very fast.

Chapter 6

Conclusions

The solution of linear programming problems is required in many industry areas and academic projects. The development of presolve and crash techniques for LP is essential, allowing larger problems to be handled by the solvers, and solution time to be accelerated. Preprocessing techniques are required in most practical applications of significant or even medium scale. Many preprocessing techniques are developed and used commercially and it is also essential to take them into consideration within the scientific academic community.

This thesis presents a novel analysis of a quadratic penalty type algorithm, known to perform very well on certain classes of optimization problems. The crash was previously of interest to both the academic community and industrial OR developers. Presolve for LP is examined in detail, with an additional focus on dual postsolve, some basic rules previously not presented, and an illustration of the significant reduction of overall solution time.

Software design and scaling is another aspect crucial to the usage of solvers within various applications. Poor design at any stage of development can lead to numerous problems in the future and may eventually require large segments of the code to be completely rewritten. That often takes a significant amount of resources and is a major reason for many software projects to struggle with performance and maintenance. Integrating solvers and preprocessing techniques along with utilities, allowing the users to interact with the program, requires thoughtful consideration of use cases and potential issues. This thesis additionally presents an examination of the design and engineering of HiGHS, with respect to conceptual integrity.

The optimization software HiGHS, created with careful design and integration of preprocessing algorithms, is now providing users worldwide with several of the best performing solvers within the open source community.

Bibliography

[1] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

[2] Warren P Adams and Terri A Johnson. Improved linear programming-based lower bounds for the quadratic assignment problem. *DIMACS series in discrete mathematics and theoretical computer science*, 16:43–77, 1994.

[3] Erling D Andersen and Knud D Andersen. Presolving in linear programming. *Mathematical Programming*, 71(2):221–245, 1995.

[4] David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. *Advances in Neural Information Processing Systems*, 34, 2021.

[5] David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *arXiv preprint arXiv:2105.12715*, 2021.

[6] Cevdet Aykanat, Ali Pinar, and Ümit V Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on scientific computing*, 25(6):1860–1879, 2004.

[7] Michel Berkelaar. Linear programming solver. `https://sourceforge.net/projects/lpsolve/`. Accessed: 11/11/2021.

[8] Robert E Bixby. Implementing the simplex method: The initial basis. *ORSA Journal on Computing*, 4(3):267–284, 1992.

[9] Frederick P Brooks. The mythical man-month. *Datamation*, 20(12):44–52, 1974.

[10] COIN-OR. Cbc. `https://github.com/coin-or/Cbc`. Accessed: 09/11/2021.

[11] COIN-OR. Clp. `https://github.com/coin-or/Clp`. Accessed: 09/11/2021.

[12] COIN-OR. Computational Infrastructure for Operations Research. `https://www.coin-or.org/`. Accessed: 11/11/2021.

[13] AR Conn, NIM Gould, and Ph L Toint LANCELOT. A fortran package for large-scale nonlinear optimization (release a). *Springer Ser. Comput. Math*, 17, 1992.

[14] Timothy A Davis and William W Hager. A sparse proximal implementation of the lp dual active set algorithm. *Mathematical programming*, 112(2):275–301, 2008.

[15] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM review*, 59(2):295–320, 2017.

[16] Yu G Evtushenko, AI Golikov, and N Mollaverdy. Augmented lagrangian method for large-scale linear programming problems. *Optimization Methods and Software*, 20(4-5):515–524, 2005.

[17] Michael C Ferris and Jeffrey D Horn. Partitioning mathematical programs for parallel solution. *Mathematical programming*, 80(1):35–61, 1998.

[18] Ivet L Galabova and Julian AJ Hall. The 'idiot'crash quadratic penalty algorithm for linear programming and its application to linearizations of quadratic assignment problems. *Optimization Methods and Software*, 35(3):488–501, 2020.

[19] Gerald Gamrath, Thorsten Koch, Alexander Martin, Matthias Miltenberger, and Dieter Weninger. Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7(4):367–398, 2015.

