A Flow-Based Approach to Solving Resource-Constrained Scheduling Problems

Jürgen Teich
Lothar Thiele
Computer Engineering and Communication Networks Lab (TIK)
Swiss Federal Institute of Technology (ETH)
Gloriastrasse 35, 8092 Zurich
Switzerland

TIK-Report
Nr. 17, April 1996

Contents

Abstract

A novel *integer linear programming* (ILP) formulation for resource-constrained scheduling problems is presented. Existing ILP-based methods for the same problem typically have a complexity (measured in terms of the number of unknown integer variables) that is proportional to the latency bound of the problem. Hence, they are restricted in scope (e.g., high-level synthesis). Our approach consists in (1) overlaying a *flow graph* to a given *problem graph* $G(V, E)$ (i.e., task, precedence graph) which allows to model the impact of resource and scheduling constraints in a single graph model. (2) Based on this model, we formulate a number of resource-constrained scheduling problems as problems of optimal resource flow. The complexity of the model can be measured in terms of $\mathcal{O}(|V|^2)$ binary variables that model the flow of resources from node to node, and, in case there is a non-zero flow, activate serialization constraints of allowed schedules. (3) We analyze the polyhedral structure of this ILP model and prove that only a subset of flow variables have to be declared integer. (4) Also, we explain how to eliminate flow variables. Procedures for optimizing the polyhedral structure are outlined including the addition of facet defining inequalities and the optimization of constant parameters of the model. (5) Finally, we present strategies for efficiently solving the ILP-model.

Chapter 1

Introduction

Resource- constrained scheduling problems exist in many application areas, e.g., in the planning of industrial processes (*operations research*), in software synthesis (code scheduling), and on several levels of hardware synthesis.

In many cases, a problem graph is given in which the nodes represent operations that have to be scheduled, and edges represent data dependencies. Furthermore, a set of resources is given and a relation that describes for each operation the set of resources which may execute this operation. Finally, costs of the operations (e.g., execution times) are known.

In the following, we consider optimization problems such as the minimization of the latency of a schedule under resource constraints or the minimization of the costs of a schedule given a latency bound. It is known (see, e.g., [GJ76]) that all of these problems except a few special cases are $\mathcal{NP}$- hard.

Nevertheless, *integer linear program* (ILP) based models have become attractive for a number of reasons given next.

1.1 ILP techniques for solving scheduling problems

The following results are known from the domain of high-level synthesis that use ILP formulations: [HLH91, GE92, GE93, BST93, CWM94]. All approaches assume that a data flow graph or control data flow graph (CDFG) is given. We are going to use the term *problem graph* in the following. Hwang et al. [HLH91] describe the start times of operations of a problem graph as a weighted sum of binary variables. Using ASAP-, ALAP- and list scheduling techniques, the number of variables and hence, the search space is reduced so that typical problem instances encountered in high-level synthesis can be optimally solved. The approach described in [HLH91] has been extended to model memory and buses in [BST93].

In order to improve the structure of the solution polytopes, Gebotys et al. [GE92, GE93] have analyzed the structure of the constraints of the ILP models. These ideas have been developed further and extended by Chaudhuri et al. in [CWM94], an excellent work in which the exact structure of the constraints is analyzed using polyhedral theory. They showed that subproblems of the scheduling problems with resource constraints can be exactly solved by solving *LP-relaxations*, because the polyhedra resulting in that cases can be shown to possess integral vertices (so called *integral polyhedra*). A LP-relaxation of an ILP denotes the linear program with identical cost function and same set of constraints, however, with all integrality constraints omitted. A linear program (LP) can be solved in polynomial time. Chaudhuri et al. showed further that the complete polytope which can be described by the intersection of integral polytopes, however, is not integral. In order to improve the solution polytope, they presented a method that adds a number of additional constraints.

The advent of ILP techniques for solving computationally hard problems can be recognized also in other problem areas like for example the traveling salesman problem [JRT92]. Due to Reinelt [JRT92], the main interests in these techniques lies in the improvement of the structures of an ILP formulation, or, in other words, in finding a „good formulation“ and in the determination of provably good solutions within a given time bound. Hence, ILP formulations can be attractive for a large number of reasons.

1.2 Overview

Here, we will 1) present a different ILP model for resource constrained scheduling problems in which resource constraints are modeled in a so called *flow graph* that is an extended problem graph. We will show that this model has a complexity (number of integral variables) that is independent of the latency and computation times of nodes. Hence, this formulation is not restricted to model scheduling problems only in the domain of high-level synthesis.

2) Similar to the work of Gebotys [GE92, GE93] and Chaudhuri [CWM94], we will analyze the polyhedral structure of our new model. In particular, we will show that many variables need not be declared integer. The complexity of our model is, however, $\mathcal{O}(|V|^2)$ given a problem graph with $|V|$ nodes.

Therefore, we 3) investigate methods to improve the structure of the formulation. These include a) the elimination of redundant variables, b) the determination of optimal constants, and c) the addition of facet defining inequalities.

Finally, we 4) present methods to accelerate the solution of an optimal solution. These include a) the determination of good initial solutions, and b) iterative tightening of the polyhedral constraints during the optimization. The underlying solution strategy is summarized in Fig. 1.1.

Here is a short summary of the contents of the following sections. First, a formal definition of the optimization problems are given (Section 2.1). Then, existing ILP-based approaches are introduced and compared (Section 2.2) followed by the introduction of our new model that we will call *flow model* in the following. In Section 3, the structure of the solution polytope of the flow model is analyzed. Methods to improve the model structure are outlined in Section 3.3. After that we explain, how iterative scheduling problems (loops) can be handled by the flow model. Also, it is

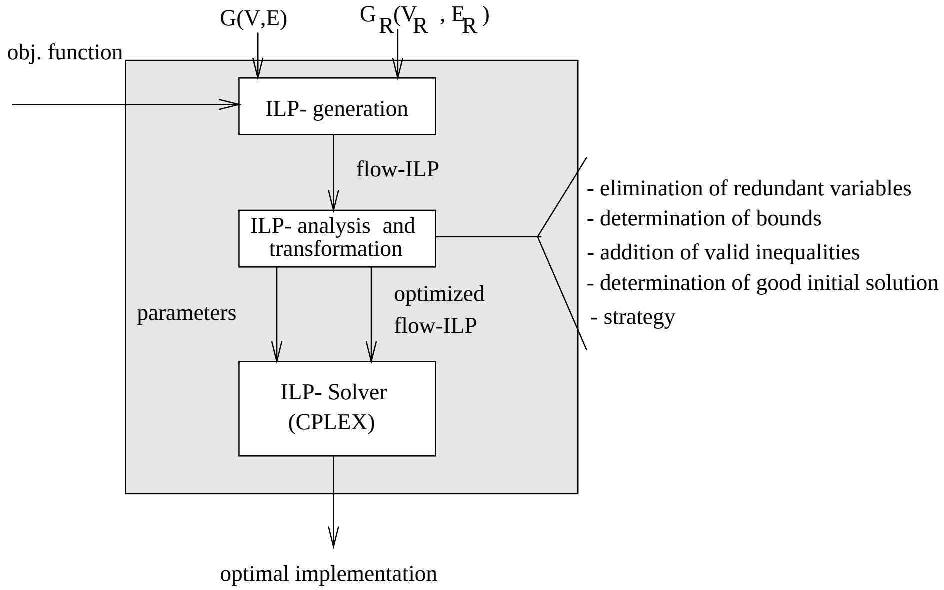

Figure 1.1: Solution strategy for solving resource- constrained scheduling problems

shown that modules with pipelining as well as functional pipelining and loop folding can be modeled (Section 4). Finally, we analyze solution strategies (Section 5).

The flow model will be compared to existing approaches using several examples (Section 6). We close with a summary and give an overview of future work.

Chapter 2

Flow-based Approach to Solving Scheduling Problems

2.1 Notation and problem formulation

The operations to be scheduled and the data dependencies between operations are modeled by a problem graph.

Definition 2.1 (Problem graph) *A* problem graph $G(V, E)$ *is a directed, acyclic graph with the set of nodes V and the set of edges E. Each node $v_i \in V$ represents an object that is to be scheduled (e.g., task, statement, elementary operation) and each edge $e = (v_i, v_j) \in E$ represents a data dependency.*

Example 2.1 *We consider the algorithm*

$$\begin{aligned} x_1 &= f_1(); \\ x_2 &= f_2(x_1); \\ x_3 &= f_3(x_1); \end{aligned}$$

The problem graph of this algorithm is shown in Fig. 2.1a). There exist data dependencies between the nodes v_1 and v_2 and between the nodes v_1 and v_3.

The relationship between the tasks and the resources (e.g., processors, arithmetic logical units, gates) is described by a *resource graph.*

Definition 2.2 (Resource graph) *The* resource graph $G_R(V_R, E_R)$ *is a bipartite graph. The node set $V_R = V \cup V_T$ contains the set V of nodes of the problem graph $G(V, E)$. Each node $r_k \in V_T$ represents a resource type (e.g., processor, ALU, adder, multiplier, etc.). An arc $(v_i, r_k) \in E_R$ with $v_i \in V$ and $r_k \in V_T$ models the realizability of v_i on an instance of resource type r_k. Furthermore, there is a*

- *cost function $c : V_T \to \mathbf{Z}_0^+$ that assigns the costs $c(r_k)$ to each resource type $r_k \in V_T$ and a*

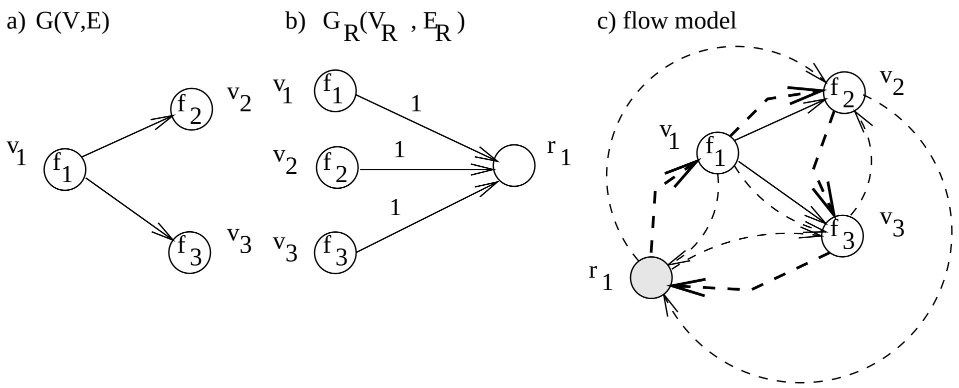

Figure 2.1: Problem graph a), resource graph b) and flow graph c)

- *weighting function* $w : E_R \rightarrow \mathbf{Z}_0^+$ *that assigns the execution time* $w((v_i, r_k))$ *(or simply* $w_{i,k}$*) to each edge* $(v_i, r_k) \in E_R$*.* $w_{i,k}$ *denotes the execution time of* v_i *on* r_k*.*

Example 2.2 *The resource graph shown in Fig. 2.1b) contains only one resource type* r_1 *(e.g., processor) which is able to execute each of the three nodes of the problem graph in Fig. 2.1a). The execution time of each node on resource type* r_1 *is* $w_{1,1} = w_{2,1} = w_{3,1} = 1$*.*

A pair comprising a problem graph $G(V, E)$ and a resource graph $G_R(V_R, E_R)$ is called *specification* in the following. Now, our goal is to a) determine a *schedule*, b) an *allocation* of resources, and c) a *binding* of nodes to resources such that a certain objective function is optimized. For this purpose, we define:

Definition 2.3 (Schedule) *A* schedule *of a problem graph* $G(V, E)$ *is a function* $\tau : V \rightarrow \mathbf{Z}_0^+$ *that assigns to each node* $v_i \in V$ *the start time* $t_i = \tau(v_i)$ *such that*

$$\tau(v_i) \geq \tau(v_j) + w((v_j, \beta(v_j))) \quad \forall (v_j, v_i) \in E$$

$\beta(v_j) \in V_T$ *specifies the resource type on which* v_j *is computed.*

Definition 2.4 (Binding) *Given a specification including a problem graph* $G(V, E)$ *and the resource graph* $G_R(V_R, E_R)$*. The* binding *is a tuple of functions*

- $\beta : V \rightarrow V_T$ *with* $\beta(v_i) = r_k \in V_T$ *and* $(v_i, \beta(v_i)) \in E_R$*;*
- $\gamma : V \rightarrow \mathbf{Z}^+$ *with* $\gamma(v_i) \leq \alpha(\beta(v_i))$*.*

Here, $r_k = \beta(v_i)$ *denotes the resource type and* $\gamma(v_i)$ *the instance of resource type* r_k *on which* v_i *is executed.* $\gamma(v_i)$ *must be smaller or equal to the number* $\alpha(r_k)$ *of allocated resources of type* r_k *(if there is such a limit).*

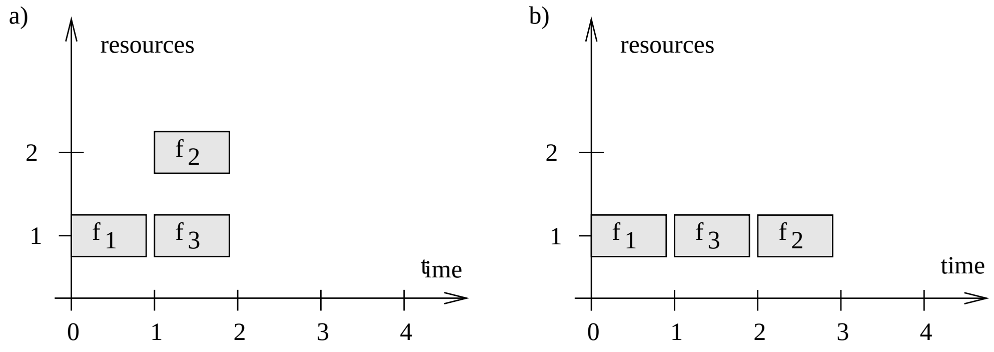

Figure 2.2: Schedules for the specification in Fig. 2.1 using a) two resources and b) one resource

Example 2.3 *In Fig. 2.2, two schedules are displayed for the specification with the problem graph and resource graph shown in Fig. 2.1a),b), respectively. The schedules correspond to an allocation of two resources (case a)) and one resource (case b)). The start times of operations and the binding is given as follows:*

a) $t_1 = 0, t_2 = t_3 = 1,\ \beta(v_1) = \beta(v_2) = \beta(v_3) = r_1,\ \gamma(v_1) = \gamma(v_2) = 1,\ \gamma(v_3) = 2,$

b) $t_1 = 0, t_2 = 1, t_3 = 2,\ \beta(v_1) = \beta(v_2) = \beta(v_3) = r_1,\ \gamma(v_1) = \gamma(v_2) = \gamma(v_3) = 1.$

Often, the number of available resources is a given constraint called *allocation*:

Definition 2.5 (Allocation) *Given a specification* $G(V, E)$, $G_R(V_R, E_R)$. *An* allocation *is a function* $\alpha : V_T \to \mathbf{Z}_0^+$ *that assigns the number* $\alpha(r_k)$ *of available resources to each resource type* $r_k \in V_T$.

The allocation may be considered as a user-defined constraint. Another important user-defined constraint is the *latency bound*.

Definition 2.6 (Latency) *The* latency L *of a schedule* τ *(given a binding* β*) is given as*

$$L = \max_{v_i \in V}\{\tau(v_i) + w((v_i, \beta(v_i)))\} - \min_{v_i \in V}\{\tau(v_i)\}$$

and denotes the number of time steps of the smallest time interval that contains the execution intervals of all nodes $v_i \in V$.

Example 2.4 *For the schedule shown in Fig. 2.2a), the latency is two and three in case b).*

Now, an upper bound on the maximal latency of allowed schedules is called *latency bound* and will be denoted $\bar{L}$ in the following. We will use the notation $l_i = \tau(v_i)^S$ and $h_i = \tau(v_i)^L$ to denote the earliest (ASAP-) and latest (ALAP-) starting times of nodes $v_i \in V$. Furthermore, let $N(v_i)$ (or simply N_i) with $N_i = [l_i, \cdots, h_i]$ denote the intervals of possible starting times of node v_i.

Using these definitions, we are able to define the term *feasible implementation* (with respect to resource and latency constraints).

