Slicing Up Bonsai: Adding Cutting Planes to Mixed 0-1 Programs

by

Cheryl Michele Petreman

B.Sc., Simon Fraser University, 1990

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

APPROVAL

Name:	Cheryl Michele Petreman
Degree:	Master of Science
Title of thesis:	Slicing Up Bonsai: Adding Cutting Planes to Mixed 0-1 Programs

Examining Committee:

Chair

Dr. L. Hafer
Senior Supervisor
Associate Professor of Computing Science

Dr. R. Krishnamurti
Assistant Professor of Computing Science

Dr. P. Hell
External Examiner
Professor of Computing Science

Date Approved: Aug 3/93

Abstract

We investigate a method of reducing the effort required to solve resource-constrained scheduling problems using mixed-0/1 linear programming. In particular we examine a formulation of the decision as to whether two activities of variable duration will be serialized or not. From this subset of the constraint system, we derive cutting plane constraints which are expressed in terms of variable upper and lower bounds.

We describe the integration of these cuts into *bonsai* — a system which implements branch and bound search with partial arc consistency. Each time the arc consistency routines are able to restrict variable domains, the cutting plane is tightened to reflect the tighter bound.

The computational results show a decrease in the average number of subproblems required to solve large examples. However, there is also an increase in the total number of pivots.

Acknowledgments

Thanks go to Lou for his time and enthusiasm.

Dedication

To David for putting up with me....

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The focus of this thesis is to investigate a method of reducing the effort required to solve resource-constrained scheduling problems using mixed-0/1 linear programming. In particular we examine a formulation of the decision as to whether two activities of variable duration will be serialized or not; and if so, in what order. We then describe the derivation of a set of cutting plane constraints using the lift and project method of Balas, Ceria, and Cornuéjols [1, 2]. These constraints are expressed in terms of the upper and lower bounds of the timing variables. We further describe the integration of these cuts into *bonsai* — a system which implements branch and bound search with partial arc consistency. Each time the arc consistency routines are able to restrict variable domains, the cutting plane is tightened to reflect the tighter bound.

This chapter contains some required definitions, a review of the required integer programming background, and an overview of the motivation and organization of this thesis.

1.1 Definitions

Linear Programming

This section starts with a brief review of linear programming terms. Further discussion of this material can be found in [4, 9].

A *linear constraint* is an equation or inequality having one of the following forms:

$$\begin{aligned} c_1x_1 + c_2x_2 + ... + c_nx_n &\leq b \\ c_1x_1 + c_2x_2 + ... + c_nx_n &= b \\ c_1x_1 + c_2x_2 + ... + c_nx_n &\geq b \end{aligned}$$

where

- $c_1, c_2, ..., c_n$ are real numbers,
- $x_1, x_2, ..., x_n$ are real variables, and
- b is a real number.

Linear Programming (LP) is a mathematical algorithm that facilitates the allocation of scarce resources while optimizing some objective. The mathematical formulation of an LP problem consists of a linear objective function and a set of linear constraints which together describe a real world decision situation. In *standard form* an LP problem is written as:

$$\begin{array}{ll} \textit{maximize} & \sum_{j=1}^{n} c_jx_j \\ \textit{subject to} & \sum_{j=1}^{n} a_{ij}x_j \leq b_i \quad (i = 1, 2, ..., m) \\ & \check{x}_j \leq x_j \leq \hat{x}_j \quad (j = 1, 2, ..., n) \end{array} \tag{1.1}$$

Introducing *slack* variables $x_{n+1}, x_{n+2}, ..., x_{n+m}$, $x_{n+i} \geq 0$ allows constraints of (1.1) to be rewritten as equalities.

$$\sum_{j=1}^{n} a_{ij}x_j + x_{n+i} = b_i$$

The system of linear equations can then be represented by a *dictionary* where the objective function, z, and a subset of the variables are expressed in terms of the remaining variables.

The variables appearing on the left hand side of the dictionary are termed *basic* while those on the right are *non-basic* (N).

$$\begin{aligned} x_k &= \bar{b}_i - \sum_{x_j \in N} \bar{a}_{ij} x_j \qquad (i = 1, 2, ..., m) \\ z &= z_0 + \sum_{x_j \in N} + \bar{c}_j x_j \end{aligned}$$

In a *basic optimal* solution (one representable by a dictionary), non-basic variables x_j have the value $\check{x}_j$ if $\bar{c}_j < 0$ or $\hat{x}_j$ if $\bar{c}_j > 0$.

An *Integer Linear Programming* (ILP) problem can be formulated as (1.1) plus additional requirements which force a subset of the variables to take on integer values. For pure-integer problems, all variables must have integer values in a solution. If only some of the variables in the model are required to be integral then it is called a mixed-integer linear program (MILP).

A mixed-0/1 linear programming problem is expressed using continuous and Boolean variables and can be formulated as follows:

$$\begin{array}{ll} maximize & cx + hy \\ st & Ax + Gy \leq b \quad x \in \Re^n \\ & y \in \{0,1\}^p \end{array}$$

where

- p is the number of boolean variables,
- c is the cost coefficients corresponding to the variables in x,
- h is the cost coefficients corresponding to the variables in y,
- A is the columns of the coefficient matrix corresponding to the variables in x, and
- G is the columns of the coefficient matrix corresponding to the variables in y.

We shall use the abbreviation MILP to refer to mixed-0/1 linear programs since they are the only type of mixed-integer linear programs that we are concerned with in this thesis.

The *linear relaxation* of a mixed-0/1 programming problem is the LP problem that results from relaxing the integrality constraints to allow continuous values, *i.e.* $y_j \in \{0,1\}$ is relaxed to $0 \leq y_j \leq 1$

The term *row bound* is used to refer to the bound which we can calculate for the left hand side of a constraint (using arc-consistency techniques described below).

The term *incumbent* refers to the best integral solution which has been discovered so far in a tree search strategy.

Polyhedral theory

We now provide a review of some polyhedral concepts. Details can be found in [12, 13].

A set P of points $x \in \Re^n$ is called a *polyhedron* if $P = \{x \ : \ Ax \leq b\}$ for some matrix A and vector b.

An inequality $\pi x \leq \pi_0$ (denoted (π, π_0)) is called a *valid inequality* for P if it is satisfied for all $x \in P$.

If $x \in P$ and there do not exist $x^1, x^2 \in P$, $x^1 \neq x^2$ such that $x = \frac{1}{2}x^1 + \frac{1}{2}x^2$ then x is an *extreme point* of P.

Let P be the polyhedron defined by the linear relaxation of the MILP. A point x of P is called *non-integer* if $\exists x_i$ such that x_i has been constrained in the MILP to be an integer but takes on a non-integer value at x.

Let $P^0 = \{r \in \Re^n \ : \ Ar \leq 0\}$. If $P = \{x \in \Re^n \ : \ Ax \leq b\} \neq \emptyset$, then $r \in P^0 \setminus \{0\}$ is called a *ray* of P.

If r is a ray of p and there do not exist r^1, $r^2 \in P$, $r^1 \neq \lambda r^2$ for any $\lambda \geq 0$, $\lambda \in \Re$, such that $r = \dfrac{1}{2}r^1 + \dfrac{1}{2}r^2$, then r is called an *extreme ray* of P.

The *projection* of a point $(x, y) \in \Re^n \times \Re^p$ onto the subspace $H = \{(x, y) : y = 0\}$ is the point $(x, 0)$. The projection of polyhedron $P = \{(x, y) \in \Re^n \times \Re^p \ : \ Ax + Gy \leq b\}$ from

(x, y)-space to x-space is defined as

$$proj_x(P) = \{x \in \Re^n \ : \ v^t(b - Ax) \geq 0 \ \forall t \in T\}$$

where $\{v^t\}_{t \in T}$ are the extreme rays of $Q = \{v \in \Re^m_+ \ : \ v \geq 0, \ vG = 0\}$. The projection of a polyhedron is a polyhedron.

The *polar* of P is defined to be the set of points

$$P^* = \{(\pi, \pi_0) \in \Re^{n+1} : \pi x \leq \pi_0 \ \forall x \in P\}.$$

For any polyhedron P, the *integer hull* of P is defined to be the convex hull of the integral lattice points in P. Let us then define the *mixed-integer hull* of P for a MILP M as the convex hull of the set of points x such that $x \in P$ and $x_j \in \mathcal{Z}$ if there is an integrality restriction on x_j in M.

The term *cutting plane* will refer to a linear constraint that separates a non-integer extreme point from the mixed-integer hull of the MILP. To *tighten* a cutting plane constraint, we replace it with a more effective one — a constraint which cuts off more points of the relaxed polyhedron than were cut off by the previous constraint.

We now describe a few approaches to solving mixed-0/1 programming problems. While these algorithms were developed in isolation, they are often used in combination.

Branch-and-Bound Approach

Since the possible values of Boolean variables can be enumerated, a branch-and-bound algorithm can be used to solve a MILP problem. This procedure uses a depth first tree exploration strategy where

- the linear relaxation of the root of each subtree provides an upper bound for any solution in the subtree, and
- the *incumbent* — the best integral optimal solution found to date — provides a lower bound.

The *live* set is the set of nodes which have been explored, but not fathomed off. The most promising node in the active set will be restored each time the current node is determined to have no ancestors. The current node begin explored is said to be *active.*

The following steps are performed at each node:

1. Solve the linear relaxation.
2. Examine the results.
   - If the relaxation is infeasible, then the node is fathomed (not explored further).
   - If the solution to the linear relaxation of the node is less than the incumbent, the node is fathomed.
   - Let S be the set of Boolean variables which have solution values $\notin \{0, 1\}$. If $S \neq \emptyset$ then go to step 3.
   - Otherwise an integer solution has been achieved and is compared against the current incumbent. If the solution is better, it becomes the new incumbent. The node is not explored further.
3. Select a set of branching variables $V \subseteq S$. Generate all possible $\{0, 1\}$ vectors of length $|V|$. Create a subproblem for each of these vectors by introducing constraints which force each branching variable v_i to be the appropriate $\{0, 1\}$ value while not excluding any feasible integer solutions. Each subproblem is added to the search tree as a child of the current node.
4. No problems remaining? Then the current incumbent solution is optimal. Otherwise, select an unexplored node and return to step 1.

Arc Consistency

Partial arc consistency techniques developed by Sidebottom [14] enable us to prune continuous variable domains by restricting the lower and upper bounds of the variables. In particular, the fixing of branching variables might allow us to restrict the domains of other variables. The mathematics involved are described in Section 3.2 while the logic is described below:

```
Algorithm : Propagate variable bound change

    Given that variable $x_j$ has had its bounds tightened:

    For each constraint $i$ in which $a_{i,j} \neq 0$

        Calculate new row bounds for constraint $i$.

        If the change in row bounds will affect the bounds of other variables in the
        constraint then add constraint $i$ to the propagation set.

    End For

Algorithm : Explore propagation set

    While the propagation set is non-empty

        Remove constraint $i$ from the propagation set

        For each variable $x_t$ in constraint $i$

            Calculate variable $x_t$'s upper and lower bounds.

            If the variable bounds have tightened then call propagate variable bound
            change.

        End For

    End For

Algorithm : arc consistency

    Initialize the propagation set to $\emptyset$.

    For each branching variable

        Call propagate variable bound change.

    End For

    Call Explore propagation set.
```

Incorporating partial arc consistency into the branch-and-bound algorithm, as described above, provides a significant reduction in the amount of work required to explore the search tree (Hafer [6]).

Using Cutting Planes

Another approach to solving integer programming problems involves the construction of cutting planes to cut off non-integer optimal solutions. This procedure is described as follows:

1. Solve the linear relaxation.
2. Examine the optimal solution. If all variables which are required to be integer have integer values then the optimal solution has been achieved and we can stop. Otherwise, proceed to step 3.
3. Construct a cutting plane to cut off the non-integer optimal solution. Add the cutting plane to the constraint system.
4. go to step 1

Merging cutting planes with the branch-and-bound algorithm leads to the more efficient *branch-and-cut* approach where a cutting plane is constructed after solving the linear relaxation at each node.

1.2 Thesis Overview

Bonsai implements an LP-based branch-and-bound algorithm incorporating arc consistency techniques (Hafer [6, 8]). It was developed to solve the scheduling problems that arise from digital hardware synthesis[1]. A fundamental decision arising in the formulation of these problems is whether or not two activities will be serialized; and if so, in what order. This decision is formulated using the following model which takes into account the variable duration of these types of activities:

$$\begin{array}{rcrcrcl} -T_s(x_2) & + & T_r(x_1) & - & \hat{T}\alpha_{1,2} & \leq & 0 \\ -T_s(x_1) & + & T_r(x_2) & - & \hat{T}\alpha_{2,1} & \leq & 0 \\ \alpha_{1,2} & + & \alpha_{2,1} & - & \phi_{1,2} & = & 1 \end{array} \tag{1.2}$$

where

- $T_s(x_i)$ is the start time of activity x_i,
- $T_r(x_i)$ is the release time of activity x_i, and
- $\hat{T}$ represents the maximum possible completion time of the schedule.
- $T_s(x_i),\ T_r(x_j) \in \Re$,
- $\alpha_{i,j}, \phi_{i,j} \in \{0,1\}$,

Variable assignments in (1.2) are interpreted as follows:

If $\alpha_{1,2} = 0$ then the first constraint becomes $T_s(x_2) \geq T_r(x_1)$ so the activities are serialized with x_1 executing first.

If $\alpha_{1,2} = 1$ then the first constraint is trivially satisfied since it states that the difference between the two times is less than the largest possible time.

If $\alpha_{2,1} = 0$ then the second constraint becomes $T_s(x_1) \geq T_r(x_2)$ so the activities are serialized with x_1 executing first.

[1]Capsule description: The algorithm to be implemented is described as transformations of data and storage of values (collectively, activities). Activities must be scheduled on a set of components (resources/machines).

If $\alpha_{2,1} = 1$ then the second constraint is trivially satisfied since it states that the difference between the two times is less than the largest possible time.

If $\phi_{1,2} = 1$ then no execution order is enforced.

If $\phi_{1,2} = 0$ then the activities are serialized.

The third constraint ensures that exactly one serialization order is chosen when serialization is forced.

In an effort to make these serialization decisions more efficiently, we replace constraints of the form

$$-T_s(x_i) + T_r(x_j) - \hat{T}\alpha_{i,j} \leq 0 \tag{1.3}$$

by the following cutting plane constraints (derived in Chapter 2):

$$\begin{array}{rcrcrcr} -T_s(x_i) & + & T_r(x_j) & + & (\check{T}_s(x_i) - \hat{T}_r(x_j))\alpha_{i,j} & \leq & 0 \\ -T_s(x_i) & & & + & (\check{T}_s(x_i) - \check{T}_r(x_j))\alpha_{i,j} & \leq & \check{T}_r(x_j) \\ & & Tr(x_j) & + & (\hat{T}_s(x_i) - \hat{T}_r(x_j))\alpha_{i,j} & \leq & \hat{T}_s(x_i) \end{array}$$

These constraints are then tightened each time one of the variable domains $T_s(x_i) \in [\check{T}_s(x_i), \hat{T}_s(x_i)]$, or $T_r(x_j) \in [\check{T}_r(x_j), \hat{T}_r(x_j)]$ is tightened by the arc consistency routines. The constraints must also be rewritten to match the restored bounds when a node is restored during the branch-and-bound search.

The computational results presented in Chapter 5 show a decrease in the average number of subproblems required to solve large examples. However, there is also an increase in the number of pivots required to solve each relaxation.

The notation used in this thesis appears in Table 1.1. The generic cutting plane equations are derived in Chapter 2. A description of *Bonsai* appears in Chapter 3. Chapter 4 contains a discussion of how the generic cutting planes are integrated into *bonsai*. Chapter 5 presents the computational results and the conclusions follow in Chapter 6.

$\check{x}$	lower bound of variable x
$\hat{x}$	upper bound of variable x
UB_i	row upper bound of the left hand side of constraint i
LB_i	row lower bound of the left hand side of constraint i
$\hat{T}$	maximum possible completion time of the schedule
A_i^+	the set of indices j such that $a_{i,j} > 0$
A_i^-	the set of indices j such that $a_{i,j} < 0$
$S \setminus t$	the set difference $S - \{t\}$
P^*	the polar of polyhedron P
(π, π_0)	$\pi x \leq \pi_0$ is a valid inequality for the polyhedron

Table 1.1: Notation

Chapter 2

Cutting Plane Derivation

2.1 Algorithm

We use the sequential convexification procedure of Balas *et al.* [2] to lift and project the constraint system to generate the cutting plane equations. This procedure is described below:

Let $K = \{x \in \Re^n : Ax \geq b,\ x \geq 0,\ x_j \leq 1,\ j = 1, 2, ..., p\} = \{x \in \Re^n : \tilde{A}x \geq \tilde{b}\}$.

1. Select $j \in \{1, ...p\}$.
2. Multiply $\tilde{A}x \leq \tilde{b}$ by $(1 - x_j)$ and x_j to obtain the nonlinear system
$$\begin{aligned} (1 - x_j)(\tilde{A}x - \tilde{b}) &\leq 0 \\ x_j(\tilde{A}x - \tilde{b}) &\leq 0 \end{aligned} \tag{2.1}$$
3. Linearize (2.1) by substituting y_i for $x_i x_j$, $i = 1, ..., n, i \neq j$ and x_j for x_j^2. Call the resulting polyhedron $M_j(K)$. This step cuts off points x with $0 < x_j < 1$ but does not eliminate any points x for which $x_j \in \{0, 1\}$ (see [1] for details).
4. Project $M_j(K)$ onto the x-space by eliminating y_i, $i = 1, ..., n, i \neq j$. Call the resulting polyhedron $P_j(K)$.

