CENTER FOR
PARALLEL OPTIMIZATION

SOLUTION OF MULTIPLE-CHOICE KNAPSACK PROBLEM ENCOUNTERED IN HIGH-LEVEL SYNTHESIS OF VLSI CIRCUITS

by

Renato De Leone, Rajiv Jain and Kenneth Straus

Computer Sciences Technical Report #980

November 1990

Solution of Multiple-Choice Knapsack Problem Encountered in High-Level Synthesis of VLSI Circuits

Renato De Leone
Center for Parallel Optimization
Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706

Rajiv Jain
Electrical and Computer Engineering Department
University of Wisconsin
1415 Johnson Drive
Madison, WI 53706

Kenneth Straus
Center for Parallel Optimization
Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706

Solution of Multiple-Choice Knapsack Problem Encountered in High-Level Synthesis of VLSI Circuits

Abstract

In computer-aided design, performance of digital designs can be enhanced by transformations to the input specification. the purpose of these transformations is to reduce total chip area or chip delay. Examples of such transformations are tree-height reduction and hierarchical decomposition. Using existing formal predictive models of cost and performance, the impact of these transformations on the design implementation can be evaluated. In this paper we address the question of what transformations should be applied and how many times should each transformation be applied in order to achieve an optimal chip design. We formulate the above problem as a multiple-choice knapsack problem and propose a Lagrangian relaxation technique for finding an approximate solution. Two simple but effective post-optimal heuristics which improve the relaxation solution are also discussed. Results for several randomly generated problems indicate that the proposed approach is highly effective. In almost all cases considered a design with 97% or above of optimality is achieved.

Keywords: High-level synthesis, digital design, computer-aided design of digital systems, multiple-choice knapsack problem, integer programming, Lagrangian relaxation

Subject Classification: Programming: integer, heuristics, relaxation.

1 Introduction

The mapping of a behavioral description (an algorithm expressed as a data flow graph or a hardware descriptive language) to a register-transfer level (RTL) design is known as high-level synthesis. An example data flow graph and an RTL design which can implement this data flow graph are given in Figures 1 and 2 respectively [1]. A variety of candidate RTL designs can implement a given behavioral description. However, not all designs are of equal quality. A design implementation is inferior when there exists at least one other implementation which performs better in one or more figures of merit, all other figures of merit being at least equal. All designs lie within a boundary in the *design space*, which in our context is a 2-dimensional plane with area and delay as its two axes[2]. A *design point* represents a design in this space. In the design space, the cheapest (least area) and the fastest designs delimit the design space boundary for a given behavioral description and design library (see Table 1 for an example design library). A plot of the area versus delay of the non-inferior design points gives a lower-bound area-delay tradeoff curve for the input description.

Let us consider the behavioral description given in Figure 1a in the form of a data flow graph. Several non-inferior pipelined designs for this input description are shown in Figure 3 (marked as "before"). Each point represents the area-delay characteristics of an RTL design which will satisfy the input behavioral requirements. We observe in Figure 3 that a fast design has a large area while a cheap design is slow. In general a decrease in delay is achieved only at the expense of area and vice versa.

In order to achieve an area-delay efficient implementation of a given input specification, designers generally apply transformations which modify the input specification without altering the intended behavior of the final design. Examples of such transformations are tree height reduction [11] [17], dead code elimination [1] [16] [18], loop winding and loop unwinding [5], and bit-width decomposition. The effect of these transformations is to move the area-delay curves in the design space; Figure 3 shows the area-delay curves before and after applying a "decompose" transformation. Each transformation modifies the curve differently. While some move the curve towards the origin others may move it away from the origin. Furthermore, some transformations may increase (or decrease) the number of design points or may extend (or restrict) the design boundary (as in Figure 3).

[1] The RTL design was generated using a 0-1 integer linear programming formulation of data path synthesis [2].

[2] The design space is not restricted to area vs. delay alone. A dimension could be any metric important to the designer, like power or design time.

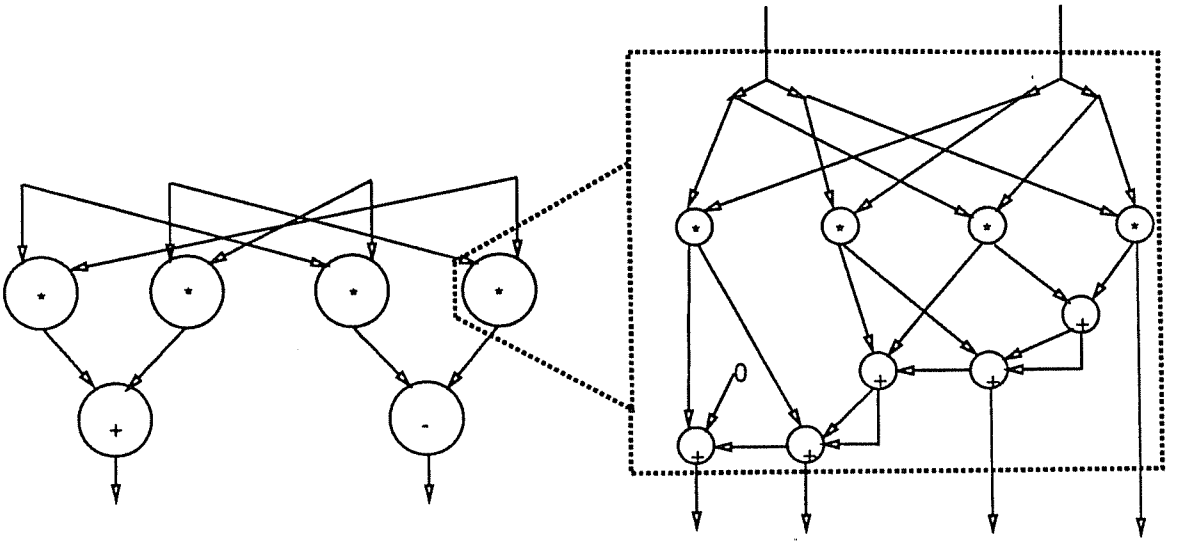

Figure 1: Hierarchical decomposition transformation

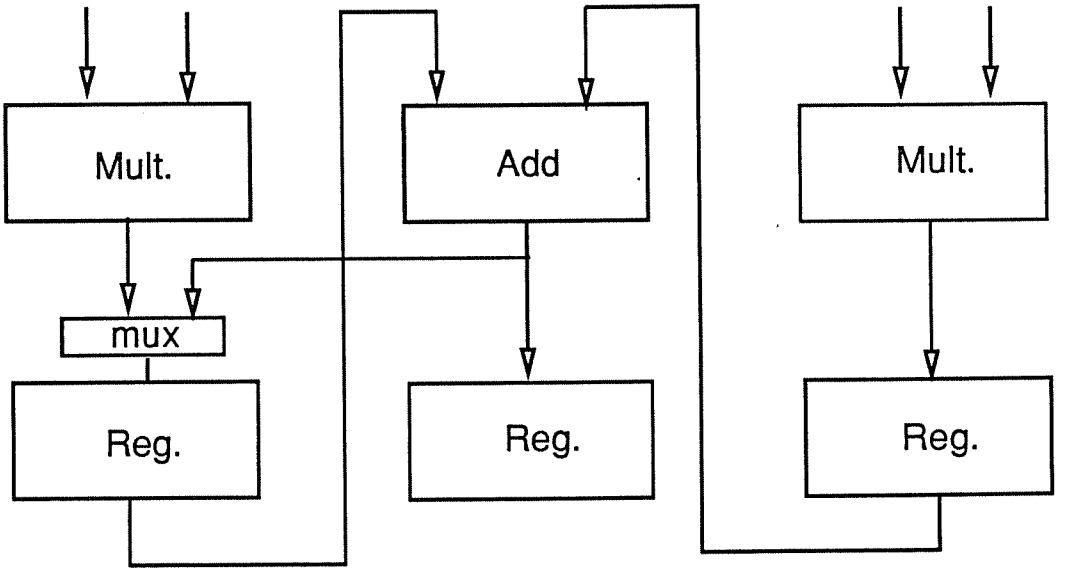

Figure 2: An RTL design

Function	Area mil^2	Delay ns	Bit Width
addition	2000	225	16
addition	4200	340	32
subtraction	4200	340	32
multiplication	13800	250	8
multiplication	49000	375	16

