Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem

Samit Chaudhuri, Robert A. Walker, John E. Mitchell

Abstract— **In integer linear programming (ILP), formulating a "good" model is of crucial importance to solving that model [1]. In this paper, we begin with a mathematical analysis of the structure of the assignment, timing, and resource constraints in high-level synthesis, and then evaluate the structure of the scheduling polytope described by these constraints. We then show how the structure of the constraints can be exploited to develop a well-structured ILP formulation, which can serve as a solid theoretical foundation for future improvement. As a start in that direction, we also present two methods to further tighten the formulation. The contribution of this paper is twofold: (1) it provides the first in-depth formal analysis of the structure of the constraints, and it shows how to exploit that structure in a well-designed ILP formulation, and (2) it shows how to further improve a well-structured formulation by adding new valid inequalities.**

I. Introduction

The scheduling problem in high-level synthesis [2] is concerned with sequencing the operations of a control/data flow graph (cdfg) into correct order. This is an optimization problem, and is specified in several ways depending on the goal: (1) *unconstrained* scheduling (UCS) minimizes a function of the number of hardware resources and the number of control steps; (2) *resource-constrained* scheduling (RCS) minimizes the number of control steps when the number of hardware resources is fixed; (3) *time-constrained* scheduling (TCS) minimizes the number of resources when the number of control steps is fixed. We can also consider a fourth problem called *time- and resource-constrained* scheduling (TRCS), which optimizes a given objective function when both the number of hardware resources and the number of control steps are fixed. The decision problem [3] corresponding to TRCS is known to be NP-complete [4]; therefore at present, no polynomial time exact algorithm exists to solve any of the scheduling problems.

To solve the scheduling problem, both heuristic scheduling algorithms, which find approximate (or suboptimal) solutions, and exact algorithms, which find optimal solutions, have been used. A wide variety of heuristic algorithms are used, including the transformational [5], list [6], and force-directed [7] scheduling heuristics. In contrast, most exact algorithms employ integer linear programming (ILP) to compute the optimal solutions. Although solving an ILP formulation is NP-hard [3], significant progress has been made in the development of efficient ILP algorithms. ILP-based schedulers, such as OASIC[8] and ALPS[9], have produced better schedules than heuristic algorithms, in comparable time for medium-sized problems. In this paper we will focus on the structure of the constraints in ILP approach to the scheduling problem, and on exploiting that structure in a well-designed ILP formulation.

A. Background

Integer programming problems [10] either maximize or minimize an objective function of many variables, subject to: (a) equality and inequality constraints, and (b) integrality restrictions on all of the variables. It is also common to use linear objective functions and linear constraints, and to require the variables to be nonnegative. An *Integer Linear Programming* (ILP) formulation is written as:

$$\begin{aligned} z_{IP} &= \min \{ c^T x \mid x \in P_F \; ; \; x \text{ integer} \} \quad (1) \\ \text{where } P_F &= \{ Ax \leq b, x \in \mathbb{R}^n_+ \} \end{aligned}$$

where $\mathbb{R}^n_+$ is the set of nonnegative real $(n \times 1)$ vectors, c is a $(n \times 1)$ real vector, b is a $(m \times 1)$ vector of integers, and A is a $(m \times n)$ matrix of integers.

A wide variety of problems can be represented by ILP formulations. However, solving a general ILP formulation is known to be NP-hard [3], so solving a problem by using a poorly-designed ILP formulation is not a very practical strategy.

Fortunately, not all ILP formulations are equally difficult to solve. Recent research has lead to the understanding of properties that make some ILP formulations well-solvable. We refer to such formulations as *good*, or *structured*, because the constraints of these formulations have some special structure. Use of good formulations has led to significant success of the ILP approach for solving NP-complete problems in other areas, such as the traveling salesman problem (TSP) [11], 0-1 integer programming [12], and minimum perfect 2-matchings [13]. It has been said in [1] that "*formulating a good model is of crucial importance to solving the model*". Therefore, to efficiently solve the

Manuscript received August 31, 1993; revised April 22, 1994 and July 6, 1994. This work was supported by the National Science Foundation under Grant No. MIP-9211323, and the Office of Naval Research under Grant No. N00014-90-J-1714.

S. Chaudhuri is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA. E-mail: chaudhus@cs.rpi.edu .

R. A. Walker is with the Department of Computer Science and the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA

J. E. Mitchell is with the Department of Mathematics, Rensselaer Polytechnic Institute, Troy, NY 12180 USA

IEEE Log Number 9405512

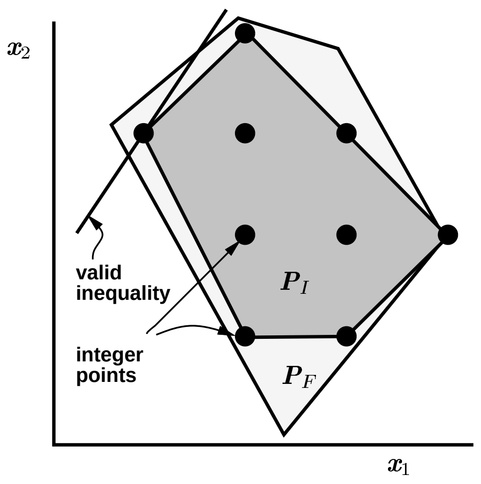

Fig. 1. Feasible Regions of an ILP Formulation

scheduling problem, it is important to use a good, or structured formulation instead of just an ad-hoc formulation.

Before we can discuss why some particular combinatorial problems can be efficiently solved using ILP formulations, note that we can solve the ILP presented in (1) by solving the following problem [10]:

$$\begin{aligned} z_{IP} &= \min\{\, c^T x \mid x \in P_I \,\} \qquad (2)\\ \text{where } P_I &= conv\{\, x \in P_F \mid x \text{ integer}\} \end{aligned}$$

where *conv* denotes convex hull.

A geometrical view of P_F and P_I is given in Figure 1. The feasible solutions to the ILP are denoted by the integer points inside P_F; depending on the objective function, one of them represents an optimum solution. The feasible region of (1) consists of only the integer points; in contrast, the feasible region of (2), P_I, consists of the convex hull of the same integer points.

Since P_F is described using only equality and inequality constraints (no integrality constraints are required), any linear objective function can be minimized over P_F in polynomial time using a *linear program*. This motivates us to define a related problem, called the *LP-relaxation* of the ILP (2), as follows:

$$z_{LP} = \min\{\, c^T x \mid x \in P_F \,\} \qquad (3)$$

Since $P_I \subseteq P_F$, we can conclude from (2) and (3) that $z_{LP} \leq z_{IP}$. We say that the polyhedron P_F is *integral* if $P_I = P_F$, and when P_F is integral, $z_{LP} = z_{IP}$, and the ILP formulation can be solved in polynomial time by solving its LP-relaxation. Therefore, while formulating an ILP, one should attempt to find equality and inequality constraints such that P_F will be integral.

Unfortunately, except for a very few special problems, it is not possible to find a set of constraints that describe P_F as an integral polyhedron. In general, $P_I \subset P_F$, and the LP-relaxation provides a lower bound on the objective function. Most integer programming algorithms require this bound, and the efficiency of the algorithm is very dependent on the sharpness of the bound [1]. The sharpness of the bound increases as P_F approximates P_I more closely, so for efficient solution of an ILP formulation, it is extremely important that P_F be close to P_I.

For combinatorial problems in other areas, advances in solving the ILP formulations have been made in several ways. First, a formal analysis of the structure of the constraints has helped to find *tight* descriptions of P_F, that more closely approximate P_I. As a result, sharp bounds on the objective function could be found. Second, formal analysis has led to new *valid inequalities* (the inequality constraints that arise due to the integrality of the variables), and as a result, the preliminary formulations of the problem were further tightened. Finally, the increased understanding of the constraints has also led to better relaxation and branching strategies. Taken together, all these factors have made efficient solutions of the ILP formulations of these problems possible.

In contrast, most of the research on ILP-based scheduling algorithms in the area of high level synthesis has concentrated solely on describing a correct formulation. Thus the focus has been only to express the design issues in terms of mathematical equations, while the structure of the formulation has not received much attention. If we are to develop more efficient solution techniques, we must analyze the structure of the constraints of the scheduling problem, and exploit the structure in a well-designed ILP formulation.

B. Previous Work

Hafer [14] was perhaps the first to employ combinatorial optimization in the high level synthesis area. Although his work led to a formal definition of the problem, the formulation was too complex to design a chip of reasonable size. Other ILP-based schedulers are GRAD [15], ALPS [9], and OASIC [8]. A comparison of the details of these schedulers will be presented in Section VI after our notation is introduced.

At this point, it should be noted that each of the formulations used by these previous systems will yield the optimal schedule, although they describe the problem in different ways. The real challenge here is to analyze the structure of the constraints, to design a formulation that exploits the structure, and to indicate ways to further tighten it. To date, there has been no mention of the structure of the formulation used by ALPS or GRAD. In contrast, Gebotys [8] identified the similarities between the scheduling problem constraints and other known types of ILP constraints, and proved that the constraints used in OASIC are tighter than ALPS. For further improvement in the ILP approach, such work has to continue, so that the guaranteed optimal results of ILP solutions can be produced with more efficiency.

Recent work on ILP-based scheduling algorithms has instead concentrated on including more design parameters (number of busses, clock length, binding etc.) [16], [17], [18] into the model in an approximate way. Because of the robustness of the general ILP model, a correct formulation can always be extended to additional design parameters. However, formal analysis is still necessary for further improvement in solution efficiency, which is the main impediment to practical use of the ILP approach.

There has also been extensive research in scheduling in other areas [19]. Unfortunately, unlike TSP, or the knapsack problem, very little work has been done on describing the scheduling polyhedra. Analyzing the scheduling polyhedra is difficult, and only very recently some work has been published on the single-machine scheduling problems [20], [21]. However, the models considered in such literature are not appropriate for use in high-level synthesis.

C. Motivation and Outline of the Paper

As we discussed in Section I-A, to develop efficient solution techniques, we must analyze the structure of the constraints of the scheduling problem, and exploit that structure in a well-designed ILP formulation. Our survey of previous work presented in Section I-B indicates that more research is needed in order to explore the structure of the scheduling constraints. The motivation for this paper is not to present just another ILP model for the scheduling problem, but to formally analyze the ILP approach to the scheduling problem, which will serve as a theoretical basis for future improvement to this approach. In Sections II and III, we describe the ILP constraints of the scheduling problem. Sections IV and V examine the structure of these constraints, and present an ILP formulation that exploits that structure.

For further improvement in the solution efficiency of a well-designed ILP formulation, new ways have to be found to tighten the original formulation. In [22], Nemhauser said, "Preprocessing and polyhedral theory have yielded at least an order of magnitude improvement in branch-and-bound algorithms for solving mixed-integer programs". This prospect of tightening an ILP formulation has so far remained unexplored in case of the scheduling problem. In Section VII we discuss a preprocessing method to modify the convectional resource-constrained ASAP and ALAP algorithms, and present strong valid inequalities using polyhedral theory. Finally, Section VIII discusses how to achieve design goals using our ILP formulation. Experimental results are presented in Section IX to show the validity of the predictions made from the analysis in the previous sections.