Definition 2.7 (Implementation) *Given a specification with problem graph $G(V, E)$ and resource graph $G_R(V_R, E_R)$ (and possibly a latency bound $\bar{L}$ and/or an allocation α). A* (feasible) implementation *is a quadruple $(\tau, \beta, \gamma, \alpha)$ that satisfies the following conditions:*

- *$L \leq \bar{L}$ in case of a given latency bound*
- *$|\{v_i : \beta(v_i) = r_k \ \wedge \ \tau(v_i) \leq t \leq \tau(v_i) + w(v_i, r_k)\}| \leq \alpha(r_k) \quad \forall r_k \in V_T, \forall t = 0, \cdots, L-1$ in case of given resource constraints.*

Example 2.5 *Figs. 2.2a) and b) show two feasible implementations with $\alpha(r_1) = 2$. However, the schedule and binding shown in Fig. 2.2a) is infeasible for an allocation of $\alpha(r_1) = 1$.*

Now, we are interested in solving certain optimization problems when deriving an implementation. The most interesting optimization problems in this context can be characterized as follows (we simply assume that the *costs* of an implementation are given as a linear function of the allocation α):

- *latency minimization under resource constraints*:
  Find an implementation with schedule τ with minimal latency L given an allocation α. [1].
- *cost minimization under latency constraints*:
  In this case, the goal is to find an implementation that satisfies a latency bound $\bar{L}$ and produces minimal costs.
- *admissible scheduling*:
  Find a feasible implementation given a latency bound $\bar{L}$ and an allocation α.
- *weighted minimization of latency and cost under latency and resource constraints*:
  Find an implementation that, given $\bar{L}$ and α, minimizes a cost function that is a linear function of cost and latency.

2.2 ILP-models for scheduling problems

Now, we would like to introduce ILP-models for the above named optimization problems. First, we would like to introduce informally our new idea that considers resource constraints as a flow problem on a graph. After that, we will discuss existing

[1] Without loss of generality, the binding γ does not need to be considered explicitly, because one could define each individual resource instance $r \leq \alpha(r_k)$ as an individual resource type with allocation 1. Hence, the binding β can represent both resource type binding and binding to a resource instance. This model would only increase the complexity of the resource graph. Furthermore, we would like to remark that for non-iterative scheduling problems, a valid γ that satisfies the allocation α can always be found in polynomial time by coloring an interval graph (given τ and β)

approaches using ILP formulations and analyze their complexity. After that, we will formally introduce the *flow model* and compare its complexity with those of existing models.

2.2.1 Idea of the Flow Model

In order to explain the difference to existing models, we introduce the flow model using an example.

Consider the specification shown in Fig. 2.1a), b). Fig. 2.1c) explains the basic idea of our approach. Shown is a so-called *flow graph* containing the problem graph as a subgraph. Furthermore, the flow graph contains another special node called *resource node* (depicted gray) that serves as a source and drain of resources.

Now, one can imagine that the resource node is able to emit resources to all nodes using the dashed *flow edges* for transport. Furthermore, any node can transmit resources to other nodes.

Example 2.6 *For the flow graph shown in Fig. 2.1c), the following ILP model for cost minimization under latency constraints ($\bar{L} = 3$) (here the costs are simply modeled as the number of allocated resources) can be derived:*

$$\begin{array}{rl}
\min & x_{r_1,v_1} + x_{r_1,v_2} + x_{r_1,v_3} \\
\text{subj. to} & \\
(F1.1) & x_{r_1,v_1} - x_{v_1,v_2} - x_{v_1,v_3} - x_{v_1,r_1} = 0 \\
(F1.2) & x_{v_1,v_2} + x_{v_3,v_2} + x_{r_1,v_2} - x_{v_2,v_3} - x_{v_2,r_1} = 0 \\
(F1.3) & x_{v_1,v_3} + x_{v_2,v_3} + x_{r_1,v_3} - x_{v_3,v_2} - x_{v_3,r_1} = 0 \\
(F2.1) & x_{r_1,v_1} = 1 \\
(F2.2) & x_{v_1,v_2} + x_{v_3,v_2} + x_{r_1,v_2} = 1 \\
(F2.3) & x_{v_1,v_3} + x_{v_2,v_3} + x_{r_1,v_3} = 1 \\
(D.1) & t_2 - t_1 \geq 1 \\
(D.2) & t_3 - t_1 \geq 1 \\
 & t_1, t_2, t_3 \leq \bar{L} - 1 \\
(S.1) & t_2 - t_3 \geq 1 + (x_{v_3,v_2} - 1)M \\
(S.2) & t_3 - t_2 \geq 1 + (x_{v_2,v_3} - 1)M \\
 & x_{v_2,v_3}, x_{v_3,v_2} \in \{0, 1\}
\end{array}$$

The model and the different inequalities can be explained by answering a number of questions:

1. How to model an allocation α?
   One can specify a constraint that the sum of resources that are emitted on outgoing edges of the resource nodes must be smaller or equal to the allocation. For this purpose, we define a number of so-called *flow variables* $x_{v_i,v_j,k} \in \{0, 1\}$ that may be considered as edge weights of edges $e = (v_i, v_j)$ of the flow graph.

Using these variables, the sum of flow variables of outgoing edges of each resource node r_k must be smaller or equal to $\alpha(r_k)$. First, we assume that the flow graph consists of $|V_T|$ subgraphs where the $k-$th subgraph is the complete graph containing all nodes $v_i \in V$: $(v_i, r_k) \in E_R$ plus the resource node r_k itself. (Remark: In Example 2.6 there exists only one resource type. Hence, the third index k of variables x has been omitted).

2. How to express a binding β?
Assume there is exists not just one resource type. Hence, the flow graph consists of $|V_T|$ subgraphs. In order to guarantee that each node $v_j \in V$ will be bound to exactly one resource type, one formulates a condition that the sum of flow variables of all ingoing edges is equal to one (this means each node must receive exactly one resource on which it is scheduled) (see inequalities $(F2.*)$ in Example 2.6). In this way, the k for which $x_{v_i,v_j,k}$ is one represents the binding $\beta(v_j) = r_k$. (In Example 2.6, we have $\beta(v_i) = r_1, i = 1, \cdots, 3$. The allocation is evaluated from the sum of flow variables of outgoing edges of the resource nodes).

3. How is guaranteed that the number of resources is constant?
For the reason nodes cannot consume resources, we need a constraint that guarantees that each resource that a node receives on ingoing edges must leave the node again on a outgoing edge. For this purpose, a constraint is formulated for each node that the sum of flow variables of ingoing edges equals to the sum of flow variables on the outgoing edges (see equalities $(F1.*)$ in Example 2.6).

4. How do I see which nodes are bound to the same resource and how do I guarantee that the execution of nodes bound to a resource do not overlap (schedule τ)?
Two nodes $v_i, v_j \in V$ are bound to the same instance of a resource of type r_k if and only if there exists a 1-path in the $k-$th subgraph of the flow graph (a path whose flow variables are all one) from v_i to v_j or the other way round. The orientation of this path describes the scheduling order on this resource. A sequentialization constraint is added for each flow edge that becomes active in case there is a flow on the edge and is inactive elsewise (see inequalities (S.1) and (S.2) in Example 2.6).

Example 2.7 *The only integer variables of the flow model of the specification shown in Fig. 2.1c) are the binary flow variables x_{v_2,v_3} and x_{v_3,v_2}. The correctness of this model and the proof why only two integer variables must be defined will be result of our analysis in Section 3. In the ILP model of Example 2.6, M is a large constant, $\bar{L}$ is the latency bound. The variables are the start times $t_i, i = 1, \cdots, 3$ of the flow variables x_{v_i,v_j}. The cost function says that the flow of resources out of resource node r_1 is to be minimized.*

Summarizing our model, there exist different types of constraints:

- data dependencies (type D): these guarantee that data dependencies are satisfied.

- flow constraints (type F): these consist of
  - binding constraints (type F1) (each node of a graph G must be bound to a resource type);
  - flow conservation (type F2) (in-flow equals out-flow); and
- sequentialization constraints (type S): data dependencies which become active depending on the flow, e.g., for $x_{v_2,v_3} = 1$ (flow from v_2 to v_3)

  $$t_3 - t_2 \geq 1 + (1 - 1)M$$

  This constraint implies that v_3 can start only after v_2 is finished. In case $x_{v_2,v_3} = 0$, we have

  $$t_3 - t_2 \geq 1 - M << 0$$

  In this case, the sequentialization constraint is not active.
- integrality constraints.

Example 2.8 *With $\bar{L} = 3$, we obtain the solution $x_{r_1,v_1} = 1$; either $x_{v_2,v_3} = 1$ or $x_{v_3,v_2} = 1$, but never both at the same time, because the activation of both sequentialization constraints ((S.1) and (S.2)) wouldn't allow an assignment of start times t_2 and t_3 (positive cycle)). Fig. 2.1c) shows a solution to the ILP model where flow edges with positive flow are shown fat. In this case, only one resource is needed. One can easily show that all cycles of activated flow edges include a resource node.*

In Example 2.8 the flow graph was not a complete graph. In Section 3.3, we will present some results on which flow variables can never be activated, and hence, are redundant. For example, sequentialization constraints of flow edges opposed to data dependencies can never be active.

Before formally introducing the flow model, we would like to compare our approach with existing models.

2.2.2 Existing Approaches

Given a specification consisting of a problem graph $G(V, E)$ and a resource graph $G_R(V_R, E_R)$. We consider the formulation of Hwang et al. [HLH91] that is realized in the design system ALPS for high-level synthesis and a formulation of Gebotys [GE92, GE93] that has been implemented in the design system OASIC.

ALPS

The methodology described in [HLH91] represents one of the first approaches to solving scheduling problems with resource constraints in the context of high-level synthesis using ILP-techniques. The system ALPS implements this methodology and supports numerous features like chaining, multi-cycle operations, and pipelining. The ILP model for cost minimization under latency constraints looks as follows:

- *cost function*:
  minimize the total costs:

$$\min \sum_{k=1}^{|V_T|} c_k \cdot a_k \tag{2.1}$$

  where $c_k = c(r_k)$ represents the cost of a resource of type $r_k \in V_T$ and $a_k = \alpha(r_k)$.

- *constraints*:

  1. *binding constraints*:
     Each node must be started exactly once and must be bound to exactly one resource type. Contrary to the flow model, binary variables $x_{i,k,t}$ are introduced where i stands for node v_i, k stands for resource type r_k, and the index t represents the start time of v_i on an instance of resource type k. We obtain:

$$\sum_{k:(v_i,r_k)\in E_R} \sum_{t \in N(v_i)} x_{i,k,t} = 1 \qquad \forall i = 1, \cdots, |V| \tag{2.2}$$

  2. *data dependencies*:
     Using (2.2), the execution time $d_i = d(v_i)$ of node v_i can be easily computed as

$$d_i = \sum_{k:(v_i,r_k)\in E_R} \sum_{t \in N(v_i)} x_{i,k,t} \cdot w((v_i, r_k))$$

     and the start time $t_i = \tau(v_i)$ as

$$t_i = \sum_{k:(v_i,r_k)\in E_R} \sum_{t \in N(v_i)} t \cdot x_{i,k,t}$$

     Hence, the data dependencies become:

$$t_j - t_i \geq d_i \quad \forall (v_i, v_j) \in E \tag{2.3}$$

  3. *latency constraint*:
     Each node must have finished its operation by the latency bound $\bar{L}$:

$$t_i + d_i \leq \bar{L} \quad \forall i = 1, \cdots, |V| \tag{2.4}$$

  4. *resource constraints*:
     For each resource type $r_k \in V_T$, no more than $a_k = \alpha(r_k)$ nodes may be busy at the same time:

$$\sum_{i:(v_i,r_k)\in E_R} \sum_{p'=\max\{0,t-h_i\}}^{\min\{w((v_i,r_k))-1,t-l_i\}} x_{i,k,t-p'} \leq a_k \tag{2.5}$$
$$\forall k = 1, \cdots, |V_T| \ \wedge \min_{i=1,\cdots,|V|}\{l_i\} \leq t \leq \max_{i=1,\cdots,|V|}\{h_i\}$$

Traditionally, the complexity of an ILP formulation is measured by the number of integer variables and the number of inequalities. However, a word of caution is in order: Often, the structure of the solution polytope plays a not less important role for the solution time.

In the ALPS-model, the number of integer (binary) variables is upper bounded by $u_x \cdot |E_R|$, where $u_x = \max_{i=1,\cdots,|V|}\{h_i - l_i + 1\}$. The number of inequality constraints evaluates to $|V|$ (2.2), $|E|$ (2.3) and $u_t \cdot |V_T|$ (2.5) with $u_t = \max_{i=1,\cdots,|V|}\{h_i\} - \min_{i=1,\cdots,|V|}\{l_i\} + 1$.

Hence, the complexity of the ILP model in (2.1)$\sim$(2.5) is: $\mathcal{O}(u_x \cdot |E_R|)$ binary variables and $\mathcal{O}(|V| + |E| + u_t \cdot |V_T|)$ constraints.

We present a small example to demonstrate this model.

Example 2.9 *Again, we consider the specification in Fig. 2.1a), b) assuming that the costs for resource type r_k are equal to $c(r_k) = 1$. With the latency bound $\bar{L} = 3$, we obtain $l_1 = 0, l_2 = l_3 = 1$ and $h_1 = 1, h_2 = h_3 = 2$. Below, the ILP model for cost optimization under latency constraint (for reasons that there exists only one resource type, the index k has been omitted):*

$$\min\ a$$

subj. to

$$\begin{array}{lllllllll}
 & x_{1,1} & & & & & \leq \bar{L}-1 \\
 & & x_{2,1} & +2x_{2,2} & & & \leq \bar{L}-1 \\
 & & & & x_{3,1} & +2x_{3,2} & \leq \bar{L}-1 \\
 & -x_{1,1} & +x_{2,1} & +2x_{2,2} & & & \geq 1 \\
 & -x_{1,1} & & & +x_{3,1} & +2x_{3,2} & \geq 1 \\
x_{1,0} & +x_{1,1} & & & & & = 1 \\
 & & x_{2,1} & +x_{2,2} & & & = 1 \\
 & & & & x_{3,1} & +x_{3,2} & = 1 \\
x_{1,0} & & & & & & \leq a \\
 & x_{1,1} & +x_{2,1} & & +x_{3,1} & & \leq a \\
 & & & x_{2,2} & & +x_{3,2} & \leq a
\end{array}$$

OASIC

Gebotys [GE92, GE93] tried to improve the structure of the solution polytope of Hwang's [HLH91] model. Her improvements consist in a reformulation of the data dependencies such that a *Weighted Node Packing* problem is obtained that can be easier solved due to certain properties of the solution polytope. In particular, the equalities in (2.3) have been reformulated as follows:

$$\sum_{k:(v_j,r_k)\in E_R} \sum_{t'\leq t, t'\in N(v_j)} x_{j,k,t} + \sum_{k:(v_i,r_k)\in E_R} \sum_{t'\geq t-w((v_i,r_k))+1, t'\in N(v_i)} x_{i,k,t} \leq 1$$
$$\forall\ (i,j) \in E\ \wedge\ t \in \{(N(v_i) + w((v_i,r_k)) - 1) \cap N(v_j)\} \qquad (2.6)$$

The other constraints remain the same as in the ALPS-model. The same binary variables are used. The number of constraints in (2.6) is bounded here by $u_x \cdot |E|$. Hence, the complexity of this model is $\mathcal{O}(u_x \cdot |E_R|)$ binary variables and $\mathcal{O}(u_x \cdot |E| + |V| + u_t \cdot |V_T|)$ inequalities.

Example 2.10 *Again, we consider the example from Fig. 2.1. Only the data dependency constraints that differ from the ALPS model are shown.*

$$\begin{array}{ccccl} x_{1,1} & + & x_{2,1} & \leq 1 \\ x_{1,1} & + & x_{3,1} & \leq 1 \end{array}$$

2.2.3 ILP flow model

The fact that in both of the above formulations, the bound $u_x = \max_{i=1,\cdots,|V|}\{h_i - l_i + 1\}$ appears as a factor in the number of integer variables means that the complexity is dependent of the chosen latency bound $\bar{L}$, because ALAP-times $h_i, i = 1, \cdots, |V|$ depend on this bound. In order for the model to be solvable in a reasonable amount of time, only problems with a small latency bound can be solved. This restricts the above models to the domain of high-level synthesis.

Now, we would like to formally introduce the flow model in which resource constraints are considered as flow constraints in a flow graph. In this model, one could imagine that a resource flows through a set of nodes of the problem graph and executes the nodes in exactly this order. If a node $v_i \in V$ receives a resource of type r_k it occupies it for an interval of $w((v_i, r_k))$. Then, the resource may be passed to another node $v_j \in V$. This corresponds to a flow from v_i to v_j and implies a sequentialization. In order to model this flow-dependent operation sequentialization, a flow graph is introduced:

Definition 2.8 (Flow graph) *Given a specification with problem graph $G(V, E)$ and resource graph $G_R(V_R, E_R)$ where $V_R = V \cup V_T$ and V_T represents the set of resource types of G_R. The* flow graph *$G_F(V_F, E_F)$ is a directed graph that is defined as follows:*

- *$V_F = \bigcup_{k=1}^{|V_T|} V_k$ is the set of nodes where $V_k = \bigcup_{v_i \in V : \exists (v_i, r_k) \in E_R} v_i \cup \{r_k\}$ and r_k: resource node (source and drain of flows) of type k.*

- *$E_F = \bigcup_{k=1}^{|V_T|} E_k$ is the set of so-called* flow edges *(also sequentialization edges in the following) where $E_k = V_k \times V_k$ for all $k = 1, \cdots, |V_T|$ (complete graph).* [2]

Hence, the flow graph comprises $|V_T|$ complete subgraphs where the $k-$th subgraph consists of all nodes that may be executed on an instance of resource type k and the resource node r_k.