Theoretically, the *best* cutting plane would be the inequality (π, π_0) that is valid for the polyhedron $P_j(K)$ and is orthogonally as far away as possible from X^*, the non-integer extreme point to be cut off. However, since distance is a non-linear function, we shall instead select the inequality (π, π_0) from the polar of $Mj(K)$ which has the maximum slack at X^* in the projection polyhedra $P_j(K)$. This criteria can be formulated as the following objective function:

maximize $U(A'X^* - b')$

where

- U is a vector of non-negative multipliers,
- X^* is a non-integer extreme point to be cut off the original polyhedron, and
- $A'x \leq b'$ defines $Mj(K)$.

To eliminate the y_i terms in the projection step of the algorithm, we use the following constraints:

$$\sum_{i=1}^{m} u_i y_{i,j} = 0, \text{ for } j = 1, ..., n \; j \neq i$$

To ensure that the LP will not be unbounded, we place a limit on the growth of the cutting plane constraint within the polar by adding the following constraint:

$$\sum_{i=1}^{m} u_i \leq 1$$

Thus, symbolically solving the following LP will provide us with a linear combination of constraints that generate a cutting plane equation:

$$\begin{array}{lllll} \text{maximize} & U(A'X^* - b') & & & \\ \text{st} & \sum_{i=1}^{m} u_i y_{i,j} & = & 0, & \text{for } j = 1, ..., n \; j \neq i \\ & \sum_{i=1}^{m} u_i & \leq & 1 & \end{array} \tag{2.2}$$

2.2 Derivation

We examine the input constraint form introduced as (1.3) on page 10:

$$-T_s(x_i) + T_r(x_j) - \hat{T}\alpha_{i,j} \leq 0,$$

To simplify notation we perform the variable substitutions:

$$\begin{aligned} s &= T_s(x_i) \\ r &= T_r(x_j) \\ \alpha &= \alpha_{i,j} \end{aligned}$$

Adding constraints enforcing the variable upper and lower bounds[1] yields the following constraint system:

$$\left.\begin{array}{cccccccc} -s & + & r & - & \hat{T}\alpha & \leq & 0 \\ -s & & & & & \leq & -\check{s} \\ s & & & & & \leq & \hat{s} \\ & - & r & & & \leq & -\check{r} \\ & & r & & & \leq & \hat{r} \\ & & & - & \alpha & \leq & 0 \\ & & & & \alpha & \leq & 1 \end{array}\right\} \quad (\tilde{A}x \leq \tilde{b}) \qquad (2.3)$$

The constraint system (2.3) defines the polyhedron for which cutting planes will be derived (shown in Figure 2.1).

[1]The bounds on s and r are required in the linear combinations which calculates the cutting plane equations.

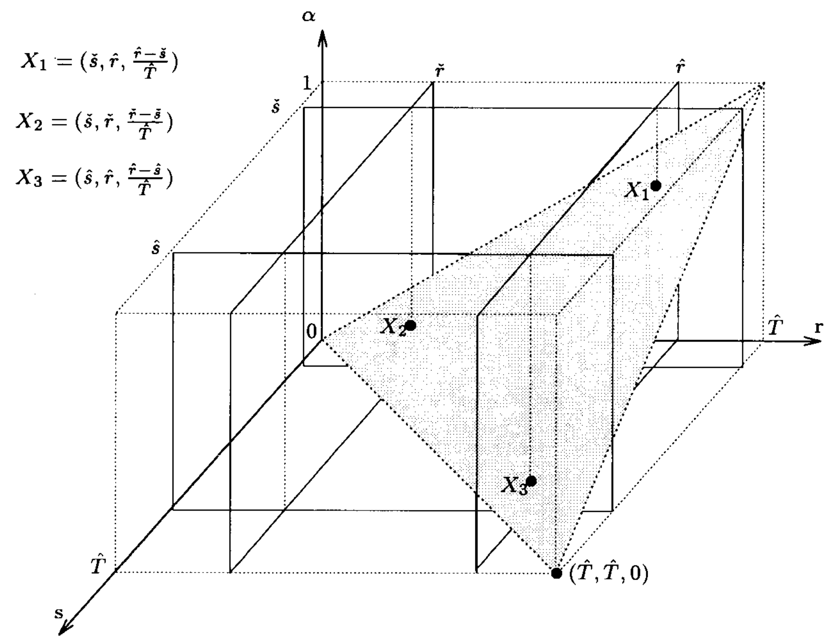
$X_1 = (\check{s}, \hat{r}, \frac{\hat{r}-\check{s}}{\hat{T}})$
$X_2 = (\check{s}, \check{r}, \frac{\check{r}-\check{s}}{\hat{T}})$
$X_3 = (\hat{s}, \hat{r}, \frac{\hat{r}-\hat{s}}{\hat{T}})$
α
1
$\check{r}$
$\hat{r}$
$\check{s}$
$\hat{s}$
0
X_1
X_2
X_3
$\hat{T}$
r
$\hat{T}$
s
$(\hat{T}, \hat{T}, 0)$

Figure 2.1: Non-integer extreme points to cut off

Lift the constraints (step 2.1 from Section 2.1) by multiplying (2.3) by α and by $(1-\alpha)$.

$$\left.\begin{array}{rrrrrrrrrrrr}
 & & & - & s\alpha & + & r\alpha & - & \hat{T}\alpha^2 & & \leq & 0 \\
 & & & - & s\alpha & & & & & + \ \check{s}\alpha & \leq & 0 \\
 & & & & s\alpha & & & & & - \ \hat{s}\alpha & \leq & 0 \\
 & & & & & - & r\alpha & & & + \ \check{r}\alpha & \leq & 0 \\
 & & & & & & r\alpha & & & - \ \hat{r}\alpha & \leq & 0 \\
 & & & & & & & - & \alpha^2 & & \leq & 0 \\
 & & & & & & & & \alpha^2 & - \ \alpha & \leq & 0
\end{array}\right\} \quad \alpha(\tilde{A}x \leq \tilde{b})$$

$$\left.\begin{array}{rrrrrrrrrrrr}
-s & + & r & + & s\alpha & - & r\alpha & + & \hat{T}\alpha^2 & - \ \hat{T}\alpha & \leq & 0 \\
-s & & & + & s\alpha & & & & & - \ \check{s}\alpha & \leq & -\check{s} \\
s & & & - & s\alpha & & & & & + \ \hat{s}\alpha & \leq & \hat{s} \\
 & - & r & & & + & r\alpha & & & - \ \check{r}\alpha & \leq & -\check{r} \\
 & & r & & & - & r\alpha & & & + \ \hat{r}\alpha & \leq & \hat{r} \\
 & & & & & & & - & \alpha^2 & + \ \alpha & \leq & 0 \\
 & & & & & & & - & \alpha^2 & + \ 2\alpha & \leq & 1
\end{array}\right\} \quad (1-\alpha)(\tilde{A}x \leq \tilde{b})$$

Replace α^2 by α (step 3 from Section 2.1) :

$$A' = \begin{array}{c} \begin{array}{ccccc} s & r & \alpha & s\alpha & r\alpha \end{array} \\ \left[\begin{array}{ccccc} 0 & 0 & -\hat{T} & -1 & 1 \\ 0 & 0 & \check{s} & -1 & 0 \\ 0 & 0 & -\hat{s} & 1 & 0 \\ 0 & 0 & \check{r} & 0 & -1 \\ 0 & 0 & -\hat{r} & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ -1 & 1 & 0 & 1 & -1 \\ -1 & 0 & -\check{s} & 1 & 0 \\ 1 & 0 & \hat{s} & -1 & 0 \\ 0 & -1 & -\check{r} & 0 & 1 \\ 0 & 1 & \hat{r} & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array}\right] \end{array}, b' = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -\check{s} \\ \hat{s} \\ -\check{r} \\ \hat{r} \\ 0 \\ 1 \end{array}\right] \tag{2.4}$$

The cutting plane equations are then calculated as the linear combination of (2.4) derived using following linear program which eliminates any non-linear terms[2]

$$\begin{array}{lll} \textit{maximize} & U(A'X^* - b') & \\ \textit{st} & -u_1 - u_2 + u_3 + u_8 + u_9 - u_{10} & = 0 \\ & u_1 - u_4 + u_5 - u_8 + u_{11} - u_{12} & = 0 \\ & \sum_{i=1}^{14} u_i & = 1 \end{array} \tag{2.5}$$

where

- U is a set of non-negative multipliers, and
- X^* is a non-integer extreme point to be cut off.

[2]This incorporates the projection step 4 in Section 2.1.

To solve (2.5) symbolically we performed the following steps using *Maple* [3]:

1. Calculate the objective function.
2. Substitute numeric values for the variable bounds to acquire a numeric LP.
3. Solve the numeric LP using the simplex method.
4. Format the symbolic LP as a dictionary using the set of basic variables from the numeric solution.

Then it is necessary to manually verify that the symbolic solution is optimal — *i.e.* test that all coefficients in the dictionary representation of the objective function are negative.

Depending on the values of the domains of s and r, there can be a maximum of three non-integer extreme points in (2.3) These points have the following coordinates (see Figure 2.1):

$$\begin{array}{ll} X_1 & (\check{s}, \hat{r}, (\hat{r} - \check{s})/\hat{T}) \\ X_2 & (\check{s}, \check{r}, (\check{r} - \check{s})/\hat{T}) \\ X_3 & (\hat{s}, \hat{r}, (\hat{r} - \hat{s})/\hat{T}) \end{array}$$

Since we can derive one cutting plane from each non-integer extreme point, we will repeat the algorithm of Section 2.1 three times to generate our three cutting plane equations. Figures 2.2, 2.3, and 2.4 illustrate examples of these cutting planes and contain labeled points having the following coordinates:

$$\begin{array}{ccccccc} A & = & (& \check{s}, & \hat{r}, & 1 &) \\ B & = & (& \check{s}, & \check{r}, & 1 &) \\ C & = & (& \check{r}, & \check{r}, & 0 &) \\ D & = & (& \hat{s}, & \hat{s}, & 0 &) \\ E & = & (& \hat{s}, & \hat{r}, & 1 &) \end{array}$$

Cutting off X_1

Substituting $X_1 = (\check{s}, \hat{r}, (\hat{r}-\check{s})/\hat{T}, 0, 0)$ for X^* in $U(A'X^* - b')$ we have:

$$obj_1 = \left(\frac{\hat{r}-\check{s}}{\hat{T}}\right) \Bigg(-\hat{T}u_1 + \check{s}u_2 - \hat{s}u_3 + \check{r}u_4 - \hat{r}u_5 - u_7 + \hat{T}u_8 - \check{s}u_9 + \left(\hat{s} - \frac{(\hat{s}-\check{s})\hat{T}}{\hat{r}-\check{s}}\right)u_{10} - \left(\check{r} + \frac{(\hat{r}-\check{r})\hat{T}}{\hat{r}-\check{s}}\right)u_{11} + \hat{r}u_{12} + \left(1 - \frac{\hat{T}}{\hat{r}-\check{s}}\right)u_{13}\Bigg)$$

Note that X_1 is an extreme point of the polyhedron (2.3) only when $\check{s} \le \hat{r}$. By selecting u_2, u_5, and u_8 as basic variables, obj_1 can be rewritten as z_1 below.

$$\begin{aligned}
u_2 &= 1/3(1 - 2u_1 + u_3 - 2u_4 - u_6 - u_7 + u_9 - 3u_{10} - 2u_{12} - u_{13} - u_{14}) \\
u_5 &= 1/3(1 - 2u_1 - 2u_3 + u_4 - u_6 - u_7 - 2u_9 - 3u_{11} + u_{12} - u_{13} - u_{14}) \\
u_8 &= 1/3(1 + u_1 - 2u_3 - 2u_4 - u_6 - u_7 - 2u_9 - 2u_{12} - u_{13} - u_{14}) \\
z_1 &= \left(\frac{\hat{r}-\check{s}}{3\hat{T}}\right)(\hat{T} - \hat{r} - \check{s} + c_1u_1 + c_3u_3 + c_4u_4 + c_7u_7 + c_9u_9 \\
&\qquad + c_{10}u_{10} + c_{11}u_{11} + c_{12}u_{12} + c_{13}u_{13})
\end{aligned}$$

where

$$\begin{aligned}
c_1 &= -2(\hat{r}-\check{s})(\hat{T} - (\hat{r}-\check{s})) \\
c_3 &= -(\hat{r}-\check{s})(2\hat{T} - 2(\hat{r}-\hat{s}) + (\hat{s}-\check{s})) \\
c_4 &= -(\hat{r}-\check{s})(\hat{r} + 2\check{s} + 2\hat{T} - 3\check{r}) \\
c_6 &= -(\hat{r}-\check{s})(\hat{T} - (\hat{r}-\check{s})) \\
c_7 &= -(\hat{r}-\check{s})(\hat{T} - (\hat{r}-\check{s}) + 3) \\
c_9 &= -2(\hat{r}-\check{s})(\hat{T} - (\hat{r}-\check{s})) \\
c_{10} &= -3(\hat{s}-\check{s})(\hat{T} - (\hat{r}-\check{s})) \\
c_{11} &= -3(\hat{r}-\check{r})(\hat{T} - (\hat{r}-\check{s})) \\
c_{12} &= -2(\hat{r}-\check{s})(\hat{T} - (\hat{r}-\check{s})) \\
c_{13} &= -(3 + (\hat{r}-\check{s}))(\hat{T} - (\hat{r}-\check{s})) \\
c_{14} &= -(\hat{r}-\check{s})(\hat{T} - (\hat{r}-\check{s}))
\end{aligned}$$

Given $\check{s} \le \hat{r}$, all of the coefficients of z_1 are negative. Hence the above dictionary represents an optimal solution.

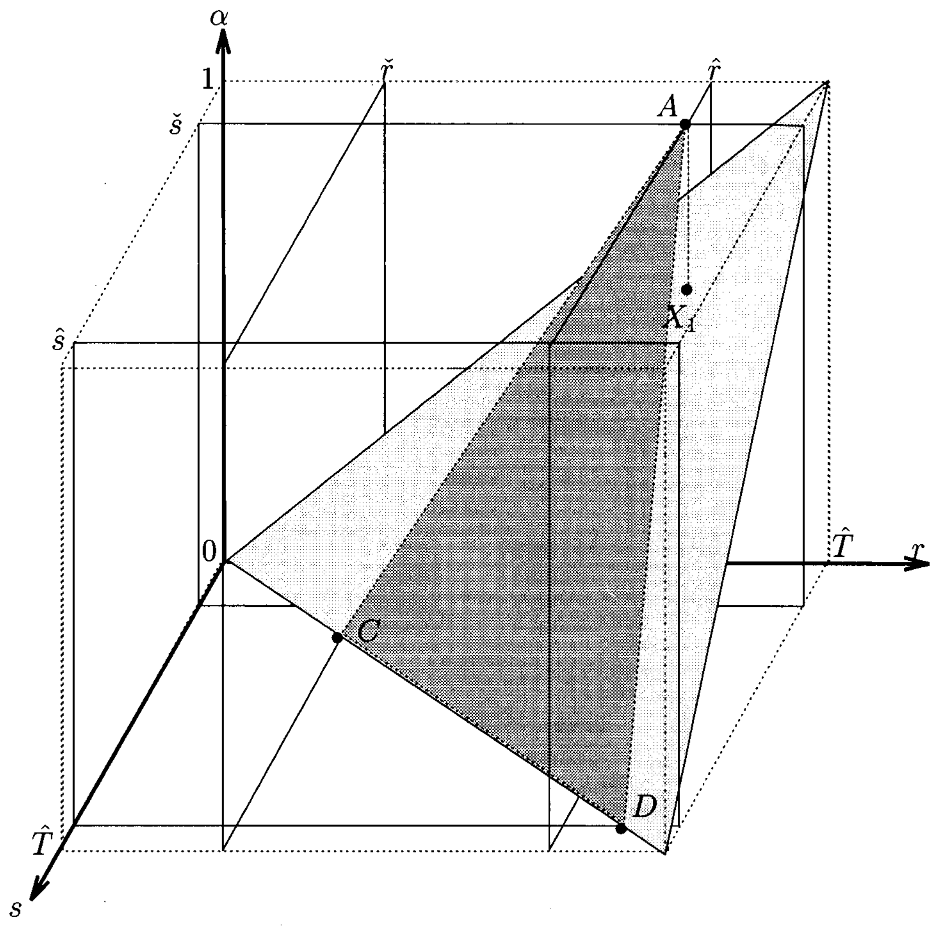

Figure 2.2: ACD cuts off X_1

Since $u_2, u_5, andu8$ all have the same value, we can calculate the first cutting plane constraint (shown in Figure 2.2) as the sum of the 2nd, 5th, and 8th constraints of $M_j(K)$ (2.4):

$$\begin{array}{ccccccccccc}
 & & & - & s\alpha & & & + & \check{s}\alpha & \le & 0 \\
 & & & & & & r\alpha & - & \hat{r}\alpha & \le & 0 \\
-s & + & r & + & s\alpha & - & r\alpha & \le & 0 & & \\
\hline
-s & + & r & & & & & + & (\check{s}-\hat{r})\alpha & \le & 0
\end{array}$$