II. Preliminaries

Suppose a given cdfg is to be scheduled onto a set S of control steps. Let I be the index set of all operations, and let each arc a_{ij} of the cdfg indicate a timing relation:

$$a_{ij} \Rightarrow o_i \xrightarrow{d_{ij}} o_j$$

Each timing relation specifies a delay d_{ij}, such that $t(j) \geq t(i)+d_{ij}$, where $t : I \to S$, and $t(i)$ denotes the control step in which operation o_i starts execution. The delay d_{ij} can be used to indicate either a data dependency or a timing constraint. An arc a_{ij} with positive d_{ij} (resp. a back-arc a_{ji} with negative d_{ji}) implies a minimum (resp. maximum) timing constraint between o_i and o_j. A fixed timing constraint t_f between o_i and o_j can be easily incorporated by making $d_{ij} = -d_{ji} = t_f$. Since maximum timing constraints are denoted by back-arcs which create cycles in the cdfg, we assume, to ensure consistency, that the total delay of every cycle is ≤ 0.

As-soon-as-possible (ASAP) and as-late-as-possible (ALAP) schedules give a continuous range S_i of control steps, called the *schedule interval*, over which an operation o_i can be scheduled.

The *type* of a functional unit (FU) indicates its functionality (eg., multiplication or addition). Let K be the set of types that are available. Let a_k and m_k, respectively, be the area and number of functional units of type $k \in K$. The type of the operations are determined by the type function $\tau : I \to K$. $\tau(i) = k$ means operation o_i is executed on a functional unit of type k.

The values of m_k and S can be fixed or can be variables, depending on the scheduling problem being solved. In UCS, a function of area and time, $c_1|S|+c_2 \sum_{k\in K} a_k m_k$, is minimized. In TCS, the total area $\sum_{k\in K} a_k m_k$ is minimized for a fixed S. In RCS, $|S|$ is minimized for fixed values of m_k. In TRCS, both S and m_k are fixed, and some objective function is optimized. In all cases, similar constraints, to be presented in the next section, can be used with different objective functions.

Consider the set of nodes $V = \{ (i, s) \mid i \in I \,;\, s \in S_i \}$, where a node (i, s) indicates that operation o_i is scheduled in control step s. Each operation o_i corresponds to a set of nodes $V_i = \{ (i, s) \mid s \in S_i \}$ (see V_2 in Figure 2 (b)). Furthermore, each functional unit type k relates, for each control step s, to a set of nodes $V_{k,s} = \{ (i, s) \mid s \in S_i \,;\, \tau(i) = k \}$.

Each feasible schedule $Q \subseteq V$ contains exactly one node from each V_i, satisfies all the timing constraints between operations, and uses no more than the available number of functional units. The feasible schedules will be described by the following notation:

$\mathcal{Q}$	Set of all feasible schedules. This is a set of subsets of V.
x^Q	A vector in $\mathbb{R}^{\vert V\vert}$, called the *incidence* or *characteristic* vector of Q, where $Q \subseteq V$, defined as follows:

$$x^Q_{i,s} = \begin{cases} 1, & \text{if } (i,s) \in Q \\ 0, & \text{if } (i,s) \notin Q \end{cases}$$

In the next section, we will describe the set of feasible schedules by specifying a set of equality and inequality constraints on the incidence vectors, and by imposing integrality constraints, from which one can formulate an ILP. Our goal is a representation of the ILP by an LP that has the same optimal solution. Therefore, as discussed in Section I-A, we need to consider the convex hull of the feasible schedules, which can be described as:

$$P_I(\mathcal{Q}) = conv\{ x^Q \in \mathbb{R}^{|V|} \mid Q \in \mathcal{Q} \}$$

$P_I(\mathcal{Q})$ is called the *scheduling polytope.* If we can describe $P_I(\mathcal{Q})$ without using integrality restrictions on x, then we can solve the ILP by solving an LP using $P_I(\mathcal{Q})$ as the feasible region.

Since the scheduling problem is NP-hard, such a description of $P_I(\mathcal{Q})$ is probably not achievable; we can only attempt to describe $P_I(\mathcal{Q})$ as tightly as possible, so that the ILP solution can be found by solving a relatively small number of LP-relaxations.

III. The ILP Constraints

In this section we will present the ILP constraints of the scheduling problem, and give a preliminary description of the scheduling polytope.

The constraints of the scheduling problem can be divided into three types, namely assignment, timing and resource constraints. The resource constraints are described in Section III-C. In the next two sections, we will discuss the assignment and timing constraints only. For this purpose, we will consider the set $\mathcal{N}$ of feasible schedules, each of which satisfies the assignment and timing constraints (but not necessarily the resource constraints).

The assignment and timing constraints can be represented using a graph model, called a constraint graph. In Section III-A, we will discuss the constraint graph G_p used by the previous systems. We will show that G_p is not well-structured (its node-packing polytope could be non-integral), so we will use a different graph G_c which is described in Section III-B. The purpose of using G_c is to find and prove the tightest possible description of $P_F(\mathcal{N})$.

A. Graph Models for Scheduling Constraints

Let us first consider the set $\mathcal{N}$ of feasible schedules, each of which must contain exactly one node from each V_i and satisfy all the timing constraints specified in the cdfg. The conditions under which the set $N \subseteq V$ constitutes such a feasible schedule (i.e., $N \subseteq \mathcal{N}$) can be expressed by listing the pairs of nodes that can not coexist in N. These forbidden pairs can be conveniently expressed as edges between nodes in a graph, which we will refer to as a *constraint graph.* For the time being, we will ignore the constraints on the number of FU's, and introduce them in a later section.

Each edge uv in the constraint graph represents a constraint for a feasible schedule N. This constraint can be described as an *edge constraint* $x_u + x_v \leq 1$, where x_u, x_v are 0-1 variables used for the incidence vector description of N. For example, one can construct the constraint graph $G_p = (V, E_p)$ in Figure 2 (b), in order to schedule the cdfg of Figure 2 (a) in 6 control steps. A simple (and very loose) way of describing $P_F(\mathcal{N})$ would be: $P_F(\mathcal{N}) = \{x \in \mathbb{R}_+^{|V|} \mid x_u + x_v \leq 1;\, uv \in E_p\}$; then we could describe the scheduling polytope by adding intergrality constraints as $P_I(\mathcal{N}) = conv\{\, x \in P_F(\mathcal{N} \mid x \text{ integer}\,\}$.

Section I-A stated that, while formulating an ILP, one should attempt to describe $P_F(\mathcal{N})$ as tightly as possible (since the goal is to obtain an integral polytope). Two

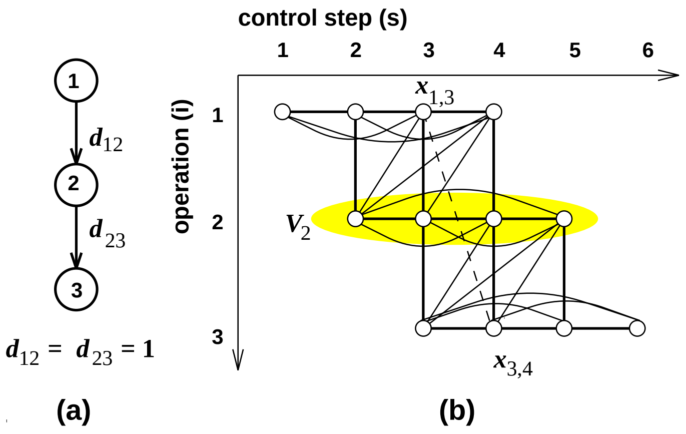

Fig. 2. A Constraint graph (a) Timing Relations in the cdfg (b) The Corresponding Constraint Graph G_p. The Dotted Edge is not in G_p, and is only Implied by the Timing Relations.

methods for obtaining a tighter description of $P_F(\mathcal{N})$ are discussed below.

In the first method, the above description of $P_F(\mathcal{N})$ is tightened by adding new edge constraints for edges that are implied by the timing constraints but are not explicitly included in G_p. For example, in Figure 2 (a), operation 1 cannot be scheduled into cstep 3 (i.e., $x_{1,3} = 1$) when operation 3 is scheduled into cstep 4 (i.e., $x_{3,4} = 1$). This condition is denoted by the dotted edge in Figure 2 (b), and represents the edge constraint $x_{1,3} + x_{3,4} \leq 1$.

Since all *integer points* inside $P_F(\mathcal{N})$ satisfy the above edge constraint, it is not mandatory to include this constraint in the description of $P_F(\mathcal{N})$ (thus the dotted edge can be ignored while forming the constraint graph G_p). However, it is easy to show that there exists a *fractional point* inside $P_F(\mathcal{N})$ that violates this constraint. Therefore, we could add this constraint to our description of $P_F(\mathcal{N})$ to make it tighter.

Instead of considering the edges of G_p, the second method considers the cliques of G_p, and replaces all the edge constraints with *clique constraints* to give an alternative description of $P_F(\mathcal{N})$. Since clique constraints are tighter than edge constraints (more specifically, clique constraints are facets of $P_I(\mathcal{N})$), this description of $P_F(\mathcal{N})$ is tighter than the description in terms of edge constraints.

Gebotys [8] used clique constraints to describe $P_F(\mathcal{N})$ as the node-packing polytope (see Definition 5) of G_p, and showed that this description is tighter than the one used in ALPS [9]. However, in the general case, the node packing polytope of a graph could be non-integral, and it is not clear how tight this description of $P_F(\mathcal{N})$ is.

As described in the next section, we use a combination of the two methods above in order to derive the tightest possible description of $P_F(\mathcal{N})$. First, we add edges to G_p and consider a larger, well-structured constraint graph G_c whose node-packing polytope is integral. Then we describe $P_F(\mathcal{N})$ as the node-packing polytope of G_c, thus leading to an integral polytope.

B. The Constraint Graph G_c

As discussed in the previous section, the node-packing polytope of a graph could be non-integral, so describing $P_F(\mathcal{N})$ as the node-packing polytope of G_p might not lead to the tightest description of $P_F(\mathcal{N})$. Therefore, instead of using G_p, we add some edges that are implied by the timing constraints but not explicitly included in G_p, and consider a larger constraint graph G_c. Edges are added in such a way that the new constraint graph G_c is well-structured, and has an integral node-packing polytope. Thus when we describe $P_F(\mathcal{N})$ using the clique constraints of G_c, we can claim that we have the tightest possible description of $P_F(\mathcal{N})$.

Although a larger graph might imply a large number of constraints, we show that all the clique constraints of G_c are linear combinations of a small number of basic constraints. There are two types of basic constraints, given in (A) and (T) below, that will be shown in Section III-B.2 to be sufficient for describing all the clique constraints of G_c.

B.1 Construction of Constraint Graph G_c from the Skeleton Digraph G_s

We generate the constraint graph G_c from a *skeleton digraph*, $G_s = \{V, A_s\}$, which is discussed in this section.

The arcs of the skeleton digraph G_s belong to two classes described as follows:

Assignment arcs: From every node $(i, s) \in V$, we create an arc a_{iis} directed to $(i, s-1)$ and denote it as:

$$a_{iis} \triangleq (i, s) \rightarrow (i, s-1)$$

(for example, see arc a_{113} in Figure 3).