Table 1: Design library

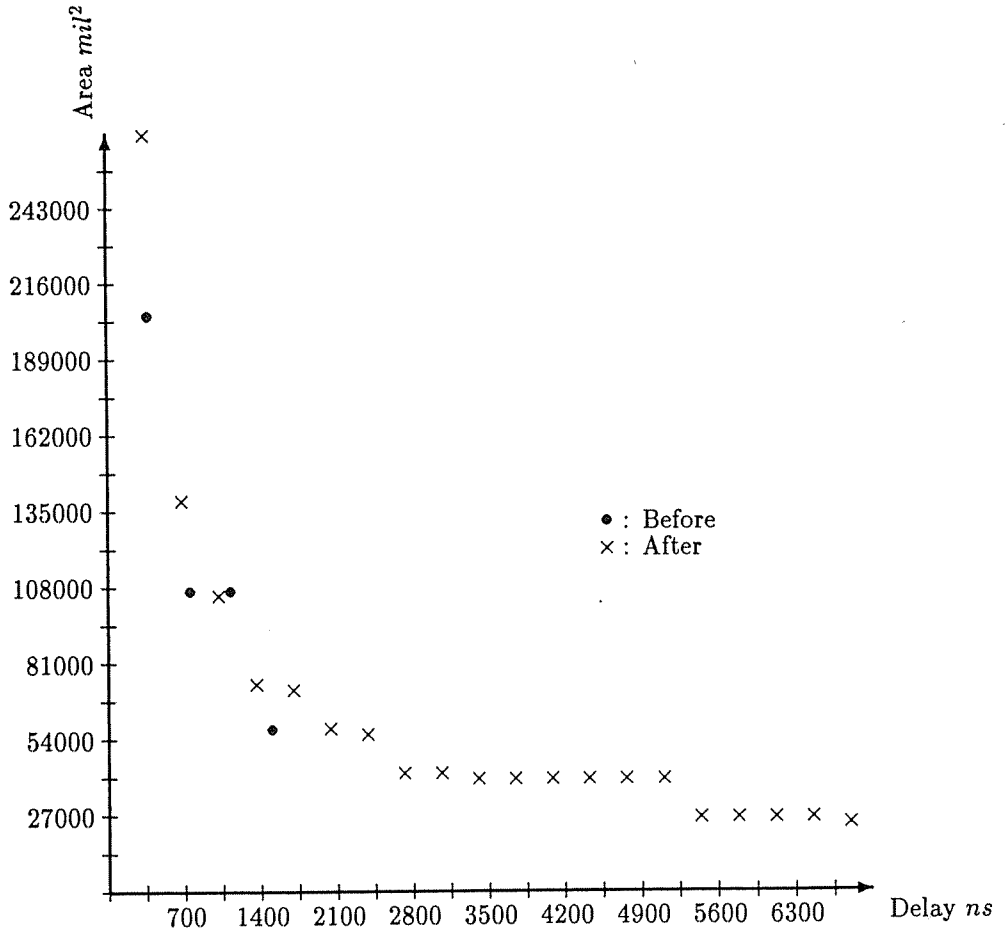

Figure 3: Hierarchical Decomposition

It is important to choose the right transformations to apply to the behavioral description before synthesis since a wrong transformation can result in a poor design inspite of efforts which may be made at the chip implementation level to improve the quality. The benefits of some transformations like dead code elimination are obvious, but for many other transformations this is not so obvious. For these transformations the impact on the area-delay characteristics can be assessed only by synthesizing all design points in the design space following each transformation (or sequence of transformations). Since the synthesis process is computationally expensive (at least $O(n^4)$ for each design point for n operations in the input description, and $O(n^5)$ for the entire design curve [14]) one could use prediction tools as an alternative means of assessing the impact [10] [9]. We will refer to these area-delay prediction tools as "AT predictors". The method proposed here is not limited to the predictors given in [10] [9]. The designer could very well choose to use some of his or her own prediction tools instead. Also, these predictors could predict measures other than area and delay.

For an example of a transformation, consider the data flow graph given in Figure 1a to be implemented as a pipelined design using 32-bit adder and 16-bit multiplier modules (Table 1). Using the AT predictor [10], we get the design curve consisting of all lower-bound design points (assuming pipelined implementation), shown in Figure 3. These points are marked as "before". Now let us apply the "decompose" transformation shown in Figure 1b and decompose all 16-bit multiplication operations into 8-bit multiplication operations. Once again, using the AT predictor we can get the area-delay curve for the transformed data flow graph. This is marked as "after" in Figure 3. As a result of the transformation, we notice that design points of the transformed data flow graph move away from the origin, i.e., towards the inferior design region. However, the fastest design offered by the transformed data flow graph is faster than any design produced by the original data flow graph. There is a penalty, however - an increase in design area. If the design constraint permits the area to be greater than $204,400\ mil^2$, or constraints the area to be less than $57,400\ mil^2$, the the transformed data flow graph offers better design solutions. Otherwise the transformation should not be applied. Thus, a transformation can be either "good" or "bad" depending upon the constraints and design goals.

This leads us to the problem which we wish to address in this paper: Given a design constraint (for example, the area of the design should not exceed 50,000 mil^2) and a pool of transformations, select the transformations that result in better design solutions and also determine the number of times each should be applied so as to achieve an area-delay efficient implementation for the given input behavioral description.

2 Literature Survey

There are four aspects to the transformation problem:

1. Defining a transformation.
2. Identifying transformations which can be applied to the given input description.
3. Computing the impact of each applicable transformation on the metrics important to the designer.
4. Finally, selecting the appropriate transformations which will help the designer in achieving design goals while satisfying the design constraints.

Snow [16], Trickey [17] and Walker [18] have researched the first aspect of the transformation problem. Snow [16] was the first to demonstrate the use of transformations in preparing the input description for synthesis. He defined the following transformations: Constant folding, insertion of no-operations, dead code elimination, redundant code elimination (also used in logic minimization as expression compaction), activity splitting (adding redundant code), and moving code to improve scheduling and resource sharing. One other important transformation he defined is the loop unwinding transformation. Many of these transformations are also used in compiler optimization [1]. The techniques described in this paper can be used, along with some measure of code optimization, to optimize compiler generated code as well.

Trickey [17] has proposed some additional transformations, the main one being tree height reduction which reduces critical path delay of the input description. A special case of the tree-height reduction where all nodes in the data flow graph are addition operations has been researched in [7]. Another recent work on transformations is by Walker [18]. Examples of transformations described in [18] are moving code in and out of *if* conditions and loops, creation of inter-processes communication operations and using the inter-processes communication operations to create pipeline stages.

Situations where these transformations can be applied and their effects on the input description in isolated examples have been discussed in [16], [17], and [18]. However, none of these references have mathematically characterized or systematically evaluated the impact of the transformations on the area-delay characteristics (or any other design characteristics) of the synthesized design.

The second aspect of the problem, i.e. identifying the transformations which can be applied to the given input description, is a non-trivial task. In this sub-problem we have to consider all possible sequences of transformations which can be applied to the input description. For example, the application of one transformation may open up possibilities for the application of another transformation. Examining every possible combination of the transformation applicability is a combinatorially exhaustive process, and how to prune the search space to limit the combinatorial growth is an open research problem. In [16], [17], and [18] the transformations are sequenced in a demand driven fashion. That is, the transformations are prioritized according to some criteria and then applied. For example, the transformation which reduces area most is applied first [17]. This technique of applying transformations is weak and leads to locally optimal solution as against globally optimal solutions. In this paper we assume that the second aspect has been performed, either manually or automatically. The techniques developed here can be used to fathom the explosive growth of this aspect of transformation problem efficiently.

The third aspect of the transformation problem was first demonstrated in [8] where some transformations have been evaluated. The evaluation technique is based on the area-delay models developed for pipelined and non-pipelined design styles. These models examine the functional module's area and delay parameters to predict a lower-bound area-delay tradeoff curve for the input description. The models, and consequently the evaluation proposed in [8], ignores the affect of register and multiplexer area on the area-delay curve. Mlinar [12] has derived a model which estimates register and multiplexer area and can be added to the model of [8] to provide an accurate measure which can be used for transformation evaluation.

The fourth aspect of the transformation problem is the selection of the transformation which would meet the design constraints and optimize the design goal, thus generating the most efficient design. It is this aspect of the problem we are concerned with in this paper. To the best of the authors knowledge this aspect of the problem has not been researched previously.

In what follows we define our problem and develop the multiple-choice knapsack formulation for the problem (Section 3). We then define the Lagrangian relaxation of the problem and use two search technique for finding the Lagrangian multiplier (Section 4). Section 5 discusses post-optimal heuristic methods employed to improve the Lagrangian relaxation solutions. Finally, some results are presented in Section 6.

3 Theoretical Foundations

In this paper we will limit ourselves to evaluating the effect of transformations on the area-delay characteristics since we have formal models to predict these measures. In the paper itself, for the sake of brevity, we will restrict ourselves to the model given in [10]. Other features that might be important to the designer are pin count, testability, and power dissipation. However, the lack of formal models at this time restrict us to area and delay measures alone. Further we will restrict ourselves to pipelined design styles using the model in [10] to characterize the area-delay quantities. Similar results for non-pipelined design style can be generated using the model in [9].

Given a pool of transformations and a design constraint along with formal models to capture the area-delay characteristics, our objective is to solve the following problem: Which transformations should be applied, and how many applications of each from the set of applicable transformations to get an area-delay efficient implementation?

3.1 Characterizing Transformations

A data flow graph transformation r_i operates on an input behavioral description B and produces a new behavioral description B' which is functionally equivalent to B. Let A and T be the lower-bound area and delay of the design produced for the behavioral description B (A and T can be computed using the model given in [10].) Similarly, let A' and T' be the lower-bound area and delay of the design produced for the behavioral description B'. In the remainder of this section we show how one might characterize a transformation, i.e. quantify the delay penalty $\Delta T = T' - T$ and the area penalty $\Delta A = A' - A$ imposed by the transformation.

Some transformations such as the decomposition transformation can be completely characterized (that is, their impact on the input description computed) with information about the module set alone. For other transformations such as the critical path reduction transformation, the actual instance of application needs to be specified. That is, each individual application of the transformation can have a different impact. Consequently each individual application instance is treated as a separate transformation.

We illustrate the characterization step with the aid of an example. Let us evaluate the impact of decomposition transformation (Figure 1) on the area-delay curves on the

pipelined design style. There are two subcases to be considered. First, if the operation which is being decomposed is not the slowest operation in the input description then the change in delay $\Delta T = T' - T = 0$ (since the clock cycle depends only on the slowest module in the pipeline and thus remains unaltered). This implies that for zero change in performance, there may be a change in the area. In the second subcase, let the the slowest operation type be decomposed. Also, let there be n slowest operations in the input description. Since the change ΔT is entirely dependent on the slowest operation in the input description, ΔT remain zero until all the n slowest operations have been decomposed.