[20] David M Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.

[21] Jacek Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, 9(1):73–91, 1997.

[22] Google. Or-tools. `https://developers.google.com/optimization`. Accessed: 11/11/2021.

[23] Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension reduction via colour refinement. In *European Symposium on Algorithms*, pages 505–516. Springer, 2014.

[24] Osman Güler. Augmented lagrangian algorithms for linear programming. *Journal of optimization theory and applications*, 75(3):445–470, 1992.

[25] William W Hager. The lp dual active set algorithm. In *High Performance Algorithms and Software in Nonlinear Optimization*, pages 243–254. Springer, 1998.

[26] William W Hager. The dual active set algorithm and its application to linear programming. *Computational Optimization and Applications*, 21(3):263–275, 2002.

[27] Julian AJ Hall. Highs. Website: `www.highs.dev`, 2019. Accessed: 17/02/2022.

[28] Julian AJ et al. Hall. Highs github repository. Website: `https://github.com/ERGO-Code/HiGHS`, 2022. Accessed: 17/02/2022.

[29] Magnus R Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969.

[30] Qi Huangfu. *High performance simplex solver*. PhD thesis, University of Edinburgh, 2013.

[31] Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

[32] Nocedal Jorge and J Wright Stephen. Numerical optimization, 2006.

[33] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007.

[34] A. O. Makhorin. GNU linear programming kit. `https://www.gnu.org/software/glpk/`. Accessed: 11/11/2021.

[35] István Maros and Gautam Mitra. Finding better starting bases for the simplex method. *Operations Research Proceedings 1995*, pages 7–12, 1995.

[36] István Maros and Gautam Mitra. Strategies for creating advanced bases for large-scale linear programming problems. *INFORMS Journal on Computing*, 10(2):248–260, 1998.

[37] Cs Mészáros and Uwe H Suhl. Advanced preprocessing techniques for linear and quadratic programming. *OR Spectrum*, 25(4):575–595, 2003.

[38] Hans D Mittelmann. Benchmarks for optimization software. `http://plato.la.asu.edu/bench.html`, 2018. Accessed: 16/02/2018.

[39] Hans D Mittelmann. Benchmark of barrier lp solvers. `http://plato.asu.edu/ftp/lpbar.html`, 2022. Accessed: 15/02/2022.

[40] Hans D Mittelmann. Benchmark of simplex lp solvers. `http://plato.asu.edu/ftp/lpsimp.html`, 2022. Accessed: 15/02/2022.

[41] Hans D Mittelmann. Benchmarks for optimization software. `http://plato.asu.edu/bench.html`, 2022. Accessed: 15/02/2022.

[42] Hans D Mittelmann. The miplib2017 benchmark instances. `http://plato.asu.edu/ftp/milp.html`, 2022. Accessed: 15/02/2022.

[43] Christopher E Nugent, Thomas E Vollmann, and John Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16(1):150–173, 1968.

[44] Michael JD Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*, pages 283–298, London, 1969. Academic Press.

[45] Robert Ramey. How to write effective documentation for c++ libraries with minimal effort. CppCon, 2017.

[46] Mauricio GC Resende, KG Ramakrishnan, and Zvi Drezner. Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. *Operations Research*, 43(5):781–791, 1995.

[47] R Tyrrell Rockafellar. Augmented Lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12(2):268–285, 1974.

[48] Judah Ben Rosen and Robert S Maier. Parallel solution of large-scale, block-angular linear programs. *Annals of Operations Research*, 22(1):23–41, 1990.

[49] Edmund Smith. *Parallel solution of linear programs.* PhD thesis, The University of Edinburgh, 2013.

[50] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[51] Ian En-Hsu Yen, Kai Zhong, Cho-Jui Hsieh, Pradeep K Ravikumar, and Inderjit S Dhillon. Sparse linear programming via primal and dual augmented coordinate descent. *Advances in Neural Information Processing Systems*, 28, 2015.