Cutting off X_2

Substituting $X_2 = (\check{s}, \check{r}, (\check{r}-\check{s})/\hat{T}, 0, 0)$ for X^* in $U(A'X^* - b')$ we have:

$$obj_2 = \left(\frac{\check{r}-\check{s}}{\hat{T}}\right) \Bigg(-\hat{T}u_1 + \check{s}u_2 - \hat{s}u_3 + \check{r}u_4 - \hat{r}u_5 - u_7 + \hat{T}u_8 - \check{s}u_9 + \left(\frac{(\hat{s}-\check{s})\hat{T}}{\check{r}-\check{s}} - \hat{s}\right)u_{10} - \check{r}u_{11} + \left(\frac{(\hat{r}-\check{r})\hat{T}}{\check{r}-\check{s}} + \hat{r}\right)u_{12} + \left(1 + \frac{\hat{T}}{\check{r}-\check{s}}\right)u_{13}\Bigg)$$

X_2 is an extreme point of the polyhedron (2.3) only when $\check{s} \le \check{r}$. By selecting u_2, u_8, and u_{11} as basic variables, obj_2 can be rewritten as z_2 below.

$$\begin{aligned}
u_2 &= 1/3(1 - 2u_1 + u_3 - 2u_4 - u_6 - u_7 + u_9 - 3u_{10} - 2u_{12} - u_{13} - u_{14})\\
u_8 &= 1/3(1 + u_1 - 2u_3 - 2u_4 - u_6 - u_7 - 2u_9 - 2u_{12} - u_{13} - u_{14})\\
u_{11} &= 1/3(1 - 2u_1 - 2u_3 + u_4 - 3u_5 - u_6 - u_7 - 2u_9 + u_{12} - u_{13} - u_{14})\\
z_2 &= \left(\frac{\check{r}-\check{s}}{3\hat{T}}\right)(\hat{T} + \check{s} - \check{r} + c_1u_1 + c_3u_3 + c_4u_4 + c_5u_5 + c_7u_7\\
&\qquad + c_9u_9 + c_{10}u_{10} + c_{12}u_{12} + c_{13}u_{13})
\end{aligned}$$

where

$$\begin{aligned}
c_1 &= -2(\check{r}-\check{s})(\hat{T} - (\check{r}-\check{s}))\\
c_3 &= -(\check{r}-\check{s})(2\hat{T} + (\hat{s}-\check{s}) + 2(\hat{s}-\check{r}))\\
c_4 &= -2(\check{r}-\check{s})(\hat{T} - (\check{r}-\check{s}))\\
c_5 &= -3(\check{r}-\check{s})(\hat{r}-\check{r})\\
c_6 &= -(\check{r}-\check{s})(\hat{T} - (\check{r}-\check{s}))\\
c_7 &= -(\check{r}-\check{s})(\hat{T} - (\check{r}-\check{s}) + 3)\\
c_9 &= -2(\check{r}-\check{s})(\hat{T} - (\check{r}-\check{s}))\\
c_{10} &= -3(\hat{s}-\check{s})(\hat{T} - (\check{r}-\check{s}))\\
c_{12} &= -(2(\check{r}-2\check{s}) + (\hat{r}-\check{r}))(\hat{T} - (\check{r}-\check{s}))\\
c_{13} &= -(3 + (\check{r}-\check{s}))(\hat{T} - (\check{r}-\check{s}))\\
c_{14} &= -(\check{r}-\check{s})(\hat{T} - (\check{r}-\check{s}))
\end{aligned}$$

Given $\check{s} \le \check{r}$, all of the coefficients of z_2 are negative. Hence the above dictionary represents an optimal solution.

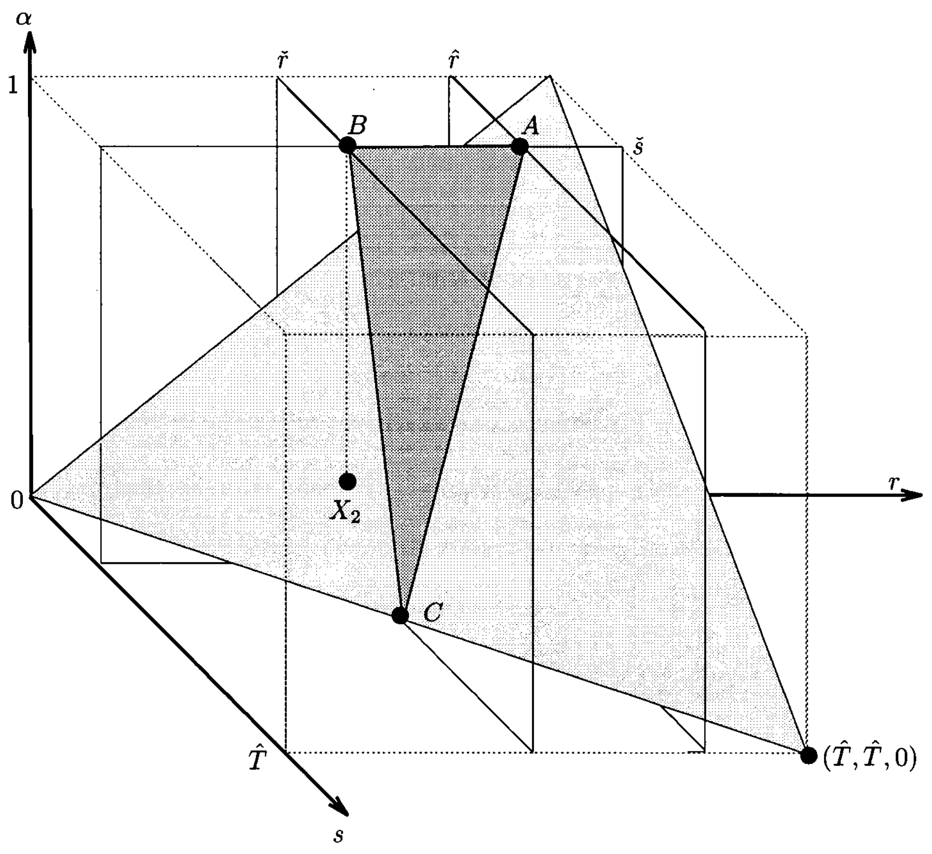

Figure 2.3: ABC cuts off X_2

Since $u_2, u_8, and u_{11}$ all have the same value, we can calculate the the second cutting plane constraint (shown in Figure 2.3) as the sum of the 2nd, 8th, and 11th constraints of $M_j(K)$ (2.4):

$$\begin{array}{rrrrrrrrrrr}
 & & & - & s\alpha & & & + & \check{s}\alpha & \leq & 0 \\
-s & + & r & + & s\alpha & - & r\alpha & & & \leq & 0 \\
 & - & r & & & + & r\alpha & - & \check{r}\alpha & \leq & -\check{r} \\
\hline
-s & & & & & & & + & (\check{s}-\check{r})\alpha & \leq & \check{r}
\end{array}$$

Cutting off X_3

Substituting $X_3 = (\hat{s}, \hat{r}, (\hat{r} - \hat{s})/\hat{T}, 0, 0)$ for X^* in $U(A'X^* - b')$ we have:

$$obj_3 = \left(\frac{\hat{r} - \hat{s}}{\hat{T}}\right) \; (-\hat{T}u_1 + \check{s}u_2 - \hat{s}u_3 + \check{r}u_4 - \hat{r}u_5 - u_7 + \hat{T}u_8 - \left(\frac{\hat{T}(\hat{s} - \check{s})}{\hat{r} - \hat{s}} + \check{s}\right) u9$$
$$+\hat{s}u_{10} - \left(\frac{\hat{T}(\hat{r} - \check{r})}{\hat{r} - \hat{s}} + \check{r}\right) u_{11} + \hat{r}u_{12} + \left(1 - \frac{\hat{T}}{\hat{r} - \hat{s}}\right) u_{13})$$

X_3 is an extreme point of the polyhedron (2.3) only when $\hat{s} \leq \hat{r}$. By selecting u_5, u_8, and u_{10} as basic variables, obj_3 can be rewritten as z_3 below.

$$\begin{aligned}
u_8 &= 1/3(1 + u_1 - 2u_3 - 2u_4 - u_6 - u_7 - 2u_9 - 2u_{12} - u_{13} - u_{14}) \\
u_5 &= 1/3(+1 - 2u_1 - 2u_3 + u_4 - u_6 - u_7 - 2u_9 - 3u_{11} + u_{12} - u_{13} - u_{14}) \\
u_{10} &= 1/3(1 - 2u_1 - 3u_2 + u_3 - 2u_4 - u_6 - u_7 + u_9 - 2u_{12} - u_{13} - u_{14}) \\
z_3 &= \left(\frac{\hat{r} - \hat{s}}{3\hat{T}}\right) \; (\hat{T} - \hat{r} - \hat{s} + c_1u_1 + c_2u_2 + c_3u_3 + c_4u_4 + c_7u_7 \\
&\qquad\qquad +c_9u_9 + c_{11}u_{11} + c_{12}u_{12} + c_{13}u_{13})
\end{aligned}$$

where

$$\begin{aligned}
c_1 &= -2(\hat{r} - \hat{s})(\hat{T} - (\hat{r} - \hat{s})) \\
c_2 &= -3(\hat{r} - \hat{s})(\hat{s} - \check{s}) \\
c_3 &= -2(\hat{r} - \hat{s})(\hat{T} - (\hat{r} - \hat{s})) \\
c_4 &= -(\hat{r} - \hat{s})(2\hat{T} + 2(\hat{s} - \check{r}) + (\hat{r} - \check{r})) \\
c_6 &= -(\hat{r} - \hat{s})(\hat{T} - (\hat{r} - \hat{s})) \\
c_7 &= -(\hat{r} - \hat{s})(\hat{T} - (\hat{r} - \hat{s}) + 3) \\
c_9 &= -(\hat{T} - (\hat{r} - \hat{s}))(2(\hat{r} - \check{s}) + (\hat{s} - \check{s})) \\
c_{11} &= -3(\hat{r} - \check{r})(\hat{T} - (\hat{r} - \hat{s})) \\
c_{12} &= -2(\hat{r} - \hat{s})(\hat{T} - (\hat{r} - \hat{s})) \\
c_{13} &= -((\hat{r} - \hat{s}) + 3)(\hat{T} - (\hat{r} - \hat{s})) \\
c_{14} &= -(\hat{r} - \hat{s})(\hat{T} - (\hat{r} - \hat{s}))
\end{aligned}$$

Given $\hat{s} \leq \hat{r}$ all of the coefficients of z_3 are negative. Hence the above dictionary represents an optimal solution.

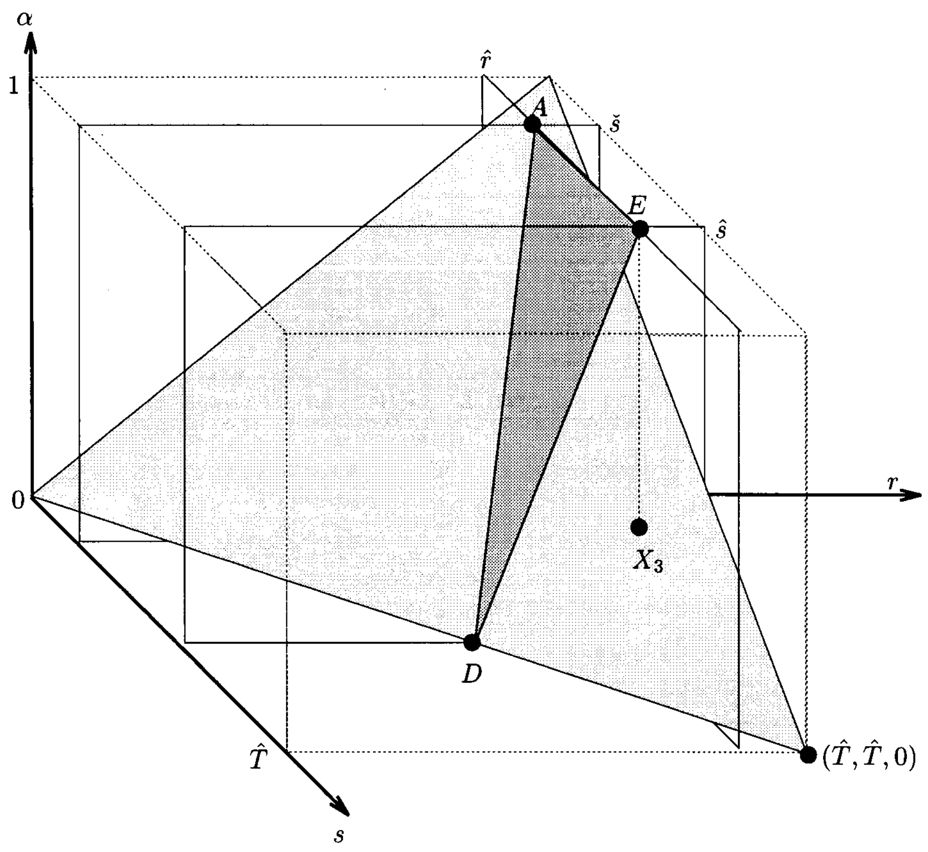

Figure 2.4: ADE cuts off X_3

Since $u_5, u_8, and u_{10}$ all have the same value, we can calculate the the third cutting plane constraint (shown in Figure 2.4) as the sum of the 5th, 8th, and 10th constraints of $M_j(K)$ (2.4):

$$\begin{array}{rrrrrrrrr}
 & & & & r\alpha & - & \hat{r}\alpha & \leq & 0 \\
-s & + & r & + & s\alpha & - & r\alpha & \leq & 0 \\
s & & & - & s\alpha & + & \hat{s}\alpha & \leq & \hat{s} \\
\hline
 & & r & & & + & (\hat{s}-\hat{r})\alpha & \leq & \hat{s}
\end{array}$$

Table 2.1 shows the generic cutting plane equations derived in this chapter. These equations will be used to generate cutting planes to replace each constraint appearing in the input having appropriate form. The cutting planes will then be tightened each time the variable bounds are tightened.

$$\boxed{\begin{array}{lcl} A & = & (\ \check{s},\ \hat{r},\ 1\ ) \\ B & = & (\ \check{s},\ \check{r},\ 1\ ) \\ C & = & (\ \check{r},\ \check{r},\ 0\ ) \\ D & = & (\ \hat{s},\ \hat{s},\ 0\ ) \\ E & = & (\ \hat{s},\ \hat{r},\ 1\ ) \end{array}}$$

$$\begin{array}{rccccccc} ACD: & -s & + & r & + & (\check{s}-\hat{r})\alpha & \leq & 0 \\[1em] ABC: & -s & & & + & (\check{s}-\check{r})\alpha & \leq & \check{r} \\[1em] ADE: & & & r & + & (\hat{s}-\hat{r})\alpha & \leq & \hat{s} \end{array}$$

Table 2.1: Cutting Plane Constraints

Chapter 3

Bonsai

Bonsai solves mixed-0/1 programming problems. It implements a LP-based branch-and-bound algorithm incorporating partial arc consistency and supports binary and continuous variables. Only the components of *bonsai* relevant to this thesis are described below. For a detailed discussion see [6, 8].

3.1 Branch-and-Bound

The main cycle of the branch-and-bound algorithm (shown in Figure 3.1) begins by selecting a set of Boolean variables to branch over. The active subproblem is expanded using these branching variable(s). The successor subproblems are then evaluated. The objective function value, infeasibility, and integrality are all used in an attempt to fathom each subproblem. Improving the bound on the subproblem's objective function is attempted through a cycle of arc consistency, solving the LP relaxation, and penalty calculations which is repeated until there are no more variables which can be fixed.

The best unfathomed successor is selected to become the new active subproblem. The remaining unfathomed nodes are placed in the set of live subproblems and will be explored at a later time. If all the active node's successors are fathomed, then a new active subproblem is selected from the live set.