In all our following representations, flow edges will be represented by dashed lines.

[2]For purpose of simplicity, we assume that an edge may appear in E_F several times. Each of these edges must, however, belong to a different set E_k (module selection).

Example 2.11 *We consider the problem graph $G(V, E)$ and resource graph $G_R(V_R, E_R)$ in Fig. 2.1. The corresponding flow graph consists of ($|V_T| = 1$) one component and is shown in Fig. 2.3.*

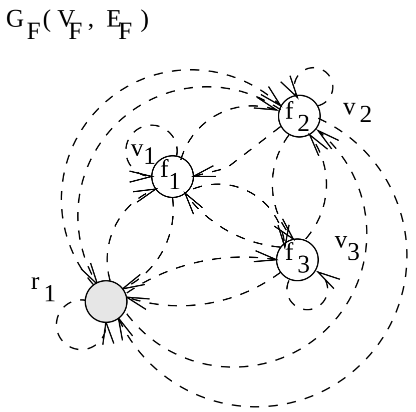

Figure 2.3: flow graph

One possible flow with one allocated resource has been shown in Fig. 2.1c) in bold: The resource node r_k sends the resource to node v_1 which after execution, passes the resource to node v_2, then to v_3, and finally back to the resource node. One can recognize that not only the binding β can be computed from the flow of resources (the k for which a node $v_i \in V$ obtains a positive flow), but also γ: Each cycle of activated flow edges starting in r_k includes all nodes bound to one and the same resource instance. This is an advantage over other ILP models like in ALPS where from the variables $x_{i,k,t}$ only the resource type k (β) but not the resource instance (γ) can be deduced.

Before we will now formally introduce the flow model, we would like to present one more example.

Example 2.12 *We consider the problem graph $G(V, E)$ and the resource graph $G_R(V_R, E_R)$ in Fig. 2.4a), b) respectively. The corresponding flow graph $G_F(V_F, E_F)$ is shown in Fig. 2.4c). In this case, it consists of two subgraphs.*

We would like to remark that in case a node $v_i \in V$ may be scheduled on more than one resource type $r_k \in V_T$ (module selection), corresponding subgraphs are connected.

The following optimization model is obtained:

- *Objective function*:

$$minimize \sum_{k=1}^{|V_T|} c(r_k) \cdot \sum_{(r_k, v_j) \in E_k : k \neq j} x_{r_k, v_j, k} \tag{2.7}$$

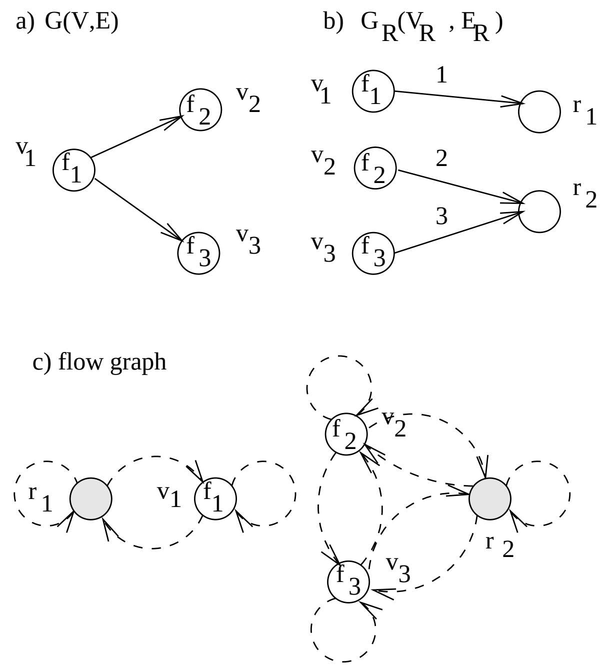

Figure 2.4: problem graph G a), resource graph G_R b) and flow graph G_F c)

For each edge $(v_i, v_j) \in E_k, i \neq j$, a binary *flow variable* $x_{v_i,v_j,k}$ is introduced with the meaning that in case $x_{v_i,v_j,k} = 1$ ($= 0$) there is (no) flow of a resource of type k from node v_i to node v_j passed on edge $(v_i, v_j) \in E_k$. The first two indices denote node identifiers. The third index k denotes the resource type.

The objective function serves to minimize the overall costs, represented as the sum of costs of resource types k multiplied by the number of flows on edges that are leaving the k-th resource node.

- *Constraints*:

  1. *flow constraints* (type F):
     For any node $v_i \in V$, the sum of active in-edges must be equal to the sum of out-edges. This requirement must hold for any type k for which there exists an edge $(v_i, r_k) \in E_R$. This means:

$$\sum_{(v_j,v_i)\in E_k: i\neq j} x_{v_j,v_i,k} - \sum_{(v_i,v_j)\in E_k: i\neq j} x_{v_i,v_j,k} = 0 \quad \forall v_i \in V \wedge \forall k : \exists (v_i, r_k) \in E_R \tag{2.8}$$

     (Remark: resource nodes must not be considered separately.)
     Now, each node $v_i \in V$ must receive exactly one resource that executes it:

$$\sum_{k:\exists (v_i,r_k)\in E_R} \sum_{(v_j,v_i)\in E_k: i\neq j} x_{v_j,v_i,k} = 1 \quad \forall i = 1, \cdots, |V| \tag{2.9}$$

     The flow that enters a node must be equal to one for any node $v_i \in V$.

2. *data dependencies* (type D):
   The execution time d_i of node $v_i \in V$ is given by:

$$d_i = \sum_{k:\exists(v_i,r_k)\in E_R} \sum_{(v_j,v_i)\in E_k: i\neq j} x_{v_j,v_i,k} \cdot w((v_i,r_k))$$

   Hence, data dependencies can be formulated as follows:

$$t_j - t_i \geq d_i \quad \forall (v_i, v_j) \in E \tag{2.10}$$

   Herein, $t_i, i = 1, \cdots, |V|$ denote real-valued variables.

3. *latency constraints* (type Z): For a given latency bound $\bar{L}$, the following constraint must be satisfied for the start times t_i:

$$t_i + d_i \leq \bar{L} \quad \forall i = 1, \cdots, |V| \tag{2.11}$$

4. *sequentialization constraints* (type S):

$$\begin{aligned} t_j - t_i \;\; \geq \;\; & d_i + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k) \\ & \forall (v_i, v_j) \in E_k, k = 1, \cdots, |V_T|, v_i, v_j \in V, i \neq j \end{aligned} \tag{2.12}$$

5. *resource constraints* (type R):

$$\sum_{(r_k,v_j)\in E_k: k\neq j} x_{r_k,v_j,k} \leq a_k \quad \forall k = 1, \cdots, |V_T| \tag{2.13}$$

   For each resource type, the number of activated out-edges must be smaller or equal to the allocation $a_k = \alpha(r_k)$.

6. *integrality constraints* (type G):

$$\begin{aligned} & x_{v_i,v_j,k} \in \{0, 1\} \\ & \qquad \forall (v_i, v_j) \in E_k, k = 1, \cdots, |V_T|, v_i, v_j \in V, i \neq j \end{aligned} \tag{2.14}$$

Here, $\delta(v_i, v_j, k) \in \mathbf{Z}_0^+$ is a „large" constant, the values of which, however, have an important influence on the structure of the solution polytope. For the start times $t_i, i = 1, \cdots, |V|$, we require $t_i \in \mathbf{R}_0^+$.

We will see later that in general, the number of integrality constraints in Eqn. (2.14) can be greatly reduced. Also, in case of resource minimization, the resource constraints (type R) do not exist, and in case of latency minimization, the time constraints (type Z) do not appear.

Remark:
In case of latency minimization as well as in case of the weighted minimization of costs and latency, the latency also appears in the objective function. In order to determine L exactly, the constraints of the form

$$L \geq t_i + d_i \quad \forall i = 1, \cdots, |V|$$

are added.

Semantics of the sequentialization constraints: Let $x_{v_i,v_j,k} = 1$, then v_j is bound to resource type k and receives a resource from a node v_i which also is bound to the same resource. Hence, v_j can only start after v_i has finished its computation $(t_j - t_i \geq d_i + 0)$.In this case the sequentialization constraint becomes active. In case $x_{v_i,v_j,k} = 0$, v_j should not be sequentialized after v_i $(t_j - t_i \geq d_i - \delta(v_i, v_j, k) << 0)$. Hence, the sequentialization constraints create the connection between flow of resources (resource constraints) and valid schedules.

Example 2.13 *We consider the problem presented in example 2.12 with the problem graph and resource graph shown in Fig. 2.4 and an allocation of* $\alpha(r_1) = \alpha(r_2) = 1$. *The flow-model for latency minimization looks as follows:*

$$
\begin{array}{rl}
\min & L \\
\text{subj. to} & \\
(F1.1) & x_{r_1,v_1,1} - x_{v_1,r_1,1} = 0 \\
(F1.2) & x_{r_2,v_2,2} + x_{v_3,v_2,2} - x_{v_2,r_2,2} - x_{v_2,v_3,2} = 0 \\
(F1.3) & x_{r_2,v_3,2} + x_{v_2,v_3,2} - x_{v_3,r_2,2} - x_{v_3,v_2,2} = 0 \\
(F2.1) & x_{r_1,v_1,1} = 1 \\
(F2.2) & x_{r_2,v_2,2} + x_{v_3,v_2,2} = 1 \\
(F2.3) & x_{r_2,v_3,2} + x_{v_3,r_2,2} = 1 \\
(D.1) & t_2 - t_1 \geq 1 \\
(D.2) & t_3 - t_1 \geq 1 \\
 & L \geq t_1 + 1, L \geq t_2 + 2, L \geq t_3 + 3 \\
(S.1) & t_2 - t_3 \geq 3 + (x_{v_3,v_2,2} - 1)\delta(v_3, v_2, 2) \\
(S.2) & t_3 - t_2 \geq 2 + (x_{v_2,v_3,2} - 1)\delta(v_2, v_3, 2) \\
(R.1) & x_{r_1,v_1,1} \leq \alpha(r_1) \\
(R.2) & x_{r_2,v_2,2} + x_{r_2,v_3,2} \leq \alpha(r_2) \\
(G) & x_{v_2,v_3,2}, x_{v_3,v_2,2} \in \{0, 1\}
\end{array}
$$

The ILP contains two binary variables (G).

Theorem 2.1 (Correctness flow model) *Given a specification with problem graph* $G(V, E)$ *and resource graph* $G_R(V_R, E_R)$. *The solution of the ILP in Eqns. (2.8-2.14) returns a cost optimal schedule* $(t_i \in \mathbf{Z}_0^+, i = 1, \cdots, |V|)$ *and binding* (β, γ) *which satisfies the latency bound* $\bar{L}$ *if such a solution exists and returns the values ,,not solvable" else.*

We do not prove Theorem 2.1 here. The model has a worst case complexity of the number of binary variables of $\mathcal{O}(|V_T||V|^2)$ and $\mathcal{O}(|E_R| + |E| + 2|V| + |V|^2 + |V_T|)$ constraints.

In the following, we will present a number of results on how to reduce the number of integrality constraints based on polyhedral analysis and by introducing a number of optimizing functions for preprocessing the polyhedron before solving it.

Chapter 3

Analysis of the flow-model

3.1 Polyhedral Theory

Corresponding to Nemhauser [NW88], a „good“ formulation of the ILP-model is a basic requirement for being able to solve it. For this reason, we will analyze the structure of the solution polytope of the flow-model and prove some interesting properties that turn out to be relevant for being able to solve typical problems in an acceptable amount of time.

Some basic definitions of polyhedral theory are required for this purpose. A general ILP-model can be written as follows:

$$z_{ILP} = \min\{c^T x \mid x \in P_F,\ x \in \mathbf{Z}^n\} \tag{3.1}$$

where P_F denotes the *solution polytope* with

$$P_F = \{Ax \geq b,\ x \in \mathbf{R}^n_{\geq 0}\} \tag{3.2}$$

and $A \in \mathbf{Z}^{m \times n}$, $b \in \mathbf{Z}^{m \times 1}$ and $c^T \in \mathbf{Z}^{1 \times n}$.

Alternatively, the same problem may be formulated as

$$z_{ILP} = \min\{c^T x \mid x \in P_I\} \tag{3.3}$$

with

$$P_I = \mathrm{conv}\{x \in P_F \mid x \in \mathbf{Z}^n\} \tag{3.4}$$

where conv denotes the operator that returns the *convex hull* of a set. P_I is called *integer (convex) hull* of P_F.

Geometrically, these definitions are outlined in Fig. 3.1.

The *admissible solutions* of the ILP are marked by the fat points within P_F in Fig. 3.1.

Let the *LP-relaxation* of the ILP be given by

$$z_{LP} = \min\{c^T x \mid x \in P_F\} \tag{3.5}$$

Obviously, $P_I \subseteq P_F$ and $z_{LP} \leq z_{ILP}$ holds. Hence, the LP-relaxation always provides a lower bound of the objective function of the corresponding ILP.

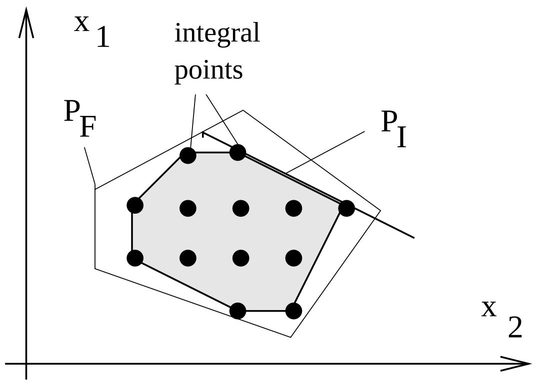

Figure 3.1: Two-dimensional polytope P_F and integer convex hull P_I

Now, ILP-models for which $P_I = P_F$ holds are easy to solve, because then, $z_{ILP} = z_{LP}$ holds and the ILP-model can be simply solved in polynomial time by solving its LP-relaxation. A polytope P_F with this property is called *integral polytope*.

Definition 3.1 (Integral polytope) *A polytope P_F is called* integral*, if $P_I = P_F$. Then, $z_{LP} = z_{ILP}$ holds, and the ILP may be solved in polynomial time by solving its LP-relaxation.*

Clearly, an integral polytope has the property that all its vertices are integral points. As it is known that any linear objective function has its minimum in one of the vertices of the solution polytope, it can be concluded that the solution of the LP-relaxation always will find an integer solution. Hence, the integrality constraints on variables can be simply omitted if $P_I = P_F$.

It is unlikely that the solution polytope of the flow model is integral. However, it is worth to spend some time in finding good approximations of P_I (many facets of P_F should be integral), because the solution times when using standard ILP-solvers such as a branch-and-bound- algorithm will strongly depend on the goodness of the approximation (see Fig. 3.1).

3.2 Analysis of the solution polytope of the flow-ILP

Now, we will investigate the polyhedral structure of the polytope of the flow-model. First, we will simply assume that each node $v_i \in V$ may only be mapped to exactly one resource type: (i.e., the number of $k : \exists(v_i, r_k) \in E_R$ is one for all $i = 1, \cdots, |V|$, no module selection). Then, we obtain the simplified constraint system: (Only difference: The execution times d_i are constants. As a result, the flow constraints (type F) and data dependency constraints (type D) are decoupled):

- *flow constraints* (type F):

$$\sum_{(v_j,v_i)\in E_k : i\neq j} x_{v_j,v_i,k} - \sum_{(v_i,v_j)\in E_k : i\neq j} x_{v_i,v_j,k} = 0 \quad \forall v_i \in V \wedge k : (v_i, r_k) \in E_R \tag{3.6}$$

$$\sum_{(v_j,v_i)\in E_k : i\neq j} x_{v_j,v_i,k} = 1 \quad \forall v_i \in V \;\wedge\; k : (v_i, r_k) \in E_R \tag{3.7}$$

- *data dependency constraints* (type D):

$$t_j - t_i \geq d_i \quad \forall (v_i, v_j) \in E \tag{3.8}$$

- *latency constraints*:

$$t_i \leq \bar{L} - d_i \quad \forall i = 1, \cdots, |V| \tag{3.9}$$

- *sequentialization constraints* (type S):

$$\begin{aligned} t_j - t_i \;\; \geq \;\; & d_i + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k) \\ & \forall (v_i, v_j) \in E_k, k = 1, \cdots, |V_T|, v_i, v_j \in V, i \neq j \end{aligned} \tag{3.10}$$

With $v_i \prec v_j$, we denote the property that there exists a directed path in G using edges of E from node $v_i \in V$ to node $v_j \in V$.