Let k be the number of slowest operations which are decomposed, $0 \leq k \leq n$. $k = 0$ implies that none of the operations are decomposed. Let $\Delta A_k = A' - A$ and $\Delta T_k = T' - T$ be the area and delay penalty respectively, if the transformation is applied k times (note that a negative ΔA_k implies decrease in area). For $k = 4$, $T' = maximum(d_{8mult}, d_{16adder}, d_{32adder}, d_{32subtractor})$, for $k < 4$ $T' = maximum(d_{8mult}, d_{16adder}, d_{16mult}, d_{32adder}, d_{32subtractor})$, and $T = maximum(d_{16\ mult}, d_{32adder}, d_{32subtractor})$. For our example,

$$\Delta T_k = \begin{cases} T' - T & \text{if } k = n \\ 0 & \text{otherwise} \end{cases}$$

and,

$$\Delta A_k = k(4a_{8mult} + 5a_{16adder} - a_{16mult})$$

where, a_{8mult}, $a_{16adder}$ and a_{16mult} are the areas of the 8-bit multiplier, 16-bit adder and 16-bit multiplier respectively. Using these values from Table 1, we have $\Delta A_k = 16,200k$, and $\Delta T_1 = \Delta T_2 = \Delta T_3 = 0$ and $\Delta T_4 = -35$. We observe that for the decomposition transformation ΔA_k and ΔT_k can be computed based on the module set parameters and hence are completely defined by the module set.

3.2 Combining Transformations

So far we have shown how to characterize the effect of a single transformation on a single design point of the area-delay curve. In this section we will outline the formulation of the problem whereby we select only those transformations which lead to efficient implementation and determine the number of times to apply them.

Denote the set of all possible transformations by $R = \{r_1, r_2, \cdots r_K\}$. Assume that design area constraint is A_{max} and the design goal is to minimize delay (i.e. maximize

throughput)[3]. Let ΔA_{ik} and ΔT_{ik} be the area and delay penalty of applying k times the transformation r_i. Then, the problem **P** of selecting appropriate transformations and determine the number of times to apply each transformation is formulated as a multiple-choice knapsack problem [15] as follows:

$$\text{minimize} \quad \sum_{i=1}^{K}\sum_{k=1}^{d_i}(Y_{ik} \times \Delta T_{ik})$$

$$\text{subject to} \quad \begin{cases} \sum_{i=1}^{K}\sum_{k=0}^{d_i}(Y_{ik} \times \Delta A_{ik}) \leq A_{max} - A \\ \sum_{k=0}^{d_i} Y_{ik} = 1, \quad i = 1, 2, \cdots, K \\ Y_{ik} = \{0, 1\}, \quad k = 1, \cdots, d_i;\ i = 1, \cdots, K \end{cases}$$

where d_i is the maximum number of times transformation r_i can be applied on the input description. The first constraint ensures that the area constraint imposed by the designer is not violated while remaining constraints ensure that transformation r_i is either not applied, in which case $Y_{i0} = 1$, or applied exactly k times, in which case $Y_{ik} = 1$.

In the remainder of this section we will derive an equivalent ILP (Integer Linear Programming) formulation of the 0-1 ILP problem given above. A Lagrangian relaxation of the ILP formulation is then obtained.

Let $t_i(x_i)$ be the savings in time when transformation r_i is applied x_i times ($t_i(x_i) = -\Delta T_{ix_i}$). Similarly, let $a_i(x_i)$ be the area penalty (increase) when transformation r_i is applied x_i times ($a_i(x_i) = \Delta A_{ix_i}$). Note that x_i are non-negative integers and are bounded above by d_i, the maximum number of times transformation r_i can be applied. The area increase cannot exceed $b = A_{max} - A$. Furthermore, we assume that for $i = 1, \ldots, K$:

$$t_i(0) = 0, \quad t_i(x_i) \geq 0 \text{ when } x_i > 0, \quad \text{and} \quad t_i(d_i) > 0$$
$$a_i(0) = 0, \quad a_i(x_i) \geq 0 \text{ when } x_i > 0, \quad \text{and} \quad a_i(d_i) > 0$$

We assume $b > 0$ and, for all i, t_i and a_i are increasing (not necessarily strictly). An equivalent formulation of our problem is:

[3]A similar problem can be formulated for a delay constraint and a goal of minimizing area.

$$
\begin{array}{rcll}
z_{\mathrm{P}} & := & \text{maximize} & \displaystyle\sum_{i=1}^{K} t_i(x_i) \\
& & \text{subject to} & \displaystyle\sum_{i=1}^{K} a_i(x_i) \le b \\
& & & 0 \le x_i \le d_i,\ x_i \text{ integer } i = 1, \ldots, K
\end{array}
\tag{1}
$$

Under these assumptions, a solution of (1) always exists. Note that no assumptions of linearity or continuity are made for any of our functions; in fact, they are only defined on a discrete set. To further simplify the notation we define

$$
\begin{aligned}
x &:= [x_1, x_2, \ldots, x_{\mathrm{K}}], & t(x) &:= \sum_{i=1}^{K} t_i(x_i) \\
d &:= [d_1, d_2, \ldots, d_{\mathrm{K}}], & a(x) &:= \sum_{i=1}^{K} a_i(x_i) \\
S &:= \{x :\ 0 \le x_i \le d_i,\ x_i \text{ integer } i = 1, \ldots, K\}
\end{aligned}
$$

The set S is finite with $|S| = \prod_{i=1}^{K}(d_i + 1)$. The problem (1) can be rewritten in more compact form as

$$
\begin{array}{rcll}
z_{\mathrm{P}} & := & \text{maximize} & t(x) \\
& & \text{subject to} & a(x) \le b \\
& & & x \in S
\end{array}
\tag{2}
$$

4 Relaxed Lagrangian Approach

In this section we define the Lagrangian relaxation of (2) and two methods of solving the relaxation.

Let u be a nonnegative real variable. The Lagrangian relaxation of problem (2) is defined as:

$$
z_{\mathrm{LR}}(u) := \max_{x \in S}\ t(x) - u\,(a(x) - b) \tag{3}
$$

and its Lagrangian dual is given by,

$$z_{\text{LD}} := \min_{u \geq 0} z_{\text{LR}}(u) \tag{4}$$

Theorem 4.1 (Weak Duality [13]) *For all $u \geq 0$, $z_{\text{LR}}(u) \geq z_{\text{P}}$ and therefore $z_{\text{LD}} \geq z_{\text{P}}$.*

The theorem states that z_{LD} is an upper bound for z_{P}, the optimal value of problem (2). Let x^u denote a solution of the Lagrangian relaxation (3) for a fixed u, that is,

$$x^u := \operatorname*{argmax}_{x \in S} \left(t(x) - u(a(x) - b) \right) \tag{5}$$

The next lemma [3] shows that x^u solves a perturbed version of our original problem:

Lemma 4.1 (Right-hand Side Perturbation) *$\forall u \geq 0$, x^u solves the following problem:*

$$\begin{array}{ll} \text{maximize} & t(x) \\ \text{subject to} & \left\{ \begin{array}{c} a(x) \leq a(x^u) \\ x \in S \end{array} \right. \end{array}$$

Corollary 4.1 *If for some $\hat{u} \geq 0$ $a(x^{\hat{u}}) = b$, then $x^{\hat{u}}$ solves (2).*

Corollary 4.2 *Let u_1, $u_2 \geq 0$ and x^* be a solution of (2). Then, $b \leq a(x^{u_1}) \leq a(x^{u_2})$ implies that $t(x^*) \leq t(x^{u_1}) \leq t(x^{u_2})$. Also, $a(x^{u_1}) \leq a(x^{u_2}) \leq b$ implies that $t(x^{u_1}) \leq t(x^{u_2}) \leq t(x^*)$.*

Corollary 4.2 shows that the difference $a(x^u) - b$ is a measure of error between $t(x^u)$ and optimal solution z_{P}. More precisely, if we define

$$\begin{array}{rll} \hat{u}_{\text{feas}} := \operatorname*{argmin}_{u \geq 0} & b - a(x^u) \\ \text{s.t.} & b \geq a(x^u) \end{array} \tag{6}$$

and

$$\hat{u}_{\text{infeas}} := \begin{array}{ll} \underset{u \geq 0}{\text{argmin}} & a(x^u) - b \\ \text{s.t.} & a(x^u) \geq b \end{array} \tag{7}$$

and let $x^{\text{feas}} := x^{\hat{u}_{\text{feas}}}$ and $x^{\text{infeas}} := x^{\hat{u}_{\text{infeas}}}$, then, among all feasible x^u, $t(x^{\text{feas}}) \geq t(x^u)$ and, similarly, for all infeasible x^u we have $t(x^u) \geq t(x^{\text{infeas}})$. That is, x^{feas} and x^{infeas} are the best feasible and infeasible solutions of (2).

Note that, for a fixed value of u problem (3) is easy to solve. First, since b and u are constants within the problem, the term ub can be dropped. Next, the problem can be decoupled into the following K maximization subproblems, each of which can be solved by simple enumeration. For $i = 1, \ldots, K$, solve:

$$\begin{array}{ll} z_i'(u) := \max & t_i(x_i) - u a_i(x_i) \\ \text{s.t.} & 0 \leq x_i \leq d_i, \text{ and integer,} \end{array}$$

and sum the solutions of the K subproblems:

$$z_{\text{LR}}(u) = \sum_{i=1}^{K} z_i'(u) + ub$$

Furthermore, being independent of each other, these subproblems can be solved in parallel. Thus, if we can find $\hat{u}_{\text{feas}}$ and $\hat{u}_{\text{infeas}}$ the relaxation of (2) can be quickly solved. The following lemma [3] can be used to find $\hat{u}_{\text{feas}}$ and $\hat{u}_{\text{infeas}}$.

Lemma 4.2 *Let f and g be any real-valued functions defined on a set W. For $u \geq 0$, let x^u be a solution of*

$$\max_{x \in W} f(x) - ug(x).$$

Then $u_2 \geq u_1$ implies that $g(x^{u_2}) \leq g(x^{u_1})$.