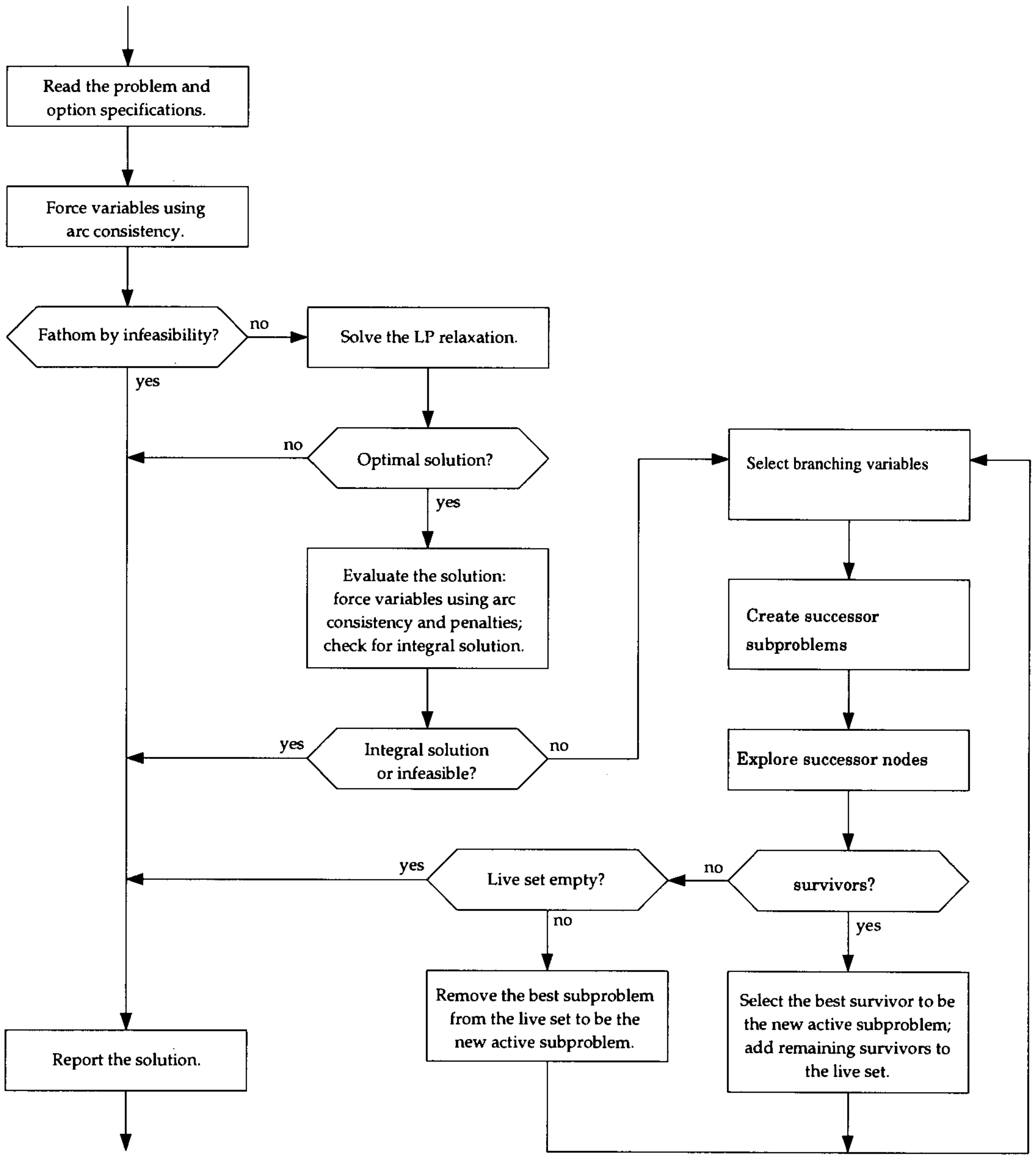
Read the problem and option specifications.
Force variables using arc consistency.
Fathom by infeasibility?
no
yes
Solve the LP relaxation.
no
Optimal solution?
yes
Evaluate the solution: force variables using arc consistency and penalties; check for integral solution.
yes
Integral solution or infeasible?
no
Select branching variables
Create successor subproblems
Explore successor nodes
yes
Live set empty?
no
no
survivors?
yes
Remove the best subproblem from the live set to be the new active subproblem.
Select the best survivor to be the new active subproblem; add remaining survivors to the live set.
Report the solution.

Figure 3.1: Branch-and-bound algorithm

Whenever a new solution to the MILP problem is discovered, it is tested against the current incumbent. If the new objective function value is better then this solution becomes the new incumbent and the set of live subproblems is winnowed to remove any subproblems which no longer have the potential to produce a better solution.

Penalty Calculations

After solving each linear relaxation, *bonsai* calculates a penalty for forcing each basic non-integer variable to either 0 or 1. The penalty is the minimum deterioration in the objective function after the first dual pivot to restore primal feasibility. The reader is directed to [6] for details since for the purpose of this thesis, our only interest in this procedure is that it provides another method besides branching for setting boolean variables. These domain restrictions are propagated using the arc consistency routines and this propagation will in turn cause tightening of cutting plane constraints.

3.2 Arc Consistency

Bonsai uses a partial arc consistency algorithm to propagate restrictions on the domain of one variable onto the domains of other variables. The algorithm is termed partial because it does not necessarily eliminate all inconsistent values from the domain of a variable. It prunes the domain only by restricting the lower and upper bounds of the variable. A variable is fixed at a value if its lower and upper bounds are set equal to that value. The partial arc consistency algorithm used by *bonsai* (shown in Figure 3.2) is a specialization of the algorithm described in [14] which relies on the knowledge that all constraints are summations, and the right hand side of each constraint is a constant.

Background

The i-th linear constraint can be represented in the following format:

$$\sum_{j \in A_i^+} a_{i,j} x_j + \sum_{j \in A_i^-} a_{i,j} x_j \diamond b_i \text{ where } \diamond \in \{\leq, =, \geq\} \tag{3.1}$$

Isolating x_t in the equality constraint i ($\diamond$ is '=') of (3.1) yields the following:

$$x_t = \frac{b_i - \left(\sum_{j \in A_i^+ \setminus t} a_{i,j} x_j + \sum_{j \in A_i^- \setminus t} a_{i,j} x_j \right)}{a_{i,t}} \tag{3.2}$$

Given that $\hat{x}_j$ and $\check{x}_j$ are the upper and lower bounds of variable x_j, we can calculate the upper and lower bounds on the value of the left hand side of the i-th constraint as

$$\begin{aligned} \mathrm{UB}_i &= \sum_{j \in A_i^+} a_{i,j} \hat{x}_j + \sum_{j \in A_i^-} a_{i,j} \check{x}_j \\ \mathrm{LB}_i &= \sum_{j \in A_i^+} a_{i,j} \check{x}_j + \sum_{j \in A_i^-} a_{i,j} \hat{x}_j \end{aligned}$$

A lower bound on x_t is obtained by minimizing the right hand side of (3.2):

$$a_{i,t} > 0 \;\Rightarrow\; \check{x}_t = \frac{b_i - (\mathrm{UB}_i - a_{i,t}\hat{x}_t)}{a_{i,t}}$$

$$a_{i,t} < 0 \;\Rightarrow\; \check{x}_t = \frac{b_i - (\mathrm{LB}_i - a_{i,t}\hat{x}_t)}{a_{i,t}}$$

An upper bound on x_t is obtained by maximizing the right hand side of (3.2):

$$a_{i,t} > 0 \;\Rightarrow\; \hat{x}_t = \frac{b_i - (\mathrm{LB}_i - a_{i,t}\check{x}_t)}{a_{i,t}}$$

$$a_{i,t} < 0 \;\Rightarrow\; \hat{x}_t = \frac{b_i - (\mathrm{UB}_i - a_{i,t}\check{x}_t)}{a_{i,t}}$$

Given an inequality constraint i ($\diamond$ is '$\leq$' or '$\geq$'), the unique bound of x_t which can be tightened depends on the type of inequality and the sign of $a_{i,t}$.

If i is a "$\geq$" constraint and $a_{i,t} > 0$ then:

$$\begin{aligned} x_t &\geq \frac{b_i - \left(\sum_{j \in A_i^+ \setminus t} a_{i,j} x_j + \sum_{j \in A_i^- \setminus t} a_{i,j} x_j \right)}{a_{i,t}} \\ \check{x}_t &= \frac{b_i - (\mathrm{UB}_i - a_{i,t}\hat{x}_t)}{a_{i,t}} \end{aligned}$$

If i is a "$\geq$" constraint and $a_{i,t} < 0$ then:

$$\begin{aligned} x_t &\leq \frac{b_i - \left(\sum_{j \in A_i^+ \backslash t} a_{i,j} x_j + \sum_{j \in A_i^- \backslash t} a_{i,j} x_j \right)}{a_{i,t}} \\ \hat{x}_t &= \frac{b_i - (\mathrm{UB}_i - a_{i,t} \check{x}_t)}{a_{i,t}} \end{aligned}$$

If i is a "$\leq$" constraint and $a_{i,t} > 0$ then:

$$\begin{aligned} x_t &\leq \frac{b_i - \left(\sum_{j \in A_i^+ \backslash t} a_{i,j} x_j + \sum_{j \in A_i^- \backslash t} a_{i,j} x_j \right)}{a_{i,t}} \\ \hat{x}_t &= \frac{b_i - (\mathrm{LB}_i - a_{i,t} \check{x}_t)}{a_{i,t}} \end{aligned} \quad (3.3)$$

If i is a "$\leq$" constraint and $a_{i,t} < 0$ then:

$$\begin{aligned} x_t &\geq \frac{b_i - \left(\sum_{j \in A_i^+ \backslash t} a_{i,j} x_j + \sum_{j \in A_i^- \backslash t} a_{i,j} x_j \right)}{a_{i,t}} \\ \check{x}_t &= \frac{b_i - (\mathrm{LB}_i - a_{i,t} \hat{x}_t)}{a_{i,t}} \end{aligned} \quad (3.4)$$

Propagating incumbent bounds

From the objective function:

$$maximize/minimize \sum_{j=1}^{n} c_j x_j$$

we create the following constraint:

$$\sum_{j=1}^{n} c_j x_j - z \diamond 0$$

where $\diamond$ is "$\geq$" for maximization problems and "$\leq$" for minimization problems. Each time an incumbent is discovered in *bonsai*, the bounds on z are tightened and the arc consistency routines are run to propagate this change.

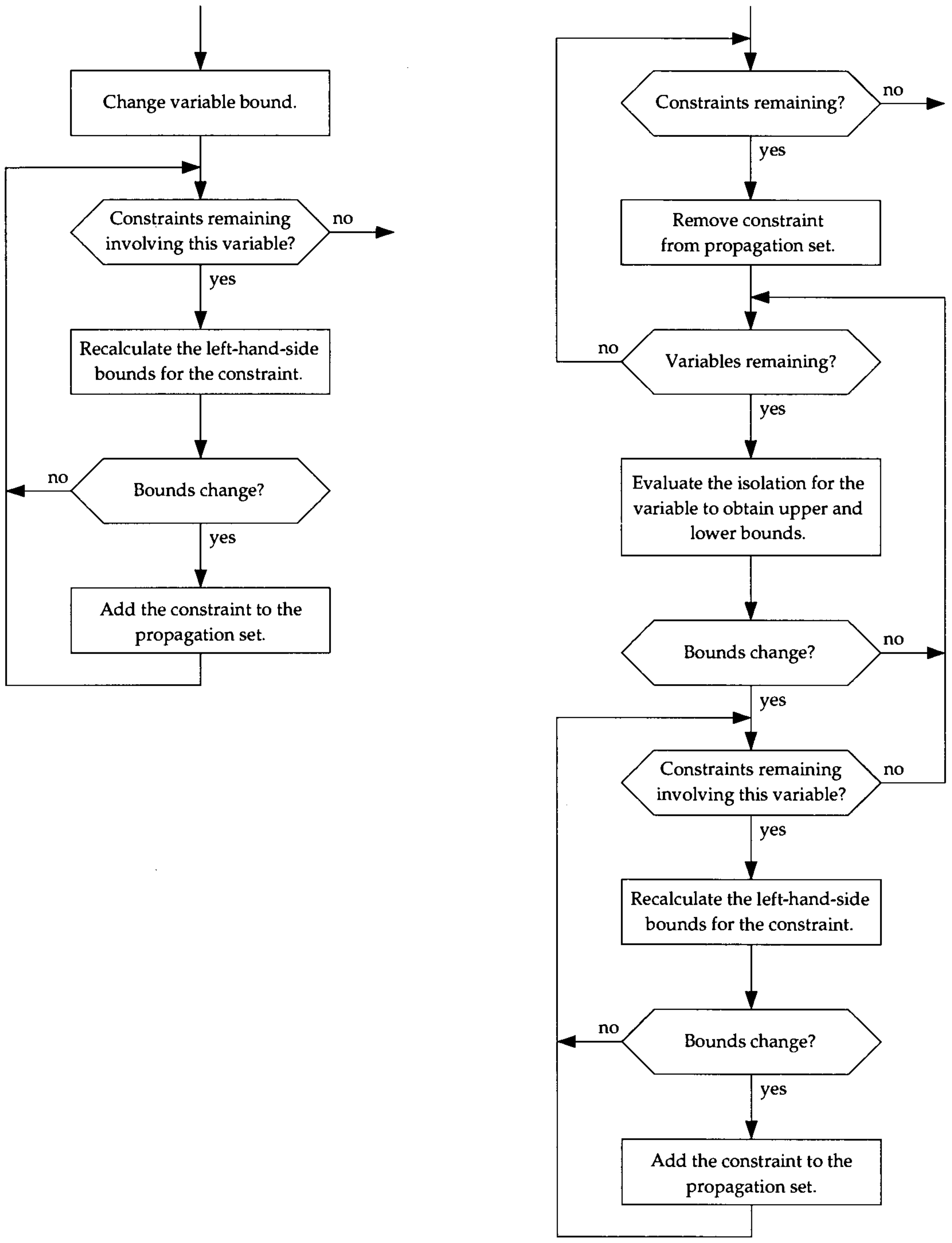
Change variable bound.
Constraints remaining involving this variable?
no
yes
Recalculate the left-hand-side bounds for the constraint.
Bounds change?
no
yes
Add the constraint to the propagation set.
Constraints remaining?
no
yes
Remove constraint from propagation set.
Variables remaining?
no
yes
Evaluate the isolation for the variable to obtain upper and lower bounds.
Bounds change?
no
yes
Constraints remaining involving this variable?
no
yes
Recalculate the left-hand-side bounds for the constraint.
Bounds change?
no
yes
Add the constraint to the propagation set.

Figure 3.2: Partial arc consistency algorithm

Chapter 4

Integration with Bonsai

This chapter discusses how the generic cutting plane equations are introduced into the *bonsai* algorithm. In particular we discuss how the cutting planes interact with the bound tightening performed by the arc consistency component.

4.1 Implementation Details

Due to restrictions in the underlying data structures of *bonsai* which prevent efficient addition or deletion of constraints, it was necessary to have all possible cutting planes in the constraint system at all times even though most of these will be redundant or inactive.

Initialization

Continuous variable domains $[\check{x}_j, \hat{x}_j]$ are initialized to $[0, \hat{T}]$. Thus after reading in the constraint system, each constraint having the form

$$-s + r - \hat{T}\alpha \leq 0$$

is replaced with the initial cutting plane constraints:

$$\begin{array}{ccccccccc} -s & + & r & + & -\hat{T}\alpha & \leq & 0 \\ -s & & & + & & \leq & 0 \\ & & r & + & & \leq & \hat{T} \end{array}$$

This is an equivalent constraint system to the original problem although more redundant since we already had bound constraints on all variables.

Rewriting Cutting Plane Constraints

Whenever the bounds on the variable are changed — either during the arc consistency routines or when a node is restored from the active set during the branch-and-bound search — all cutting planes are examined and those whose equations use the changed bound will be rewritten. Row bounds for cutting plane constraints are then recalculated.

Figure 4.1 illustrates how cutting plane ACD is tightened as $\check{s}$ increases and $\hat{r}$ decreases.

4.2 When Cuts Are Effective

The relative ordering of the variable bounds determines which cuts are effective and which are actually redundant.

So long as either $\check{s} > 0$ or $\hat{r} < \hat{T}$ then cutting plane ACD: $-s + r + (\check{s} - \hat{r})\alpha \leq 0$ is a tighter constraint than the original $-s + r - \hat{T}\alpha \leq 0$. Since variable bounds are never loosened in *bonsai*[1], the cutting plane will only become tighter whenever either $\check{s}$ or $\hat{r}$ is tightened.

[1]Lower bounds become monotonically larger while upper bounds become monotonically smaller

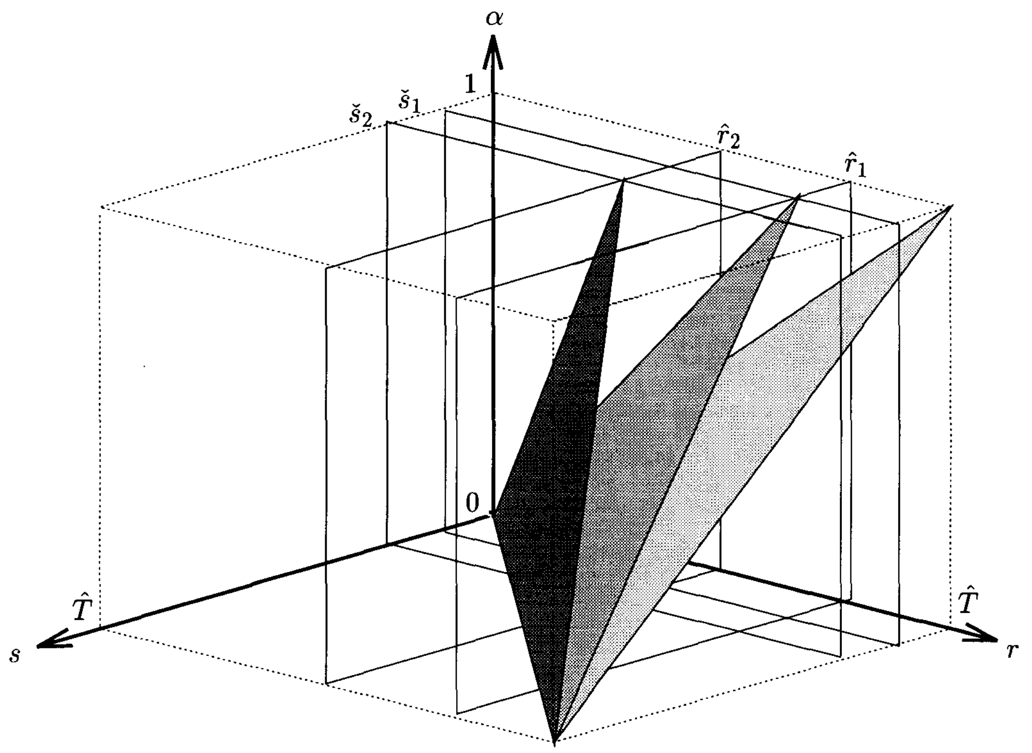

Figure 4.1: Tightening constraint $-s + r + (\check{s} - \hat{r})\alpha \leq 0$

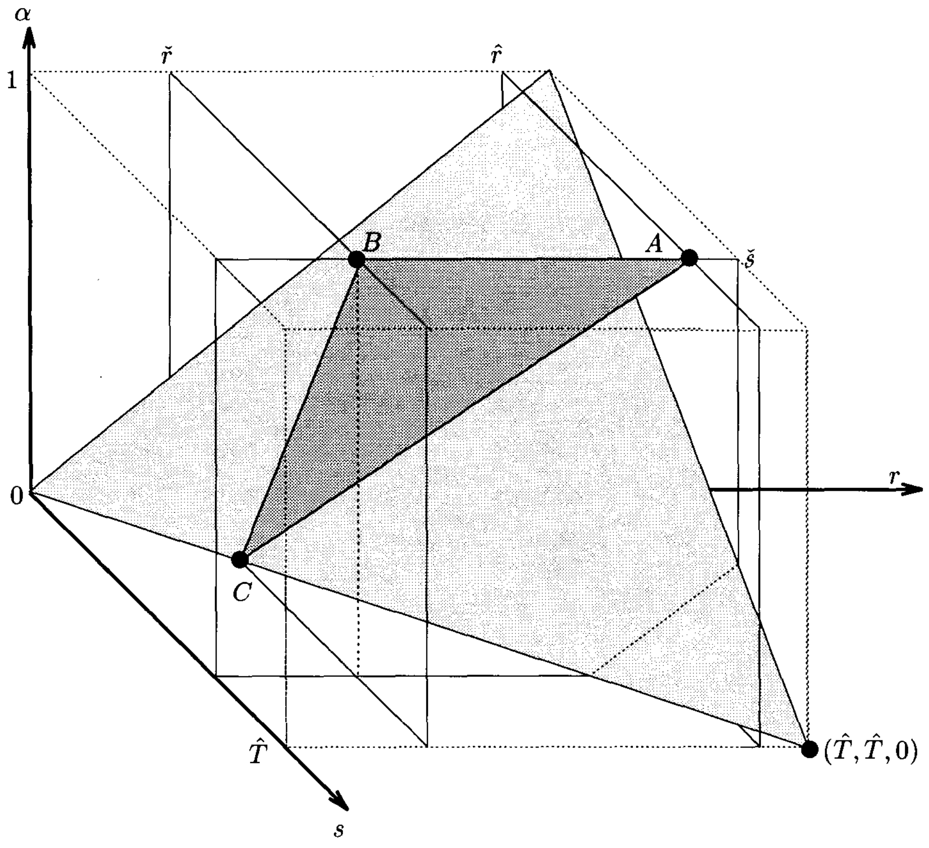

Figure 4.2: When $\check{r} < \check{s}$ then $s \geq \check{s}$ makes constraint ABC redundant