Timing arcs: For each timing arc $a_{ij} \Rightarrow o_i \xrightarrow{d_{ij}} o_j$ in the cdfg, we construct an arc a_{ijs} directed from (i, s) to $(j, s + d_{ij} - 1)$, and denote it as:

$$a_{ijs} \triangleq (i, s) \rightarrow (j, s + d_{ij} - 1)$$

(for example, see arc a_{235} in Figure 3).

The arcs of the skeleton digraph corresponding to Figure 2 (a) are shown in Figure 3 using thick arrows.

Let $G_s^* = (V, A_s^*)$ denote the *transitive closure* of the skeleton digraph G_s. We use the underlying graph G_c of the digraph G_s^* as the constraint graph. For example, G_c can be obtained by ignoring the directions on the edges in Figure 3. It can be easily seen that G_p is a subgraph of G_c.

It should be noted that although our constraint graph G_c contains more edges than the initial constraint graph G_p, it does not not include all possible edges that are implied by the timing constraints. Edges are included only if they are necessary to make G_c well-structured. For example, the dotted edge in Figure 2, although implied by the timing constraints, is not considered for inclusion in G_c because we can prove that G_c is already well-structured without this edge. Therefore, it is not necessary to consider this edge in the tightest possible description of $P_F(\mathcal{N})$.

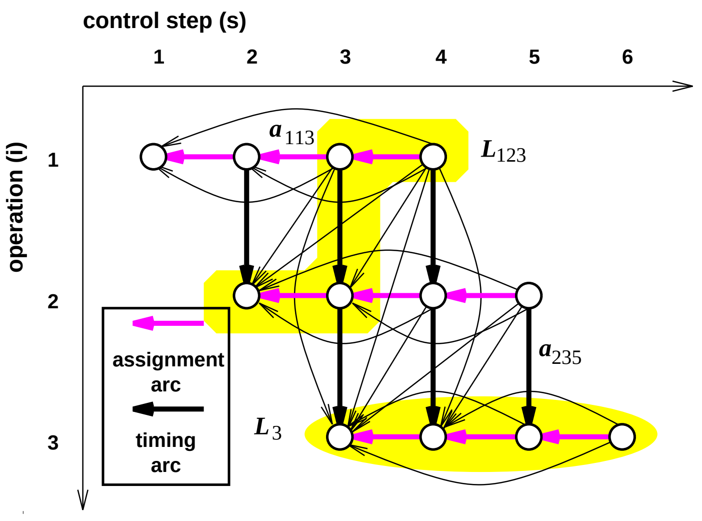

Fig. 3. Skeleton Digraph G_s corresponding to Figure 2 (arcs denoted by thick arrows), and its Transitive Closure G_s^* (all arcs). Constraint Graph G_c is the Same as G_s^*, with no Directions on the Edges.

B.2 Generating all Cliques of the Constraint Graph G_c

Although the general problem of generating all the cliques is NP-complete, we will show that all the clique constraints of the graph G_c are linear combinations of two small basic classes of constraints. Our proof consists of two steps: first, we show that for every maximal clique in G_c, there exists a maximal path in G_s with the same set of nodes (Corollary 1), and then we present the minimal set of path constraints in G_s that are sufficient to generate all the clique constraints of G_c (Theorem 2).

Definition 1: A *tournament* is a complete graph in which each edge is assigned a direction.

Given a graph G, we will use the notation $F \subseteq G$ to indicate that F is a subgraph of G, and we will use $V(F)$ to denote the set of nodes of subgraph F.

Definition 2: Given a graph G, a subgraph $F \subset G$ is called a *maximal* clique (resp. path, tournament) if (1) F is a clique (resp. path, tournament), and (2) there exists no other clique (resp. path, tournament) H such that $F \subset H \subseteq G$.

Claim 1: For every path $L \subseteq G_s$, there exists a tournament $T_L \subseteq G_s^*$ such that $V(T_L) = V(L)$ (i.e, T_L contains exactly the same set of nodes as L).

Proof: Each pair of nodes that lie on L are directly connected by an arc in G_s^*, due to transitive closure. Therefore such nodes form a tournament T_L in G_s^*, such that $V(T_L) = V(L)$. □

Theorem 1: For every maximal tournament $T \subseteq G_s^*$, there exists a unique maximal path $L_T \subseteq G_s$ such that $V(L_T) = V(T)$ (i.e., L_T contains exactly the same set of nodes as T).

Proof: Let T be any maximal tournament in G_s^*. First we construct a path $L_T \subseteq G_s$ such that $V(L_T) = V(T)$; the uniqueness and the maximality of L_T will be proven in the next part of the proof.

From Theorem 14 in [23], we know that T has at least one spanning path L, i.e., $V(L) = V(T)$. Since $T \subseteq G_s^*$, the spanning path $L \subseteq G_s^*$. Furhtermore, since G_s^* is the transitive closure of G_s, $G_s \subseteq G_s^*$. However, from the above, we can not necessarily conclude $L \subseteq G_s \subseteq G^*$. Although each arc of L belongs to A_s^*, it might not belong to A_s.

In general, there may exist an arc (a,b) in L, such that $(a,b) \notin A_s$. It can be concluded that (a,b) must have been created while taking the transitive closure on G_s. Therefore, there must be an $a - b$ path $L_{ab} \subseteq G_s$. By replacing each such $(a,b) \notin A_s$ with its respective L_{ab}, we can obtain a walk [24] L_T, such that each arc of L_T belongs to A_s and $L_T \subseteq G_s$.

In order for the walk L_T to be a path, it is required that L_T does not go through the same node twice. Since G_s is acyclic (proof given in Appendix I), no node can appear twice in L_T; hence L_T is a path in G_s.

While constructing L_T from L, we retained all the nodes that were originally in L. Therefore we can write:

$$V(L) \subseteq V(L_T) \tag{4}$$

Since L is a spanning path of T, $V(L) = V(T)$. Furthermore, from Claim 1, we know that there exists a tournament $T_{L_T} \subseteq G_s^*$, such that $V(L_T) = V(T_{L_T})$. Using these values of $V(L)$ and $V(L_T)$ in (4), we find $V(T) \subseteq V(T_{L_T})$. However, $V(T) \not\subset V(T_{L_T})$ because we have assumed T to be maximal, and we can conclude $V(T) = V(T_{L_T}) = V(L_T)$.

The uniqueness of path L_T follows from the acyclic property of G_s. The maximality of path L_T can be proven by contradiction as follows. If L_T is not a maximal path in G_s, then there must be another path L' in G_s such that $L' \supset L_T$. However, this implies $T_{L'} \supset T$, which contradicts our assumption that T is maximal. Therefore, L_T must be a maximal path in G_s. □

Corollary 1: For each maximal clique C of G_c, there exists a maximal path L_C in G_s such that $V(C) = V(L_C)$.

If, for any $U \subseteq V$ we define $|x^U|$ as:

$$|x^U| \triangleq \sum_{u \in U} x_u, \quad \text{for any } U \in V$$

then a maximal clique constraint for clique C can be written as $|x^{V(C)}| \leq 1$. From Corollary 1, we know $V(C) = V(L_C)$, and we can write:

$$\text{Constraint for clique } C\text{:} \qquad |x^{V(L_C)}| \leq 1 \tag{5}$$

Thus every maximal clique constraint for G_c corresponds to a maximal path in G_s, and all the maximal clique constraints for G_c can be found by examining the maximal paths in G_s. We will show that it is enough to consider the constraints for only two classes of maximal paths in G_s; the constraints for all other maximal paths are linear combinations of these two classes of constraints. The two classes of constraints are described as follows:

(1) For every $i \in I$, we consider the path L_i consisting of all the *assignment arcs* $a_{iis} \in A_s$. Such a path connects all the nodes in V_i. For example, path L_3 is indicated by a shaded ellipse in Figure 3. The clique constraint for L_i is given below:

$$|x^{V(L_i)}| \triangleq \sum_{s \in S_i} x_{i,s} = 1, \quad \forall\, i \in I \tag{A}$$

Constraints (A) are called *assignment* constraints; they ensure that the feasible schedule contains exactly one node per operation.

(2) For every *timing arc* $a_{ijs} \in A_s$, we consider a path L_{ijs} consisting of the timing arc a_{ijs}, and the assignment arcs $\{\, a_{iis'} \mid s' > s\, ;\, a_{iis'} \in A_s \,\}$ and $\{\, a_{jjs'} \mid s' \leq s + d_{ij} - 1\, ;\, a_{jjs'} \in A_s \,\}$. For example, path L_{123} is indicated by a shaded polygon in Figure 3. The clique constraint for L_{ijs} is given below:

$$|x^{V(L_{ijs})}| \triangleq \sum_{\substack{s' > s - d_{ij} \\ s' \in S_i}} x_{i,s} + \sum_{\substack{s' \leq s \\ s' \in S_j}} x_{j,s} \leq 1 \qquad \begin{array}{l} \forall\, a_{ijs} \in A_s \\ \forall\, s \in (S_i + d_{ij} - 1) \cap S_j \end{array} \tag{T}$$

Constraints (T) are called *timing* constraints, and they prevent two nodes that are in timing conflict from being in the same schedule.

Theorem 2: Any maximal clique constraint of G_c is a linear combination of constraints (A) and (T).

The proof of this theorem is presented in Appendix II.

In this section, we have presented the constraint graph G_c, and the assignment constraints (A) and timing constraints (T) for the scheduling problem. We have proven that (A) and (T) generate all the clique constraints of G_c. In a later section (Section IV-A) we will show that the constraint graph G_c is well-structured, which allows us to conclude that we are able to describe the polytope $P_F(\mathcal{N})$ as tightly as possible, even in the most general case when minimum, maximum and fixed timing constraints are present.

C. Description of the Scheduling Polytope

In the previous section, we have presented the assignment constraints (A) and timing constraints (T) for the scheduling problem. To ensure that the schedule does not use more than the available number of FU's, we need a third type of constraints, namely the *resource constraints*, which, for uni-cycle operations, are given below:

$$|x^{V_{k,s}}| \triangleq \sum_{i : \tau(i) = k} x_{i,s} \leq m_k, \quad s \in S, \ \forall\, k \tag{R}$$

The constraints (A), (T), and (R) can be represented in the form $\{\, x \in \mathbb{R}_+^{|V|} \mid M_a x = 1\ ;\ M_t x \leq 1\ ;\ M_r x \leq n\ ;\ x \text{ integer} \,\}$, where M_a is the coefficient matrix due to the assignment constraints, M_t is the coefficient matrix due to the timing constraints, and M_r is the coefficient matrix due to the resource constraints. If we denote the *fractional scheduling polytope* as:

$$P_F(\mathcal{Q}) = \{\, x \in \mathbb{R}_+^{|V|} \mid M_a x = 1\ ;\ M_t x \leq 1\ ;\ M_r x \leq m \,\} \tag{6}$$

then we can write:

$$P_I(\mathcal{Q}) = conv\{\, x \in P_F(\mathcal{Q}) \mid x \text{ integer} \,\} \tag{7}$$

In this section we have given a description of $P_I(\mathcal{Q})$, that in addition to equality and inequality constraints, also requires the variables to be integral. Using this preliminary description, we will examine the structure of $P_F(\mathcal{Q})$ to see how close it is to $P_I(\mathcal{Q})$. As mentioned in Section I-A, it is extremely important that $P_F(\mathcal{Q})$ be close to $P_I(\mathcal{Q})$.