Then, we can deduce the following important theorem:

Theorem 3.1 (Integrality constraints) *Given a specification with problem graph G and resource graph G_R as well as the flow-ILP in Eqns. (3.6-3.10).*

1. *The number of integer (binary) optimization variables may be reduced to*

$$N_{ILP} = \sum_{k=1}^{|V_T|} \sum_{(v_i,v_j)\in E_k : v_i,v_j \in V \,\wedge\, i\neq j} n(i,j) \tag{3.11}$$

$$n(i,j) = \begin{cases} 1 & if \;\; i \not\prec j \;\wedge\; j \not\prec i \\ 0 & sonst \end{cases} \tag{3.12}$$

2. *Furthermore, all inequalities in (3.10) are redundant for pairs i, j for which $n(i,j) = 0$ holds, hence may be eliminated.*

3. *The integrality constraints consider only the N_{ILP} variables $x_{v_i,v_j,k}$ that exist after elimination of the redundant inequalities (2.) in (3.10).*

Proof: First, part 2) will be proven. We consider the constraint with flow variable $x_{v_i,v_j,k}$ in (3.10). In particular, we have to show that the inequality is redundant if $n(i,j) = 0$. Now, $n(i,j) = 0$ holds, if either 1) $v_i \prec v_j$ or 2) $v_j \prec v_i$. Case 1): $v_i \prec v_j$ holds. Then, there exists a set of data dependencies $(v_i, \cdot), \cdots (\cdot, v_j) \in E$ and therefore a set of data dependency constraints in (3.8) that imply $t_j - t_i \geq d_i$. Hence, the inequality is redundant. 2) Case 2: We have $v_j \prec v_i$. We show that in each valid solution, $x_{v_i,v_j,k} = 0$ holds. Hence, the constraint is not active and can also be eliminated. The proof is by contradiction. Let $x_{v_i,v_j,k} = 1$. Then, the inequality $t_j - t_i \geq d_i > 0$ is active. Due to $v_j \prec v_i$, $t_i - t_j \geq 0$ holds. Both inequalities can never be active. Therefore, $x_{v_i,v_j,k} = 0$ in any valid solution.

Part 1) will be proven as follows: Let $X_S \in \{0,1\}^{N_{ILP}\times 1}$ be the vector of those N_{ILP} flow variable in (3.10) that are not redundant. For any admissible solution X_S (all constraints in (3.6-3.10) are satisfied) that is integral, we define a polytope $P_F(X_S)$ in which X_S is fixed. The theorem is proven if we can show that any $P_F(X_S)$ is integral. This is done in the next theorem. □

It turns out that the proof of all $P_F(X_S)$ to be integral is easier in matrix notation.

Let $C \in \{0,+1,-1\}^{|V|\times|E|}$ denote the *(node-edge) incidence matrix* of G, let $T \in \mathbf{R}^{|V|\times 1} = (t_1, t_2, \cdots, t_{|V|})^T$ denote the vector of start times of the nodes, and let $w \in \mathbf{Z}^{|E|\times 1} = (w_1, w_2, \cdots, w_{|E|})^T$ be a constant vector of node execution times with $w_l = d_i \; \forall l = 1, \cdots, |E| : e_l = (v_i, v_j) \in E$. Then, the constraints in (3.8) become

$$C^T T \geq w$$

Let $X = (X_{\bar{S}}^T,\ X_S^T)^T \in \{0,1\}^{|E_{ges}|\times 1}$ with $E_{ges} = |E_F| - |V| - |V_T|$ ($|V| + |V_T|$ corresponds to the number of not introduced flow variables for self loops) denote the vector of flow variables with $X_{\bar{S}} \in \{0,1\}^{|E_{ges}|-N_{ILP}\times 1}$ and $X_S \in \{0,1\}^{N_{ILP}\times 1}$. Let $(C_{\bar{S}},\ C_S) \in \{-1,0,1\}^{|V|\times|E_{ges}|}$ denote the incidence matrix of the flow graph (without self loops) $C_{\bar{S}} \in \{-1,0,1\}^{|V|\times(|E_{ges}|-N_{ILP})}$ and $C_S \in \{-1,0,1\}^{|V|\times N_{ILP}}$. Then, the constraints in (3.6) can be formulated as

$$(C_{\bar{S}},\ C_S)\begin{pmatrix} X_{\bar{S}} \\ X_S \end{pmatrix} = 0$$

Finally, let $(D_{\bar{S}},\ D_S) \in \{-1,0,1\}^{|V|\times|E_{ges}|}$ denote the constraints in (3.7) with $D_{\bar{S}} \in \{0,1\}^{|V|\times(|E_{ges}|-N_{ILP})}$ and $D_S \in \{0,1\}^{|V|\times N_{ILP}}$. Then, then constraints in (3.7) can be written as

$$(D_{\bar{S}},\ D_S)\begin{pmatrix} X_{\bar{S}} \\ X_S \end{pmatrix} = 1$$

There remain the sequentialization constraints in (3.10) that, formulated in matrix notation, become

$$C_S^T T \;-\; M_S X_S \geq m_S$$

with

$$M_S \in \mathbf{Z}^{N_{ILP}\times N_{ILP}} \;=\; \begin{pmatrix} \ddots & & \\ & \delta(v_i, v_j, k) & \\ & & \ddots \end{pmatrix}$$

and

$$m_S \in \mathbf{Z}^{N_{ILP}\times 1} \;=\; (\cdots, d_i - \delta(v_i, v_j, k), \cdots)^T$$

Hence, the complete ILP can be formulated in matrix notation as follows:

$$z_{ILP} = \min\{c^T \cdot (T^T,\ X_{\bar{S}}^T,\ X_S^T)^T \mid (T^T,\ X_{\bar{S}}^T,\ X_S^T)^T \in P_F, X_S \in \mathbf{Z}^{N_{ILP}\times 1}\} \tag{3.13}$$

with

$$P_F = \{x \in \mathbf{R}_{\geq 0}^{|V|+|E_{ges}|} \mid Ax \ (= = \geq \geq)^T \ b\} \tag{3.14}$$

and

$$A = \begin{pmatrix} 0 & C_{\bar{S}} & C_S \\ 0 & D_{\bar{S}} & D_S \\ C^T & 0 & 0 \\ C_S^T & 0 & -M_S \end{pmatrix}, \quad x = \begin{pmatrix} T \\ X_{\bar{S}} \\ X_S \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 0 \\ 1 \\ d \\ m_S \end{pmatrix} \tag{3.15}$$

Theorem 3.2 (Integrality of $P_F(X_S)$) *Given a specification with problem graph G and resource graph G_R as well as the ILP in Eqns. (3.6-3.10). The solution polytope P_F is given by Eqns. (3.14-3.15). Let $X_S \in \{0,1\}^{N_{ILP}\times 1}$ be the vector of those N_{ILP} flow variables in (3.10) that are not redundant corresponding to Theorem 3.1. For any X_S of admissible solutions, let $P_F(X_S)$ denote that subpolytope of P_F, in which $X_S \in \{0,1\}^{N_{ILP}\times 1}$ is fixed, all other variables not determined. Then, for any such X_S:*

$$P_F(X_S) \quad \text{is integral.}$$

Proof: Let $X_S \in \{0,1\}^{N_{ILP}\times 1}$ be fixed. Then, $P_F(X_S)$ can be written in matrix notation as follows:

$$P_F(X_S) = \{x' \in \mathbf{R}_{\geq 0}^{|V|+|E_{ges}|-N_{ILP}} \mid A'x' \ (= = \geq \geq)^T \ b'\} \tag{3.16}$$

with

$$A' = \begin{pmatrix} 0 & C_{\bar{S}} \\ 0 & D_{\bar{S}} \\ C^T & 0 \\ C_S^T & 0 \end{pmatrix}, \quad x' = \begin{pmatrix} T \\ X_{\bar{S}} \end{pmatrix} \quad \text{und} \quad b' = \begin{pmatrix} -C_S X_S \\ 1 - D_S X_S \\ d \\ m_S + M_S X_S \end{pmatrix} \tag{3.17}$$

Because X_S is integral, b' is integral and constant. Furthermore, the solution polytope is decomposed into two orthogonal subpolytopes $P_{F,T}(X_S)$ and $P_{F,X_{\bar{S}}}(X_S)$ with

$$P_{F,T}(X_S) = \{T \in \mathbf{R}_{\geq 0}^{|V|} \mid A_T T = b_T\}$$

and

$$P_{F,X_{\bar{S}}}(X_S) = \{X_{\bar{S}} \in \mathbf{R}_{\geq 0}^{|E_{ges}|-N_{ILP}} \mid A_{X_{\bar{S}}} X_{\bar{S}} \geq \ b_{X_{\bar{S}}}\}$$

such that $P_F(X_S) = P_{F,T}(X_S) \cap P_{F,X_{\bar{S}}}$. The matrices become:

$$A_T = \begin{pmatrix} C^T \\ C_S^T \end{pmatrix}, \quad b_T = \begin{pmatrix} d \\ m_S + M_S X_S \end{pmatrix} \tag{3.18}$$

and

$$A_{X_{\bar{S}}} = \begin{pmatrix} C_{\bar{S}} \\ D_{\bar{S}} \end{pmatrix}, \quad b_{X_{\bar{S}}} = \begin{pmatrix} -C_S X_S \\ 1 - D_S X_S \end{pmatrix} \tag{3.19}$$

It remains to prove that both subpolytopes are integral. For this purpose, we apply the following corollary proven in [PS82]:

Corollary 3.1 *given a polytope $P_F = \{x : Ax \geq b, x \geq 0\}$ (or $P_F = \{x : Ax = b, x \geq 0\}$). If the matrix A* is totally unimodular *(TU), then P_F is integral for any constant, integral vector b, if P_F is non-empty.*

Now, it is known (see [PS82]) that incidence matrices of directed graphs are TU. Using the following corollary from [NW88], page 540, we will show that A_T is TU:

Corollary 3.2 *The following statements are equivalent:*

1. *A is TU.*
2. *A^T, the transposed of A, is TU.*
3. *...*

Now, $A_T = (C,\ C_S)^T$ is the transposed of the incidence matrix of the graph with node set V and edge set E (incidence matrix C) plus additional N_{ILP} edges of flow arcs defined in (3.10). Hence, $(C,\ C_S)$ is an incidence matrix and hence, TU. Now, as A_T is the transposed of this matrix, A_T is also TU due to Corollary 3.2.

The proof is finished if we can show that $A_{X_{\bar{S}}}$, too, is TU: $C_{\bar{S}}$ is also an incidence matrix of the graph with node set V and those $|E_{ges}| - N_{ILP}$ flow edges that do not appear in any of the Eqns. (3.10). The matrix $D_{\bar{S}}$ has the property that in each column there exists exactly one one, and zeroes else (because any edge (variable in $X_{\bar{S}}$) may only be incident to exactly one node). Hence, any quadratic submatrix of $D_{\bar{S}}$ is unimodular and thus $D_{\bar{S}}$. We show now that $A^T_{X_{\bar{S}}} = (C^T_{\bar{S}},\ D^T_{\bar{S}})^T$ is TU and hence, using Corollary 3.2 also $A_{X_{\bar{S}}}$.

Fur this purpose, we apply the following theorem from [NW88], page 542:

Theorem 3.3 ([NW88], page 542) *The following statements are equivalent:*

1. *$A \in \{-1, 0, 1\}^{m \times n}$ is TU.*
2. *For any set $J \subseteq N = \{1, \cdots, n\}$ of columns, there exists a partition J_1, J_2 of columns of J such that*

$$s_i = |\sum_{j \in J_1} a_{i,j} - \sum_{j \in J_2} a_{i,j}| \leq 1 \quad \forall i = 1, \cdots, m \tag{3.20}$$

In $A^T_{X_{\bar{S}}}$, any row contains a maximum of three non-zero elements: There is at most one (-1)- element and a maximum of two 1-elements. Condition 2 in Theorem 3.3 is obviously satisfied for all subsets $J \subseteq N$ that contain only a) columns in $C^T_{\bar{S}}$ ($J_2 = \emptyset$) ($|\sum_{j \in J_1} a_{i,j}|$ is either 0 or 1), b) columns of $D^T_{\bar{S}}$ ($J_2 = \emptyset$) ($|\sum_{j \in J_1} a_{i,j}|$ is either 0 or 1). The case c), where J contains a subset of columns $C^T_{\bar{S}}$ and $D^T_{\bar{S}}$, is harder to prove. We consider the following example:

Example 3.1 *We consider the ILP-model for the problem in Example 2.11, and obtain the following matrix* $A^T_{X_{\bar{S}}}$*:*

$$A^T_{X_{\bar{S}}} = \begin{array}{c|ccc|ccc|} & & C^T_{\bar{S}} & & & D^T_{\bar{S}} & \\ X_{\bar{S}} & (F1.1) & (F1.2) & (F1.3) & (F2.1) & (F2.2) & (F2.3) \\ \hline x_{r_1,v_1} & 1 & 0 & 0 & 1 & 0 & 0 \\ x_{r_1,v_2} & 0 & 1 & 0 & 0 & 1 & 0 \\ x_{r_1,v_3} & 0 & 0 & 1 & 0 & 0 & 1 \\ x_{v_1,v_2} & -1 & 1 & 0 & 0 & 1 & 0 \\ x_{v_1,v_3} & -1 & 0 & 1 & 0 & 0 & 1 \\ x_{v_1,r_1} & -1 & 0 & 0 & 0 & 0 & 0 \\ x_{v_2,r_1} & 0 & -1 & 0 & 0 & 0 & 0 \\ x_{v_3,r_1} & 0 & 0 & -1 & 0 & 0 & 0 \end{array} \tag{3.21}$$

In any column $D^T_{\bar{S}j}$ *of* $D^T_{\bar{S}}$*, there exists a unique column in* $C^T_{\bar{S}c(j)}$ *that has exactly a one-entry where* $D^T_{\bar{S}j}$ *has a one-entry (property of the flow constraints of type F1 and F2).*

The proof can now be easily finished: If a set $J \subseteq N$ consists only of columns $C^T_{\bar{S}}$ and $D^T_{\bar{S}}$, then we derive the partition into blocks J_1 and J_2 as follows: For each column of $D^T_{\bar{S}j} \in J$, let $D^T_{\bar{S}j} \in J_2$, if column $C^T_{\bar{S}c(j)} \in J$, else $D^T_{\bar{S}j} \in J_1$. For all columns $C^T_{\bar{S}j} \in J$ we let: $C^T_{\bar{S}j} \in J_1$.
In the first case, column j and column $c(j)$ are subtracted. Hence, the absolute value of the difference of these two columns becomes either $s_i = 0$ or $s_i = 1$ in each row. The addition of additional columns from $D^T_{\bar{S}}$ does not change this, as there is exactly one one in each row. Also the addition of $C^T_{\bar{S}}$ does not change Eqn. (3.20) to hold, because in this case, only zeroes or one -1 are added in case $s_i = 0$ or, in case $s_i = -1$, only zeroes or one 1 may be added (property of the incidence matrix).

If $C^T_{\bar{S}c(j)} \notin J$, then $D^T_{\bar{S}j} \in J_1$. In this case, the sum of all columns $C^T_{\bar{S}} \in J$ can be either zero or -1 in each row where arbitrary additive columns $D^T_{\bar{S}j} \in J$ may add only zero or a one. This finishes the proof. □

Example 3.2 *Again, we consider the problem of Example 2.11. Corresponding to Theorem 3.1, the only integrality constraints are necessary for the two flow variable* $x_{2,3}$ *and* $x_{3,2}$*, because* v_2 *and* v_3 *denote the only node pair for which neither* $v_2 \prec v_3$ *nor* $v_3 \prec v_2$ *(*$N_{ILP} = 2$*) holds.*

The following example serves to reveal some interesting implications of the proven results.

Example 3.3 *Consider a problem graph* $G(V, E)$ *with the property that for each pair of nodes* $v_i, v_j \in V, i \neq j$*, there exists either a path from* v_i *to* v_j *or from* v_j *to* v_i*. Under our assumption that to each node there is assigned a unique resource type, we can deduce from Theorem 3.1 that the flow ILP contains no integrality constraints, and hence, can be solved in polynomial time by solving its LP-relaxation.*

Theorem 3.4 (Integrality of P'_F) *Given a specification with problem graph G and resource graph G_R as well as the ILP-model in Eqns. (3.6-3.10) with the solution polytope P_F as described by Eqns. (3.14-3.15). Now, we eliminate all sequentialization constraints (Eqns. in (3.10)) obtaining the polytope $P_{F'}$ with*

$$P'_F = \{x \in \mathbf{R}_{\geq 0}^{|V|+|E_{ges}|} \mid Ax \; (= = \geq)^T \; b\} \tag{3.22}$$

and

$$A = \begin{pmatrix} 0 & C_{\bar{S}} & C_S \\ 0 & D_{\bar{S}} & D_S \\ C^T & 0 & 0 \end{pmatrix}, \quad x = \begin{pmatrix} T \\ X_{\bar{S}} \\ X_S \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 0 \\ 1 \\ d \end{pmatrix} \tag{3.23}$$

Then:

$$P_{F'} \quad \text{ist integral.}$$

Proof: In direct analogy to the proof of Theorem 3.2, one shows that as well C^T as the matrix

$$\begin{pmatrix} C_{\bar{S}} & C_S \\ D_{\bar{S}} & D_S \end{pmatrix}$$

are TU and hence, the complete matrix A in En. (3.23). □

This implies that the solution polytope of the flow-ILP, when eliminating all of the sequentialization constraints has only integral vertices, and hence, $P_{F'} = \text{conv}(P_{F'})$ holds.