Observation 1 *If $\check{r}$ is strictly less than $\check{s}$ then cutting plane constraint ABC is made redundant by the constraint $s \geq \check{s}$.*

Proof $A = (\check{s}, \hat{r}, 1)$ and $B = (\check{s}, \check{r}, 1)$ are in the plane $s = \check{s}$. $C = (\check{r}, \check{r}, 0)$ is cut off by $s \geq \check{s}$ since by assumption $\check{r} < \check{s}$. Thus, cutting plane ABC is redundant given that $0 \leq \alpha \leq 1$ (see Figure 4.2). ■

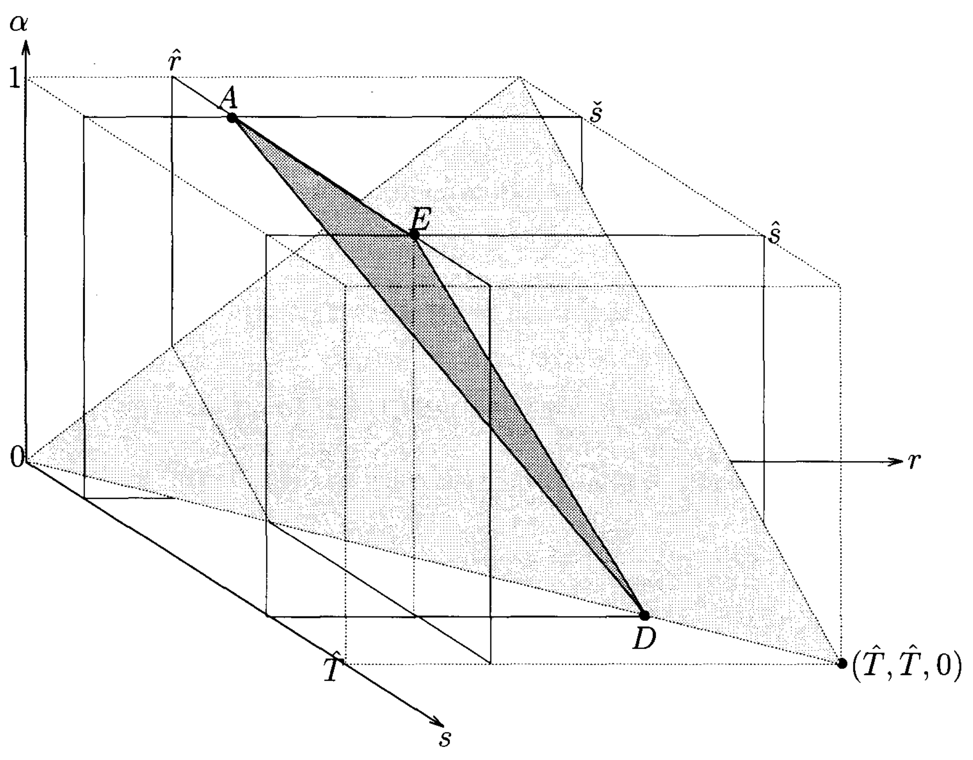

Figure 4.3: When $\hat{r} < \hat{s}$ then $r \leq \hat{r}$ makes constraint ADE redundant

Observation 2 *If $\hat{r}$ is strictly less than $\hat{s}$ then cutting plane constraint ADE is made redundant by the constraint $r \leq \hat{r}$.*

Proof $A = (\check{s}, \hat{r}, 1)$ and $E = (\hat{s}, \hat{r}, 1)$ are in the plane $r = \hat{r}$. $D = (\hat{s}, \hat{s}, 0)$ is cut off by $r \leq \hat{r}$ since by assumption $\hat{r} < \hat{s}$. Thus, cutting plane ADE is redundant given that $0 \leq \alpha \leq 1$ (see Figure 4.3). ■

4.3 An Examination of Row Bounds

Previous incarnations of *Bonsai* read in the constraint matrix and then treated it as a static entity. Now however, the coefficients of the Boolean variables and the right hand side of cutting plane constraints are modified as the bounds of the continuous variables are tightened.

In *bonsai*, bounds are only ever tightened; lower bounds are only replaced by larger values and upper bounds only by smaller values. We need to examine the tightening of the cutting plane constraints to ensure that this property has been maintained.

Recall from Section 3.2 that only the constraint lower bound is used for $\leq$ constraints and it is calculated as follows:

$$\mathrm{LB}_i = \sum_{j \in A_i^+} a_{i,j}\check{x}_j + \sum_{j \in A_i^-} a_{i,j}\hat{x}_j \tag{4.1}$$

Note that the relative magnitudes of the variable bounds determines the sign of the coefficient of α in the cutting plane constraints. The sign of the coefficient is then used to determine whether the variable's upper or lower bound will be used in calculating the constraint bounds.

Expanding (4.1) for cutting plane constraint ACD: $-s + r + (\check{s} - \hat{r})\alpha \leq 0$ we have:

$$\mathrm{LB}_{\mathrm{ACD}} = \begin{cases} \check{r} - \hat{s} + \check{s} - \hat{r} & \text{if } \check{s} - \hat{r} < 0 \text{ and } \hat{\alpha} = 1, \text{or} \\ & \phantom{\text{if }} \check{s} - \hat{r} > 0 \text{ and } \check{\alpha} = 1 \\ \check{r} - \hat{s} & \text{if } \check{s} - \hat{r} < 0 \text{ and } \hat{\alpha} = 0, \text{or} \\ & \phantom{\text{if }} \check{s} - \hat{r} > 0 \text{ and } \check{\alpha} = 0 \end{cases} \tag{4.2}$$

Examining (4.2) it is clear that $\mathrm{LB}_{\mathrm{ACD}}$ cannot decrease by increases in $\check{s}$ and $\check{r}$ or decreases in $\hat{s}$ and $\hat{r}$.

Similarly, the following expansion of (4.1) for constraint ABC: $-s + (\check{s} - \check{r})\alpha \leq -\check{r}$ shows that $\mathrm{LB}_{\mathrm{ABC}}$ can not be loosened:

$$\mathrm{LB}_{\mathrm{ABC}} = \begin{cases} -\hat{s} + \check{s} - \check{r} & \text{if } \check{s} - \check{r} < 0 \text{ and } \hat{\alpha} = 1, \text{or} \\ & \phantom{\text{if }} \check{s} - \check{r} > 0 \text{ and } \check{\alpha} = 1 \\ -\hat{s} & \text{if } \check{s} - \check{r} < 0 \text{ and } \hat{\alpha} = 0, \text{or} \\ & \phantom{\text{if }} \check{s} - \check{r} > 0 \text{ and } \check{\alpha} = 0 \end{cases}$$

The expansion of constraint ADE: $r + (\hat{s} - \hat{r})\alpha \leq \hat{s}$ shows that it is also true that $\mathrm{LB}_{\mathrm{ADE}}$ cannot be loosened.

$$\mathrm{LB}_{\mathrm{ADE}} = \begin{cases} \check{r} + \check{s} - \hat{r} & \text{if } \hat{s} - \hat{r} < 0 \text{ and } \hat{\alpha} = 1, \text{or} \\ & \phantom{\text{if }} \hat{s} - \hat{r} > 0 \text{ and } \check{\alpha} = 1 \\ \check{r} & \text{if } \hat{s} - \hat{r} < 0 \text{ and } \hat{\alpha} = 0, \text{or} \\ & \phantom{\text{if }} \hat{s} - \hat{r} > 0 \text{ and } \check{\alpha} = 0 \end{cases}$$

Replacing α with its complement

The above analysis describes the happy circumstance of everything working out. However, the constraints in the test data actually used $\overline{\alpha} = 1 - \alpha$ and had the form

$$\begin{array}{rcrcrcl} -T_s(x_2) & + & T_r(x_1) & + & \hat{T}\overline{\alpha} & \leq & \hat{T} \\ -T_s(x_1) & + & T_r(x_2) & + & \hat{T}\overline{\alpha} & \leq & \hat{T} \\ \alpha_{1,2} & + & \alpha_{2,1} & - & \phi_{1,2} & \leq & 1 \end{array}$$

This switch to the complement allowed constraint bounds to loosen when variable bounds are tightened.

Substituting α for $\overline{\alpha}$, the cutting plane equations become

$$\begin{array}{lrcrcrcl} ACD: & -s & + & r & + & (\hat{r} - \check{s})\alpha & \leq & \hat{r} - \check{s} \\ ABC: & -s & & & + & (\check{r} - \check{s})\alpha & \leq & -\check{s} \\ ADE: & & & r & + & (\hat{r} - \hat{s})\alpha & \leq & \hat{r} \end{array}$$

Expanding (4.1) for constraint ACD: $-s + r + (\hat{r} - \check{s})\alpha \geq \hat{r} - \check{s}$ we have:

$$\mathrm{LB}_{\mathrm{ACD}} = \begin{cases} -\hat{s} + \check{r} + \hat{r} - \check{s} & \text{if } \hat{r} - \check{s} < 0 \text{ and } \check{\alpha} = 1, \text{or} \\ & \phantom{\text{if }} \hat{r} - \check{s} > 0 \text{ and } \hat{\alpha} = 1 \\ -\hat{s} + \check{r} & \text{if } \hat{r} - \check{s} < 0 \text{ and } \check{\alpha} = 0, \text{or} \\ & \phantom{\text{if }} \hat{r} - \check{s} > 0 \text{ and } \hat{\alpha} = 0 \end{cases}$$

We see that tightening $\hat{r}$ or $\check{s}$, $\mathrm{LB}_{\mathrm{ACD}}$ can be loosened.

For example, assume that

$$\begin{array}{rcccr} 800 & \leq & s & \leq & 1800 \\ 300 & \leq & r & \leq & 700 \\ 1 & \leq & \alpha & \leq & 1 \end{array}$$

If $\hat{r}$ is tightened from 700 to 600, $\mathrm{LB}_{\mathrm{ACD}}$ is loosened from -1600 to -1700. However, since b_{ACD} is also reduced by 100 the variable bounds calculated are not changed as a result of loosening the constraint bound. This can be seen using (3.3) and (3.4) from page 29:

$$\begin{aligned} \Delta\check{s} &= -\Delta b_{\mathrm{ACD}} + \Delta \mathrm{LB}_{\mathrm{ACD}} \\ &= -(-100) + (-100) \\ &= 0 \end{aligned}$$

$$\begin{aligned} \Delta\hat{r} &= \Delta b_{\mathrm{ACD}} - \Delta \mathrm{LB}_{\mathrm{ACD}} \\ &= -100 - (-100) \\ &= 0 \end{aligned}$$

$$\begin{aligned} \Delta\hat{\alpha} &= \Delta b_{\mathrm{ACD}} - \Delta \mathrm{LB}_{\mathrm{ACD}} \\ &= -100 - (-100) \\ &= 0 \end{aligned}$$

Any loosening of $\mathrm{LB}_{\mathrm{ACD}}$ as a result of tightening $\check{s}$ is similarly cancelled out by the reduction of b_{ACD}.

Expanding (4.1) for constraint ABC: $-s + (\check{r} - \check{s})\alpha \leq -\check{s}$ we have:

$$\mathrm{LB}_{\mathrm{ACD}} = \begin{cases} -\hat{s} + \hat{r} - \check{s} & \text{if } \check{r} - \check{s} < 0 \text{ and } \check{\alpha} = 1, \text{or} \\ & \phantom{\text{if }} \check{r} - \check{s} > 0 \text{ and } \hat{\alpha} = 1 \\ -\hat{s} & \text{if } \check{r} - \check{s} < 0 \text{ and } \check{\alpha} = 0, \text{or} \\ & \phantom{\text{if }} \check{r} - \check{s} > 0 \text{ and } \hat{\alpha} = 0 \end{cases}$$

which can be loosened by tightening $\check{s}$. However, any loosening of $\mathrm{LB}_{\mathrm{ABC}}$ as a result of tightening $\check{s}$ is cancelled out by the reduction of b_{ABC}.

Expanding (4.1) for constraint ADE: $r + (\hat{r} - \hat{s})\alpha \leq \hat{r}$ we have:

$$\mathrm{LB}_{\mathrm{ACD}} = \begin{cases} \check{r} + \hat{r} & \text{if } \check{r} - \check{s} < 0 \text{ and } \check{\alpha} = 1, \text{or} \\ & \quad \check{r} - \check{s} > 0 \text{ and } \hat{\alpha} = 1 \\ \check{r} & \text{if } \check{r} - \check{s} < 0 \text{ and } \check{\alpha} = 0, \text{or} \\ & \quad \check{r} - \check{s} > 0 \text{ and } \hat{\alpha} = 0 \end{cases}$$

which can be loosened by tightening $\hat{r}$. However, any loosening of $\mathrm{LB}_{\mathrm{ADE}}$ as a result of tightening $\hat{r}$ is cancelled out by the reduction of b_{ADE}.

Since in all cases, $\Delta\mathrm{LB}_i = \Delta b_i$ it is sufficient to store the changed constraint bound without propagating it. Therefore, the arc consistency routines need not pay any attention to the changing α coefficient when propagating variable bound changes.

This chapter described the theory of how the cutting plane constraints are integrated into *bonsai*. There were also several assumptions underlying the implementation of *bonsai* which were broken when the cutting planes were introduced. These are described in Appendix A. The computational results of the integration are presented in the next chapter.

Chapter 5

Computational Results

5.1 Applying Bonsai to Digital Hardware Synthesis

Bonsai is tested on problems arising from research into automated synthesis of register-transfer level digital logic. These problems can be characterized as deterministic machine scheduling problems with *variable-duration* activities and other complicating constraints.

In digital hardware design, the scheduling problem is phrased in terms of activities which must be executed by hardware components. An algorithmic description of the behaviour which is to be implemented is translated into a data flow graph – a representation of the algorithm in terms of data values flowing between activities that transform the values. The activities in the data flow graph are the jobs which must be performed. The components made available to construct the implementation are the machines.

More formally, a set of n activities $x_a, a = 1, ..., n$, must be performed on a set of m components $f_d, d = 1, ..., m$. The components f_d can only perform one activity at a time.

In a digital circuit, the output of a component is *not* stable; it is simply a voltage asserted on a wire, and the component must continue to assert the voltage for as long as the output is in use. The components f_d are assumed to be combinational logic. By definition, when the inputs to combinational logic change, the output will change after a specified propagation delay. This means that the active interval during which a component f_d will be occupied with an activity x_a is *not* a constant. The inputs i_a required by an activity x_a must be held until the outputs o_a are no longer required. Although a minimum length for the interval can be developed [7], the maximum length of the interval depends on how long the output is required as an input to components executing other activities.

Formally, variables must be defined for the start and end of the intervals when inputs, outputs, and activities are active. For example, for an activity x_a, let the variable $T_s(x_a)$ represent the time when the activity starts, and the variable $T_r(x_a)$ represent the time when it ends. The interval required to perform activity x_a on some component is then the interval $[T_s(x_a), T_r(x_a)]$.

In order to release a component and inputs occupied in producing a value $o_{a,c}$, a storage activity can *optionally* be introduced which captures and holds the value. A component s_e is required to implement a storage activity, and it remains in use as long as the value is required as input to some activity. A component s_e is assumed to be sequential logic; by definition, it is capable of retaining a value after the value has been withdrawn from its input.