IV. The Structure of the Constraints

As we stated in Section I-A, the success of an ILP-based scheduling algorithm depends on tightly defining $P_F(\mathcal{Q})$ so that it closely approximates $P_I(\mathcal{Q})$. The purpose of this section is to examine, by analyzing the structure of the constraints, how close $P_F(\mathcal{Q})$ is to $P_I(\mathcal{Q})$. Although a thorough examination is as hard as solving the scheduling problem itself, we can get some useful information by selectively dropping some of the constraints. In Section 3.1 and 3.2 we will drop the resource and timing constraints, respectively, and in Section 3.3 we will consider all the constraints together to see how they interact.

In [8], Gebotys applied polyhedral theory to analyze only the structure of the timing constraints. In contrast, we present a more comprehensive analysis that considers timing as well as resource constraints. Furthermore, we show new theoretical results that provide additional insight into the structural properties of the problem, and will serve as a basis for future improvement of scheduling algorithms.

A. The Polytope of the Assignment and the Timing Constraints

In this section we will drop the resource constraints, and consider the subsets of V, called *feasible timing allocations*, that satisfy the assignment and timing constraints. Let $\mathcal{N}$ be the set of all feasible timing allocations. The convex hull of the incidence vectors of all the feasible timing allocations constructs the *timing-assignment* polytope, which can be described as follows:

$$\begin{aligned} P_F(\mathcal{N}) &= \{x \in \mathbb{R}_+^{|V|} \mid M_a x = 1 \,;\, M_t x \le 1\} \\ P_I(\mathcal{N}) &= \text{conv}\{x \in P_F(\mathcal{N}) \mid x \text{ integer}\} \end{aligned}$$

Instead of considering only the feasible timing allocations, if we also include all the subsets of each feasible timing allocation, then we get the *monotone timing-assignment* polytope; its fractional counterpart is given below:

$$P_F(\tilde{\mathcal{N}}) = \{x \in \mathbb{R}_+^{|V|} \mid M_a x \le 1 \,;\, M_t x \le 1\} \tag{8}$$

Definition 3: [10]. A *node packing* on a graph $G = \{V, E\}$ is a set $U \subseteq V$ with the property that no two nodes in U are joined by an edge.

Definition 4: [10]. A *clique matrix* of a graph G is a 0-1 matrix K whose columns correspond to the nodes of G and rows correspond to the incidence vectors of all the maximal cliques of G.

Definition 5: [10]. The *fractional node packing polytope* of a graph G is $P = \{x \in \mathbb{R}_+^n,\ Kx \le 1\}$, where K is the clique matrix of G.

We have shown in Section III-B.2 that the constraints (A) and (T) generate all the cliques of the constraint graph G_c. Therefore, for G_c, the rows of the clique matrix are nothing but the rows of M_a and M_t, and its *fractional node-packing polytope* is the same as its monotone fractional timing-assignment polytope, as can be verified from (8).

Definition 6: [25]. A graph $G = \{V, E\}$ is called *transitively orientable* if each edge can be assigned a one-way direction in such a way that the resulting oriented graph (V, F) satisfies the following property:

$$(a, b) \in F \text{ and } (b, c) \in F \;\; implies \;\; (a, c) \in F$$

Proposition 2: The constraint graph G_c is transitively orientable

Proof: The proof follows immediately from the construction of G_c, which has been described in Section III-B.1. Since G_c is the underlying graph of digraph G_s^*, it is enough to examine whether G_s^* satisfies the transitive property mentioned in Definition 6. It is obvious that G_s^* satisfies the transitive property, because G_s^* itself is the transitive closure of digraph G_s. □

Proposition 3: The fractional monotone timing assignment polytope is integral, i.e $P_F(\tilde{\mathcal{N}}) = P_I(\tilde{\mathcal{N}})$.

Proof: A transitively orientable graph is a *perfect graph* [25], hence the constraint graph G_c is also a perfect graph. By definition, the fractional node-packing polytope of a perfect graph is integral [10], which implies that $P_F(\tilde{\mathcal{N}})$ is an integral polytope. □

The above result immediately leads to the integrality of the fractional timing assignment polytope, and is stated in the following corollary:

Corollary 4: The polytope defined by the assignment and the timing constraints is integral,
i.e. $P_F(\mathcal{N}) = P_I(\mathcal{N})$

Proof: To prove our claim, we shall use a well-known result [25] in linear programming:

Lemma 1: Given bounded polyhedra S and T, where S has a finite number of extrema, $S = T$ iff $\max\{c^T x \mid x \in S\} = \max\{c^T x \mid x \in T\}$ ($\forall c$, integral)

Let us consider any $c \in \mathbb{Z}^{|V|}$.

$$\begin{aligned} & \max\{c^T x \mid x \in P_F(\mathcal{N})\} \\ = \; & \max\{c^T x \mid M_a x = 1 \,;\, x \in P_F(\tilde{\mathcal{N}})\} \\ = \; & \max\{c^T x \mid M_a x = 1 \,;\, x \in P_I(\tilde{\mathcal{N}})\} \\ = \; & \max\{c^T x \mid x \in P_I(\mathcal{N})\} \end{aligned}$$

The above lemma, when applied to this result, leads to the conclusion, $P_F(\mathcal{N}) = P_I(\mathcal{N})$ □

In this section we have shown that, for any cdfg with no positive length cycles, the polytope $P_F(\mathcal{N})$ is integral; hence, the assignment constraints (A), and the timing constraints (T) describe $P_F(\mathcal{N})$ as tightly as possible. This property remains valid even when additional architectural features, such as pipelining, loop-folding, multi-cycle operations and chaining are added to the model.

B. The Polytope of the Assignment and the Resource Constraints

In this section we will drop the timing constraints, and consider the family $\mathcal{R}$ of subsets of V that satisfy the resource and the assignment constraints. These subsets are called *feasible resource allocations*. The *resource-assignment* polytope $P_I(\mathcal{R})$ is the convex hull of the incidence vectors of all the feasible resource allocations, and is described as:

$$\begin{aligned} P_F(\mathcal{R}) &= \{x \in \mathbb{R}_+^{|V|} \mid M_a x = 1 \; ; \; M_r x \leq m\} \\ P_I(\mathcal{R}) &= \text{conv}\{x \in P_F(\mathcal{R}) \mid x \text{ integer}\} \end{aligned}$$

We want to show that the assignment and the resource constraints completely describe $P_I(\mathcal{R})$, i.e. $P_F(\mathcal{R}) = P_I(\mathcal{R})$. Before proceeding further, we need to prove the following lemma.

Lemma 2: The coefficient matrix $A = \begin{bmatrix} M_a \\ M_r \end{bmatrix}$ describing the constraints of $P_F(\mathcal{R})$ is totally unimodular (TU).

Proof: The rows of A are nothing but the incidence vectors of V_i and $V_{k,s}$, $i \in I, k \in K, s \in S$. From the definition it is clear that each node $(i,s) \in V$ belongs to exactly one V_i and one $V_{k,s}$. Thus each node has an entry 1 in exactly two rows; in other words, each column of A contains exactly two 1's, one in M_a and one in M_r. This is a sufficient condition for a matrix to be TU (Chapter III.1, Corollary 2.8 in [10]). □

Proposition 5: For any fixed $m \in \mathbb{Z}_+^{|K|}$, the polytope defined by the assignment and the resource constraints is integral, i.e. $P_F(\mathcal{R}) = P_I(\mathcal{R})$

Proof: It is a well-known result that if A is TU then $P = \{x \in \mathbb{R}_+^n \mid Ax \leq b\}$ is integral when the elements of b are integral [10]. When this fact is applied to the description of $P_F(\mathcal{R})$ along with the result of the previous lemma, the proof is obvious. □

When multi-cycle operations (operations with latency greater than 1) are considered, let $l_{\tau(i)}$ be the latency of FU-type $\tau(i)$ on which operation o_i is executed. Operation o_i can be decomposed into $l_{\tau(i)} + 1$ uni-cycle operations. The resource constraints, in this case, would be similar to (R), and $P_F(\mathcal{R})$ would be integral. New fixed timing constraints would be needed to ensure that the decomposed operations are scheduled into consecutive control steps. However, these new constraints would still imply an integral timing-assignment polytope.

Another method is to modify the resource constraints as follows:

$$\sum_{i:\tau(i)=k} \sum_{s'=s-l_{\tau(i)}+1}^{s} x_{i,s'} \leq m_k, \quad s \in S, \; \forall \, k \qquad (\text{R}')$$

However, in this case, the resource-assignment polytope is not always integral.

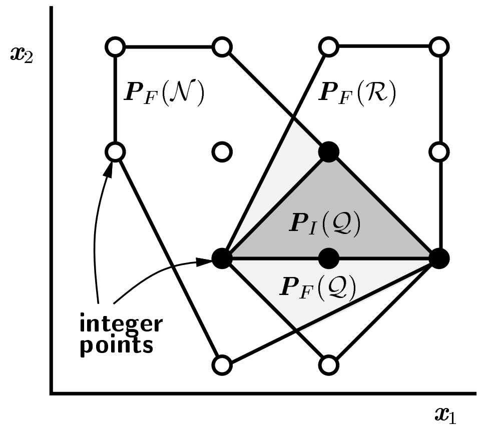

Fig. 4. Feasible Regions of the RCS and TRCS Scheduling Problems

Note that in Proposition 5, the vector m is fixed; therefore, the integrality results are applicable only to scheduling problems where the number of FU's are fixed (eg., RCS or TRCS). If the number of FU's are treated as variables, $P_F(\mathcal{R})$ is not integral even for uni-cycle operations. Therefore, fixing the elements of m leads to a tighter formulation.

In this section we have shown that the polytope $P_F(\mathcal{R})$ is integral; hence the assignment constraints (A) and the resource constraints (R) describe $P_F(\mathcal{R})$ as tightly as possible.

C. The Polytope of the Assignment, Timing, and the Resource constraints

In the previous sections, we have analyzed the structure of the resource-assignment polytope and the timing-assignment polytope; in this section, we will investigate the structure of the fractional scheduling polytope $P_F(\mathcal{Q}) = P_F(\mathcal{R}) \cap P_F(\mathcal{N})$. As previously mentioned, our motivation is to investigate how close $P_F(\mathcal{Q})$ is to $P_I(\mathcal{Q})$.

From Proposition 5 we know that $P_F(\mathcal{R})$ is integral for the RCS and the TRCS problems. Furthermore, from Corollary 4 we know that $P_F(\mathcal{N})$ is integral. More specifically, for the RCS and the TRCS problems, $P_F(\mathcal{Q})$ is the intersection of two integral polytopes, as illustrated in Figure 4. However, as can be seen from Figure 4, this structure of $P_F(\mathcal{Q})$ does not necessarily imply that it is integral. There can be instances of the scheduling problem in which $P_F(\mathcal{Q})$ is fractional, as can be demonstrated with a counterexample.

