Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

Master Thesis

Linear programming on Cell/BE

by

Åsmund Eldhuset

Supervisor: Dr.Ing. Lasse Natvig
Co-supervisor: Dr. Anne C. Elster

Trondheim, June 1, 2009

Abstract

(TODO)

Acknowledgements

(TODO)

Contents

List of Figures

List of Tables

Listings

List of Symbols and Abbreviations

Abbreviation	Description	Definition
LP	Linear programming	page 3

Chapter 1

Introduction

(TODO)

Chapter 2

Background

(TODO)

Chapter introduction

2.1 Linear programming

(Natvig)

Do we need section introductions too?

2.1.1 Problem formulation. Standard and slack forms

The term *linear programming* (LP) refers to a type of optimisation problems in which one seeks to maximise or minimise the value of a linear function of a set of variables that are constrained by a set of linear equations and/or inequalities[1].

Linear programming is a fairly general problem type, and many important problems (TODO) can be cast as LP problems — for instance, network flow problems and shortest path problems (see [?]).

(other than those problems that are initially formulated as an LP problem)

Throughout this report, we will consistently use n to refer to the number of variables and m to refer to the number of inequalities. The variables will typically be (TODO: spell "label(l)ed") x_1 through x_n.

The function to be optimised is called the *objective function*. (TODO) However, since this term (TODO), we drop it from the objective function, which can then be written as $f = c_1x_1 + c_2x_2 + \ldots + c_nx_n = \sum_{j=1}^{n} c_jx_j$, where c_j are the coefficient values.

In the real world situation that gives rise to an optimisation problem, the function may contain a constant term.

(TODO)

The equations and inequalities that (together with the objective function) constitute an LP problem may be represented in different forms. We shall first consider the *standard form*, in which only less-than-or-equal-to inequalities with all variables on the left hand side are allowed. (TODO) A problem containing an equalities of the form $a_1x_1+\ldots+a_nx_n = b$ (Natvig) may be rewritten by splitting

Nonnegativity of variables, which is often the case in real world prolems.

Why are not less-than allowed?

Should I label the coefficients $a_{i1}, \ldots, a_{in}$ instead to maintain consistency with the standard/slack forms?

[1]Hence, LP is not (as the name would seem to suggest) a programming technique.

each equality into two inequalities: $a_1x_1+\ldots+a_nx_n \leq b$ and $-a_1x_1-\ldots-a_nx_n \leq -b$. Also, the goal must be to maximise the objective function (if the original problem is to minimize f, we let our objective function be $-f$). A linear program in standard form can be expressed as follows: (TODO)

How to indent?

Maximise

$$f = \sum_{j=1}^{n} c_j x_j$$

with respect to

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad \text{for } i = 1, \ldots, m.$$

The other common representation, which is employed by the simplex algorithm (to be presented shortly), is *slack form*, which only allows a set of equations (and a nonnegativity constraint for each variable). An inequality of the form $a_1x_1 + \ldots + a_nx_n \leq b$ is converted to an equation (TODO) by adding a *slack variable* w. Together with the condition that $w \geq 0$, the equation $a_1x_1 + \ldots + a_nx_n + w = b$ is equivalent to the original inequality (whose difference, or "slack", between the left and right hand sides is represented by w).

equation or equality?

A proposed solution of a linear program (that is, a specification of a value for each variable) is called:

Feasible if it does not violate any of the constraints

Infeasible if it violates any constraint

Basic (TODO)

Optimal if it is feasible and no other feasible solutions yield a higher value for the objective function

(TODO)

The ??? theorem (TODO: citation)) states that the optimal solution of a linear program, if it exists, occurs when n variables are set to zero and the n others are nonzero. CHECK first?

2.1.2 Simplex algorithm

The *simplex algorithm* was the (TODO) systematic algorithm developed for solving linear programs. It requires the program to be in slack form. (TODO) The nonnegativity constraints are not represented explicitly anywhere. (TODO)

The variables in the leftmost column are referred to as the *basic variables*, and the variables inside the tableau are called *nonbasic variables*. It should be noted that the slack form must have been created from a standard form, because this ensures that there are n slack variables, where each slack variable occurs in excactly one equation.

For now, let us assume that the solution that is obtained by setting all nonbasic variables to zero is feasible. This solution will provide a lower bound for the value of the objective function (namely, the constant term). We will now select one nonbasic variable x_j and consider what happens if we increase its value (since all nonbasic variables are currently zero, we cannot decrease any of them). Since our goal is to maximise the objective function, we should select a variable whose coefficient c_j in the objective function is positive. If no such variables exist, we cannot increase the objective function value further, and the current solution is optimal (we can be certain of this since linear functions do not have local maxima). How far can we increase this variable? Recall that each line in the tableau expresses one basic variable as a function of all the nonbasic variables; hence we can increase x_j until one of the basic variables becomes zero. Let us look at line i. If a_{ij} is positive, we can increase x_j indefinitely without w_i ever becoming negative, and in that case, we have determined the problem to be *unbounded*. If $a_{ij} = 0$, this equation is not affected at all by any change in x_j, and the problem (TODO) is said to be *(TODO)*. If a_{ij} is negative, the value of w_i will decrease as x_j increases, so the largest allowable increase is limited by the current value of w_i — which is b_i, since all nonbasic variables initially are zero. Thus, by setting $x_j = -\frac{b_i}{a_{ij}}$, w_i becomes zero. (TODO)

or tableau?

limited by lowest value

The variable selected is called the *entering variable*, since it is about to enter the collection of basic variables. We also need a *leaving variable* to be removed from said collection. (TODO) We can eliminate the entering variable from (and introduce the leaving variable into) the set of *nonbasic* variables (the "main" part of the tableau) by rewriting the selected equation and adding appropriate multiples of it to each of the other equations:

how to find it?

The algorithm presented so far is capable of solving linear programs whose initial basic solution (the one obtained by setting all nonbasic variables to 0) is feasible. (TODO) This may not always be the case. We get around this by introducing an *auxiliary problem* which will

Phase I and Phase II

Example

We will now demonstrate one iteration of the simplex algorithm, on the following problem: (TODO)

2.1.3 Interior point algorithms

2.1.4 Use of LP to solve advanced flow problems

2.2 Cell Broadband Engine

2.2.1 Architecture

2.2.2 Programming methods

Chapter 3

Design

(TODO)

Chapter 4

Implementation and testing

(TODO)

Chapter 5

Evaluation

(TODO)

Chapter 6

Conclusion

(TODO)

Future work

Bibliography

Appendices

Appendix A

Schedule

This appendix will obviously be deleted before submission.

Week 8 Finish the implementation of a dense Simplex for a regular CPU and test with `netlib` datasets. Implement a vectorised (SIMD) dense Simplex on the PPE

Week 9 Implement a vectorised dense Simplex running in parallel on the SPEs

Week 10 Study interior point algorithms

Week 11 Implement a dense, non-parallelised interior point algorithm

Week 12 Decide on whether to pursue simplex or interior point. Making a test plan. Experiment with different approaches to sparse storage; look into numerical stability with single-precision values

Week 13 — " —

Week 14 First draft of report

Week 15 Easter vacation

Week 16 Look into autotuning?

Week 17

Week 18

Week 19

Week 20 Performance measurements and graphing

Week 21 Frenetic report writing

Week 22 — " —

Week 23 Ordinary submission deadline. Will try to submit as close to this date as possible

Week 24

Week 25

Week 26

Week 27 Natvig goes on vacation

Week 28

Week 29 Final deadline: July 19