Summary of the results of this section: In Theorem 3.1, we have shown that 1) the total number of integrality constraints can be reduced to N_{ILP}. Hence, neither the start times t_i of nodes nor those flow variables which do not appear in any sequentialization constraint in Eqn. (3.10) must be declared integer. 2) The theorem has further shown that many of the sequentialization constraints in (3.10) are redundant. N_{ILP} was the best bound that we found for the number of inequalities in (3.10) and therefore for the number of integer (binary) variables without optimizing the model. [1] 3) Finally, we have shown that the polytope $P_{F'}$ (obtained from P_F by removal of sequentialization constraints) is integral. Note that these results hold also by consideration of resource constraints of type R in Eqn. (2.13) and hence, for any of the here presented optimization problems.

Hence, the complexity of the model (no module selection) is $\mathcal{O}(N_{ILP})$ binary variables and $\mathcal{O}(|E_R| + |V| + |E| + |V| + N_{ILP} + |V_T|)$ constraints.

If N_{ILP} is smaller than $|V| \times u_x$, the number of integer variables is smaller than in the ALPS-model and the OASIC-model. In the following, we will show that also the choice of the constants plays an important role for obtaining a good approximation of the integer convex hull of the solution polytope.

[1] In some cases, the number of variables may be further reduced by maximally $|V| - 1$ variables, if from the N_{ILP} variables, those $|V| - 1$ variables are selected that create a spanning tree in the flow graph.

3.3 Improving the polyhedral structure

In the preceding section, we showed that the complexity of the flow-model is caused by those flow variables $x_{v_i,v_j,k}$ for which neither $v_i \prec v_j$ nor $v_j \prec v_i$ holds. The question that interests us now is: How good does the polytope P_F approximate P_I? In order to find a better approximation, we will make use of transformations. In particular, the following techniques will be exploited:

- *Elimination of redundant variables*:
  We will introduce path analysis techniques in order to reduce the number of flow variables.

- *Improvement of polyhedral structure*:
  In general, $P_I \subset P_F$, and the LP-relaxation provides a lower bound on the objective function. The efficiency of an ILP-solver using a branch-and-bound-algorithm depends to a large amount on the property, how many facets of P_F are integral, because in that case, it is more likely that large subtrees may be pruned, because „good“ integer solutions have been found. In this context, we will show that the choice of the constants $\delta(v_i, v_j, k)$, declared so far as being large number, may be optimized in the sense to give better approximations of P_I.

- *Addition of valid inequalities*:
  The addition of inequalities that are facets of P_I may further reduce the search space.

3.3.1 Elimination of flow variables

Obviously, some flow variables $x_{v_i,v_j,k}$ can never have the value $x_{v_i,v_j,k} = 1$ as has been pointed out already in Theorem 3.1, see, e.g., in Fig. 3.2, where the activation of a flow variable would result in a cycle of data dependencies.

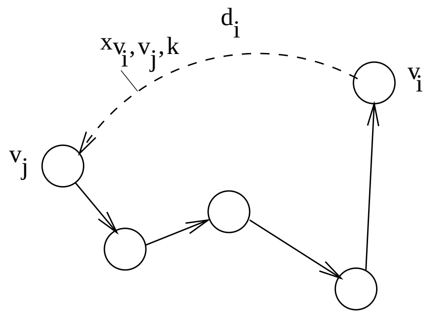

Figure 3.2: Elimination of flow variables $x_{v_i,v_j,k}$

Such variables and the corresponding inequalities that can never be active, can be simply eliminated.

A condition for the cases in which the activation of a flow variable would lead to a positive cycle, is given as follows:

Lemma 3.1 *Given a flow-ILP with the constraints in (2.8) - (2.14). Variable $x_{v_i,v_j,k}$ may be eliminated if $v_j \prec v_i$.*

Proof: We show that in any admissible solution of the constraint set, $x_{v_i,v_j,k} = 0$ must hold. Hence, the corresponding variable and inequality can be discarded. The proof is by contradiction. Let $x_{v_i,v_j,k} = 1$. Then, the corresponding inequality $t_j - t_i \geq d_i > 0$ is active. Due to $v_j \prec v_i$, $t_i - t_j \geq 0$ holds. Both inequalities cannot be simultaneously active. Hence, $x_{v_i,v_j,k} = 0$ in any admissible solution. □

Example 3.4 *We consider the flow graph in Fig. 2.1c). Corresponding to Lemma 3.1, the flow variables x_{v_2,v_1} and x_{v_3,v_1} are redundant. In Fig. 2.1c), they have already been left out.*

Lemma 3.2 *Given a flow-ILP with the constraints in Eqns. (2.8) - (2.14). Variable $x_{v_i,v_j,k}$ may be eliminated if*

$$d_i > h_j - l_i$$

where l_i denotes the ASAP-time of $v_j \in V$, and h_j denotes the ALAP-time of node $v_j \in V$.

Proof: We show that in any admissible solution of the constraint set, $x_{v_i,v_j,k} = 0$ and hence, $x_{v_i,v_j,k}$ can be eliminated. The proof is by contradiction. Let $x_{v_i,v_j,k} = 1$. Then the corresponding inequality $t_j - t_i \geq d_i$ is active. Now, we show that there is no admissible schedule that satisfies this condition: Obviously, the maximal value of $t_j - t_i$ is $h_j - l_i$. By definition, $d_i > h_j - l_i$ holds. Therefore, $t_j - t_i \geq d_i > h_j - l_i$ cannot be satisfied for any values t_j, t_i. Hence, $x_{v_i,v_j,k} = 0$ in any admissible solution. □

Example 3.5 *We consider the flow graph in Fig. 3.3. Let the execution times of the nodes be $d_1 = d_2, \cdots = d_6 = 1$. There is only one resource node.*

- *With $\bar{L} = 3$, we obtain the ASAP- and ALAP-times: $l_1 = l_4 = h_1 = h_4 = 0$, $l_2 = l_5 = h_2 = h_5 = 1$ and $l_3 = l_6 = h_3 = h_6 = 2$. The flow variables that may be eliminated corresponding to Lemma 3.2 are $x_{1,4}$, $x_{2,4}$, $x_{3,4}$, $x_{4,1}$, $x_{5,1}$, $x_{6,1}$, $x_{3,5}$, $x_{6,2}$, $x_{2,5}$, $x_{3,6}$, $x_{5,2}$ and $x_{6,3}$. The flow graph after elimination of these flow edges is shown in Fig. 3.3b).*

- *With $\bar{L} = 4$, we obtain the ASAP- and ALAP-times: $l_1 = l_4 = 0$, $h_1 = h_4 = 1$, $l_2 = l_5 = 1$, $h_2 = h_5 = 2$ and $l_3 = l_6 = 2$, $h_3 = h_6 = 3$. The flow variables that may be eliminated corresponding to Lemma 3.2 are now only the variables $x_{2,4}$, $x_{3,4}$, $x_{5,1}$, $x_{6,1}$, $x_{3,5}$, and $x_{6,2}$.*

- *For values larger than $\bar{L} = 6$, no flow variables may be eliminated corresponding to Lemma 3.2.*

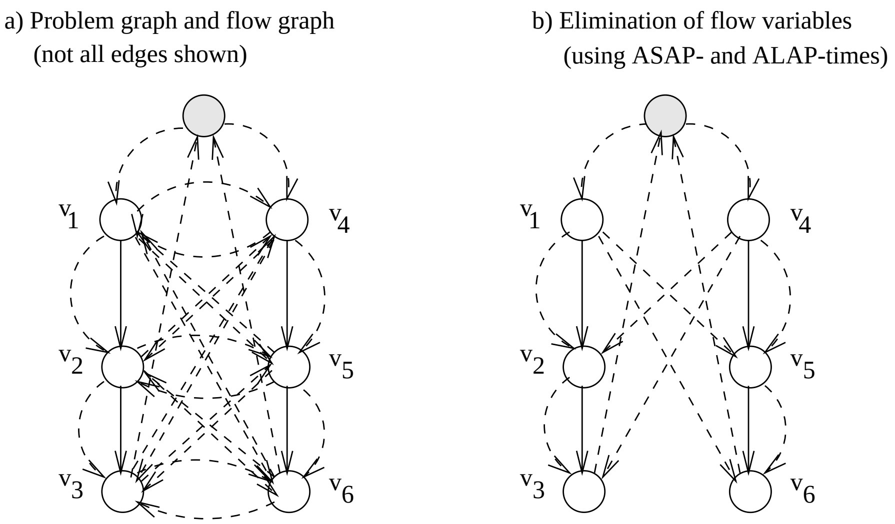

Figure 3.3: Elimination of flow variables

As could be seen in the previous example, the amount of reduction is strongly dependent on the determined latency bound. In case of latency minimization under resource constraints, the latency bound should be minimized beforehand with a heuristic such as list-scheduling or Force-Directed Scheduling. For the problem of cost minimization, the reduction effect is obviously larger in case of small latency bounds.

Example 3.6 *In this example, we would like to show that in case of dedicated resources, the optimization problem becomes trivial. We consider the problem graph*

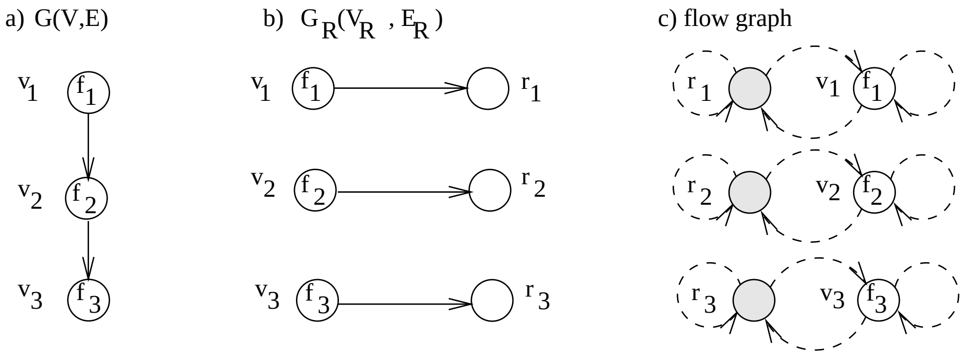

Figure 3.4: Problem graph a), resource graph b) and flow graph c)

$G(V,E)$ and the resource graph $G_R(V_R,E_R)$ in Fig. 3.4a), b) respectively. The corresponding flow graph is shown in Fig. 3.4c). We gain the following ILP-model:

$$\min \qquad x_{r_1,v_1,1} + x_{r_2,v_2,2} + x_{r_3,v_3,3}$$

$$
\begin{array}{rl}
\text{subj. to} & \\
(F1.1) & x_{r_1,v_1,1} - x_{v_1,r_1,1} = 0 \\
(F1.1) & x_{r_2,v_2,2} - x_{v_2,r_2,2} = 0 \\
(F1.1) & x_{r_3,v_3,3} - x_{v_3,r_3,3} = 0 \\
(F2.1) & x_{r_1,v_1,1} = 1 \\
(F2.2) & x_{r_2,v_2,2} = 1 \\
(F2.3) & x_{r_3,v_3,3} = 1 \\
(D.1) & t_2 - t_1 \geq 1 \\
(D.2) & t_3 - t_2 \geq 2 \\
 & t_1, t_2, t_3 \leq \bar{L} - 1
\end{array}
$$

Obviously, the model does not have any integrality constraint, because there are no sequentialization constraints (type S). Flow problem (type F) and scheduling problem (type D) are independent. The solution of the LP-relaxation provides $t_1 = 0, t_2 = 1, t_3 = 3$ *and* $L = 6$. *In the solution of the LP, all 6 flow variables* $x_{r_1,v_1,1}, x_{r_2,v_2,2}, x_{r_3,v_3,3}$, $x_{v_1,r_1,1}, x_{v_2,r_2,2}$ *and* $x_{v_3,r_3,3}$ *are one.*

The effort to apply the elimination transformation corresponding to Lemma 3.1 and 3.2 is polynomial (all-pair longest path computation using the Floyd-Warshall-algorithm ($\mathcal{O}(|V|^3)$), and determination of ASAP- ALAP-times ($\mathcal{O}(|V| + |E|)$), respectively).

3.3.2 Determination of constants $\delta(v_i, v_j, k)$

Here, we would like to show how the constants $\delta(v_i, v_j, k)$ in inequalities of the form

$$t_j - t_i \geq d_i + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k) \tag{3.24}$$

can be determined such that the model remains correct, however, that each inequality approximates P_I as good as possible.

We would like to point out the importance of this optimization by introduction of a small example.

Example 3.7 *We transform the inequality (3.24) into*

$$x_2 - \delta x_1 \geq d - \delta$$

by defining $x_2 := t_j - t_i$, $x_1 := x_{v_i,v_j,k}$ *and* $d := d_i$. *As an example, we consider two nodes* v_i *and* v_j *with the ASAP- and ASAP-times* $l_i = 3, h_i = 5$ *and* $l_j = 0, h_j = 4$. *Hence, we have* $\min\{t_j - t_i\} = l_j - h_i = -5$ *and* $\max\{t_j - t_i\} = h_j - l_i = 4 - 3 = 1$. *The constraints* $-5 \leq x_2 \leq 1$ *together with the constraints* $0 \leq x_1 \leq 1$ *define a rectangle (see Fig. 3.5). Also shown in Fig. 3.5, the constraint in (3.24) for different values of* δ *for* $d = d_i = 0$. *The hyperplanes for constraints with* $d_i > 0$ *are obtained for each value of* δ *by simply shifting the hyperplane in the point* $(x_1, x_2) = (1, 0)$ *by the ordinate* d_i. *Obviously, the inequality constraint reduces the solution polytope* *(shaded area). With the following lemma, we would like to show that the best value for δ (leading to the best approximation of P_I) is given be the smallest value that does not lead to exclusions of integral solutions at $x_1 = 0$ (where the constraint should not be active!). In the example, this value is $\delta = 5$ (dashed line)).*

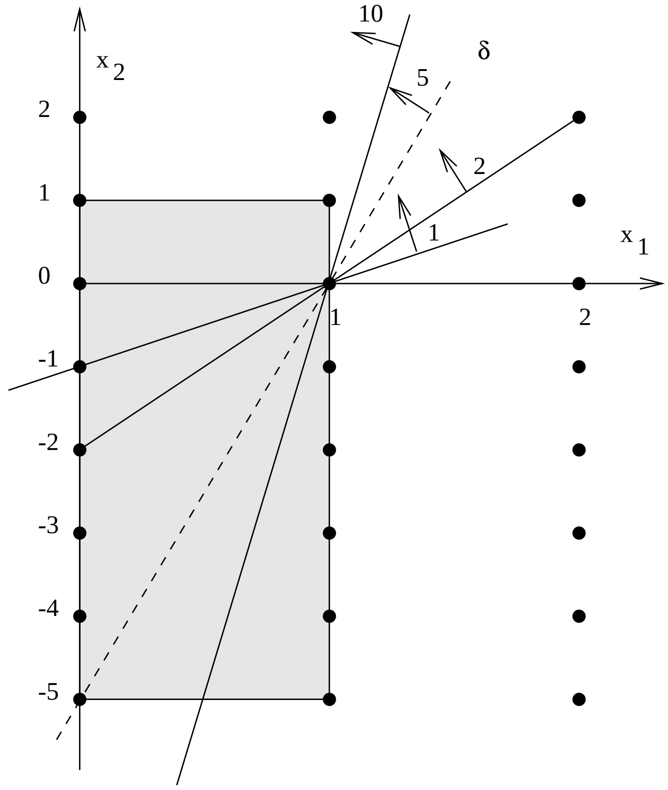

Figure 3.5: Optimization of constants $\delta(v_i, v_j, k)$

Lemma 3.3 *Given a set of sequentialization constraints in (2.12) of the form*

$$t_j - t_i \geq d_i + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k)$$

The constraints approximate the polytope P_I of the flow-ILP best if

$$\delta(v_i, v_j, k) = d_i - l_j + h_i \tag{3.25}$$

where l_j denotes the ASAP-time of v_j and h_i denotes the ALAP-time of v_i.