For the UCS and the TCS problems, $P_F(\mathcal{R})$ is non-integral, so $P_F(\mathcal{Q})$ does not approximate $P_I(\mathcal{Q})$ as tightly as for the RCS and the TRCS problems. Although this observation is intuitively plausible; it can play a major role in deciding which scheduling problem can be efficiently solved using an ILP formulation. In the next section we will present a theoretical result more along this line.

V. Using the Constraints in an ILP Formulation

In the previous sections we have described and analyzed the constraints that describe the fractional scheduling polytope $P_F(\mathcal{Q})$. These constraints can be used to formulate an

ILP that represents a TRCS, TCS, RCS, or UCS problem. In this section we will present these different formulations, and discuss how each formulation takes advantage of the structure of the constraints.

UCS The UCS (Unconstrained Scheduling) problem minimizes a function of the number of hardware resources and the number of control steps. For this problem, a sufficiently large bound on the number of control steps is usually given, that can be used to compute the schedule interval of the operations. To compute the number of control steps required to schedule a cdfg, we introduce a dummy operation o_d. All the operations that have no successors, are given o_d as their successor in order to make sure that o_d is scheduled after all other operations in the cdfg. The UCS problem is then formulated as:

$$\min \{ c_1 \sum_{s \in S_d} s x_{d,s} + c_2 \sum_{k \in K} a_k m_k \,|\, x \in P_F(\mathcal{Q})\,;\ x \ \text{ integer} \}$$

RCS An RCS (Resource-Constrained Scheduling) problem minimizes the number of control steps when the number of FU's is fixed. Since no bounds are given on the number of control steps, we find an upper bound using *list scheduling*, and then determine the schedule intervals of the operations using this bound. The total number of control steps is determined using a dummy operation, in a manner similar to the UCS problem. The RCS problem is then formulated as:

$$\min \{ \sum_{s \in S_d} s x_{d,s} \,|\, x \in P_F(\mathcal{Q})\,;\ x \ \text{ integer} \}$$

TCS A TCS (Time-Constrained Scheduling) problem minimizes a function of the number of the FU's when the number of control steps is fixed. This problem is formulated as:

$$\min \{ \sum_{k \in K} a_k m_k \,|\, x \in P_F(\mathcal{Q})\,;\ x \ \text{ integer} \}$$

where a_k is an weight (usually area) associated with an FU of type k.

TRCS A TRCS (Time- and Resource-Constrained Scheduling) problem mininimizes an objective function cx when both the number of FU's and the number of control steps are fixed. This problem is formulated as:

$$\min \{ cx \,|\, x \in P_F(\mathcal{Q})\,;\ x \ \text{ integer} \}$$

A. Choosing a Formulation

Although the descriptions of the four scheduling formulations given above are similar, they can vary widely in solution time. In this section we will discuss which formulation should be chosen so that the resulting ILP can be solved efficiently.

In considering these four different formulations, note that the number of FU's are treated as variables in UCS and TCS, while they are fixed in RCS and TRCS. It has been discussed in Section IV-C that fixed number of FU's leads to a tighter description of $P_F(\mathcal{Q})$ for RCS and TRCS, as compared to UCS and TCS.

Such a tighter formulation has two advantages: (1) it increases the likelihood that the LP-relaxation has an integer solution, and (2) it produces sharp bounds on the objective function. A sharp bound is necessary for the following reason. Since $P_F(\mathcal{Q})$ is non-integral, the LP-relaxation can produce fractional solutions, which will require us to use branch-and-bound to find the integral optimal solution. In order for the branch-and-bound approach to be successful, it is important to find a sharp bound on the objective function, so that branches can be pruned efficiently.

Furthermore, for RCS and TRCS, new techniques (to be discussed in Section VII) can be used to further tighten the formulation when the number of FU's are fixed. Therefore, fixing the number of FU's, as in RCS and TRCS, will lead to a tighter formulation and a quicker solution.

Now consider only RCS and TRCS, and note that these two formulations have similar constraint structure, and differ in the objective function only. Since the quality of the bounds is independent of the particular objective function used, the remainder of this section will use a generic objective function, and will consider the following ILP formulation, which is similar to TRCS:

$$z_{IP} = \min \{cx \,|\, x \in P_F(\mathcal{Q})\,;\ x \text{ integer}\} \qquad \text{(ILP)}$$

The ILP formulation (ILP) can incorporate pipelining, conditionals, loop-folding, chaining, and multi-cycle operations; their effects on the problem structure has already been mentioned in Section IV. As for the TCS and the UCS problems, the number of FU's can be found using a quick lower bounding scheme [26]; then the formulation (ILP) can be used to find the optimal schedule.

B. Performance of the ILP Formulation

As mentioned above, in order for a branch-and-bound algorithm to be successful, it is important to quickly find a tight bound on the objective function. The bound on the objective function value is usually obtained by solving a relaxation of the original problem, most commonly an LP-relaxation. Another kind of relaxation, called the *Lagrangian relaxation*, produces a tighter bound, and has led to the success of Lagrangian relaxation-based branch-and-bound algorithms to solve the traveling salesman problem [27], and the minimum-tardiness-scheduling problem [28]. Fisher [29] has reported that the bounds produced by Lagrangian relaxation are on the average 95% within the optimum value, and that such tight bounds allow efficient pruning of the branches.

The LP-relaxation of (ILP) is given as:

$$z_{LP} = \min \{cx \,|\, x \in P_F(\mathcal{Q})\} \qquad \text{(LP)}$$

In the following proposition we show that, for our problem, the LP-relaxation gives a bound equivalent to the Lagrangian relaxation.

Proposition 6: Let z_{AR} and z_{AP} be the Lagrangian bounds obtained when the timing and resource constraints are relaxed respectively. Then $z_{AR} = z_{AP} = z_{LP}$

Proof: We start with the definition of $P_F(\mathcal{Q})$ as given in (6):

$$\begin{aligned} & P_F(\mathcal{Q}) \\ = & \{\, x \in \mathbb{R}_+^{|V|} \mid M_a x = 1\,;\; M_t x \le 1\,;\; M_r x \le m \,\} \\ = & \{\, M_t x \le 1\,;\; x \in P_F(\mathcal{R})\} \qquad (9) \\ = & \{\, M_r x \le m\,;\; x \in P_F(\mathcal{N})\} \qquad (10) \end{aligned}$$

Applying Proposition 5 to (9), we can write:

$$P_F(\mathcal{Q}) = \{\, M_t x \le 1\,;\; x \in P_I(\mathcal{R})\} \qquad (11)$$

Similarly, Corollary 4 can be applied to (9) to give the following equation:

$$P_F(\mathcal{Q}) = \{\, M_r x \le m\,;\; x \in P_I(\mathcal{N})\} \qquad (12)$$

Now we use Theorem 6.2, Chapter II.3 in [10] to conclude the following:

$$\begin{aligned} z_{AR} &= \min\{cx \mid M_t x \le 1\,;\; x \in P_I(\mathcal{R})\} \\ &= \min\{cx \mid x \in P_F(\mathcal{Q})\} \quad \text{from (11)} \\ z_{AP} &= \min\{cx \mid M_r x \le m\,;\; x \in P_I(\mathcal{N})\} \\ &= \min\{cx \mid x \in P_F(\mathcal{Q})\} \quad \text{from (12)} \end{aligned}$$

The above expressions together with (LP) lead to the conclusion $z_{AR} = z_{AP} = z_{LP}$ □

The significance of the results of this section is that the bounds produced by the LP-relaxation are as good as the bounds from the Lagrangian relaxation. This is probably the reason why a small number of branches were required to optimally solve all the test problems we have run. Such experimental results will be presented in Section IX.

VI. Comparison with Previous Formulations

This paper has presented a formal analysis of the constraints on the scheduling problem, providing a theoretical foundation for choosing an appropriate ILP formulation. With the exception of some work in OASIC [8], most work on ILP-based schedulers (ALPS [9], GRAD [15]) has virtually ignored this analysis. Thus our work is the first in-depth analysis of the structure of the constraints on the scheduling problem.

To date, two different types of timing constraints have been used in ILP formulations. One type of constraints was used in ALPS [9], and a second type of constraints, which are tighter than the first type, was first used in GRAD [15]. OASIC employed the same type of timing constraints as GRAD, but Gebotys [8] was the first to show that the polytope constituted by the timing and assignment constraints (referred to as timing-assignment polytope $P_F(\mathcal{N})$ in Section IV-A) is similar to the node-packing polytope of the constraint graph G_p (see Figure 2).

However, in the general case, the node-packing polytope of a graph could be non-integral unless the constraint graph is well-structured. To obtain a well-structured constraint graph, we added edges that are implied by the timing constraints but that are not explicitly included in G_p, and considered a larger constraint graph G_c. We proved that the node-packing polytope of G_c is integral. Thus we were able to find and prove the tightest possible description of $P_F(\mathcal{N})$ (i.e. $P_F(\mathcal{N})$ is integral), even in the most general case when minimum, maximum and fixed timing constraints between operations are present. Although G_c has a large number of cliques, we showed that it is sufficient to consider a small number of clique constraints. The remaining constraints are linearly dependent on those constraints and are redundant.

The timing constraints that we derived from G_c are of the same type as GRAD and OASIC (clique constraints); however, we have determined the minimum number of clique constraints that are necessary to give an integral description of the timing-assignment polytope. Furthermore, it follows from our analysis that the constraints (9.3) in [8] are linearly dependent on other clique constraints, and can therefore be omitted.

The resource constraints that we considered are similar to those in ALPS and OASIC, but until now, there was no analysis of those constraints. In this paper, we have investigated the structure of the resource-assignment polytope $P_F(\mathcal{R})$, and have presented the tightest possible description. For uni-cycle operations, we used the same resource constraints as OASIC and ALPS; however, for multi-cycle operations, we have shown an alternative methodology that also leads to an integral resource-assignment polytope. Although the algebraic description of the resource constraints is intuitively clear, the knowledge of their structure opens the possibility of new applications. For example, once we knew that the resource-assignment polytope was integral, we found that we could use the resource constraints to compute highly accurate lower bounds on the number of FU's in an efficient manner [26].

In summary, our intention was not to present just another ILP formulation with different variables and constraints. We took a step backwards, considered the properties of the ILP constraints that are required for a well-structured formulation, and then analyzed the scheduling constraints to identify which descriptions of the constraints would imply a structured formulation. Future advances in the ILP approach to the scheduling problem could involve identifying new facets, and computing tight bounds on the search space. The knowledge of the problem structure is essential for progress in these directions, and the results presented here can provide a theoretical basis for such improvements.

VII. Tightening the Formulation

The formulation (ILP), described in the previous section, can produce sharp bounds on the objective function. Thus when the LP-relaxation of problem (ILP) produces a non-integral solution, we can use a branch-and-bound search and expect that the integral solution would be found in a small number of branches. At this point, the key to further improvement in the efficiency of the ILP solution lies in tightening the description of $P_F(\mathcal{Q})$ so that it approximates $P_I(\mathcal{Q})$ more closely; it has been mentioned in [1] that this line of attack has recently proven to be very successful for ILP algorithms.