To account for the option of storing outputs in an implementation, the problem definition for digital hardware design needs to be extended. A set of n activities $x_a, a = 1, ..., n$ must be performed on a set of m components $f_d, d = 1, ..., m$ and a set of optional activities storing $o_{a,c}, a = 1, ..., n, c = 1, ..., k_a$, may be performed on a set of m' components $s_e, e = 1, ..., m'$.

The constraint system can be viewed as consisting of three sets of constraints:

- One set of constraints propagates activity (job) start times forward through the network of activities.
- One set of constraints propagates activity (job) end times backward through the network of activities.
- One set of constraints ensures that the active intervals of two activities (jobs) assigned to the same component (machine) are serialized.

In a synthesis problem it is desirable to minimize both the time required to execute an algorithm and the number of components required for the implementation. These are competing objectives; generally a weighted sum of the two is acceptable as an objective function. This differs from the usual view of machine scheduling, which assumes a fixed set of machines, but is appropriate for many problems which arise in practice (*e.g.* determining both the fleet size and the schedule for a fleet of trucks).

5.2 Cutting Plane Results

The constraint systems used to test *bonsai* are generated by *iddma* [5] — a dedicated constraint generation program — from a data flow description of the algorithm to be implemented, a description of the set of available components, and a specification of the allowable uses for components. Statistical descriptions of the test examples are presented in tables 5.1 and 5.2 while their data flow diagrams appear in figures 5.1, 5.2 and 5.3.

	Data Flow Activities	Operators	Data Flow Values	Storage Elements
CrissX	4	6	6	3
Logic	5	7	8	3
Ralphl	10	13	12	7

Table 5.1: Summary of example problem descriptions

	Constraints		Variables		Non-Zero Coefficients	
	Without Cutting Planes	With Cutting Planes	Continuous	Boolean	Without Cutting Planes	With Cutting Planes
CrissX	179	259	41	77	488	728
Logic	237	345	45	106	671	995
Ralphl	577	779	78	222	1672	2308

Table 5.2: Summary of example problem constraint systems

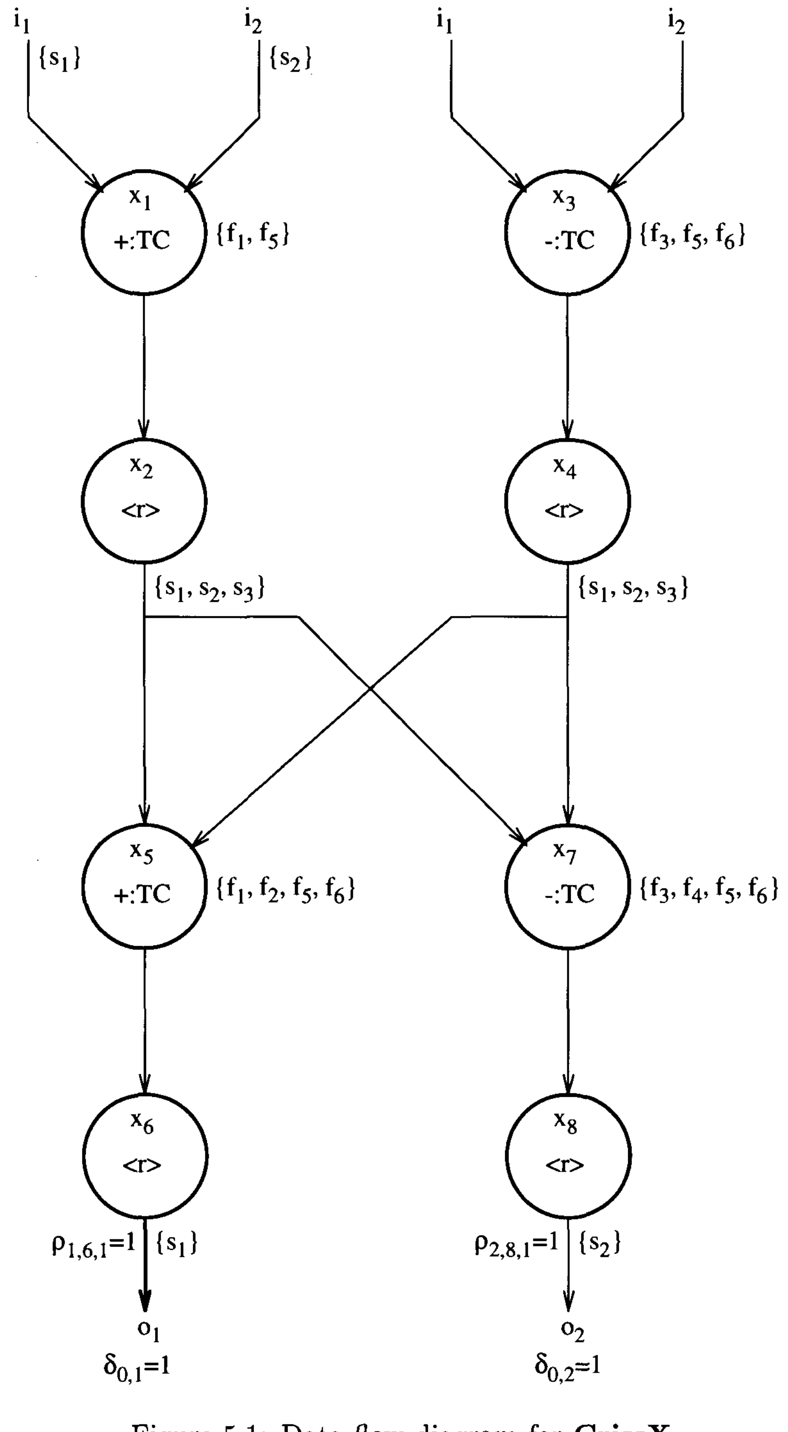

Figure 5.1: Data flow diagram for **CrissX**

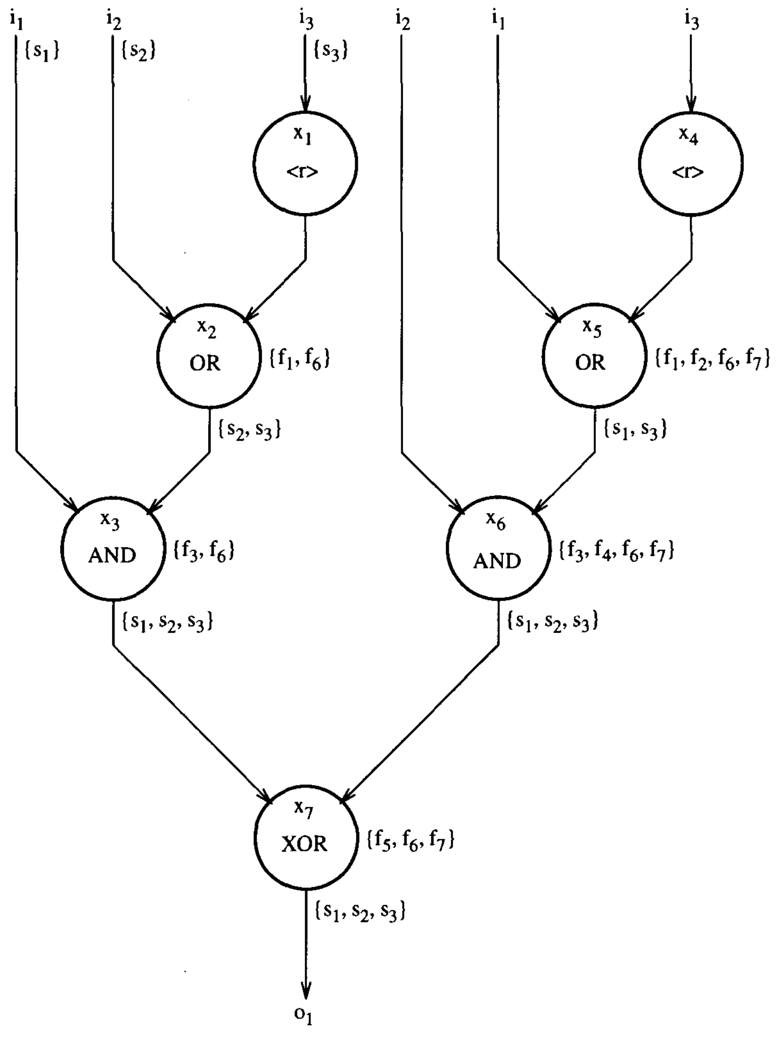

Figure 5.2: Data flow diagram for **Logic**

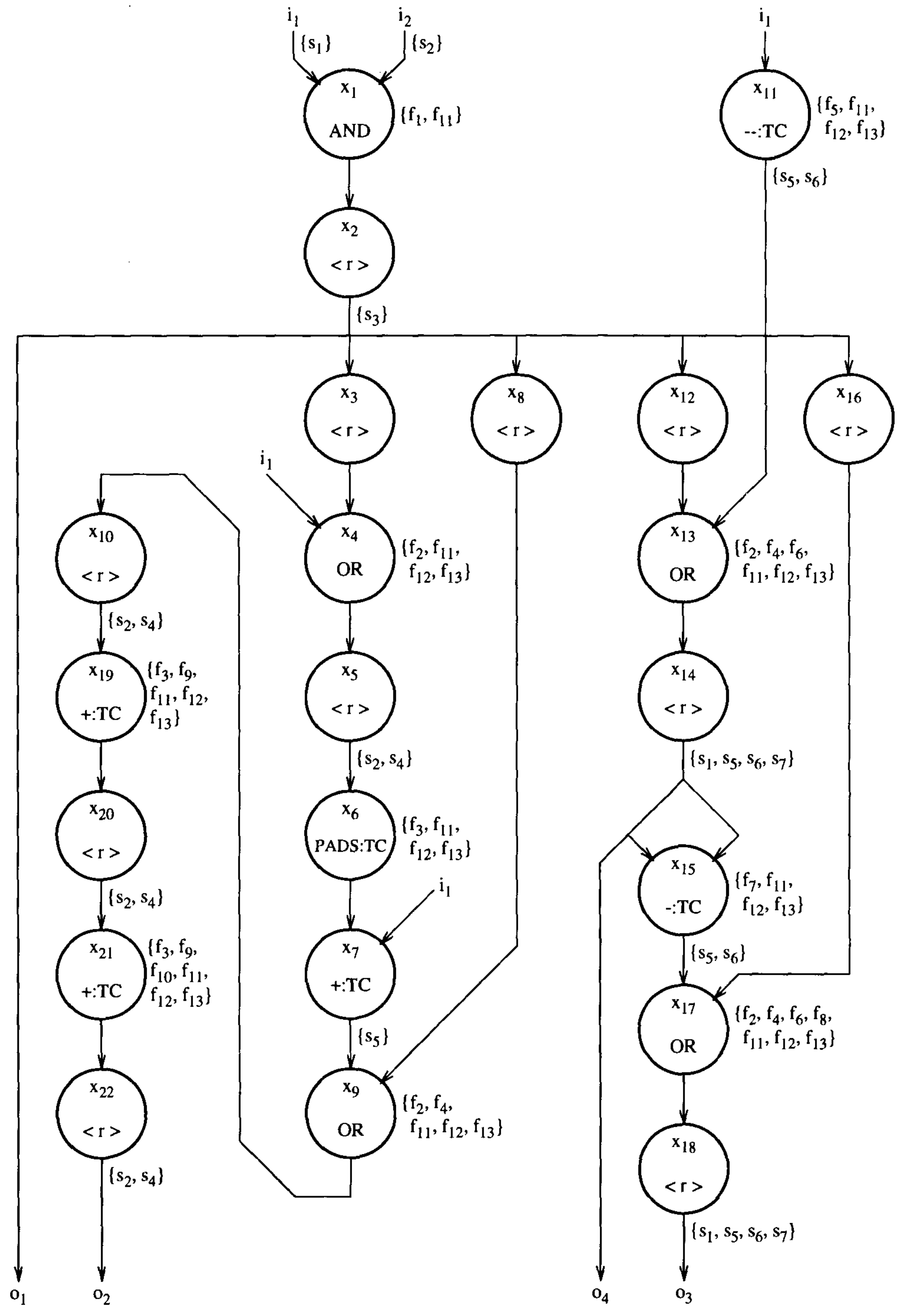

Figure 5.3: Data flow diagram for **Ralph1**

Objective Functions

There are three main types of objective functions.

- *Cost* objectives emphasize the cost of the implementation, where cost is defined to be the total cost of the components used.
- *Time* objectives emphasis the time required to complete the algorithm, where completion time is based on the time at which the final outputs of the algorithm become available to the world.
- Implementation cost and completion time are competing objectives, in the sense that serialization of the activities in the algorithm is generally required to minimize hardware, whereas maximal parallelism tends to minimize completion time. The final class of objectives, Cost/Time, attempt to balance the two by scaling the terms in the objective function so that one unit of time appears equivalent to one unit of cost.

Within each class of objective there are four functions. Three of these incorporate timing information in an attempt to distinguish solutions which would otherwise be equivalent[1].

For the Cost objectives, a small completion time component is added to all four functions to compress the schedule into the minimum time. The variations, and their codes, are as follows:

C (Cost)

Component costs dominate the objective. Completion times are incorporated with a scaling factor which limits their sum to roughly one order of magnitude smaller than the cost terms.

CI (Cost with Active Interval)

Component costs dominate the objective. Completion times and the start and release times (the active interval) for activities are incorporated with a scaling factor which limits their sum to roughly one order of magnitude smaller than the cost terms.

[1] *I.e.* there would be equivalent solutions if two independent activities could have their execution order swapped, but the implementation might still have the same cost and completion time. If, however, the objective function were to slightly favour one ordering over the other, it might be possible to fathom one solution by bound a bit earlier in the search.

CS (Cost with Start Time)

Component costs dominate the objective. Completion times and the start times for activities are incorporated with a scaling factor which limits their sum to roughly one order of magnitude smaller than the cost terms.

CSS (Cost with Scaled Start Time)

Component costs dominate the objective. Completion times and the start times for activities are incorporated with a scaling factor which limits their sum to roughly one order of magnitude smaller than the cost terms. Based on examination of the data flow for the algorithm, the weights on the start times are varied so as to prefer one order over another in cases where two activities have no data dependencies to force an ordering.

In the Time objectives, a small cost component is added to prevent the incorporation of unused hardware into the solution. The variations are as described for the Cost objectives.

The variations for the Cost/Time objectives also follow the Cost pattern.

Branching Strategies

There are nine strategy variations, some loosely based on different philosophies of hardware design and some included simply to round out the test suite. The one common feature which deserves separate explanation is that variables of class δ will always be in the first priority class. These variables represent the decision as to whether an input will obtain the value it needs directly from the combinational logic which produces it, or from a stored copy of the value. In hardware terms, this represents a tradeoff between keeping combinational logic in use to produce a value versus using a storage component to hold the value and freeing the logic for other uses. From inspection of the constraint relations, this decision is also crucial to tightening the portion of the constraint system dealing with release times, and previous work has shown that it's pointless to place it in a lower priority class.

There is little meaning in the codes used for the strategies. NG stands for No Guidance and is actually meaningful. The remainder of the codes stem from Modified No Guidance, which makes sense only in a historical context.

NG $\{\delta, \sigma, \rho, \alpha, \phi, \chi, \beta\}$

All variables are placed in a single priority class.

MNG1 $\{\delta, \sigma, \rho, \alpha, \phi\}, \{\chi, \beta\}$

The χ and β variables, which represent decisions on whether or not to allow the use of a particular component, are placed in a lower class. The underlying rationale is that decisions on which parts to use should be a consequence of other design decisions, and not drive them.

MNG2 $\{\delta, \phi\}, \{\sigma, \rho, \alpha\}, \{\chi, \beta\}$

The ϕ variable controls whether or not activities which could potentially be executed in parallel (given sufficient hardware) will be forced into a serial order. This is a fairly important decision which can have significant impact on the amount of hardware required in the implementation.

MNG3 $\{\delta\}, \{\phi, \alpha\}, \{\rho, \sigma\}, \{\beta, \chi\}$

The α variables control the order in which serialized activities will be executed. One can argue that it only makes sense to decide on the order at the same time as the decision on whether or not to serialize. ρ and σ variables control, respectively, the allocation of combinational logic to computational activities and storage elements to stored values. In an hardware design context, this priority says that scheduling decisions should drive component allocation.

MNG4 $\{\delta\}, \{\rho, \sigma\}, \{\phi, \alpha\}, \{\beta, \chi\}$

This ordering reflects the philosophy that component allocation decisions should drive the scheduling of activities.

MNG5 $\{\delta\}, \{\sigma, \rho, \alpha, \phi\}, \{\chi, \beta\}$

This ordering places equal importance on scheduling and allocation.

MNG6 $\{\delta, \phi, \alpha\}, \{\rho, \sigma\}, \{\beta, \chi\}$

MNG3, with the variation that scheduling decisions are given equal importance with input value access decisions.

MNG7 $\{\delta\}, \{\phi, \alpha\}, \{\beta, \chi\}, \{\rho, \sigma\}$

MNG3, but experiment with the idea of deciding what components to use before actually assigning them to specific activities.

MNG8 $\{\delta\}, \{\beta, \chi\}, \{\phi, \alpha\}, \{\rho, \sigma\}$

Decide what components to use, then schedule the activities, and finally decide on the details of how components should be assigned to activities.

Data

Tables 5.3, 5.4, and 5.5 report the number of subproblems which were solved in order to completely explore the branch-and-bound search tree and the average pivots required by each subproblem. Each data point in these tables is an average over 4 objective functions.