Proof: 1) It is required that in case of inactivity (corresponding to the case $x_1 = 0$ in Fig. 3.5), no integral solution for $t_j - t_i$ is excluded. 2) From the ASAP- and ALAP-time bounds, this time difference can only have values in the interval $[\min\{t_j - t_i\}, \cdots, \max\{t_j - t_i\}] = [l_j - h_i, \cdots, h_j - l_i]$. 3) Because of the fact that d_i results only in a shift of the hyperplane in the point $(x_1, x_2) = (1, 0)$ by the ordinate

d_i, the smallest possible δ represents the hardest constraint (see also in Fig. 3.5) that does not exclude a solution of $t_j - t_i$ at $x_{v_i,v_j,k} = 0$. Hence, the optimization problem becomes

$$\min\{\delta(v_i, v_j, k) \in \mathbf{Z}_{\geq 0} \mid \min\{t_j - t_i\} \geq d_i - \delta(v_i, v_j, k)\}$$

or

$$\min\{\delta(v_i, v_j, k) \in \mathbf{Z}_{\geq 0} \mid \delta(v_i, v_j, k) \geq d_i - \min\{t_j - t_i\}\}$$

4) The claim of Eqn. (3.25) follows from the relation $\min\{t_j - t_i\} = l_j - h_i$. 5) The complete solution polytope is best approximated if each of the sequentialization constraints are best approximated (principle of intersection). □

Example 3.8 *In Fig. 3.6a), the result of Lemma 3.2 can be explained: If $d_i > h_j - l_i$ (the maximal value of $x_2 = t_j - t_i$), then there does not exist a solution with $x_1 = 1$ what implies the redundancy of the corresponding variable and sequentialization constraint ($x_1 \equiv 0$). In Fig. 3.6b), the case $d_i < l_j - h_i$ is shown (the minimal value of $x_2 = t_j - t_i$). The constraint is redundant, but not the corresponding flow variable that can take values 0 or 1. However, as this variable does not appear any more in a sequentialization constraint, the integrality constraint can be dropped. This will be shown in the next paragraph.*

3.3.3 Elimination of redundant inequalities

In Fig. 3.6b), it could be seen that an inequality constraint of the form

$$t_j - t_i \geq d_i + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k)$$

(type S) is redundant if $d_i < l_j - h_i$.

The elimination of this inequality reduces at the same time the number of integrality constraints as will be proved in the next lemma.

Lemma 3.4 *Given a flow-ILP with the constraints in (2.8) - (2.14) including a constraint of type S of the form*

$$t_j - t_i \geq d_i + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k)$$

The inequality and the integrality constraint on the corresponding variable $x_{v_i,v_j,k}$ can be dropped if

$$d_i < l_j - h_i$$

where l_j denotes the ALAP-time of $v_j \in V$ and and h_i denotes the ASAP-time of $v_i \in V$.

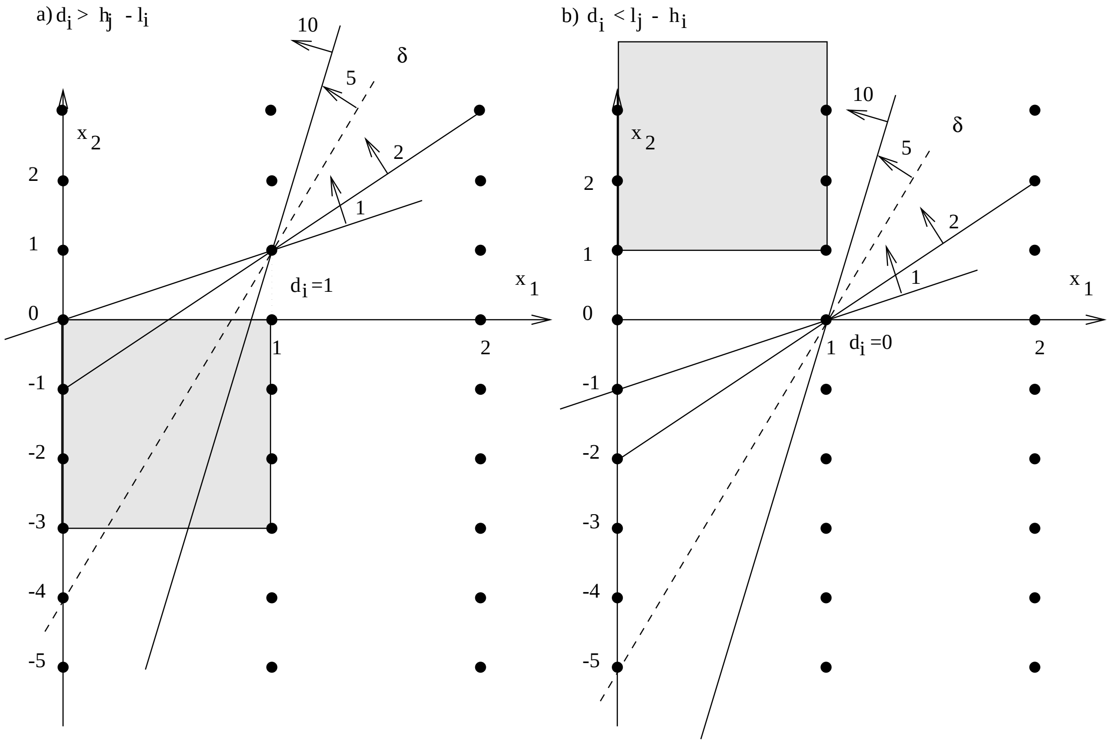

Figure 3.6: Determination of constants $\delta(v_i, v_j, k)$: special cases

Proof: 1) We show that the constraint may never be active in case $d_i < l_j - h_i$. 2) For all possible values of t_i, t_j, the difference of $t_j - t_i$ (due to ASAP- and ALAP-time limits) may only take values in the interval $[\min\{t_j - t_i\}, \cdots, \max\{t_j - t_i\}] = [l_j - h_i, \cdots, h_j - l_i]$. 3) Now, there does not exist a finite value for $\delta(v_i, v_j, k) \in \mathbf{Z}_{\geq 0}$, for which the hyperplane $t_j - t_i \geq d_i + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k)$ intersects the solution polytope corresponding to Fig. 3.5. Hence, the constraint is redundant and can be eliminated. 4) Because of Theorem 3.1 and Theorem 3.2, only those flow variables require an integrality constraint that appear in constraints of type S in (2.12) (and which are not redundant). Hence, the integrality constraint on $x_{v_i,v_j,k}$ can be dropped, too. $\square$

3.3.4 Addition of valid inequalities

Another step to improve the structure of an ILP-polytope is by determination of integral facets and by addition of corresponding inequalities to the model.

Even after carrying out the above optimization steps, there are conditions in our model that are mutually exclusive: Consider, for example, the two flow edges $(v_i, v_j) \in E_k$ and $(v_j, v_i) \in E_k$ that did not become victim of the elimination process in Lemma 3.1. Hence, neither $v_i \prec v_j$ nor $v_j \prec v_i$ holds. Therefore, there exist the two inequalities in (2.12) with

$$\begin{aligned} t_j - t_i &\geq d_i + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k) \\ t_i - t_j &\geq d_j + (x_{v_j,v_i,k} - 1) \cdot \delta(v_j, v_i, k) \end{aligned}$$

Obviously, both inequalities cannot be active simultaneously in any admissible solution (see also Fig. 3.7). Hence, we can add the following inequality constraint to our model:

Lemma 3.5 *The following inequality describes an integral facet of P_F:*

$$\begin{aligned} x_{v_i,v_j,k} \quad + \quad & x_{v_j,v_i,k} \leq 1 \\ & \forall i, j = 1, \cdots, |V| : i \neq j \ \wedge \ (v_i, v_j) \in E_k \ \wedge \ (v_j, v_i) \in E_k \end{aligned} \tag{3.26}$$

Proof: (1) (3.26) is a valid inequality constraint of P_F, because in no solution, both binary variables can be simultaneously one. (2) Eqn. (3.26) is an integral bounding hyperplane. □

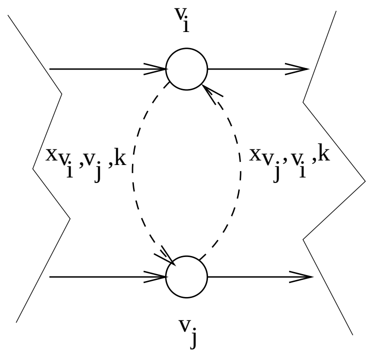

Figure 3.7: Addition of valid inequalities of the form $x_{v_i,v_j,k} + x_{v_j,v_i,k} \leq 1$

Lemma 3.6 *The following inequalities describe integral bounding hyperplane of P_F for a given latency bound $\bar{L}$:*

$$\begin{aligned} t_i &\geq l_i \quad \forall i = 1, \cdots, |V| \\ t_i &\leq h_i \quad \forall i = 1, \cdots, |V| \end{aligned}$$

where l_i denotes the ASAP-time and and h_i denotes the ALAP-time of node $v_i \in V$ for a given latency bound of $\bar{L}$.

Proof: Trivial. □

We conclude that a) the number of integer variables can be reduced from N_{ILP} by consideration of ASAP- and ALAP-times, b) that the constants $\delta(v_i, v_j, k)$ with $\delta(v_i, v_j, k) = d_i - l_j + h_i$ tighten the solution polytope, and c) that the addition of valid inequalities corresponding to Lemma 3.5 and Lemma 3.6 can further improve the structure of the solution polytope.

With these conclusions, we end our analysis and optimization of the flow-ILP. In the next section, we will introduce model extensions such as iterative scheduling problems (pipelined resources, functional pipelining).

Chapter 4

Pipelining

In this chapter, we would like to show that modules with pipelining capability can also be handled by our model. Furthermore, it is shown how to deal with iterative scheduling problems. The concepts of *functional pipelining* (concurrent planning of operations belonging to different iterations) and *loop folding* [HLH91] (functional pipelining with the additional property that start- and end times of operations may be in different iteration intervals) are also expressible in the flow model.

4.1 Pipelined resources

Modules with pipelining facility may accept input data and may produce output data in time intervals that are smaller than the execution times of corresponding operations.

Fig. 4.1 shows a schedule of a problem graph where multiplier resources are assumed to have pipelining capabilities. The execution time of a multiplication is two time steps, the execution time of an addition one time step. Obviously, a multiplier can start a new operation at each time step. However, pipelining may only be exploited by those operations between which there is no direct data dependency. Let $d_i = w((v_i, r_k))$ denote the execution time of a node v_i on a resource of type k and let k be a resource type with pipelining capability and so-called *data introduction interval* $u((v_i, r_k)) \in \mathbf{Z}_{\geq 0}$, $u((v_i, r_k)) \leq w((v_i, r_k))$. Then, a module of type k, when executing v_i, may start a new operation already after $u((v_i, r_k))$ time steps.

Theorem 4.1 (Flow-model with pipelined resources) *Given a specification with problem graph $G(V, E)$ and resource graph $G_R(V_R, E_R)$. Given further the ILP-model in Eqns. (2.8-2.14) with the inequalities in (2.12) replaced by the inequalities in Eqn. (4.1) with*

$$\begin{aligned} t_j - t_i \quad \geq \quad & u((v_i, r_k)) + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k) \\ & \forall (v_i, v_j) \in E_k, k = 1, \cdots, |V_T|, v_i, v_j \in V, i \neq j \end{aligned} \tag{4.1}$$

where $u((v_i, r_k))$ denotes the data introduction interval for planning v_i on resource type r_k. Then any admissible solution of the set of inequalities presents a solution to the scheduling problem with pipelined resources.

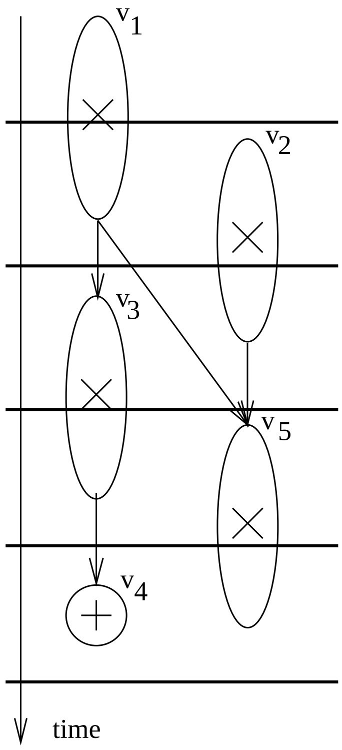

Figure 4.1: Schedule with multiplier modules having pipelining capability. Only one multiplier is required.

Proof: 1) The data dependency constraints (type D) in (2.10) are not changed, because in case of a data dependency, a successor operation must always wait until all predecessor operations are finished. 2) For the computation of the latency, the constraints $L \geq t_i + d_i$ for all $i = 1, \cdots, |V|$ are still valid, because the capability of pipelining does not change the execution time of an operation v_i. 3) In Eqn. (4.1), all edges between nodes $v_i, v_j \in V$ of E_k do appear except loops. In case the constraint is active ($x_{v_i,v_j,k} = 1$), we have $t_j - t_i \geq u((v_i, v_j))$. If now a) the edge (v_i, v_j) does not exist in the problem graph ($(v_i, v_j) \notin E$), $t_j - t_i \geq u((v_i, v_j))$ becomes the only constraint that sequentializes v_j after v_i and the soonest start time of v_j is $u((v_i, r_k))$ time steps after v_i on the same resource. If b) $(v_i, v_j) \in E$, then there exists a data dependency $t_j - t_i \geq d_i$ that is stronger or at least equally strong as the sequentialization constraint $t_j - t_i \geq u((v_i, r_k))$, because $u((v_i, r_k)) \leq d_i$. Hence, the sequentialization constraint is not active. 4) In case $x_{v_i,v_j,k} = 0$, the sequentialization constraint in Eqn. (4.1) is inactive. □

4.2 Functional pipelining and loop folding

In case of iterative scheduling, we consider algorithms of the form:

Definition 4.1 (Iterative Algorithms) *An* iterative algorithm *consists of a set of $|V|$ quantified, linearly indexed equations $S_i[n]$:*

$$S_1[n] \ldots S_i[n] \ldots S_V[n] \qquad \forall n \geq 0$$

Each equation $S_i[n]$ is of the form

$$x_i[n] = f_i(\ldots, x_j[n - s_{j,i}], \ldots)$$

where the index n denotes the iteration of the algorithm. The variables $x_i[n]$ are indexed and f_i denote arbitrary functions. Between the computation of variables $x_i[n]$ and $x_j[n]$ there may exist constant index displacements $s_{j,i} \in \mathbf{N}_0$.

The equations $S_i[n]$ are evaluated for all $n \in \mathbf{N}_0$. An index displacement $s_{j,i}$ has the meaning that the computation of x_i of index n may only start after the computation of $s_{j,i}$ of index $n - s_{j,i}$ has finished.

Example 4.1 *We consider the simple algorithm*

$$\begin{aligned} x_1[n] &= f_1(x_2[n-1], x_3[n-1]) & \forall n \geq 0 \\ x_2[n] &= f_2(x_1[n]) & \forall n \geq 0 \\ x_3[n] &= f_3(x_2[n-1]]) & \forall n \geq 0 \end{aligned}$$

Between the computation of variable x_1 and the variables x_2 and x_3 there is an index displacement of one iteration.

Again, the structure and data dependencies of an iterative algorithm may be captured by a problem graph that is extended by defining an index displacement function $s : E \to \mathbf{Z}$ that assigns each edge $(v_i, v_j) \in E$ of the problem graph the index displacement $s_{i,j} = s((v_i, v_j))$. In the following, we understand by an *iterative problem graph* a network $G = (V, E, s)$.

Example 4.2 *The iterative problem graph for the iterative algorithm introduced in example 4.1 is shown in Fig. 4.2a).*

Let *iteration interval* P denote the number of time steps between the begin of successing iterations. In the following, we assume that for all iterations $n \geq 0$, each node v_i is bound to one and the same resource (full static (= 1-periodic) binding).

The start time of the $n-$th iteration of each node v_i for a given iteration interval P is given by

$$t_i(n) = t_i + n \cdot P$$

with $t_i \in \mathbf{Z}_{\geq 0}$.

Example 4.3 *Fig. 4.2b) shows an* iterative schedule *with latency $L = 3$ and iteration interval $P = 3$ where each operation requires exactly one time step. For the start times of node iterations, we obtain:*

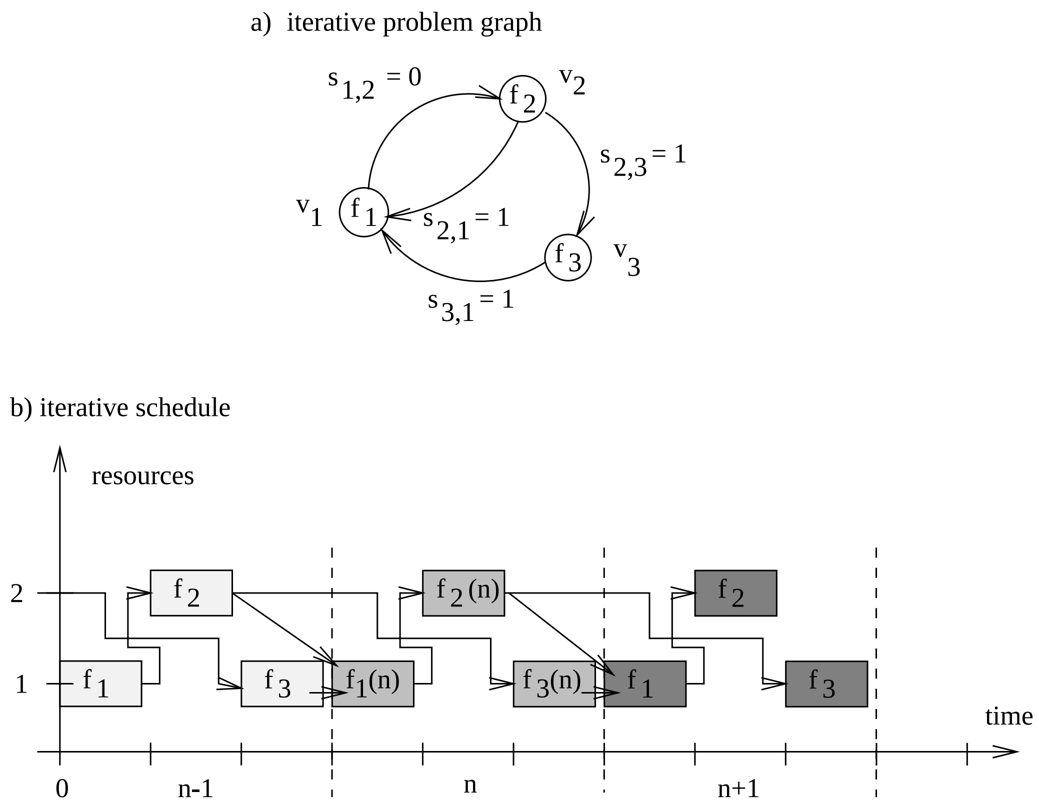

Figure 4.2: Iterative problem graph a) and iterative schedule b)

- $t_1(n) = 0 + n \cdot 3 \quad \forall n \geq 0$,
- $t_2(n) = 1 + n \cdot 3 \quad \forall n \geq 0$,
- $t_3(n) = 2 + n \cdot 3 \quad \forall n \geq 0$.