We have already analyzed the interaction between the resource and the assignment constraints in Section IV-B, and the interaction between the timing and the assignment constraints in Section IV-A; in both cases we defined the tightest possible description of the corresponding polytopes. Therefore, in order to further tighten the description of $P_F(\mathcal{Q})$ we need to consider the interaction between the resource and the timing constraints.

In this section we present two new ways to tighten the description of $P_F(\mathcal{Q})$. In Section VII-A we describe how resource constraints can be used to shorten an operation's schedule interval. In Section VII-B we present a new class of valid inequalities that can be found by considering the effect of timing edges on the resource constraints. Research in such techniques for tightening the description of $P_F(\mathcal{Q})$ are extremely important for improving the solution efficiency.

A. Preprocessing

In the original constraints of Section III, the schedule interval of each operation was determined by ASAP and ALAP scheduling. Ordinary ASAP and ALAP algorithms assume an unlimited number of FU's, but the algorithms can be modified to take into account the number of FU's. These modifications help to reduce the schedule interval of each operation, leading to a formulation using fewer variables. This procedure has the effect of dropping some variables from the preliminary constraints of Section III, and of cutting off some fractional extreme points from $P_F(\mathcal{Q})$. In the following proposition we indicate a way of modifying the ordinary ASAP algorithm so that it considers constraints on the number of FU's; a similar technique can be applied to the ALAP counterpart.

Proposition 7: Let, for operation o_i, the number of total predecessors and nonimmediate predecessors of type k be denoted by $npred_{i,k}$ and $pred_{i,k}$ respectively. An ASAP time $a_{i,k}$ of o_i is given by:

$$a_{i,k} = \max\left\{ \left\lceil \frac{pred_{i,k}}{m_k} \right\rceil , \left\lceil \frac{npred_{i,k}}{m_k} \right\rceil + 1 \right\} + 1 \tag{13}$$

Proof: In presence of resource constraints, each control step can be regarded as a bin of capacity m_k, and is labeled with the corresponding control step number. The first term is included to make sure that there are enough bins to pack all the predecessors of o_i. The second term is required to ensure that after the nonimmediate predecessors of o_i have been packed, there is one bin available to the immediate predecessors of o_i. □

For an illustration of how the above modification can affect the ASAP time of an operator, consider the data flow graph of Fig 5, with a resource constraint of 3 adders. For the shaded operation at the bottom, an ordinary ASAP scheduling algorithm would schedule the operation in the 4th control step. However, due to the resource constraint, the operation can not be placed into control step 4 in any feasible schedule. When Proposition 7 is taken into account, the following values can be computed: $npred = 7$, $pred = 9$. Using these values in (13), the operation is scheduled into the 5th control step. Thus the modification based on Proposition 7 will lead to a tighter formulation using fewer variables.

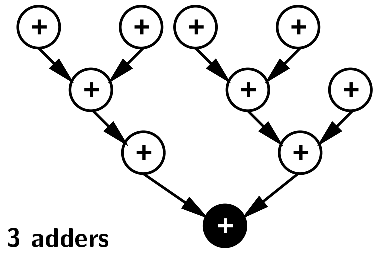

Fig. 5. Effect of Resource Constraints on the ASAP Algorithm

B. Valid Inequalities

The description of $P_F(\mathcal{Q})$ can also be made tighter when the resource constraints are modified by considering the effects of the timing edges that exist between nodes in $V_{k,s}$, as stated in Proposition 8

Definition 7: A set of cliques $\{C_l \mid l = 1, \ldots, p\}$ is called a *clique cover* of a set V of nodes if $V = \bigcup_{l=1}^{p} C_l$. A clique cover is *minimal* if each C_l contains at least one node which is not contained in any other clique $C_{l'}$, $l' = 1, \ldots p$, $l' \neq l$.

Proposition 8: Let $|x^{V_{k,s}}| \leq m_k$ be a resource constraint of $\mathcal{Q}$, and $\{C_l \mid l = 1, \ldots, p\}$ be a minimal clique cover of $V_{k,s}$, where each C_l represents a clique made by timing edges. Furthermore, for each $v \in V_{k,s}$, let p_v denote the number of cliques that contain v. When $p \geq m_k$, the following expression is a valid inequality of $\mathcal{Q}$:

$$\sum_{v \in V_{k,s}} c_v x_v \leq m_k \tag{14}$$

where $c_v = \max\{1, m_k + p_v - p\}$, and $p \geq m_k$

Proof: First we partition $V_{k,s}$ according to the value of c_v in the following way:

$$U = \{ v \in V_{k,s} \mid c_v = 1 \} \tag{15}$$
$$W = \{ v \in V_{k,s} \mid c_v > 1 \} \tag{16}$$

Since the cliques C_l, $l = 1, \ldots, p$ cover $V_{k,s}$, each node $v \in V_{k,s}$ belongs to at least one clique; which implies:

$$p_v \geq 1 \tag{17}$$

For any feasible schedule $Q \subset V$, let $V' = U \cap Q$, and $V'' = W \cap Q$. A feasible schedule does not include more than one node from any clique C_l, $l = 1, \ldots, p$. Therefore, in order for the nodes in $V' \cup V''$ to be in a feasible schedule, there must exist at least $\sum_{v \in V' \cup V''} p_v$ cliques. Thus we can write:

$$\sum_{v \in V' \cup V''} p_v \leq p \tag{18}$$

We can use the relations (17) and (18) in the following derivation to complete the proof:

$$\begin{aligned}
\sum_{v \in V_{k,s}} c_v x_v^Q &= |V'| + \sum_{v \in V''} c_v \\
&= |V'| + \sum_{v \in V''} (p_v + m_k - p) \\
&= |V'| + \sum_{v \in V''} p_v + |V''|(m_k - p) \\
&\leq \sum_{v \in V' \cup V''} p_v + |V''|(m_k - p) && \text{using (17)} \\
&\leq p + |V''|(m_k - p) && \text{using (18)} \\
&= m_k|V''| - p(|V''| - 1) \\
&\leq m_k|V''| - m_k(|V''| - 1) && \text{since } p \geq m_k \\
&= m_k
\end{aligned}$$

□

So far we have shown that the inequality (14) is valid; to show that it is strong, a more elaborate analysis is required. This will be the topic of the rest of this section.

If, in the definition of $P_F(Q)$, we change the equality sign of the assignment constraints into $\leq$ then we will get the fractional *monotone scheduling polytope*, as described below:

$$P_F(\tilde{Q}) = \{\, x \in \mathbb{R}_+^{|V|} \mid M_a x = 1 \; ; \; M_p x \leq 1 \; ; \; M_r x \leq m \,\}$$

In the above equation $\tilde{Q}$ can be thought of as the set of all the partial schedules.

Proposition 9: The extended resource constraint, $\sum_{v \in V_{k,s}} c_v x_v \leq m_k$, $c_v = \max\{1, m_k + p_v - p\}$, defines a facet of $P_I(S)$, where $S = \{\, x \in \tilde{Q} \mid x_v = 0 \,,\, \forall\, v \notin V_{k,s} \,\}$.

Proof: Clearly $dim(P_I(S)) = |V_{k,s}|$. Thus the proposition can be proved by exhibiting $|V_{k,s}|$ linearly independent points in $F = \{\, x \in P_I(S) \mid \sum_{v \in V_{k,s}} c_v x_v = m_k \,\}$.

Consider the clique cover $\{C_l \mid l = 1, \ldots, p\}$ of $V_{k,s}$ used to generate the extended resource constraints (14). Since the cover is minimal, every clique C_l has at least one node v_l that does not belong to any other clique, and let $U' = \{\, v_l,\ l = 1, \ldots, p \,\}$. From (15) it can be seen that U' is an independent set of U of size p. Suppose $U'' \subset U$, such that $|U''| = m_k + 1$. The points are found in the following way:

(i) x^{J_u}, $\forall\, u \in W$, where, $J_u = \{u\} \cup Y$, such that $Y \subset U'$, and $|Y| = m_k - c_u$. There are $|W|$ of them.
(ii) x^{J_u}, $\forall\, u \in U \setminus U'$, where, $J_u = \{u\} \cup Y$, such that $Y \subset U'$, and $|Y| = m_k - 1$. There are $|U| - |U'|$ of them.
(iii) x^{J_u}, $\forall\, u \in U''$, where, $J_u = U'' \setminus \{u\}$. There are $|U''|$ of them.
(iv) x^{J_u}, $\forall\, u \in U' \setminus U''$, where $J_u = \{u\} \cup \big(U'' \setminus \{u_1, u_2\}\big)$. There are $|U'| - |U''|$ of them.

From the description of the points above, it can be seen that there is only one point for each $u \in W$, and $u \in U \setminus U'$. Hence, the points described in the first two items are linearly independent among themselves and with the rest of the points. So, if we can prove that the points given in the last two items are linearly independent, then we are done. If we write down those x vectors as the rows of a matrix, the resulting $|U'| \times |V|$ matrix can be written, after proper interchange of columns, as $\begin{bmatrix} I1 & 0 & 0 \\ I2 & I3 & 0 \end{bmatrix}$, where,

I1 is a $|U''| \times |U''|$ matrix of 1's with zeros in the diagonal.
I2 is a $(|U'| - |U''|) \times |U''|$ matrix with 0's in the first two columns and 1's in the rest of them.
I3 is a $(|U'| - |U''|) \times (|U'| - |U''|)$ identity matrix.

I_1, and I_3 are full rank matrices. Therefore, the matrix has full rank, which means that the row vectors are linearly independent. □

Since the extended constraint is a facet of $P_I(S)$, it can be augmented to a facet of $P_I(\tilde{Q})$ by *lifting*[10] on the set $V \setminus V_{k,s}$. The above discussion leads to the conclusion that the inequality (14) is strong.

An important issue not addressed in this section is that of *separation* – for a given fractional point $\hat{x}$, find an inequality of the form (14) violated by $\hat{x}$ or demonstrate that no such inequality exists. In the examples we have tried so far, the inequalities of the form (14) could be quickly examined using an exhaustive search. However, heuristic procedures are commonly used for solving separation problems, and further research is needed to develop a suitable heuristic for separating the inequalities discussed above.

VIII. An Objective Function

The formulation (ILP), described in Section V, can be used with any objective function c. However, the choice of the objective function depends on the design goal and can vary widely. As an example, in this section we describe a particular objective function that helps to reduce the register usage.

A. An Approximate Objective Function

Consider an operation and its need for registers. An operation that is not chained to its successor needs a register to store its output after it finishes execution. If this operation needs only one input variable (in case of a binary operation, the other input may be a constant), then it needs only one register to store its input until it starts execution. Since this operation needs one register both before and after execution, scheduling the operation early or late will not affect the register count. However, if the operation needs two input variables, it would require two registers to hold its input as opposed to one at the output; therefore, it seems proper to schedule the operation as early as possible to minimize register usage. We used this intuition to formulate and minimize the following objective function:

$$cx = \sum_{i \in I} (c_i - 1) \sum_{s \in S_i} s x_{i,s} \tag{19}$$

where c_i is the number of input variables used by operation o_i. Note that this objective function does not optimally minimize the register count, but produces satisfactory results as will be shown in the results section.

B. An Exact Computation of the Number of Live Variables

If the number of live variables across any control step boundary has to be optimally minimized, additional constraints must be added to the original formulation. These constraints are not structured, and therefore make the formulation take longer time to find the result. In the rest of this section we will consider the constraints that represent the maximum number of live variables.