The above lemma shows that $a(x^u)$ is a decreasing function[4] of u (an example of $a(x^u)$ is shown in Figure 4).

[4]The solution x^u of (3) may not be unique. Thus, it appears that $a(x^u)$ is not well-defined as a function. The difficulty is only apparent; we discuss this in Remark 4.2.2.

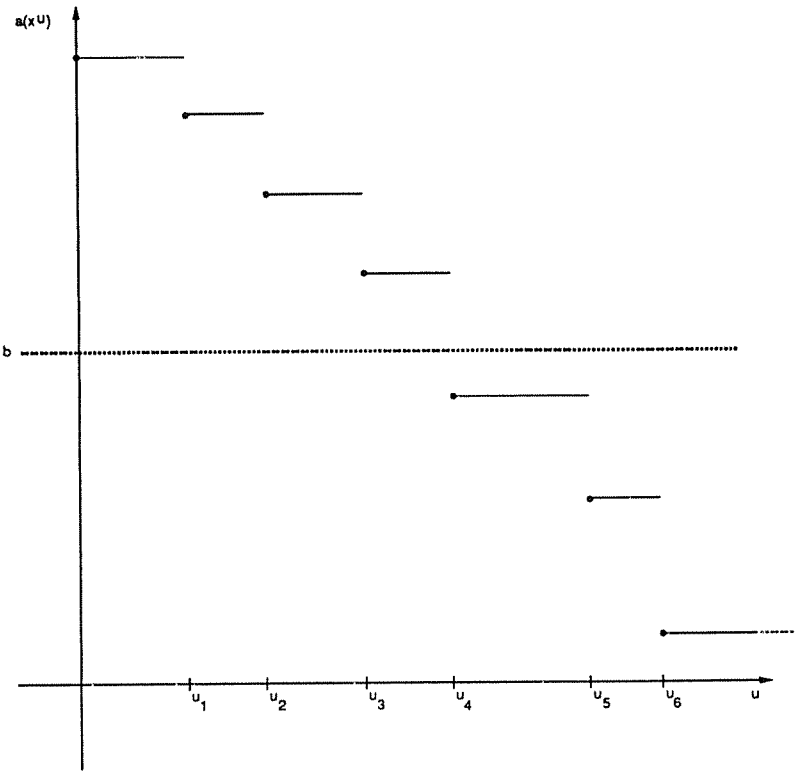

Figure 4: Graph of $a(x^u)$

4.1 Bisection Search Technique

Thus, a natural way to find a $\hat{u}$ such that $a(x^{\hat{u}}) = b$ (or, more likely, best near-solutions $\hat{u}_{\text{feas}}$ and $\hat{u}_{\text{infeas}}$) is to use a bisection method (also known as binary search technique) [4]. Furthermore, since $a(x^u)$ is a decreasing function of u, those u for which x^u is infeasible are on the left of $\hat{u}$ (or $\hat{u}_{\text{infeas}}$); we label them u_{L}. Those u for which x^u is feasible are on the right of $\hat{u}$ (or $\hat{u}_{\text{feas}}$); we label them u_{R}. The Bisection Algorithm terminates either with $\hat{u}_{\text{infeas}}$ and $\hat{u}_{\text{feas}}$ (as the final values of u_{L} and u_{R}, respectively), or with $\hat{u}$.

Bisection Algorithm

Step 0: $u_{\text{L}} := 0, u_{\text{R}} := u'$, choose $\epsilon > 0$

Step 1: $\bar{u} := \dfrac{u_{\text{L}} + u_{\text{R}}}{2}$

Step 2: solve (3) with $\bar{u}$ as lagrangian multiplier.

Step 3: if $a(x^{\bar{u}}) = b$ then stop. /* precise optimal solution */
Step 4: if $a(x^{\bar{u}}) < b$ then $u_{\mathrm{R}} := \bar{u}$; else $u_{\mathrm{L}} := \bar{u}$
Step 5: if $u_{\mathrm{R}} - u_{\mathrm{L}} < \epsilon$ then stop; else go to 1.

When $u = 0$, (3) is a relaxation of (2) and if $a(x^0) \leq b$ then x^0 is optimal for (2). Apart from this (unlikely and fortunate) possibility, in which the area constraint is actually superfluous, we can assume that $a(x^0) > b$. Thus, the choice $u_{\mathrm{L}} = 0$ in Step 0 of the Bisection Algorithm is appropriate.

Remark 4.1.1 *The quantity u' in Step 0 of the Bisection Algorithm must be chosen so that $a(x^{u'}) < b$. A possible choice is*

$$u' = \frac{t(d)}{\min\limits_{x:a(x)>0} a(x)} + 1.$$

To see this, let

$$S^0 := \{x \in S : a(x) = 0\} \quad \text{and} \quad S' := \{x \in S : a(x) \neq 0\}$$

and

$$a_{\min} := \min_{x \in S'} a(x)$$

Since the function $t(x)$ is non-decreasing in each component, we have that $t(d) \geq t(x)$, $\forall x \in S$. Therefore:

$$\frac{t(d)}{a_{\min}} \geq \frac{t(x)}{a(x)}, \ \forall x \in S'$$

Let $u' := \frac{t(d)}{a_{\min}} + 1$. Then

$$u' > \frac{t(x)}{a(x)}, \ \forall x \in S'$$

or equivalently,

$$t(x) - u'a(x) < 0, \ \forall x \in S'. \tag{8}$$

$\forall x \in S^0$, we have

$$t(x) - u'a(x) = t(x) \geq 0 \tag{9}$$

By (8) and (9), the solution $x^{u'}$ of

$$\max_{x \in S} t(x) - u'a(x)$$

must be an element of S^0, and therefore $a(x^{u'}) = 0 < b$, as required.

$\hat{u}_{\text{feas}}$ and $\hat{u}_{\text{infeas}}$ are not unique. The graph in Figure 4 is typical, in that any point from the interval $[u_3, u_4)$ may be chosen as $\hat{u}_{\text{feas}}$, while any point from $[u_4, u_5)$ may be chosen as $\hat{u}_{\text{infeas}}$. ϵ must be small enough such that at the termination of the algorithm u_{L} and u_{R} belong to two adjacent intervals.

4.2 Tangential Approximation Search Technique

The Bisection Algorithm is a simple method to obtain the best feasible and infeasible Lagrangian solutions as defined in (6) and (7). However, one shortcoming of this algorithm is that it only uses u_{L}, u_{R}, $a(u_{\text{L}})$, and $a(u_{\text{R}})$ in choosing trial points and deciding to terminate and ignores $z_{\text{LR}}(u_{\text{L}})$ and $z_{\text{LR}}(u_{\text{R}})$. The Tangential Approximation Algorithm, we will describe in this section, does not have this shortcoming. The following theorem establishes some properties of $z_{\text{LR}}(u)$ used by the Tangential Approximation Algorithm.

Theorem 4.2.2 *The function $z_{\text{LR}}(u)$ (defined in (3)) is a piecewise-linear convex function (with a finite number of pieces).*

Proof: For all $u \geq 0$,

$$z_{\text{LR}}(u) = \max_{x \in S} t(x) - u\left(a(x) - b\right).$$

For each element x^j in S define

$$f_j(u) := t(x^j) - u\left(a(x^j) - b\right)$$

$f_j(u)$ is an affine function of u. Furthermore, since the set S is finite there are a finite number of these functions. Therefore, $z_{\text{LR}}(u)$ is the maximum of a finite number of affine functions:

$$z_{\text{LR}}(u) = \max_{j \in \{1,\ldots,|S|\}} f_j(u).$$

Thus, $z_{\text{LR}}(u)$ is a piecewise-linear convex function. ■

We have:

$$z_{\text{LR}}(u) = \begin{cases} t(x^1) - u\,(a(x^1) - b)\,, & 0 \leq u < u_1 \\ t(x^2) - u\,(a(x^2) - b)\,, & u_1 \leq u < u_2 \\ \cdot & \\ \cdot & \\ \cdot & \\ t(x^n) - u\,(a(x^n) - b)\,, & u_{n-1} \leq u, < \infty \end{cases} \tag{10}$$

where $n \leq |S|$. The slope of each linear function is given by $b - a(x^i)$. Since $z_{\text{LR}}(u)$ is convex, these slopes strictly increase as i increases.

Let

$$t_i := t(x^i) \quad \text{and} \quad p_i := b - a(x^i).$$

We will refer to the interval endpoints u_i, $i = 1, \ldots, n-1$, in the definition of $z_{\text{LR}}(u)$, as "corner points".

Each $u \geq 0$ lies in a unique interval $[u_{i-1}, u_i)$ and is associated with a unique slope p_i corresponding to that interval (we sometimes refer to this as "the slope at u"). Strictly speaking, this is the right-hand derivative at u. Moreover, for each corner point u_i, $p_i u_i + t_i = p_{i+1} u_i + t_{i+1}$

Lemma 4.2.3 *For the piecewise-linear convex function $z_{\text{LR}}(u)$, the final slope (the slope of the last linear piece) is positive and, if x^0 is not feasible, the initial slope (the slope of the first linear piece) is negative.*

Proof: From Remark 4.1.1, we have that $a(x^u) = 0$ when

$$u > \frac{t(d)}{\min_{x:a(x)>0} a(x)}$$

Thus, for all u sufficiently large,

$$z_{\text{LR}}(u) = t(x^u) + ub$$

and the slope will equal b. Hence, $p_n = b > 0$. If x^0 is not feasible, then $a(x^0) > b$ and the initial slope p_1 must be negative. ■

Figure 5 depicts a typical plot of $z_{\mathrm{LR}}(u)$. The graphs in Figures 4 and 5 are closely connected: the decreasing step values of $a(x^u)$ in Figure 4 correspond exactly to the increasing slopes, $b - a(x^u)$, of the linear pieces in Figure 5.