In general, $P < L$ holds. An iterative schedule with iteration interval $P = 2$ is shown in Fig. 4.3. Both displayed schedule are, however, non-overlapping.

Iterative scheduling may also be handled in the flow model as we will show next:

Theorem 4.2 (Functional pipelining and loop folding) *Given a specification with iterative problem graph $G(V, E, s)$ and resource graph $G_R(V_R, E_R)$. Without loss of generality, we assume a given allocation α: $\alpha(r_k) = 1 \; \forall k = 1, \cdots, |V_T|$. Given also, the ILP-model in Eqns. (2.8-2.14) where the data dependency constraints (type D) in Eqn. (2.10) are replaced by Eqn. (4.2) with*

$$t_j - t_i \geq (d_i - s_{i,j} \cdot P) \quad \forall (v_i, v_j) \in E \tag{4.2}$$

and the sequentialization constraints (type S) in Eqn. (2.12) replaced by Eqn. (4.3) with

$$z_j - z_i \;\; \geq \;\; (d_i + (g_i - g_j)P) + (x_{v_i,v_j,k} - 1) \cdot \delta(v_i, v_j, k) \tag{4.3}$$

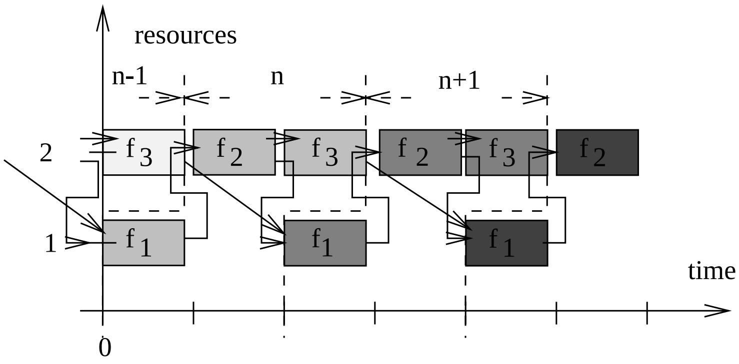

Figure 4.3: Iterative schedule with iteration interval $P = 2$

$$\forall (v_i, v_j) \in E_k, k = 1, \cdots, |V_T|, i \neq j$$
$$0 \leq z_i \leq P, t_i + d_i = z_i + g_i P + d_i \leq \bar{L}, g_i, g_j \in \mathbf{Z}_{\geq 0}$$

where

$$d_k = -P \quad \forall k = 1, \cdots, |V_T|.$$

Then any admissible solution to the corresponding set of inequalities represents a solution to the iterative scheduling problem with functional pipelining and loop folding.

Proof: Let t_i (stands for $t_i(0)$) denote the start time of v_i for the 0-th iteration.

1. The data dependencies in Eqn. (4.2) are now given as follows: If there is a data dependency from v_i to v_j over $s_{i,j}$ iterations, then obviously, $t_j(n) \geq t_i(n - s_{i,j}) + d_i$ must hold. I.e.,

$$t_j + nP \geq t_i + (n - s_{i,j})P + d_i$$

   (4.2) is exactly this inequality.

2. The sequentialization constraints in (4.3) require a deeper explanation: The difference to the former model is that now, also resource nodes $r_k \in V_T$ appear in the constraints and that the resource nodes also obtain a start time $t_k = z_k + g_k P$. To each resource node r_k, the execution time $d_k = -P$ is assigned, see Fig. 4.4.

   Now, a sequentialization constraint should be active if and only if a node v_i in the execution of the g_i-th iteration receives a resource from node v_j after executing its g_j-th iteration:

$$z_j + g_j P - z_i - g_i P \geq d_i + (x_{v_i, v_j, k} - 1) \cdot \delta(v_i, v_j, k)$$

   where $0 \leq z_i, z_j \leq P$ and $g_i, g_j \in \mathbf{Z}_{\geq 0}$ and the bounds $0 \leq z_i, z_i + g_i P + d_i \leq \bar{L}$ an g_i hold with $t_i = z_i + g_i \cdot P$. z_i presents the start time of an iteration of node

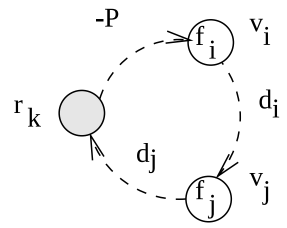

Figure 4.4: Modeling of cyclic resource flow in case of iterative scheduling

v_i during the interval $[0, \cdots, P]$. Now, we have to guarantee that a) there are still cyclic resource flows possible and b), what motivated the consideration of start times for resource nodes, that the sum of computation times of all nodes $v_i \in V$ that are bound to a resource during one iteration interval, must be smaller or equal to P. This is achieved by choosing $d_k = -P$ (see also in Fig. 4.4: There would not exist a) an admissible assignment of start times in a cyclic resource flow if $d_k \geq 0$ (positive cycle). b) $d_k = -P$ guarantees that the sum of all node computation times $v_i \in V$ that are executed on resource r_k is smaller or equal to P. Hence, in any admissible solution, a resource cannot be busy for more then P time steps.

3. Each resource node r_k is assigned the start time t_k. In case $\alpha(r_k) > 1$, it would not be possible to assign different start times to different instances of resources. Hence, loop folding would not be possible. Therefore, we require $\alpha(r_k) = 1$.

□

Example 4.4 *We consider the simple case of a graph $G(V, E, s)$ with one node v_i and one edge (v_i, v_i) (see Fig. 4.5) and $s_{i,i} = 1$. The corresponding iterative algorithm is:*

$$x_i[n] = x_i[n-1] \quad \forall n \geq 0$$

There exists one resource type r_k. The execution times of v_i on r_k is $d_i = w((v_i, r_k)) = 2$ time steps. The latency bound is $\bar{L} = 7$ time steps.

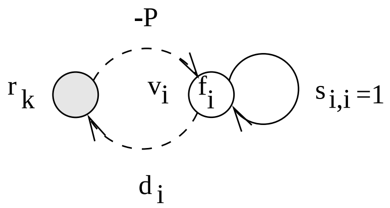

Figure 4.5: Example of an iterative problem graph and corresponding flow graph for explaining the flow-model in case of functional pipelining and loop folding

Now, let the iteration interval be

- $P = 1$*: In this case, there is no way to satisfy the data dependency constraint* $t_i - t_i = 0 \geq d_i - P$ *in Eqn. (4.2). The minimal iteration interval such that this constraint is satisfiable is* $P = d_i = 2$.

- $P = 2$*: now, the data dependency constraint is satisfiable. We consider now the two sequentialization constraints in Eqn. (4.3):*

$$z_i - z_k \geq -P + (g_k - g_i)P$$

$$z_k - z_i \geq 2 + (g_i - g_k)P$$

Both constraints are active, because the flow may only come from r_k *and must return to* r_k*. Furthermore, we have:* $0 \leq z_i, z_i + g_iP \leq 5$ *and* $0 \leq z_k, z_k + (g_k - 1)P \leq 7$*. Now, we obtain a solution with* $z_i = 0$*. For* $g_i = g_k = 0$*, we obtain* $z_k = P = 2$*. Hence,* $0 \leq g_i \leq 2$*, i.e.,* v_i *may execute three iterations for a latency bound of* $\bar{L} = 7$ *with the start times* $t_i(n) = n2$.

- $\bar{L} = 2, P = 2$*: Now, the constraints become* $0 \leq z_i, z_i + g_i2 \leq 0$ *and* $0 \leq z_k, z_k + (g_k - 1)P \leq 2$*. Hence, only the solution* $z_i = g_i = 0$ *is possible. With* $g_i = g_k = 0$*, we obtain* $z_k = t_k = P = 2$.

4.3 Computation of ASAP- and ALAP- times in case of iterative scheduling

First, consider the case where there is no module selection. Then the execution times of nodes $v_i \in V$ are constant (d_i).

From the data dependency constraints in (4.2), we can compute the earliest start time of a node v_i with $t_i(n) = z_i + n \cdot P$ by computing a longest path problem on a weight transformed graph.

Lemma 4.1 (ASAP- ALAP-times for iterative problem graphs) *Given an iterative problem graph* $G(V, E, s)$ *as well as the computation times* d_i *of nodes* $v_i \in V$ *and an iteration interval* P*. Then, we obtain the*

- *ASAP-times* l_i *for iterative scheduling*
  as time time potentials of the longest path in the network $G^S(V, E^S, d^S)$ *with* $E^S = E$, $d^S : E^S \to \mathbf{Z}$ *and weights*

$$d^S((v_i, v_j)) = d_i - s_{i,j}P$$

  and initial potentials zero.

- *ALAP-times h_i for iterative scheduling as the time potentials of the shortest path in the network $G^L(V, E^L, d^L)$ with $E^L = \mathrm{rev}(E)$ [1] , $d^L : E^L \rightarrow \mathbf{Z}$ and weights*

$$d^L((v_i, v_j)) = -d_i + s_{i,j}P$$

and initial potentials $\bar{L} - d_i$.

Proof: 1) From 4.2 the data dependencies for iterative problems become $t_j - t_i \geq d_i - s_{i,j}P = d^S((v_i, v_j))$. The computation of the longest path on the network G^S hence minimizes the start times t_i while satisfying the data dependency constraints. These are the ASAP- times t_i. 2) It is known that the ALAP-times may be computed on a graph with reversed edges and negated edge weights. □

Example 4.5 *We compute ASAP- and ALAP-times for the problem graph shown in Fig. 4.2a). In Fig. 4.6a) the network with weights $d_i - s_{i,j} \cdot P$ is shown. The longest path determines the ASAP- times. The iteration interval has been chosen as $P = \bar{L} = 3$. The computation of ALAP-times corresponding to Lemma 4.1 is shown in Fig. 4.6b) (Attention: Computation of the shortest path). In order to be able to apply an algorithm such as the Bellman-Ford- algorithm, a fictive node v_0 may be introduced as a source node In case of ASAP, this node is initialized with $\tau(v_0) = 0$, in case of ALAP, it is initialized with $\tau(v_0) = \bar{L}$. The weights of the edges (v_0, v_i) are chosen to be zero in case of ASAP and $-d_i$ in case of ALAP.*

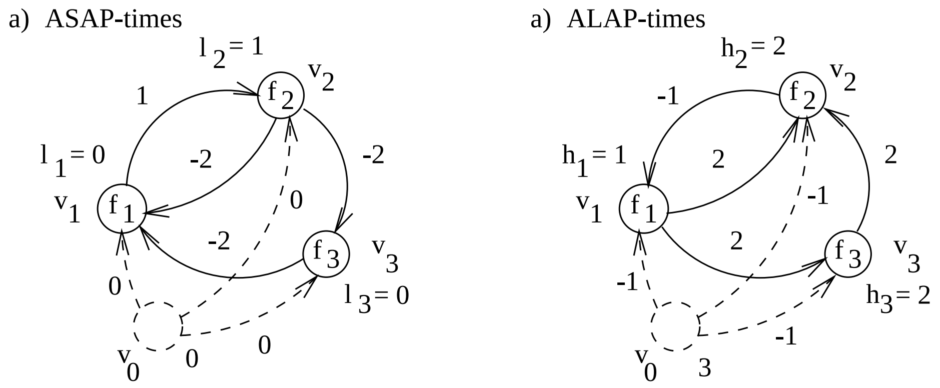

Figure 4.6: Computation of ASAP- and ALAP-times in case of iterative scheduling for the problem graph in Fig. 4.2a)

Now, in case of module selection, the execution times of nodes $v_i \in V$ are dependent on the resource type to which v_i is bound. In order not to exclude any possible solutions for all possible bindings, the ASAP- ALAP- interval must be computed with the execution times: $d_i = \min_{(v_i, r_k) \in E_R} w((v_i, r_k))$, because in this case the ASAP-times will be the smallest possible and the ALAP-times will be the largest possible for any binding and computed corresponding to Lemma 4.1.

[1] $\mathrm{rev}(E)$ denotes the operator that, given a set of edges E, returns a set of edges with same cardinality and each edge reversed.

4.4 Iterative Scheduling with fully-static bindings

We have shown so far that pipelined resources as well as functional pipelining and loop folding can be easily incorporated in our model. We would like to point out one more advantage of the flow-model over the ALPS-model for functional pipelining and loop folding [HLH91]:

In the ALPS-model there is no explicit binding of operations to instances of resources. For non-iterative scheduling problems, a valid fully-static binding (each iteration of a node is bound to the same resource instance; so-called 1-periodic binding) can be found in polynomial-time (e.g., leftedge algorithm). In case of iterative scheduling, this is no longer possible, as shows the following example:

Example 4.6 *Given the problem graph G in Fig. 4.7a) and the resource graph G_R in Fig. 4.7b). Let the execution time of each operation be two time steps. There is only one resource type (e.g., processor).*

The longest path in the problem graph is $\bar{L} = 6$ time steps. We formulate the resource constraint $\alpha(r_1) = 2$.

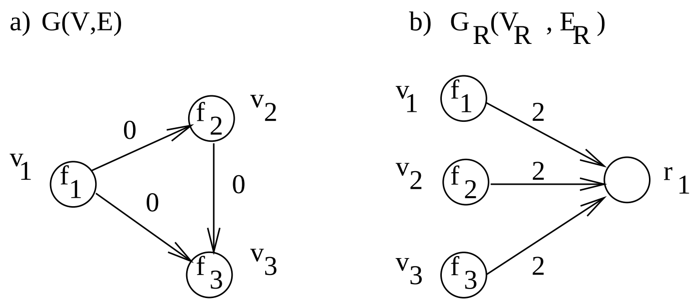

Figure 4.7: Problem graph a) and resource graph b)

Fig. 4.8a) shows a schedule that has been found for a minimal iteration interval of $P = 3$ for two available resources. In the flow-model, $P = 4$ is the smallest iteration interval for which an admissible schedule is found. This is shown in Fig. 4.8b).

One can easily see that the schedule shown in Fig. 4.8a) is a schedule with cyclo-static binding, *because it is assumed that an operation may be bound to different resources in different iterations. E.g., consider v_1: In even iterations, it is bound to resource* 1, *in odd iterations, it is bound to resource* 2. *In general, a minimal iteration interval is only achievable if cyclo-static bindings are allowed. However, the assumption that different iterations of operations may be bound to different resources is often not realistic. Hence, the binding model should be incorporated in the scheduling model. This is done in the flow-model.*

In existing models, a fully-static binding might be achieved by representing a resource type k with allocation $\alpha(r_k)$ by $\alpha(r_k)$ resource types of allocation one in order to avoid cyclo-static bindings.