Live variable constraints were first used by Gebotys in the OASIC system [8]. Compared to OASIC, the method presented here leads to fewer constraints and a sparser constraint matrix.

Consider any particular control step s, and a partial order pair $o_i \to o_j$. Whether this partial order will lead to a live variable between control steps s and $s+1$ can be indicated by the following expression, which evaluates to 1 in case of a register, and to 0 otherwise:

$$r_{i,j,s} \triangleq \sum_{s' \leq s, s' \in S_i} x_{i,s'} + \sum_{s' > s, s' \in S_j} x_{j,s'} - 1 \qquad (20)$$

To verify the correctness of the expression, two cases need to be considered:

1. If o_j is not scheduled after control step s (second term is 0), then a register is not needed between s and $s+1$. In this case, due to the timing constraints, o_i must be scheduled at or before control step s (first term is 1), and thus the expression evaluates to 0.
2. If o_j is scheduled after control step s (second term is 1), then a register is required only if o_i is scheduled at or before control step s (first term is 1), in which case the expression evaluates to 1.

If each operation has no more than one output edge in the cdfg graph, then the following constraint indicates that there can be no more than r live variables across any control step:

$$\sum_{\substack{i \in I \\ o_i \to o_j}} r_{i,j,s} \leq r \quad \forall\, s \in S \qquad \text{(L)}$$

In practice, the output of an operation often goes to several other operations, so one lifetime defining edge has to be found. Transitive reduction [25] and ALAP-analysis has been used in OASIC [8] to discard the edges that do not define the lifetime of a variable. When no such reduction is possible, constraints are generated for each possible output edge. Thus the number of constraints at a control step is the product of the number of output edges of the operations. This can give rise to a large number of constraints when many operations in the same control step have multiple output edges. This is not the case for the EWF benchmark, but a problem in case of the DCT example. Therefore, we propose the following method that will reduce the number of constraints.

After all the edges that do not define a variable have been discarded, we add a dummy node for each remaining operation o_i with multiple output edges. The dummy node has to be scheduled after the operations that receive the output of o_i. The edge between o_i and d_i is considered as the lifetime defining edge, and other output edges of o_i are not regarded for live variable constraints. This approach will produce additional assignment and timing constraints which are well structured, and reduce the number of unstructured live variable constraints. Furthermore, this approach will reduce the number of the constraints.

IX. Results

We have proven in Section V-B that our formulation (ILP) produces tight bounds on the objective function, and constitutes a well-structured model for solving a scheduling problem. The RCS and TRCS problems lead directly to the formulation (ILP). The UCS and TCS problems can also be solved using (ILP), along with a good set of lower bounds on the number of FU's as indicated in Section V-A.

In this section, we first present results of solving the TRCS problem using formuation (ILP) on two benchmark examples: the 34-operation elliptical wave filter (EWF) [30], and the 48-operation discrete cosine transform (DCT) [31]. The purpose of these results is to demonstrate that the formulation (ILP) is well-behaved, and to allow us to observe the general behavior (tight bounds, and therefore small number of branches) that was indicated by our theoretical analysis.

It should be noted here that any ILP approach produces optimal results, so we can not expect our schedules to be better than other ILP solutions. Instead, our objective was to offer a theoretical foundation for evaluating the behavior of an ILP formulation. Thus for our purposes, we will use the number of branches taken by the ILP solver as the indicator of performance. We will demonstrate that the number of branches are small, as we predicted in Section V. Of course, these results might vary somewhat if another ILP solver was used instead.

In these examples, we used the standard assumptions: the delay of the ALU is one control step, the delay of the multiplier is two control steps, and the pipelined multiplier has a latency of one control step. The columns labeled **LV** gives the maximum number of live variables across any control step, and can be considered as the number of registers. The columns labeled **Branch** indicate the number of nodes in the branch-and-bound tree for solving a particular problem instance. The ILP formulations were solved using LINDO [32] optimization software on a SPARC 2 workstation.

For the EWF benchmark, we first solved the formulation (ILP) with objective function (19) (no register constraints), which attempts to minimize the number of registers. In all cases the optimal number of registers was produced. This number was verified later by adding the register constraints (L) to the formulation, so that the exact number of live variables across any cstep could be computed and minimized. These results are shown in Table I.

TABLE I
Scheduling Results for the Elliptic Wave Filter (with Register Constraints)

No. of csteps		Non-Pipelined Mult				Pipelined Mult			
Total	Loop	ALU	Mul	LV	Branch	ALU	Mul	LV	Branch
17	17	3	3	10	0	3	2	10	0
18	18	2	2	10	0	3	1	10	0
						2	2	9	2
18	16	3	2	10	0	3	1	10	1
19	19	2	2	9	3	2	1	9	3
19	17	2	2	9	4	2	1	9	3
21	21	2	1	9	0	2	1	9	12
21	19	2	1	9	0	2	1	9	10

TABLE II
Size and Solution time of the Formulations

Csteps	Var	Eqn	CPU-time (s)	Pivots
17	85	99	0.68	79
18	123	144	1.35	150
19	161	193	3.08	263
21	237	291	8.9	551

TABLE III
Scheduling Results for the Discrete Cosine Transform Example (without Register Constraints)

No. of csteps	Pipelined Mult				Non-Pipelined Mult			
	ALU	Mul	LV	Branch	ALU	Mul	LV	Branch
7	6	5	12	1	6	8	11	1
8	5	4	12	1	5	6	13	4
9	4	3	13	2	4	6	13	1
9	4	4	13	1	5	6	13	0
9	5	4	12	1	5	7		0

TABLE IV
Scheduling results for the Discrete Cosine Transform Example (with Register Constraints)

No. of csteps	Pipelined Mult				Non-Pipelined Mult			
	ALU	Mul	LV	Branch	ALU	Mul	LV	Branch
7	6	5	12	1	6	8	11	2
8	5	4	12	9	5	6	11	19
9	4	3	12	12	4	6	13	12
9	4	4	11	18	5	6	12	15
9	5	4	11	27	5	7	12	30

Table II summarizes the size and solution time of the formulation (ILP) for the EWF benchmark. Column **Eqn** indicates the total number of constraints (including register constraints). Column **Pivots** indicates the number of simplex pivots performed to produce the final solution, and is provided for informational purposes only; direct comparison with other ILP-based schedulers (such as OASIC and GRAD) is not possible at this point because the machines, ILP-solvers and details of the ILP formulations vary from system to system.

The scheduling results of the DCT benchmark are presented in Tables III and IV. Table III shows the solution of (ILP) with objective function (19) (no register constraints). As predicted, the solution was produced in a small number of branches; however, the number of registers occasionally is not optimal. When register constraints were added, the schedules with optimal number of registers were produced, as shown in Table IV. However, as can be seen, much more branching was required since the formulation became more unstructured due to the addition of the register constraints.

Although our analysis suggests that the TCS and UCS problems should be solved using formulation (ILP), these problems can be solved directly if the number of FU's are treated as variables as discussed in Section V. We solved the TCS problems in this manner, and let the ILP solver choose the branch-and-bound-tree.

In keeping with our observation, these TCS formulations took a larger number of branches (and a longer time) to find the optimal solution. The deterioration in performance was not noticable for the EWF benchmark: to generate schedules of 17, 18, 19 and 21 csteps using non-pipelined multipliers, the TCS formulation took 0, 0, 4 and 5 branches, respectively. However, for the DCT benchmark, in the case of schedules of 7 and 8 control steps using non-pipelined multipliers, the TCS formulation took 8 and 780 branches, respectively. Fortunately, the same solutions could be computed much faster by first using an efficient lower-bounding scheme [26] (which exploits the structure of the resource constraints), and then solving formulation (ILP); this method took only 1 and 9 branches, respectively.

These results indicate that although any ILP formulation theoretically leads to optimal results, a careful choice based on theoretical study should be made to avoid explosion in computation time.

X. Summary and Conclusion

In this paper, we have presented a mathematical analysis of the ILP constraints of the scheduling problem, and have shown how to exploit the structure of the constraints in a well-designed formulation, so that efficient results can be expected. We have shown that even if all types of scheduling problems can be described as ILP's using a reasonable number of constraints and integer variables, some of them have better structure than others. These theoretical observations have been verified on benchmark examples.

One challenge to future improvement of ILP-based scheduling algorithms lies in finding new strong valid inequalities that can be used to tighten the formulation. We have shown that the resource-assignment polytope defined by the resource and assignment constraints, and the polytope defined by the timing and assignment constraints, are integral. This clearly shows that in order to find tighter constraints, the interaction between the resource and the timing constraints has to be considered. We have presented a class of such constraints in Section VII-B and have demonstrated using polyhedral theory that such inequalities are tight. An extension of this method still remains as a promising area of further investigation.

We have also presented a method to modify the conventional ASAP and ALAP algorithms in order to determine the schedule interval of each operation. Such preprocessing schemes are very important for improving the solving efficiency of the ILP formulations [22]. These modifications can also help in improving the solution quality of other heuristic methods such as force-directed-list-scheduling [7].

Acknowledgement

The authors would like to thank the anonymous reviewers for their many helpful comments, particularly those that helped us to put both our work and previous work by others, better into context.

Appendix

I. Proof of Claim 2

Claim 2: The skeleton digraph G_s is acyclic

Proof: Any arc a, denoted as $b \to c$, in a digraph represents a distance $\delta(a) = c - b$ between nodes b and c. We can calculate the distances produced by the arcs in the cdfg and G_s as follows, $\delta(a_{ij}) = j - i$, $\delta(a_{ijs}) = (j - i, d_{ij} - 1) = (\delta(a_{ij}), d_{ij} - 1)$, and $\delta(a_{iis}) = (0, -1)$.

If we denote the distance produced by a set L of arcs as $\Delta(L) = \sum_{a \in L} \delta(a)$, then for G_s we can write:

$$\Delta(L) = \sum_{a_{ijs} \in L} (\delta(a_{ij}), d_{ij} - 1) + \sum_{a_{iis} \in L} (0, -1)$$

The arcs in L can form a cycle only if $\Delta(L) = 0$, for which the above equation requires: $\sum_{a_{ijs} \in L} \delta(a_{ij}) = 0$. This implies that the set of arcs $\{a_{ij} | a_{ijs} \in L\}$ form a cycle in the cdfg. Since the cdfg does not contain any positive length cycle, $\sum_{a_{ijs} \in L} d_{ij} < 0$. This condition can be used in the expression for $\Delta(L)$ to conclude that $\Delta(L) < (0, 0)$. Therefore, no set of arcs L can form a cycle in G_s. □

II. Proof of Theorem 2

Theorem 2: Any maximal clique constraint of G_c is a linear combination of constraints (A) and (T).

Proof: The maximal clique constraint for clique C is given in (5), where L_C is a maximal path in G_s such that $V(L_C) = V(C)$. It can be easily seen, that any maximal path in G_s can be uniquely described by listing all the timing arcs in the order they appear in L_C. We describe, without loss of generality, such a list as: $a_{i_l i_{l+1} s_l}$, $l = 1, \ldots k$.