Let u^* denote $\mathrm{argmin}_{u \geq 0} z_{\mathrm{LR}}(u)$ and u_{L}^* be the final value of u_{L} generated by the algorithm. The Tangential Approximation Algorithm, exploiting the convexity of $z_{\mathrm{LR}}(u)$, takes its new trial point at the intersection of two tangent lines, one with negative slope (on the left) and one with positive slope (on the right).

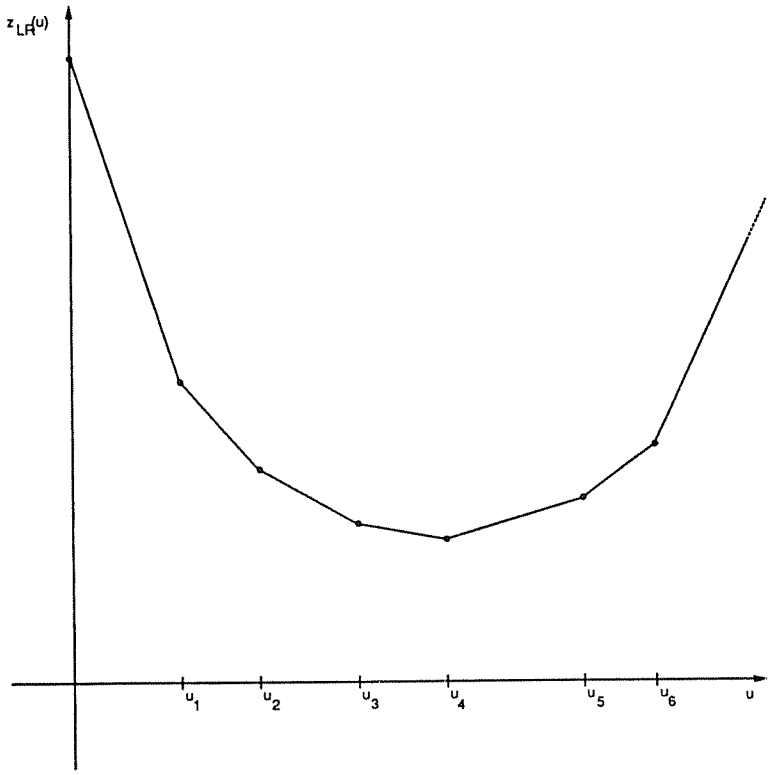

Figure 5: The Lagrangian relaxation function

Figure 6 illustrates the steps taken by the Tangential Approximation Algorithm to find u^*.

Tangential Approximation Algorithm

Step 0: $u_{\rm L} := 0,\ u_{\rm R} := u',\ t_{\rm L} := t(x^{u_{\rm L}}), p_{\rm L} := b - a(x^{u_{\rm L}})$,
$t_{\rm R} := t(x^{u_{\rm R}}),\ p_{\rm R} := b - a(x^{u_{\rm R}})$

Step 1: $\bar{u} := \dfrac{t_{\rm L} - t_{\rm R}}{p_{\rm R} - p_{\rm L}}$

Step 2: solve (3) and set $\bar{t} := t(x^{\bar{u}}),\ \bar{p} := b - a(x^{\bar{u}})$

Step 3: if $\bar{p} = 0$, then $u^* := \bar{u}$; stop.
/* We have $x^{\bar{u}}$ as the precise solution to (2) */

Step 4: if $z_{\rm LR}(\bar{u}) = p_{\rm L}\bar{u} + t_{\rm L}$, then $u^* := \bar{u}$ and $u^*_{\rm L} := u_{\rm L}$; stop.
/* $x^{u^*} = x^{\rm feas}$ and $x^{u^*_{\rm L}} = x^{\rm feas}$ */

Step 5: if $\bar{p} < 0$, then $u_{\rm L} := \bar{u},\ t_{\rm L} := \bar{t},\ p_{\rm L} := \bar{p}$;
else $u_{\rm R} := \bar{u},\ t_{\rm R} := \bar{t},\ p_{\rm R} := \bar{p}$;

Step 6: go to step 1.

In Step 0, u' must be selected such that $a(x^{u'}) < b$ (as was done in the Bisection Algorithm).

The quantity $\bar{u}$ in Step 1 is the minimum point of the piecewise linear function:

$$max\{t_{\rm L} + up_{\rm L}, t_{\rm R} + up_{\rm R}\}$$

Since $p_{\rm L} < 0$ and $p_{\rm R} > 0$ the minimum value of the above function is achieved at the intersection of these two tangent lines. The equation in Step 1 for finding a new trial point u has been taken from [6]. However, [6] does not use the notion of a piecewise-linear convex function.

The next theorem establishes an upper bound on the number of iterations required by Tangential Approximation Algorithm to find u^*.

Theorem 4.2.3 *At most $n-2$ iterations are required by the Tangential Approximation Algorithm to find u^* (thus solving the Lagrangian dual problem (4)). If u^* is unique, then u^* is some corner point u_i and we have $u^*_{\rm L} \in [u_{i-1},\ u_i)$.*

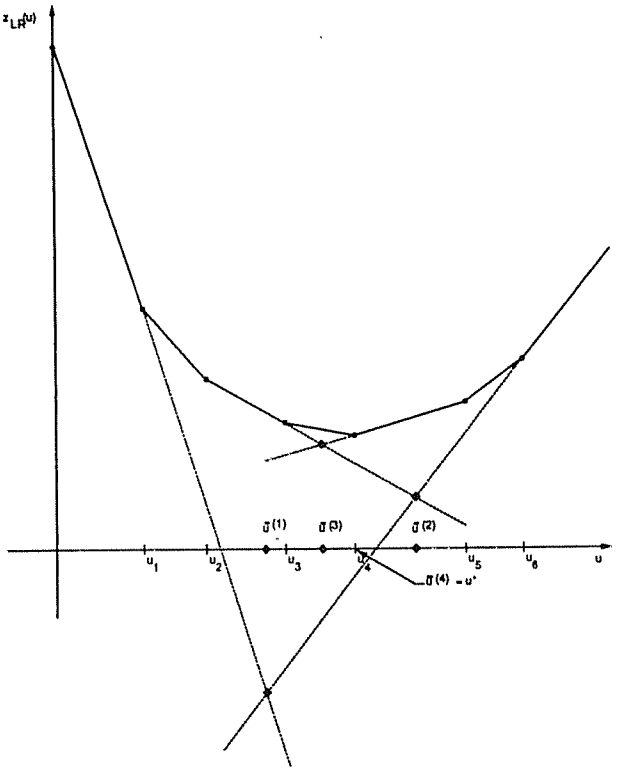

Figure 6: Tangential Approximation Algorithm

The proof is given in Appendix A.

Corollary 4.2.3 *If $z_{\mathrm{LR}}(u)$ has a unique minimum point, then $u^* = \hat{u}_{\mathrm{feas}}$ and $u_{\mathrm{L}}^* = \hat{u}_{\mathrm{infeas}}$. If the minimum is not unique, then $u^* = \hat{u}$ and x^{u^*} solves the original problem (2).*

Proof: If there is a unique minimum, then $\forall u \in [u_{\mathrm{L}}^*,\ u^*) \subset [u_{i-1},\ u_i)$, the slope of $z_{\mathrm{LR}}(u)$ is constant. Therefore, $a(x^u) = a(x^{u_{\mathrm{L}}^*})$. So, for $u < u_{\mathrm{L}}^*$,

$$a(x^u) \geq a(x^{u_{\mathrm{L}}^*}) > b.$$

For $u \geq u^*$, we have that

$$a(x^u) \leq a(x^{u^*}) < b.$$

Thus $u^* = \hat{u}_{\mathrm{feas}}$ and $u_{\mathrm{L}}^* = \hat{u}_{\mathrm{infeas}}$.

If the minimum is not unique, the slope at u^* is 0. This implies that $a(x^{u^*}) = b$, $u^* = \hat{u}$ and by Corollary 4.1 x^{u^*} solves (2). ■

The Tangential Approximation Algorithm has two advantages over the Bisection Algorithm. It terminates finitely and it has clear termination criteria that guarantees desired results. Furthermore, the worst case number of iterations required is of the order of the number of linear segments in $z_{\text{LR}}(u)$. In the experiments we conducted the algorithm terminated after a small number of iterations. Also, if criterion given in Step 4 is satisfied we have that $u_{\text{L}} = \hat{u}_{\text{infeas}}$ and $u_{\text{R}} = \hat{u}_{\text{feas}}$.

In contrast, for the Bisection Algorithm we must choose an ϵ small enough to guarantee that u_{L} and u_{R} are on adjacent intervals (and hence $u_{\text{L}} = \hat{u}_{\text{infeas}}$, $u_{\text{R}} = \hat{u}_{\text{feas}}$) when $u_{\text{R}} - u_{\text{L}} < \epsilon$. In practice, to insure proper termination of the algorithm the value of ϵ is usually chosen much smaller than needed, which results in wasted iterations. This is another reason why the Bisection Algorithm requires more iterations than the Tangential Approximation Algorithm (see Section 4).

Remark 4.2.2 Calculating $z_{\text{LR}}(u)$ implies solving (3). If (3) has more than one solution we select the one with minimum area. This makes $a(x^u)$ well-defined as a function of u and resolves point mentioned in footnote 3. This choice corresponds to selecting the right-hand derivative as the "slope at u". Multiple solutions to (3) (which occur only at the corner points u_i) do not affect our algorithms, except possibly at u^*, where a better solution to (2) might be overlooked. This is a rare occurrence and we do not attempt to handle it.