Figures 5.4 and 5.5 show the *CrissX* total subproblems and pivots of each objective function averaged over the 9 branching strategies. Figures 5.6 and 5.7 show the *Logic* total subproblems and pivots of each objective function averaged over the 9 branching strategies. Figures 5.8 and 5.9 show the computational results of *Ralph1* using the NG branching strategy. Figures 5.10 and 5.11 show the computational results of *Ralph1* using the MNG4 branching strategy. Figures 5.12 and 5.13 show the computational results of *Ralph1* using the MNG8 branching strategy.

Branching Strategy	Objective Function Emphasis	Without Cutting Planes: Subproblems Defined	Without Cutting Planes: Pivots/ Subproblem	With Cutting Planes: Subproblems Defined	With Cutting Planes: Pivots/ Subproblem
NG	Cost	42	19	54	29
	Time	11	25	5	46
	Cost/Time	184	12	229	21
MNG1	Cost	42	19	64	26
	Time	11	25	5	46
	Cost/Time	184	12	233	24
MNG2	Cost	42	19	101	23
	Time	11	25	11	32
	Cost/Time	184	12	256	20
MNG3	Cost	42	19	73	28
	Time	11	25	9	35
	Cost/Time	184	12	306	21
MNG4	Cost	42	19	191	18
	Time	11	25	8	37
	Cost/Time	184	12	144	22
MNG5	Cost	42	19	71	27
	Time	11	25	9	34
	Cost/Time	184	12	138	25
MNG6	Cost	42	19	66	28
	Time	11	25	5	46
	Cost/Time	184	12	338	20
MNG7	Cost	42	19	72	28
	Time	11	25	10	35
	Cost/Time	184	12	303	21
MNG8	Cost	42	19	72	25
	Time	11	25	9	34
	Cost/Time	184	12	110	26
Average	Cost	42	19	85	24
	Time	11	25	8	37
	Cost/Time	184	12	228	22

Table 5.3: CrissX Results

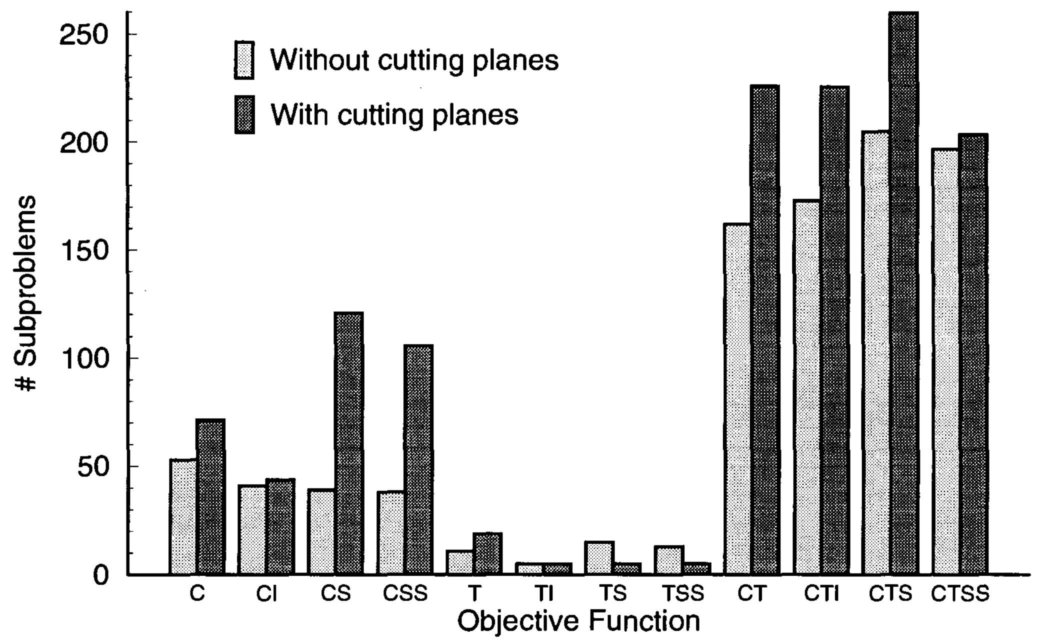

Figure 5.4: **CrissX** Subproblems — averaged over all branching strategies

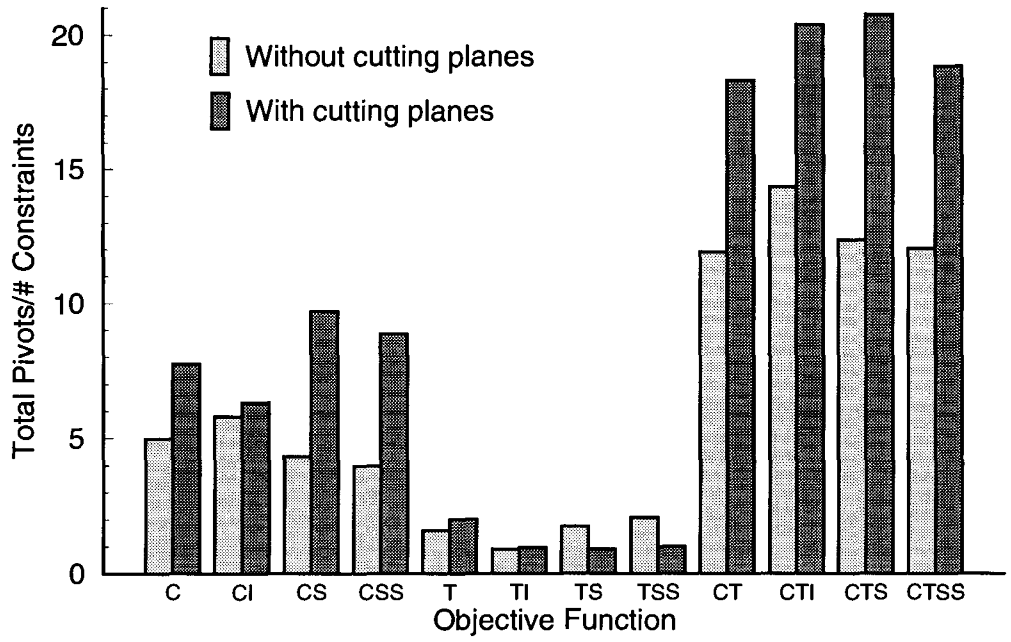

Figure 5.5: **CrissX** Pivots — averaged over all branching strategies

Branching Strategy	Objective Function Emphasis	Without Cutting Planes		With Cutting Planes	
		Subproblems Defined	Pivots/ Subproblem	Subproblems Defined	Pivots/ Subproblem
NG	Cost	55	22	105	25
	Time	7	62	1	241
	Cost/Time	16	33	14	44
MNG1	Cost	80	24	109	26
	Time	7	62	1	241
	Cost/Time	16	33	14	44
MNG2	Cost	104	21	111	23
	Time	7	62	1	241
	Cost/Time	19	32	14	42
MNG3	Cost	168	15	108	24
	Time	7	62	1	241
	Cost/Time	18	33	14	44
MNG4	Cost	77	86	81	32
	Time	15	32	1	241
	Cost/Time	28	26	30	34
MNG5	Cost	78	84	95	31
	Time	7	62	1	241
	Cost/Time	17	32	14	44
MNG6	Cost	108	20	104	23
	Time	7	62	1	241
	Cost/Time	18	33	14	44
MNG7	Cost	155	16	127	25
	Time	7	62	1	241
	Cost/Time	11	48	30	37
MNG8	Cost	84	20	73	30
	Time	7	62	1	241
	Cost/Time	18	31	20	46
Average	Cost	101	30	101	26
	Time	7	55	1	241
	Cost/Time	18	32	18	41

Table 5.4: Logic Results

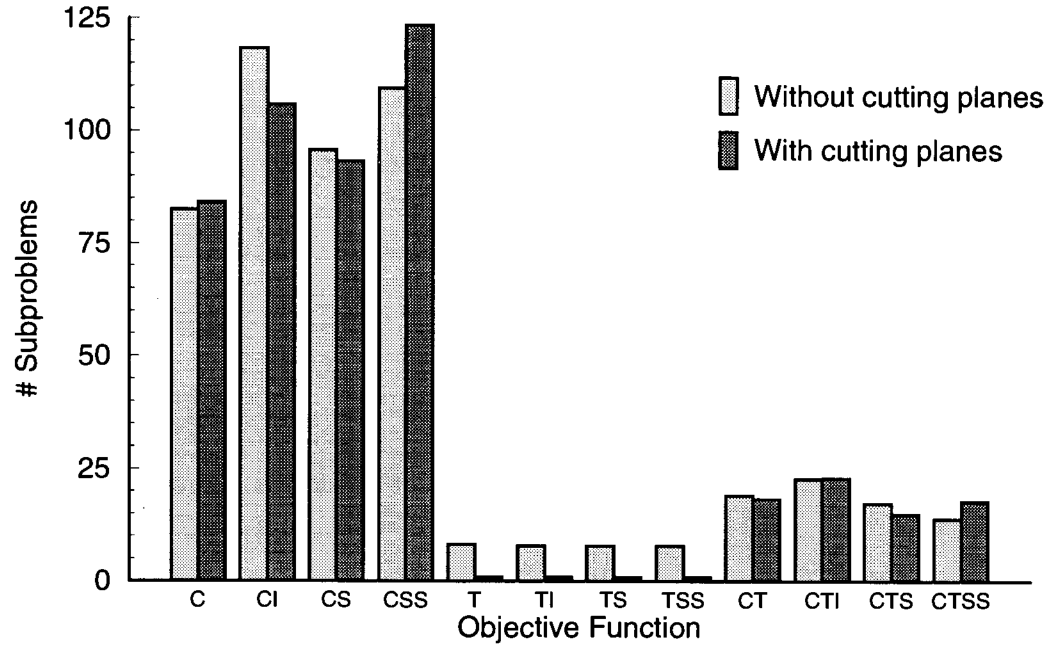

Figure 5.6: **Logic** Subproblems — averaged over all branching strategies

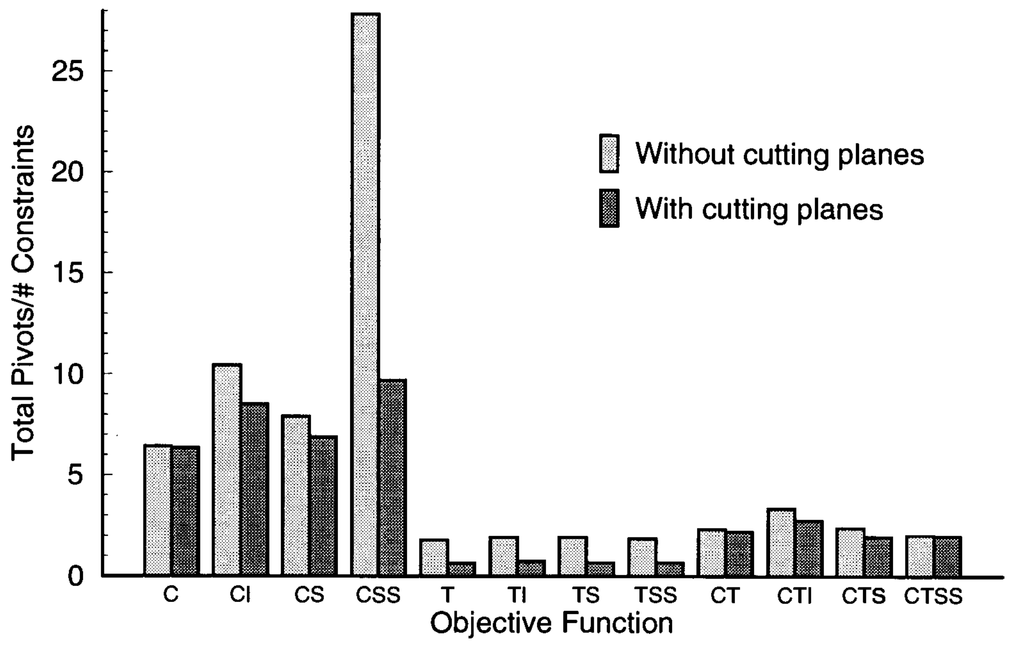

Figure 5.7: **Logic** Pivots — averaged over all branching strategies

Branching Strategy	Objective Function	Without Cutting Planes: Subproblems Defined	Without Cutting Planes: Pivots/ Subproblem	With Cutting Planes: Subproblems Defined	With Cutting Planes: Pivots/ Subproblem
NG	Cost	576	48	2936	38
	Time	136	41	69	79
	Cost/Time	14372	53	11091	88
MNG4	Cost	394	47	787	58
	Time	113	42	72	79
	Cost/Time	3180	52	656	125
MNG8	Cost	623	37	426	86
	Time	179	33	65	64
	Cost/Time	1393	64	689	123
Average	Cost	531	43	1383	47
	Time	143	38	69	74
	Cost/Time	6315	53	4145	92