To visualize L_C, let:

$$\begin{aligned} s'_1 &= max\{s \mid s \in S_{i_1}\} \\ s'_{l+1} &= s_l + d_{i_l i_{l+1}} - 1 \qquad \text{for } l = 1, \ldots, k \\ s_{k+1} &= min\{s \mid s \in S_{i_{k+1}}\} \end{aligned}$$

L_C begins at node (i_1, s'_1) and ends at node (i_{k+1}, s_{k+1}). For each l, $l = 1, \ldots k+1$, the path goes from (i_l, s'_l) to (i_l, s_l) using assignment arcs $a_{i_l i_l s}$, $s_l < s \leq s'_l$. For each l, $l = 1, \ldots k$, the path goes from (i_l, s_l) to (i_{l+1}, s'_{l+1}), using timing arc $a_{i_l i_{l+1} s_l}$. Therefore, we can write the clique constraint for L_C as:

$$|x^{V(L_C)}| = \sum_{l=1}^{k+1} \sum_{s=s_l}^{s'_l} x_{i_l,s} \tag{21}$$

For compactness of derivation, we introduce the following notations.

$$\begin{aligned} X^{<}_{i_l} &= \sum_{s \leq s'_l, s \in S_i} x_{i_l,s} \\ X^{>}_{i_l} &= \sum_{s \geq s_l, s \in S_i} x_{i_l,s} \end{aligned}$$

The original expressions for $|x^{V(L_{i_l})}|$ and $|x^{V(L_{i_l i_{l+1} s_l})}|$ (given in (A) and (T)) can be represented using the new notations as:

$$\begin{aligned} |x^{V(L_{i_l})}| &= X^{>}_{i_l} + X^{<}_{i_l} - \sum_{s=s_l}^{s'_l} x_{i_l,s} \\ |x^{V(L_{i_l i_{l+1} s_l})}| &= X^{>}_{i_l} + X^{<}_{i_{l+1}} \end{aligned}$$

In the following derivation, we use the above relations to show $|x^{V(L_C)}| \leq 1$.

$$\begin{aligned} 1 &\geq \sum_{l=1}^{k} |x^{V(L_{i_l i_{l+1} s_l})}| - \sum_{l=2}^{k} |x^{V(L_{i_l})}| \quad \text{using (A), (T)} \\ &= \sum_{l=1}^{k} \left(X^{>}_{i_l} + X^{<}_{i_{l+1}}\right) - \sum_{l=2}^{k} |x^{V(L_{i_l})}| \\ &= X^{>}_{i_1} + X^{<}_{i_{k+1}} + \sum_{l=2}^{k} \left(X^{>}_{i_l} + X^{<}_{i_l} - |x^{V(L_{i_l})}|\right) \\ &= \sum_{s=s_1}^{s'_1} x_{i_1,s} + \sum_{s=s_{k+1}}^{s'_{k+1}} x_{i_{k+1},s} + \sum_{l=2}^{k} \sum_{s=s_l}^{s'_l} x_{i_l,s} \\ &= \sum_{l=1}^{k+1} \sum_{s=s_l}^{s'_l} x_{i_l,s} \\ &= |x^{V(L_C)}| \qquad \text{using (21)} \end{aligned}$$

□

References

[1] G. L. Nemhauser and L.A. Wolsey, *Optimization*, vol. 1 of *Handbooks in Operations Research and Management Science*, chapter 6, Elsevier Science Publishers B. V., 1989.

[2] M. C. McFarland, A. C. Parker, and R. Camposano, "The High Level Synthesis of Digital Systems", *Proceedings of the IEEE*, vol. 78, no. 2, pp. 301–318, February 1990.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[4] J. D. Ullman, "NP-Complete Scheduling Problems", *J. Comput. System Sci*, vol. 10, no. 10, pp. 384–393, 1975.

[5] R. K. Brayton, R. Camposano, G. De Micheli, and R.H.J.M Otten, "The Yorktown Silicon Compiler System", in *Silicon Compilation*, pp. 204–310. Addison-Wesley, 1988.

[6] D. E. Thomas et al., *Algorithmic and Register Transfer Level Synthesis: The System Architect's Workbench*, chapter Control Step Scheduling, pp. 107–132, Kluwer Academic Press, 1990.

[7] Pierre G. Paulin and John P. Knight, "Force Directed Scheduling for the Behavioral Synthesis of ASICs", *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 6, pp. 661–679, June 1989.

[8] Catherine H. Gebotys and Mohamed I. Elmasry, *Optimal VLSI Architectural Synthesis*, Kluwer Academic Publishers, 1992.

[9] Cheng-Tsung Hwang, Jiahn-Hurng Lee, and Yu-Chin Hsu, "A Formal Approach to the Scheduling Problem in High Level Synthesis", *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 4, pp. 464–475, April 1991.

[10] George L. Nemhauser and Laurence A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, 1988.

[11] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem*, chapter Polyhedral Theory, pp. 251–305, John Wiley & Sons, 1985.

[12] H. Crowder, E. L. Johnson, and M. Padberg, "Solving Large Scale Zero-One Linear Programming Problems", *Operations Research*, vol. 31, no. 5, pp. 803–834, 1983.

[13] M. Grötschel, M. Jünger, and G. Reinelt, "A Cutting Plane Algorithm for Minimum Perfect 2-matchings", *Computing*, vol. 39, pp. 327–344, 1987.

[14] Louis J. Hafer and Alice C. Parker, "A Formal Method for The Specification, Analysis, and Design of Register-Transfer Level Design Logic", *IEEE Transactions on Computer-Aided Design*, vol. 2, no. 1, pp. 4–17, January 1983.

[15] Hyunchul Shin and Nam S. Woo, "A Cost Function Based Optimization Technique for Scheduling in Data Path Synthesis", in *Proc. of the IEEE International Conference on Computer Design*. IEEE, 1989, pp. 424–427.

[16] Catherine H. Gebotys, "Optimal synthesis of multichip architectures", in *Proc. of the International Conference on Computer-Aided Design*, Santa Clara,CA, 1992, pp. 238–241, IEEE Computer Society Press.

[17] Catherine H. Gebotys, "Optimal Scheduling and Allocation of Embedded VLSI Chips", in *Proc. of 29th ACM/IEEE Design Automation Conf.*, Anaheim, CA, 1992, pp. 116–119, IEEE Computer Society Press.

[18] T. C. Wilson, N. Mukherjee, M. K. Garg, and D. K. Banerjee, "An Integrated and Accelerated ILP Solution for Scheduling, Module Allocation, and Binding in Datapath Synthesis", in *6th International Conference on VLSI Design*, Bombay, India, Jan 1993, pp. 192–197.

[19] J. Blazewicz, K. Ecker, G. Schmidt, and J. Węglarz, *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, 1993.

[20] M. Queyranne, "Structure Of A Simple Scheduling Polyhedron", *Mathematical Programming*, vol. 58, pp. 263 – 285, 1993.

[21] J. P. Sousa and L. Wolsey, "A Time Indexed Formulation Of Non-Preemptive Single Machine Scheduling Problems", *Mathematical Programming*, vol. 54, pp. 353–367, 1992.

[22] George Nemhauser, "The Age of Optimization-Solving Large Scale Real World Problems", *Philip McCord Morse Lecture, TIMS/ORSA Joint National Meeting*, May 1993.

[23] John W. Moon, *Topics on Tournaments*, Athena Series. Holt, Rinehart and Winston, 1968.

[24] Gary Chartrand and Linda Lesniak, *Graphs and Digraphs*, The Wadsworth & Brooks/Cole Mathematics Series, 2 edition, 1986.

[25] Martin Charles Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.

[26] Samit Chaudhuri and Robert A. Walker, "Computing Lower Bounds on Functional Units before Scheduling", in *Proc. of the 7th International Symposium on High Level Synthesis*. May 1994, pp. 36–41, IEEE Computer Society Press.

[27] M. Held and R. M. Karp, "The Travelling Salesman Problem and Minimal Spanning Trees: Part II", *Mathematical Programming*, vol. 1, pp. 6–25, 1971.

[28] M. Fisher, "Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part I", *Operations Research*, vol. 21, pp. 1114–1127, 1973.

[29] M. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science*, vol. 27, no. 1, pp. 1–18, 1981.

[30] D. E. Thomas, E. D. Lagnese, R. A. Walker, J. A. Nestor, J. V. Rajan, and R. L. Blackburn, *Algorithmic and Register Transfer Level Synthesis: The System Architect's Workbench*, Kluwer Academic Press, 1990.

[31] K. Rao and P. Yap, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press, 1990.

[32] Linus E. Schrage, *Linear, Integer, and Quadratic Programming with LINDO : User's Manual*, Scientific Press, 1985.

Samit Chaudhuri (S'92) received the B. E. degree in Electronics and Telecommunications Engineering from University of Calcutta, India in 1987, and the M. Tech degree in Electrical Engineering from Indian Institute of Technology, Kanpur, India in 1989.

Since 1989 he has been working on the PhD degree at Rensselaer Polytechnic Institute, Troy, NY. His research interests include design automation, high level synthesis, and combinatorial optimization.

Robert A. Walker (S'79? - M'88? - SM'94) received the B.S. degree in Electrical Engineering (Magna Cum Laude) from the Tennessee Technological University in 1981, and the M.S. degree in Electrical Engineering (Computer Engineering) and the Ph.D. degree in Electrical and Computer Engineering from Carnegie-Mellon University, in 1982 and 1988, respectively.

From 1988 to 1989, he was a Postdoctoral Fellow at Carnegie-Mellon University. He joined the Department of Computer Science at Rensselaer Polytechnic Institute in 1989, where he is currently an Assistant Professor; he also holds a joint appointment in the Department of Electrical, Computer, and Systems Engineering. He is the co-author of the book Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench (Kluwer Academic Publishers, 1990) and the book A Survey of High-Level Synthesis Systems (Kluwer Academic Publishers, 1991). His research interests span the field of high-level synthesis, in particular those problems connected with scheduling.

Dr. Walker has served on the Advisory Board of the ACM Special Interest Group on Design Automation (ACM SIGDA) since 1992, has been actively involved with their DAC University Booth program, and is currently serving as SIGDA Secretary / Treasurer. He has served on several conference and workshop Program Committees, and is currently the Chair of the CODES Sponsors' Steering Committee and a member of the ICCAD'94 Executive Committee. He has a strong interest in teaching, and he received the Lilly Endowment Teaching Fellowship in 1990 and the Rensselaer Distinguished Teaching Fellowship in 1992. He is a member of the ACM, ACM SIGDA, ACM SIGARCH, IEEE, IEEE Computer Society, and Sigma Xi.

John E. Mitchell received the B.A. (Hons) degree in Mathematics from Cambridge University, England in 1983, and the M.S. and Ph.D. degrees in Operations Research and Industrial Engineering from Cornell University in 1986 and 1988, respectively.

He joined the department of Mathematical Sciences at Rensselaer Polytechnic Institute in 1988, where he is currently an Associate Professor. His research interests lie in optimization. In particular, he is interested in interior point methods for linear programming and in linear programming approaches for solving integer programming problems, such as cutting plane and branch and bound algorithms.

Dr. Mitchell is a member of SIAM, ORSA, and the Mathematical Programming Society.