From Corollary (4.2) we have

$$t(x^{\hat{u}_{\text{feas}}}) \leq z_{\text{P}} \leq t(x^{\hat{u}_{\text{infeas}}}).$$

Therefore an estimate of the accuracy in using $x^{\hat{u}_{\text{feas}}}$ is given by:

$$\text{Accuracy} \;\geq\; \frac{t(x^{\hat{u}_{\text{feas}}})}{t(x^{\hat{u}_{\text{infeas}}})}$$

5 Post-optimal heuristic methods

In this section we discuss two post-optimal methods to improve the solutions x^{feas} and x^{infeas} obtained from the Bisection Algorithm or the Tangential Approximation

Algorithm. We call the first heuristic the "split-the-difference" method and the second is called the "best-ratio" method. For both methods we initialize

$$x^{\text{lo}} := x^{\text{feas}} \quad \text{and} \quad x^{\text{hi}} := x^{\text{infeas}}.$$

The "split-the-difference" method computes the midpoint, x^{mid}, of the line segment joining x^{lo} and x^{hi}. A new trial point, x, is then formed with components equal to the integer part of the corresponding components of x^{mid}. If the new point is feasible, set $x^{\text{lo}} = x$, otherwise set $x^{\text{hi}} = x$. This process is repeated until $x = x^{\text{lo}}$. Note that when this algorithm terminates, we have, for each component i, either $x_i^{\text{lo}} = x_i^{\text{hi}}$ or $x_i^{\text{lo}} + 1 = x_i^{\text{hi}}$. The split-the-difference algorithm is given in Appendix B.

The "best-ratio" method (Appendix C) successively increments those components of x^{lo} that produce a feasible point and a large increase in the objective function with small increase in the area. Specifically, the "best-ratio" algorithm searches for the component i that solves:

$$\begin{aligned} \max_{i \in \{1,\ldots,K\}} \quad & \frac{t_i(x_i^{\text{lo}}+1) - t_i(x_i^{\text{lo}})}{a_i(x_i^{\text{lo}}+1) - a_i(x_i^{\text{lo}})} \\ \text{s.t.} \quad & a(x^{\text{lo}}) + a_i(x_i^{\text{lo}} + 1) - a_i(x_i^{\text{lo}}) \leq b \end{aligned}$$

The ith component of x^{lo} is incremented and the search repeated until no further feasible increase in x^{lo} can be made.

The algorithm operates on x^{hi} in a similar fashion, decrementing the components of x^{hi} that give the most decrease in area for the least decrease in time and still stay infeasible.

At the termination of these post-optimal procedures four new points, two feasible points and two infeasible points, are generated. Let $x^{*\text{lo}}$ to be the better of the two feasible points and $x^{*\text{hi}}$ to be the better of the two infeasible points [5].

Extending the error analysis presented earlier, we now have

$$\text{Accuracy} \;\geq\; \frac{t(x^{*\text{lo}})}{t(x^{\text{infeas}})}. \tag{11}$$

Note that $t(x^{*\text{hi}})$ is not, in general, an upper bound on the optimal value $t(x^*)$. However, in most cases, it turns out that $t(x^{*\text{hi}})$ is an upper bound for $t(x^*)$ and the previous bound can be sharpened.

[5] Determining the "better" solution in infeasible case is more involved than in feasible case. We select the point that violates the constraint less, unless the selected point gives a lower value for the objective function than $x^{*\text{lo}}$, in which case, we select the other infeasible point as $x^{*\text{hi}}$.

6 Computational Results

The procedures described in the preceding sections were tested on randomly generated test problems. We generated four data sets. The number of transformations was varied from five to twenty and the number of times each transformation can be applied ranged from five to ten. Twenty problems with different maximum permissible area were solved for each data set. The results are given in Tables 2 through 5 for increasing values for K and d_i. In these tables column 2 gives the optimal value of the Lagrangian relaxation problem (3); columns 3 and 4 contain the values of the objective function for the best feasible and infeasible points obtained from the Lagrangian relaxation problem. The area used by the best infeasible solution and the value of the right-hand-side of the constraint, b, are shown in column 5 (separated by a slash). The next 2 columns report the original objective function values for the best feasible and infeasible points obtained after applying post-optimal heuristics. As before, the area for the infeasible point and the value of b are then given in the following column (column 8). The number of iterations required by Tangential Approximation Algorithm and Bisection Algorithm is given in column 9 (marked Iter.). For Tables 2 and 3 the last column shows the true optimal value for the original problem. In Tables 4 and 5 the estimated percent accuracy (as defined by (11)) is shown in the last column.

In Table 2 results are reported for randomly generated problems with $K = 5$ and $d_i = 5 \ \ \forall \ i = 1, \ldots, K$. The average number of iterations required for convergence of the Tangential Approximation Algorithm for the problems in Table 2 was 4.6. We make the following observations. In many cases, the objective function value for the feasible point obtained by the Lagrangian relaxation algorithm is close to the true optimal value. In fact, the exact optimal value was obtained for Problems 4, 5, 6, 12 and 14. Using the post-optimal heuristics, we were able to substantially improve upon the initial solutions. For example, in Problem 2, we raised the accuracy from 82 % to 96 %. Note that, for Problem 12, the Lagrangian relaxation yields a solution that satisfies the constraint as an equality. Thus, post-optimal heuristics were not performed (and we reported 0 in columns 6, 7 and 8).

The violation of the area constraint for the post-optimal infeasible solution is relatively small. Thus, if the area constraint is flexible the designer may opt to use a slightly infeasible solution in order to make a substantial gain in time. For example, in Problem 17, raising the area constraint by only 6 units, yields a time gain of 64 units.

Table 3 shows the results for data-set 2 with $K = 10$ and $d_i = 7 \ \ \forall i = 1, \ldots, K$. For

these larger size problems, the solution provided by the Lagrangian relaxation method is substantially better than in Table 2, and the worst feasible solution produced by the post-optimal heuristics was within 3 % of the optimal value. Despite the increase in the problem size the number of iterations required by Tangential Approximation Algorithm and Bisection Algorithm increased by a small amount. The average number of iterations required for convergence of the Tangential Approximation Algorithm for the problems in Table 3 was 5.6.

Tables 4 and 5 show results for data-sets 3 and 4, respectively. Here, $K = 20$ and $d_i = 10 \ \forall i = 1, \ldots, K$. The problems considered in Table 4 and 5 differ in the values of the data used. On average, the values of the data for the problems in Table 5 are thrice the value of the data for the problems in Table 4. Except for a few cases the estimated percent accuracy is above 97 %, showing that the proposed approach produces good solutions for larger problems.

The average number of iterations required for convergence of the Tangential Approximation Algorithm was 6.8 for Table 4 and 6.6 for Table 5.

Figure 7 summarizes the accuracy (as defined in (11)) achived for these problems. For each data set the minimum value, the maximum value and the average accuracy is shown.

7 Conclusions

In this paper we have addressed the problem of transformation selection for an area-delay efficient RTL implementation. We have presented a solution to the problem of selecting a transformation and determining how many times each transformation must be applied. The problem has been formulated as a multiple-choice knapsack problem and an approximate solution using algorithms based on Lagrangian relaxation have been developed. Moreover, two post-optimal heuristics which improve the final solution have also been used.

We tested our approach on a large number of problems of different sizes and in almost all cases the accuracy of the solution obtained by the Lagrangian relaxation algorithm and the post-optimal heuristics was above 97%. The computer runtimes for the heuristics ranged between 0.1s and 0.3s on a SUN SPARCstation 1 with 16 Mb main memory.

References

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[2] R. De Leone and R. Jain. Optimal Resource Allocation and Binding of Non-Pipelined Designs. Technical Report TR 972, Department of Computer Sciences, University of Wisconsin, October 1990.

[3] H. Everett. Generalized Lagrangian Multiplier Methods for Solving Problems of Optimum Allocation of Resources. *Operations Research*, 11(3), 1963.

[4] B. L. Fox and D. M. Landi. Searching for the Multiplier in One-Constraint Optimization Problems. *Operations Research*, 18(2), 1970.

[5] E. Girczyc. Loop Winding - A Data Flow Approach to Functional Programming. In *Proceedings of the IEEE International Symposium on Circuits and Systems.* IEEE, May 1987.

[6] H. J. Greenberg. The One-Dimensional Generalized Lagrange Multiplier Problem. *Operations Research*, 25(2), 1977.

[7] R. I. Hartley and A. E. Casavant. Tree-Height Minimization in Pipelined Architectures. In *Proceedings of the International Conference on Computer-Aided-Design.* ACM/IEEE, November 1989.

[8] R. Jain. *High-Level Area-Delay Prediction with Application to Behavioral Synthesis.* PhD thesis, Department of Electrical Engineering, University of Southern California, July 1989.

[9] R. Jain, M. J. Mlinar, and A. C. Parker. Area-Time Model for Synthesis of Non-Pipelined Designs. In *Proceedings of the International Conference on Computer-Aided-Design.* ACM/IEEE, November 1988.

[10] R. Jain, A. C. Parker, and N. Park. Predicting Area-Time Tradeoffs for Pipelined Designs. In *Proceedings of the 24th Design Automation Conference.* ACM/IEEE, June 1987.

[11] D. J. Kuck. *The Structure of Computers and Computations - Volume 1.* John Wiley & Sons, New York, 1978.

[12] M. J. Mlinar and A. C. Parker. Estimating Register and Multiplexer Costs in VLSI Design. Technical report, Department of Electrical Engineering, University of Southern California, 1988.

[13] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, New York, 1988.

[14] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on Computer-Aided-Design*, 7(3), March 1988.

[15] P. Sinha and A. A. Zoltners. The Multiple-Choice Knapsack Problem. *Operations Research*, 27(3), May-June 1979.