Table 5.5: Ralph1 Results

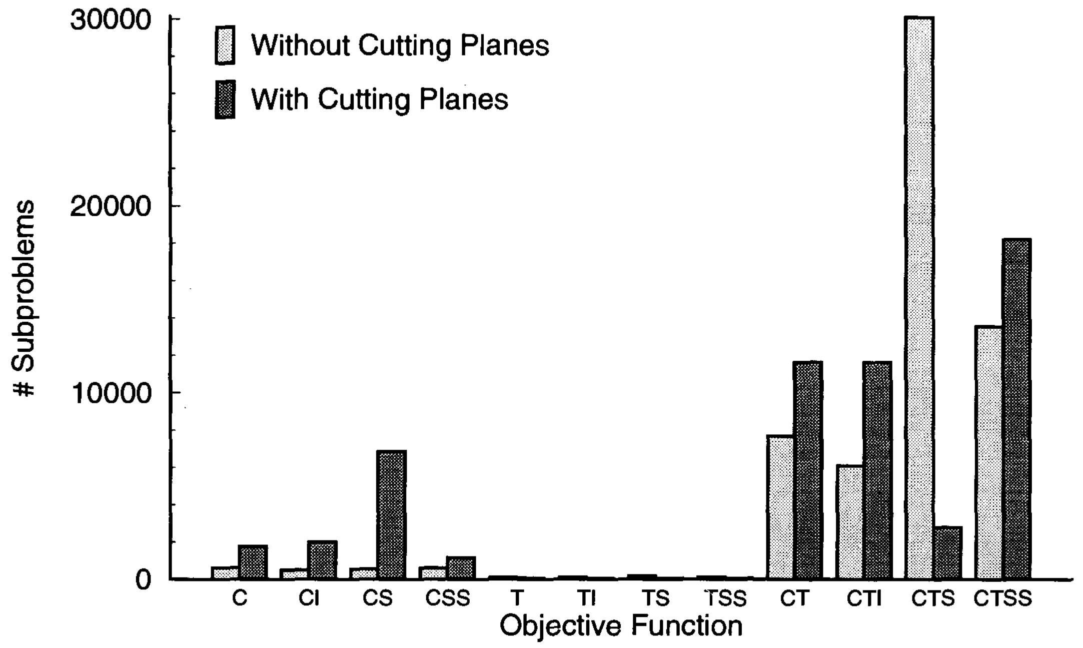

Figure 5.8: **Ralph1** Subproblems — NG branching strategy

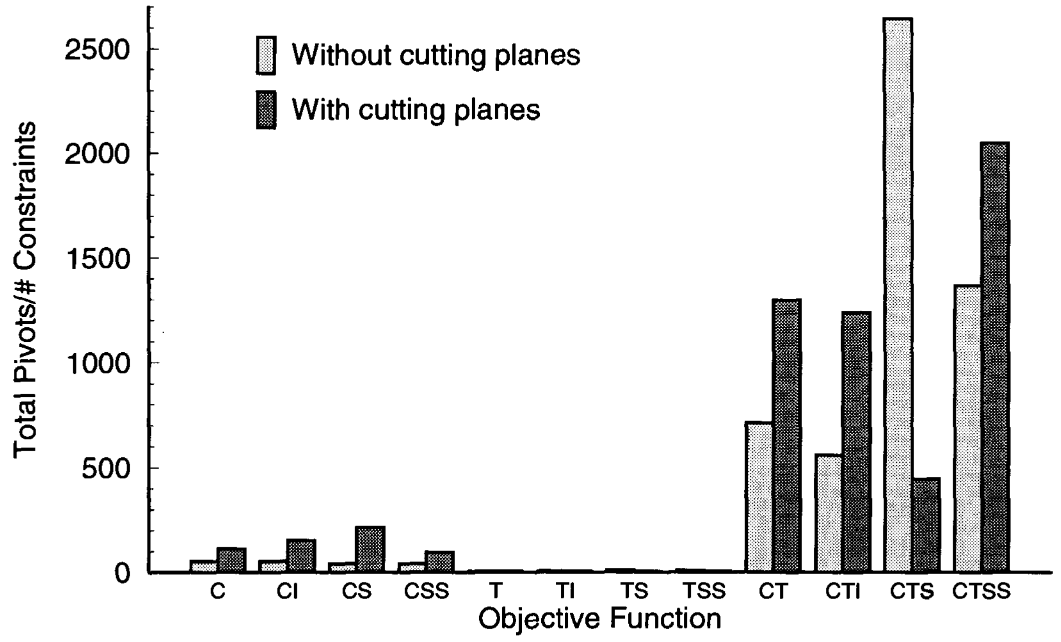

Figure 5.9: **Ralph1** Pivots — NG branching strategy

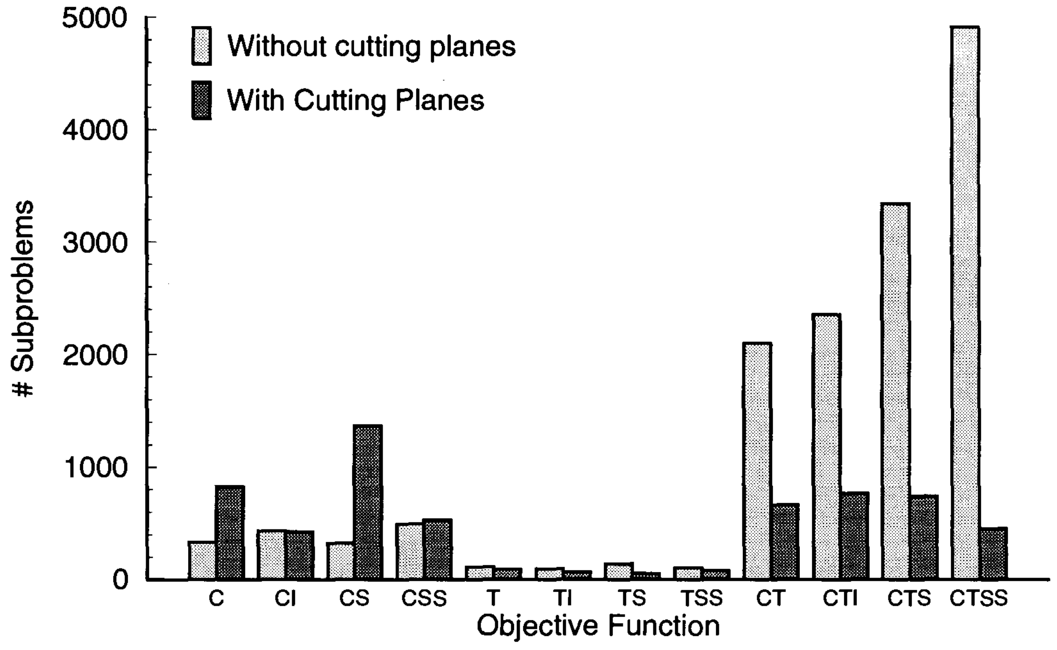

Figure 5.10: **Ralph1** Subproblems — MNG4 branching strategy

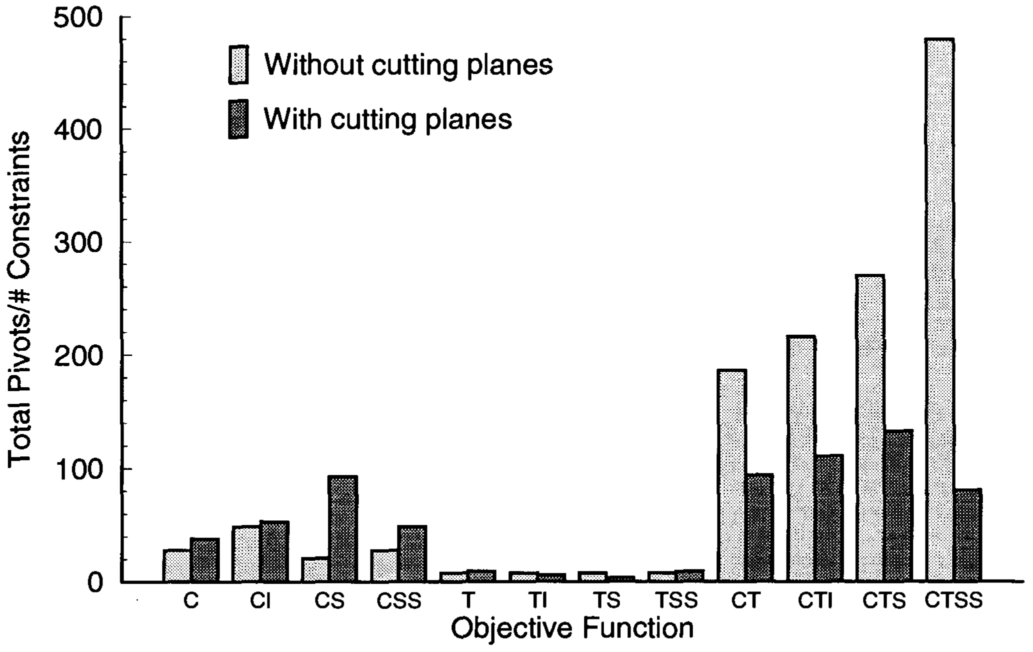

Figure 5.11: **Ralph1** Pivots — MNG4 branching strategy

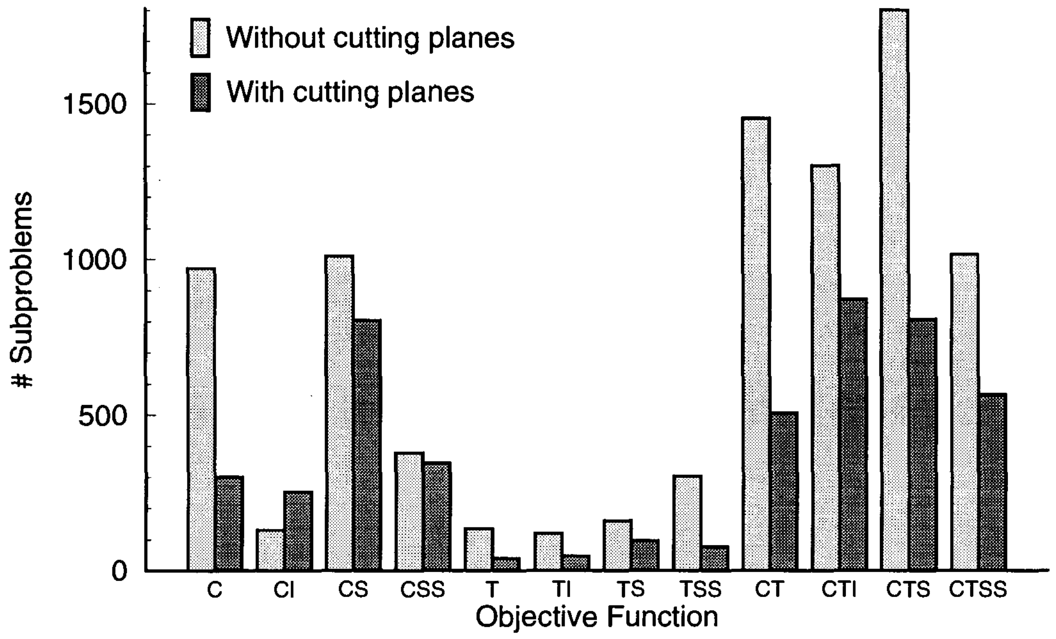

Figure 5.12: **Ralph1** Subproblems — MNG8 branching strategy

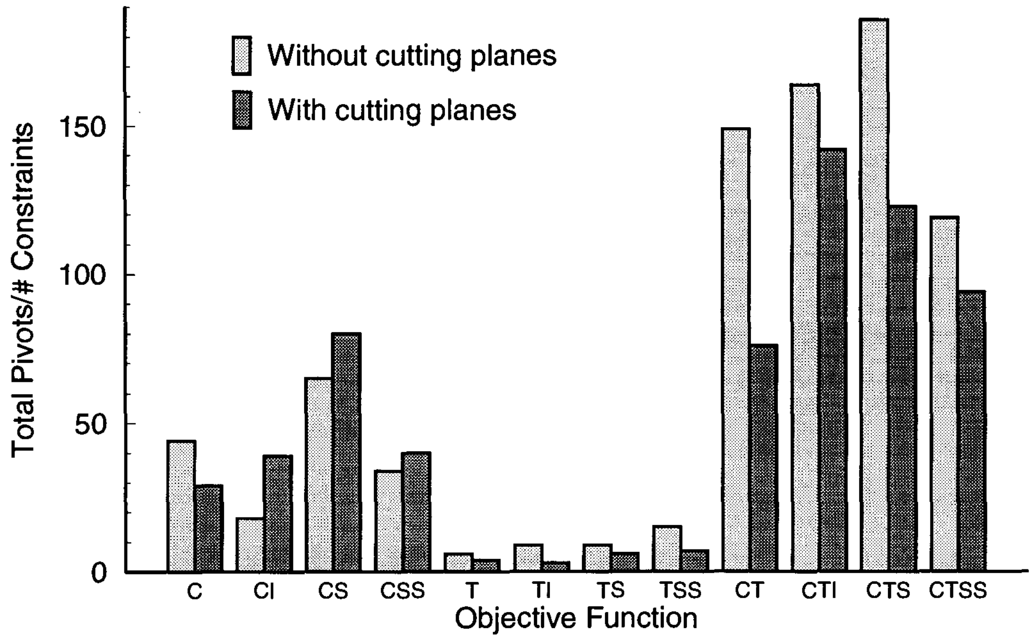

Figure 5.13: **Ralph1** Pivots — MNG8 branching strategy

Chapter 6

Conclusion and Future Work

The addition of cutting planes caused the average pivot counts to increase substantially. This is likely due to the large increases in the number of constraints in the system (seen in table 5.2). Were *bonsai* to be rewritten to allow constraints to be added and removed dynamically[1], it would not be necessary to have all cutting plane constraints in effect at all times and we believe that the number of pivots would be more in line with previous results.

The number of subproblems defined is a crucial performance measure since at least one relaxation LP is run for each one. The number of subproblems for the small *CrissX* and *Logic* examples is, on average, about the same; possibly a little bit worse. The larger *ralph1* problem shows a significant improvement in the number of subproblems in most cases and also an improvement on average.

As the size of the example increases, the value of $\hat{T}$ increases since it has to be proportional to the length of the serialized schedule. Since the cutting planes provide us with a method of tightening constraints of the form

$$-s + r - \hat{T}\alpha \leq 0$$

it makes sense that the cutting planes will become more effective as $\hat{T}$ becomes larger.

[1]Note that this is a fairly common practice and would be relatively easy given an appropriate data structure for the constraint matrix. The packed array, used by the XMP linear programming library routines [10, 11], is adequate if you only use column major, or only use row major order, however, *bonsai*'s arc consistency routines access the structure by both row and column while the simplex routines access the structure by column.

Previously, the *ralph1* examples having the CSS, CT, CTI, CTS, and CTSS objective functions were hard, *i.e.* they involved a huge branch-and-bound search tree. We have managed to improve almost all of these — the exceptions being in the NG strategy where there is no prioritization of variables for branching.

Note that the MNG4 and MNG8 strategies force the allocation of resources before performing the scheduling. Since we have a marked improvement for the Ralphl example using these strategies, it appears that the cutting plane constraints are doing their job of deciding on the order of activity execution.

6.1 Further research

- We originally started generating cutting planes for three-dimensional systems since we wanted to be able to visualize what was happening. The next step would be to design cutting planes for the seven-dimensional serialization constraint system as described as (1.2) on page 9. From there we can go into even higher dimensions for more powerful cutting planes.

- By analyzing which cutting plane constraints are active in the region near the optimal solution and throwing out the rest, we should be able to significantly reduce the number of pivots required to solve each relaxation.

  Recall the cutting plane equations from chapter 2:

$$\begin{array}{rrcrcrcl}
ACD: & -s & + & r & + & (\check{s}-\hat{r})\alpha & \leq & 0 \\
ABC: & -s & & & + & (\check{s}-\check{r})\alpha & \leq & \check{r} \\
ADE: & & & r & + & (\hat{s}-\hat{r})\alpha & \leq & \hat{s}
\end{array}$$

  If the constraint's slack variable is basic, then the constraint is not active and can probably be removed from the constraint system.

The following tests can determine whether it would be useful to add a cutting plane constraint to the constraint system:

- If α is fixed at 0 or 1 then ACD, ABC, and ADE are all redundant.
- If $\check{s} \geq \hat{r}$ then ACE, ABC, and ADE are all redundant.
- If $\check{s} = \check{r}$ then ABC is redundant since it is equivalent to $s \geq \hat{s}$.
- If $\hat{s} = \hat{r}$ then ADE is redundant since it is equivalent to $r \leq \hat{r}$.
- If $\check{r} < \check{s}$ then ABC is redundant (observation 1 page 34).
- If $\hat{r} < \hat{s}$ then ADE is redundant (observation 2 page 35).

Appendix A

Broken Underlying Assumptions

The constraint system is static

Bonsai assumes that the constraint system is static. It is stored in packed arrays in both row major and column major order to facilitate the arc consistency routines' need to use rows and columns and the simplex routines' need to look at columns. Each time a cutting plane constraint needed to be rewritten it was necessary to traverse and update both these structures.

For this reason, it was necessary to have all possible cutting planes in the constraint system at all times instead of only those that were active and non-redundant. A linked list structure which could be accessed in either order would facilitate dynamic editing of the constraint system.

Static b vector

Due to restrictions in the simplex routines used in *bonsai*, fixing a boolean variable to 1 is done by replacing the variable with its complement - which means fixing it to 0 and subtracting its coefficient from the right hand side. Because of this and the fact that the α coefficient and the b value of cutting plane constraints are both dependent on the variable bounds, maintaining a consistent right hand side is a major headache. This is especially true since there are actually three b vectors in *bonsai*: the actual right hand side, the reduced *b* vector of the parent subproblem, and the reduced b vector of the current subproblem.

The arc consistency routines only look at the actual right hand side. Each time the variable bound change propagation routine runs, the *b* value is updated for each affected cutting plane constraint in the actual b vector.

Once the arc consistency routines are finished and flagged variables have been fixed, the current subproblem's *b* vector is rewritten.

While restoring a node for further exploration, the variable bounds are restored. Then all cutting plane constraints, the actual *b* vector, and the parent's *b* vector are all corrected to match the restored bounds. The subproblem's *b* vector starts out as a copy of its parent's.

Branching variables can be selected before restoration

Branching variables are selected *before* a node is restored. However, if an incumbent solution was found between the time the node was created and the time it was restored, the restoration is likely to fix a lot of variables since the objective function value of the incumbent will be propagated during restoration. With the addition of cutting plane constraints, there are cases where all of the branching variables become fixed. Since the code is not set up to look for new branching variables at this point, we chose to allow *degenerate* tours — the parent is expanded to have a single child.

b vector branching corrections can be pre-calculated

The reductions to the right hand side, as a result of fixing branching variables, used to be calculated when the branching variables were selected. It is now necessary to wait until after the parent node has been restored.

Only non-zero coefficients are stored

Bonsai only stored non-zero coefficients. However, it is possible for the α coefficient of the cutting plane constraints to become zero. Short of copying the whole coefficient matrix to close up gaps, it is now necessary to store zero coefficients. Thus, we must test coefficient values before use in several situations. For example, in the arc consistency routines we do not want to divide by $a_{i,t}$ if it is zero. Nor can 0 coefficients be passed to the LA05 basis maintenance package.

Coefficients must be finite

Variable domains used to be initialized to [0, MAX_DOUBLE] so the α coefficient of cutting plane constraint ACD was initialized to MAX_DOUBLE. This caused all sorts of arithmetic exceptions when calculating matrix inverses since since MAX_DOUBLE^2 cannot be computed. It is therefore necessary that all coefficients have *reasonable* finite values. To ensure this, the input file *must* indicate a finite upper bound on *all* variables.

Boolean variable domains are not tightened

Bonsai assumed that the boolean variable domains remained $[0,1]$ until fixed by branching, monotones, or penalty calculations. It was hard coded that fixing a boolean variable to its upper bound meant it should be set to 1. It is necessary to use the actual variable bounds.

Variable status is not changed after a node is stored

The status of a variable — whether it was basic, non-basic, artificial, etc. — was assumed to remain static for a node after the node was visited and stored. We discovered instances where the status of variables changed upon restoring a node.

The status vector of the root node is stored and a series of status edits record the differences between the status vector of the parent and that of the child. Since the code did not check for status differences during resurrection of a node status changes were not recorded as status edits. In the case where there was no status edit for that basis position in a descendent, the variable occupying the position was random, resulting in a singular basis and excess pivots to regain feasibility and optimality.

Bibliography

[1] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. Technical Report 576, Management Science Research Group, Graduate School of Industrial Administration, Carnegie Mellon University, 1991.

[2] E. Balas, S. Ceria, and G. Cornuéjols. Solving mixed 0-1 programs by a lift-and-project method. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 232–242, 1993.

[3] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Library Reference Manual.* Waterloo Maple Software, 1991.

[4] V. Chvátal. *Linear Programming.* W.H. Freeman and Company, 1983.

[5] L. Hafer. Logic synthesis by mixed-integer linear programming. Technical Report TR 88-2, School of Computing Science, Simon Fraser University, May 1988.

[6] L. Hafer. Bonsai - teaching a clown to prune trees. Technical Report CMPT TR 91-6, School of Computing Science, Simon Fraser University, July 1991.

[7] L. Hafer. Constraint improvements for MILP-based hardware synthesis. In *ACM/IEEE 28th Design Automation Conference Proceedings*, pages 14–19. ACM SIGDA, IEEE Computer Society-DATC, June 1991.

[8] L. Hafer and E. Hutchings. Bringing up bozo. Technical Report TR90-2, School of Computing Science, Simon Fraser University, August 1990.

[9] S. M. Lee, L. J. Moore, and B. W. Taylor III. *Management Science.* Wm. C. Brown Publishers, second edition, 1981.

[10] R. Marsten. The design of the XMP linear programming library. *ACM Transactions on Mathematical Software*, 7(4):481–497, December 1981.

[11] R. Marsten. *XLP Technical Reference Manual.* XMP Software, Inc., P.O. Box 13185, Tucson, Arizona, 85732, 1987.

[12] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley and Sons Inc., 1988.

[13] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[14] G. Sidebottom. Satisfaction of constraints on non-negative integer arithmetic expressions. Open File Report 1990-15, Alberta Research Council, 6815 8th Street N.E., Calgary, AB, T2E 7H7, July 1990.