[16] E. A. Snow, D. P. Siewiorek, and D. E. Thomas. A Technology-Relative Computer-Aided Design System: Abstract Representations, Transformations, and Design Tradeoffs. In *Proceedings of the 15th Design Automation Conference.* ACM/IEEE, 1978.

[17] H. Trickey. *Compiling Pascal Programs into Silicon.* PhD thesis, Department of Computer Science, Stanford University, July 1985.

[18] R. A. Walker and D. E. Thomas. Behavioral Transformation for Algorithmic Level IC Design. *IEEE Transactions on Computer-Aided-Design*, 8(10), October 1989.

Prob.	z_{LD}	$t(x^{\text{feas}})$	$t(x^{\text{infeas}})$	$a(x^{\text{infeas}})/b$	Post-optimal			Iter.	Opt.
					$t(x^{*\text{lo}})$	$t(x^{*\text{hi}})$	$a(x^{*\text{hi}})/b$		
1	1247.462	1188	1267	989/966	1188	1267	989/966	4/21	1192
2	559.504	435	671	631/511	512	537	513/511	4/17	531
3	1113.850	1036	1206	1024/908	1079	1097	915/908	5/21	1085
4	1323.278	1318	1328	1170/1153	1318	1328	1170/1153	5/22	1318
5	907.373	902	947	909/850	902	915	895/850	5/24	902
6	1262.309	1231	1354	1218/1095	1231	1274	1128/1095	5/21	1231
7	1083.331	1033	1191	961/792	1070	1104	850/792	4/24	1070
8	1113.036	1042	1125	1053/1037	1065	1115	1049/1037	6/19	1089
9	1127.071	945	1151	757/734	1060	1080	736/734	4/19	1074
10	1341.299	1186	1478	1126/979	1288	1323	981/979	4/22	1311
11	876.867	768	906	714/676	855	906	714/676	4/21	855
12	935.000	935	1024	1230/921	0	0	0/921	2/9	935
13	989.264	820	1118	726/618	970	991	639/618	4/23	970
14	992.027	975	1038	944/863	975	980	888/863	5/22	975
15	740.618	702	797	633/560	706	739	579/560	5/18	715
16	871.364	845	903	853/805	845	903	853/805	5/19	869
17	684.456	617	690	508/502	626	690	508/502	5/19	667
18	1100.006	968	1107	755/747	1055	1066	748/747	5/21	1078
19	921.113	811	945	1185/1144	884	900	1146/1144	5/22	900
20	713.500	637	871	899/689	701	754	772/689	5/18	701

Table 2: Results for data-set 1, $K = 5$, $d_i = 5, \forall i = 1, \ldots, K$

Prob.	z_{LD}	$t(x^{\text{feas}})$	$t(x^{\text{infeas}})$	$a(x^{\text{infeas}})/b$	Post-optimal			Iter.	Opt.
					$t(x^{*\text{lo}})$	$t(x^{*\text{hi}})$	$a(x^{*\text{hi}})/b$		
1	2595.556	2545	2597	2137/2135	2547	2597	2137/2135	7/23	2556
2	1782.706	1548	1828	1276/1232	1735	1746	1250/1232	6/23	1765
3	1809.660	1783	2198	1914/1477	1787	1796	1536/1477	6/22	1799
4	3425.726	3181	3451	3115/3074	3366	3393	3081/3074	5/23	3402
5	3775.627	3684	3874	3494/3319	3735	3757	3341/3319	5/23	3761
6	2727.845	2530	2870	2610/2416	2667	2682	2418/2416	6/24	2700
7	3153.000	2983	3238	2786/2661	3064	3088	2668/2661	5/23	3121
8	2911.339	2694	3116	1984/1758	2826	2872	1760/1758	5/25	2896
9	2649.277	2331	2728	2096/2003	2557	2596	2012/2003	5/23	2613
10	1646.989	1610	1709	1157/1100	1610	1709	1157/1100	5/23	1610
11	1946.068	1873	2046	1438/1345	1900	1947	1347/1345	6/22	1929
12	2879.767	2840	3030	3282/2942	2848	2851	2971/2942	5/22	2848
13	3413.903	3413	3492	3360/3187	3413	3430	3270/3187	5/23	3413
14	2982.247	2872	3095	2419/2284	2953	2958	2286/2284	5/24	2953
15	3274.176	3226	3369	2653/2529	3248	3275	2563/2529	7/22	3263
16	2382.587	2188	2503	1751/1626	2370	2410	1662/1626	7/22	2370
17	2614.683	2362	2703	1979/1871	2567	2596	1872/1871	5/24	2591
18	3644.934	3628	3657	3525/3468	3639	3657	3525/3468	6/27	3639
19	1739.884	1659	1988	1440/1213	1734	1764	1240/1213	6/23	1734
20	3079.046	2939	3190	2974/2810	3017	3058	2810/2810	5/25	3058

Table 3: Results for data-set 2, $K = 10$, $d_i = 7, \forall i = 1, \ldots, K$

Prob.	z_{LD}	$t(x^{\text{feas}})$	$t(x^{\text{infeas}})$	$a(x^{\text{infeas}})/b$	Post-optimal			Iter.	Accuracy
					$t(x^{*\text{lo}})$	$t(x^{*\text{hi}})$	$a(x^{*\text{hi}})/b$		
1	3660.776	3445	3761	2349/2264	3621	3624	2268/2264	6/25	96%
2	5670.592	5435	5851	4031/3861	5589	5646	3862/3861	7/28	96%
3	6700.083	6411	6966	5370/5106	6670	6707	5151/5106	7/27	96%
4	8679.965	8656	8874	6954/6703	8656	8668	6728/6703	7/28	98%
5	8875.913	8683	8887	7963/7948	8831	8842	7950/7948	7/28	99%
6	5394.153	5179	5513	3661/3545	5343	5369	3549/3545	7/28	97%
7	6166.662	5919	6189	4813/4790	6113	6117	4793/4790	7/27	99%
8	7535.600	7388	7716	6596/6387	7489	7521	6401/6387	5/27	97%
9	8321.517	7969	8328	7640/7632	8261	8328	7640/7632	8/26	99%
10	4985.526	4769	5253	3481/3229	4952	4985	3297/3229	7/29	94%
11	6058.000	5567	6069	4485/4474	5939	5970	4474/4474	5/26	98%
12	7620.358	7437	7781	6269/6071	7587	7600	6092/6071	6/29	98%
13	7716.172	7681	7907	7571/7316	7700	7709	7321/7316	7/26	97%
14	4437.672	4103	4453	2925/2913	4360	4403	2915/2913	6/25	98%
15	5398.795	5306	5773	4529/4158	5394	5465	4253/4158	6/28	93%
16	7113.974	7105	7175	5823/5755	7105	7111	5755/5755	8/29	99%
17	9094.443	9043	9148	7428/7352	9086	9148	7428/7352	8/26	99%
18	4025.148	3962	4117	2677/2597	4003	4030	2610/2597	7/26	97%
19	6073.191	5782	6147	4267/4194	6023	6050	4194/4194	7/28	98%
20	6861.233	6793	7102	5686/5439	6848	6886	5466/5439	7/26	96%

Table 4: Results for data-set 3, $K = 20$, $d_i = 10, \forall i = 1, \ldots, K$

Prob.	z_{LD}	$t(x^{\text{feas}})$	$t(x^{\text{infeas}})$	$a(x^{\text{infeas}})/b$	Post-optimal			Iter.	Accuracy
					$t(x^{*\text{lo}})$	$t(x^{*\text{hi}})$	$a(x^{*\text{hi}})/b$		
1	21599.082	20926	21786	17982/17769	21513	21560	17800/17769	7/28	99%
2	22173.584	21858	22468	17672/17366	22052	22163	17387/17366	6/33	98%
3	18053.950	17947	18092	12648/12611	17981	18092	12648/12611	7/29	99%
4	16829.712	15798	17181	12541/12208	16672	16756	12256/12208	5/27	97%
5	12396.307	12027	13682	8574/7453	12307	12339	7499/7453	5/28	90%
6	11391.279	11055	13192	8544/7050	11313	11331	7051/7050	7/29	86%
7	23935.575	23001	24008	18368/18295	23625	23727	18295/18295	7/32	98%
8	22792.714	22569	22838	17942/17892	22756	22758	17906/17892	8/29	100%
9	18013.253	16950	18501	13665/13137	17850	17945	13157/13137	7/31	96%
10	16438.928	15823	16681	12993/12734	16232	16253	12745/12734	6/27	97%
11	13149.414	12568	14041	8689/7979	13051	13294	8214/7979	6/26	93%
12	12953.575	12887	14319	8663/7576	12891	12981	7653/7576	7/30	90%
13	23687.600	23371	24077	19281/18821	23597	23692	18880/18821	6/29	98%
14	24266.604	23908	24450	18642/18418	24139	24219	18419/18418	7/29	99%
15	19541.340	19450	20160	14320/13663	19510	19511	13675/13663	7/30	97%
16	17150.155	16397	17621	13737/13260	16963	17076	13264/13260	7/31	96%
17	13819.073	12374	14211	8815/8505	13507	13648	8520/8505	6/27	95%
18	12497.152	12420	13399	8967/8102	12420	12602	8210/8102	7/32	93%
19	25713.852	25431	26644	20544/19347	25662	25754	19574/19347	6/28	96%
20	21872.908	21440	22145	19301/18944	21747	21858	18958/18944	7/30	98%

Table 5: Results for data-set 4, $K = 20$, $d_i = 10, \forall i = 1, \ldots, K$

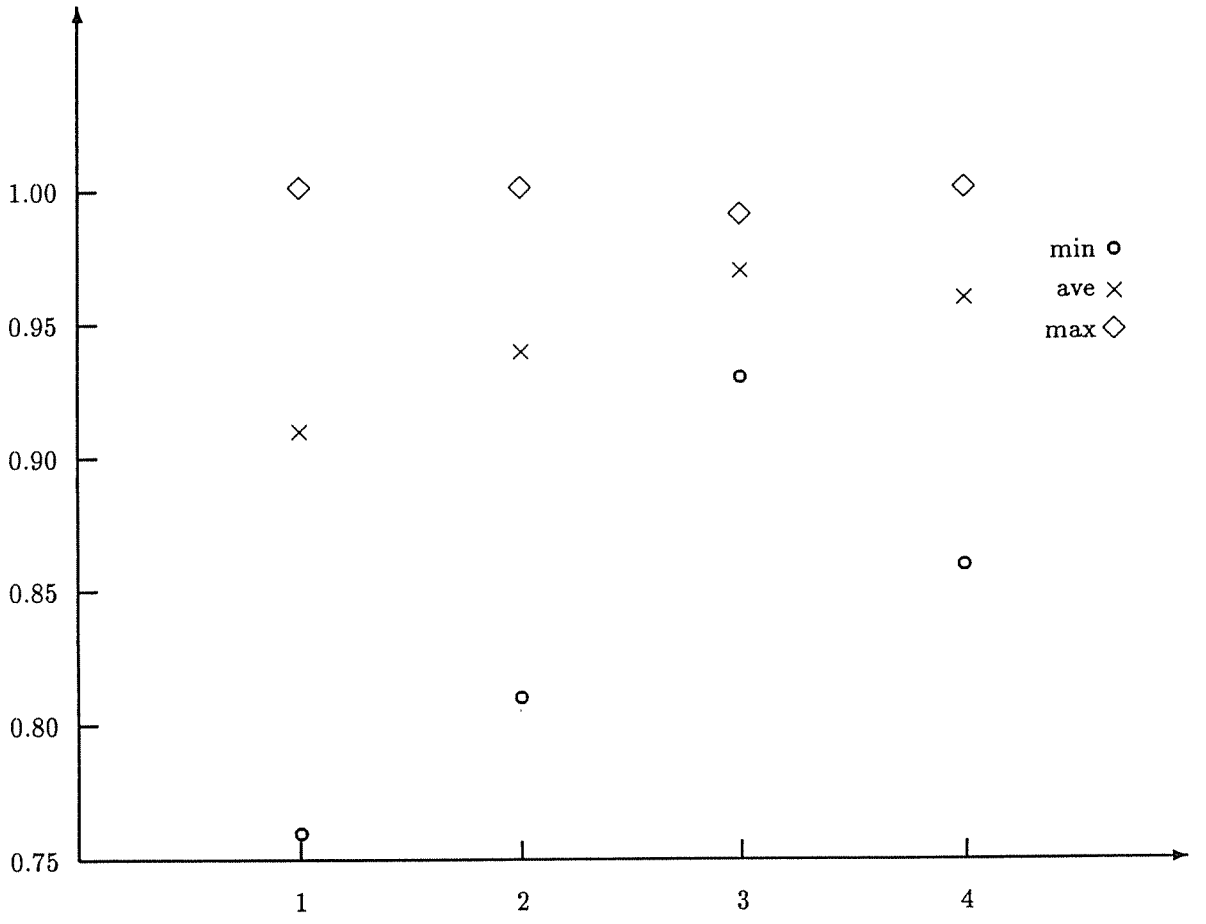

Figure 7: Solution accuracy for the data-sets 1-4

A Proof of Theorem 4.2.3

In the appendix we give a proof for Theorem 4.2.3. We use the notation given in (10) and the Tangential Approximation Algorithm. Let $\bar{I} = [u_j, u_{j+1})$ be the interval in which $\bar{u}$ lies. If $\bar{p} = 0$, i.e., there is zero slope on $\bar{I}$, then we have a local minimum at $\bar{u}$, and so, by convexity, $\bar{u} = u^*$. This u^* is clearly not unique, since any $u \in \bar{I}$ can be chosen as u^*. This covers step 3 of the algorithm.

Let $I_L = [u_{i-1}, u_i)$ and $I_R = [u_k, u_{k+1})$ be the intervals in which u_{L} and u_{R} lie, respectively.

Claim: $u_i \leq \bar{u} \leq u_k$

Proof: Since $z_{\mathrm{LR}}(u_i) = p_{\mathrm{L}} u_i + t_{\mathrm{L}}$ and p_{R} is a subgradient of $z_{\mathrm{LR}}()$ at u_{R}, we have

$$p_{\mathrm{L}} u_i + t_{\mathrm{L}} \geq p_{\mathrm{R}} u_i + t_{\mathrm{R}}.$$

Thus,

$$u_i \leq \frac{t_{\mathrm{L}} - t_{\mathrm{R}}}{p_{\mathrm{R}} - p_{\mathrm{L}}} = \bar{u}.$$

Similarly,

$$p_{\mathrm{R}} u_k + t_{\mathrm{R}} \geq p_{\mathrm{L}} u_k + t_{\mathrm{L}}$$

yields

$$u_k \geq \frac{t_{\mathrm{L}} - t_{\mathrm{R}}}{p_{\mathrm{R}} - p_{\mathrm{L}}} = \bar{u}.$$

The claim is proven. ■

Because the slopes on the intervals are strictly increasing as we move to the right, we have:

$$(1) \text{ if } \bar{u} > u_i, \text{ then } z_{\mathrm{LR}}(\bar{u}) > p_{\mathrm{L}} \bar{u} + t_{\mathrm{L}}$$

and

$$(2) \text{ if } \bar{u} < u_k, \text{ then } z_{\mathrm{LR}}(\bar{u}) > p_{\mathrm{R}} \bar{u} + t_{\mathrm{R}}$$

By (1), the termination criterion of step 4 in the Tangential Approximation Algorithm is only satisfied when $\bar{u} = u_i$. But, then, by (2) (and since $p_{\mathrm{L}} \bar{u} + t_{\mathrm{L}} = p_{\mathrm{R}} \bar{u} + t_{\mathrm{R}}$), we must also have $\bar{u} = u_k$. Thus $u_i = u_k$, i.e., I_L and I_R are adjacent segments (and $\bar{I} = I_R$). But, since $p_{\mathrm{L}} < 0$ and $p_{\mathrm{R}} > 0$, we have a strict local minimum at $\bar{u}$. Thus, by convexity, $\bar{u} = u^*$ and u^* is unique. Also, $u_{\mathrm{L}}^* = u_{\mathrm{L}} \in [u_{i-1},\ u_i)$.

If we do not terminate at step 4, then step 5 maintains the condition that $p_{\mathrm{L}} < 0$ and $p_{\mathrm{R}} > 0$. Also, since $u_i < \bar{u} < u_k$, we have

$$u_i \leq u_j \leq u_{j+1} \leq u_k$$

In other words, $\bar{I}$ is a new interval disjoint from and between I_L and I_R. When step 5 chooses $\bar{u}$ as either the new u_{L} or the new u_{R}, it is also choosing $\bar{I}$ as the new I_L or I_R. Thus, I_L and I_R are "advancing" toward each other at every iteration (unless $\bar{p} = 0$ at some stage) and, eventually, will be adjacent intervals, at which point the termination criterion of step 4 will be met. Since there are at most $n - 2$ intervals, we can have at most $n - 2$ iterations.

B Split-The-Difference Algorithm

Step 0: $x^{\text{lo}} := x^{\text{feas}}$; $x^{\text{hi}} := x^{\text{infeas}}$
Step 1: $x^{\text{mid}} := \frac{x^{\text{lo}}+x^{\text{hi}}}{2}$
Step 2: $x := floor(x^{\text{mid}})$ /* Take the integer part (in each component) of x^{mid} */
Step 3: If $x = x^{\text{lo}}$, then stop.
Step 4: If $a(x) < b$, then $x^{\text{lo}} := x$, else $x^{\text{hi}} := x$
Step 5: Go to 1.

C Best-Ratio Algorithm

Step 0: $x^{\text{lo}} := x^{\text{feas}}$; $x^{\text{hi}} := x^{\text{infeas}}$
Step 1: $I := \{i : \ x_i^{\text{lo}} + 1 \leq d_i \text{ and } a(x^{\text{lo}}) + a_i(x_i^{\text{lo}} + 1) - a_i(x_i^{\text{lo}}) \leq b\}$
Step 2: If $I = \emptyset$, then go to step 6.
Step 3: $j := \underset{i \in \{1,\ldots,K\}}{\text{argmax}} \dfrac{t_i(x_i^{\text{lo}} + 1) - t_i(x_i^{\text{lo}})}{a_i(x_i^{\text{lo}} + 1) - a_i(x_i^{\text{lo}})}$
Step 4: $x_j^{\text{lo}} := x_j^{\text{lo}} + 1$
Step 5: Go to 1.
Step 6: $I := \{i : \ x_i^{\text{hi}} - 1 \geq 0 \text{ and } a(x^{\text{hi}}) + a_i(x_i^{\text{hi}} - 1) - a_i(x_i^{\text{hi}}) \geq b\}$
Step 7: If $I = \emptyset$, then stop.
Step 8: $j := \underset{i \in \{1,\ldots,K\}}{\text{argmax}} \dfrac{a_i(x_i^{\text{hi}}) - a_i(x_i^{\text{hi}} - 1)}{t_i(x_i^{\text{hi}}) - t_i(x_i^{\text{hi}} - 1)}$
Step 9: $x_j^{\text{hi}} := x_j^{\text{hi}} - 1$
Step 10: If $t(x^{\text{hi}}) \leq t(x^{\text{lo}})$, then $x_j^{\text{hi}} := x_j^{\text{hi}} + 1$; stop.
Step 11: Go to step 6.