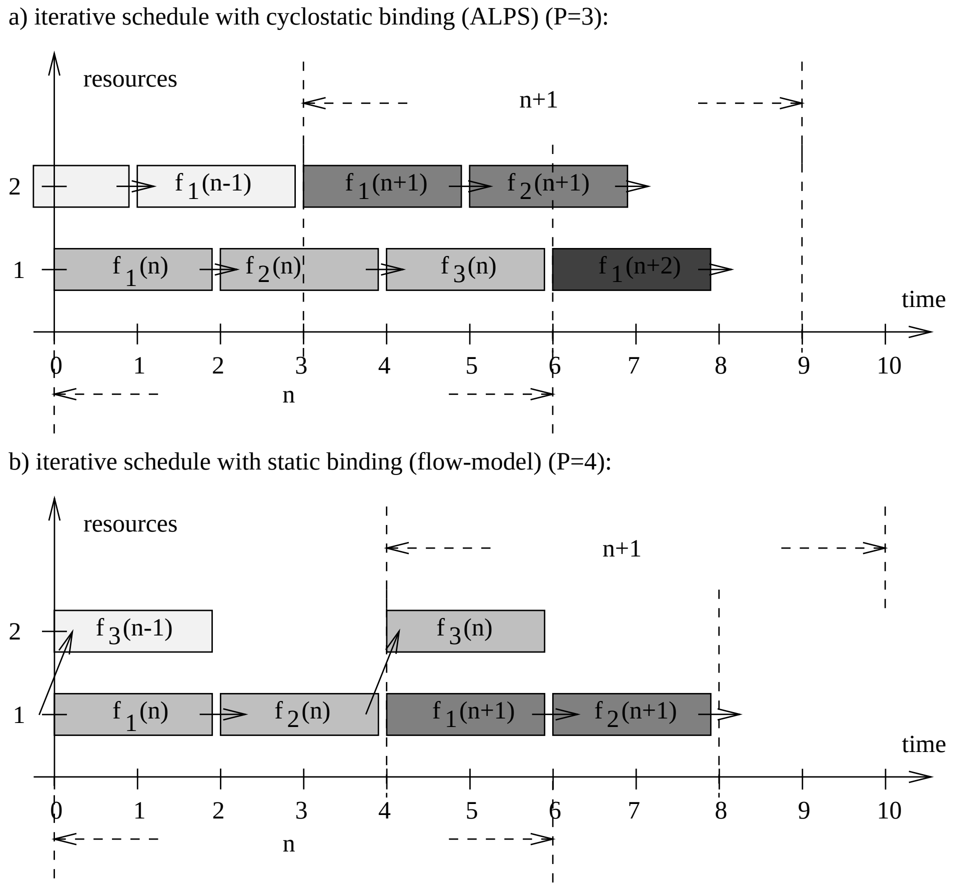

Figure 4.8: Iterative (cyclo-static) schedule with minimal iteration interval $P = 3$ found in ALPS a) and iterative (fully-static (1-periodic) binding) with optimal iteration interval $P = 4$ found with the flow-model. The solution in a) is not feasible when fully-static bindings are required.

4.5 Optimization of the flow-model for iterative scheduling problems

In principle, the results of 3.1 and 3.2 may be applied directly for the ASAP- and ALAP-times computed corresponding to Lemma 4.1.

Also, the procedure for optimizing the constants $\delta(v_i, v_j, k)$ corresponding to Lemma 3.3 as well as the addition of valid inequalities (Lemma 3.5 and Lemma 3.6) may be applied to improve the model.

Hence, all results obtained so far remain applicable.

Finally, we would like to add some remarks on how to efficiently solve the flow-model.

Chapter 5

Solution strategies

In the following, we are interested in solution strategies for solving ILPs, i.e., Branch-and-Bound- (BAB-) algorithms which are mostly used in packages for solving ILPs. A BAB-algorithm solves an ILP by solving a number of LP-relaxations.

First, we would like to show that a BAB-algorithm is suitable for efficiently solving the flow-ILP. It is known that *Lagrange- Relaxations* may provide better lower bounds than LP-relaxations. Although Lagrange-relaxations require in general a higher computational effort, their computation might have a positive result of being able to speed up the solution process by pruning a larger number of subtrees as compared to LP-relaxations. Hence, the number of nodes that have to be computed might be considerably smaller in case of good relaxation method is chosen.

Therefore, we would like to analyze the flow-model again under the aspect of finding the best relaxation method.

5.1 Relaxation methods

First, we neglect again the problem of module selection. Hence, the flow-ILP can be expressed as follows:

$$z_{ILP} = \min\{c^T(T^T,\ X_{\bar{S}}^T,\ X_S^T)^T \mid C_S^T T - M_S X_S \geq m_s,\ (T^T,\ X_{\bar{S}}^T,\ X_S^T)^T \in P'_F\} \tag{5.1}$$

with

$$P_{F'} = \{x \in \mathbf{Z}_{\geq 0}^{|V|+|E_{ges}|} \mid C_{\bar{S}} X_{\bar{S}} + C_S X_S = 0, D_{\bar{S}} X_{\bar{S}} + D_S X_S = 1, C^T T \geq d\} \tag{5.2}$$

The *Lagrange-Relaxation* [NW88] considers the problem

$$z_{LR(\lambda)} = \min\{c^T \cdot (T^T,\ X_{\bar{S}}^T,\ X_S^T)^T + \lambda(-C_S^T T + M_S X_S + m_s) |\ (T^T,\ X_{\bar{S}}^T,\ X_S^T)^T \in P_{F'}\} \tag{5.3}$$

with $\lambda \in \mathbf{R}_{\geq 0}^{N_{ILP}}$.

Interestingly, this formulation does not contain the sequentialization constraints in (3.10). Instead, they have been moved into the objective function with a so-called ,,penalty-term" $\lambda(-C_S^T T + M_S X_S + m_s)$. Because of $\lambda \geq 0$, violations of the sequentialization constraints will add a positive number to the objective function. It can be seen that the constraints will be satisfied, if λ becomes sufficiently large.

Now, (5.3) is a relaxation of (5.1) for all values of $\lambda \geq 0$. Hence, $z_{LR(\lambda)} \leq z_{ILP}$ holds for all $\lambda \geq 0$. Therefore, the best relaxation is

$$z_{LR(\lambda^*)} = \max_{\lambda \geq 0} z_{LR(\lambda)} \tag{5.4}$$

(5.4) is called *Lagrangian dual* of (5.1) *with respect to the constraints* $C_S^T T - M_S X_S \geq m_s$ (constraints of type S). Obviously, $z_{LR(\lambda^*)}$ is the best lower bound that may be obtained by relaxation of the sequentialization constraints.

Finally, we would like to show that z_{LP}, the bound obtained from the LP-relaxation is equally good as the best Lagrange-relaxation. This means: $z_{LP} = z_{LR(\lambda^*)}$. Fur this purpose, we need to compute $z_{LR(\lambda^*)}$.

Corresponding to Nemhauser [NW88], Theorem 6.2, Chapter II.3, the Lagrangian dual may be computed by solving the following linear program:

$$\begin{aligned} z_{LR(\lambda^*)} = \min\{c^T \cdot (T^T,\ X_{\bar{S}}^T,\ X_S^T)^T | \quad & C_S^T T - M_S X_S \geq m_s), \\ & (T^T,\ X_{\bar{S}}^T,\ X_S^T)^T \in \text{conv}(P_{F'})\} \end{aligned}$$

Hence, $z_{LP} = z_{LR(\lambda^*)}$ (the polytope $P_{F'}$ obtained by omitting the sequentialization constraints is integral as has been proven in Theorem 3.4) and therefore, the LP-relaxation is equally good as the best Lagrange-Relaxation with respect to the sequentialization constraints.

5.2 Determination of „good“ initial solutions and adaption of the model

We have seen that the approximation of P_I is better, the smaller the latency bound and hence, the mobility of the nodes is. Therefore, the following iterative solution strategy lies at hand:

Here: Latency minimization under resource constraints

1. Determination of initial solution using a heuristic like list scheduling or Force-Directed- Scheduling. From the solution, a binding is computed (e.g., leftedge-algorithm) and from these, the corresponding flow.

2. ILP-model and optimization of constraints corresponding to Lemma 3.1, 3.2 and 3.4. Determination of $\delta(v_i, v_j, k)$ for an actual latency bound $\bar{L}$ corresponding to Lemma 3.3. Addition of valid inequalities corresponding to Lemma 3.5 and 3.6.

3. Iterative solution with BAB-algorithm: If an integer solution is found with $\bar{L}^* < \bar{L}$ and this is a global optimum, goto step 4. Else (there exist still uncomputed subtrees) that could not be pruned: Let $\bar{L} = \bar{L}^*$ and go back to step 2.

4. End.

Experiments show that the step to go back to step two from step 3 should not always be taken but only if the number of not investigated subtrees becomes greater then a certain bound, because elsewise the computational effort in step 2 is not justified.

One can imagine that this procedure tries to find better approximations of P_F^* to P_I (shrinking the polytope) on its way to find a global optimal solution.

Chapter 6

Examples

Now, the design methodology will be tested using some concrete examples. The first example is a numerical method for solving a partial differential equation. After that a video-codec using the H.261 norm is introduced. Using this example, we would like to show that in case of nodes with high execution times, existing methods fail whereas our model can be solved equally fast than with small execution times.

As a further example from the domain of high-level synthesis, we choose an elliptical wave filter (EWF). We compare the solution times for the flow model with those of existing ILP-models for different objective functions.

The commercial ILP-solver CPLEX [CPL89] running on a SUN Sparc 4-20 has been used for solving the ILP models.

6.1 PDE-Integrator

The following program is a VHDL-specification for solving a partial differential equation of the form $y'' + 3xy' + 3y = 0$ in the interval $[x, a]$ with stepsize dx and initial values $y(0) = y$, $y'(0) = u$ using Euler's method.

```
ENTITY dgl IS
  PORT(x_in,y_in,u_in,dx_in,a_in: IN REAL;
       activate: IN BIT; y_out: OUT REAL);
END dgl;

ARCHITECTURE behavioral OF dgl IS BEGIN
  PROCESS (activate)
    VARIABLE x, y, u, dx, a, x1, u1, y1: REAL;
  BEGIN
    x := x_in; y := y_in; u := u_in; dx := dx_in; a := a_in;
    LOOP
      x1 := x + dx;
      u1 := u - (3 * x * u * dx) - (3 * y * dx);
      y1 := y + (u * dx);
      x := x1; u := u1; y:= y1;
      EXIT WHEN x1 > a;
```

```
    END LOOP;
    y_out <= y;
  END PROCESS;
END behavioral;
```

A program graph for the above specification is shown in Fig. 6.1.

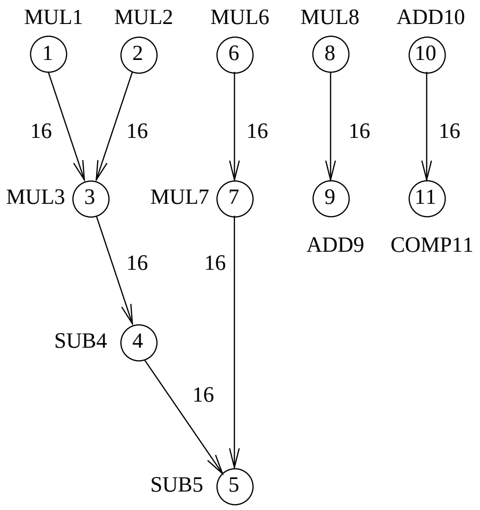

Figure 6.1: Problem graph of the PDE-integrator

In Fig. 6.2, a solution of the flow problem for latency minimization under resource constraints ($\alpha = (2, 1)$) is shown. The displayed flow shows in which order a resources visits the nodes of the problem graph. The solution of the corresponding flow-ILP provided a minimal latency of $L = 8$ time steps. The CPU-time for solving the ILP was 2.03 seconds on a SUN 4-20.

In the following, we compare the solution times of the ALPS-model with the flow model for latency and cost minimization for different constraints and different examples.

6.2 Video-Codec

Fig. 6.3 shows a block diagram of the behavior of a hybrid image sequence coder/decoder. The purpose of the coder is to compress video images using the H.261 standard. Therein, transformative and predictive coding techniques are unified. The compression factor can be further increased by a predictive method for motion estimation: blocks inside a frame are predicted from blocks of previous images. The blocks of the behavioral description in Fig. 6.3 possess the granularity of algorithms. However, this description contains no information corresponding to timing, architecture, and mapping of blocks onto an architecture.

Fig. 6.4 shows a problem graph G of the video-codec in Fig. 6.3. The problem graph contains a subgraph for the coder and one subgraph for the decoder.

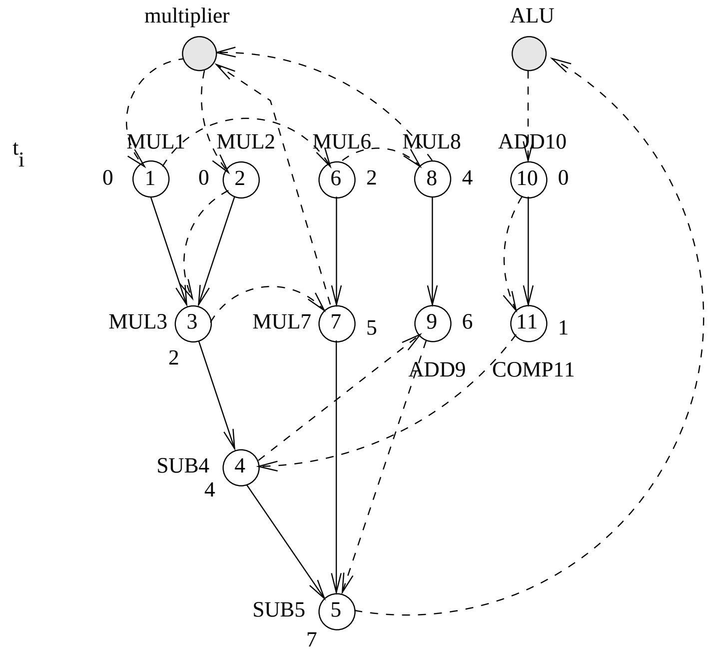

Figure 6.2: Latency minimization under resource constraints of 2 multipliers and one ALU. Shown are the flow edges that have been active in the solution. Also shown are the start times of the nodes.

We would like to map the graph onto an architecture that contains the components shown in Fig. 6.5. The architecture consists of a shared bus, several memory modules, two programmable RISC- processors, and two dedicated hardware modules: a block matching module (BMM) that is used for motion estimation and a module DCTM for computing DCT/IDCT- computations.

The allowed mappings and costs of the individual components are shown in Fig. 6.6.

Fig. 6.6a) shows that all nodes can be computed on the two RISC-processors. For ease of representation, these edges are simply left out. Node v_9 (block matching) may only be computed on resource BMM, DCT- and IDCT operations may be computed on the DCTM-module.

6.2.1 Latency minimization video-codec in the ALPS-model

In order to compare the performance of our model with that of the ALPS- model, we will first present the results obtained for latency minimization using the ALPS-model. Because ASAP- and ALAP-time optimization can heavily reduce the number of variables in the model, we present a table with results for different latency bounds $\bar{L}$, as well as for different resource allocations, namely

- 1 BMM-, 1 RISC-, 1 DCTM- module,

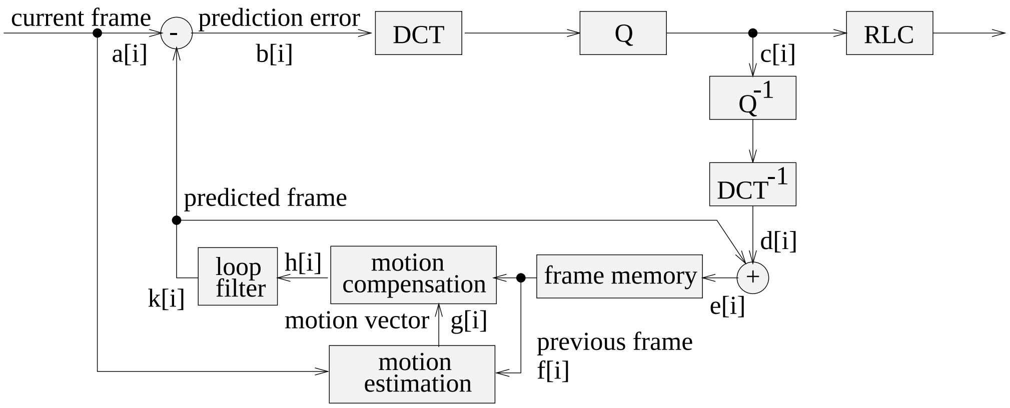

Figure 6.3: block diagram of a video-codec (H.261)

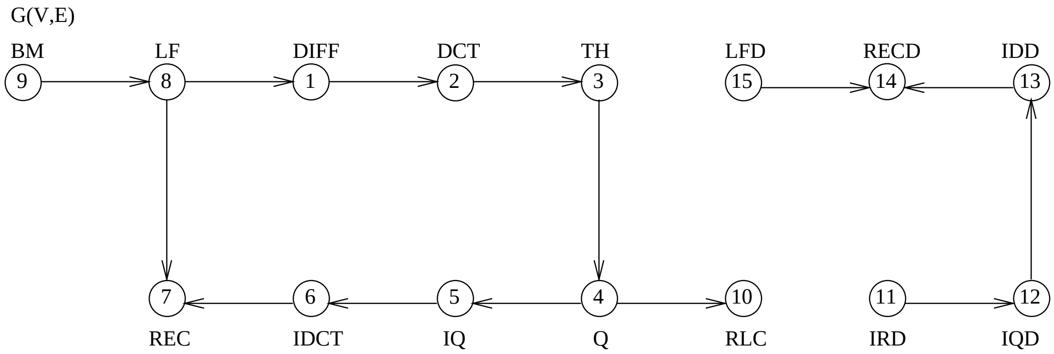

Figure 6.4: Problem graph G of the video-codec in Fig. 6.3

- 1 BMM-, 2 RISC-, 1 DCTM- module,
- 4 BMM-, 4 RISC-, 4 DCTM- modules.

In the following tables, the latency bound $\bar{L}$, the number of computed LP-relaxations (nodes), the first found valid (integral) solution, the number of nodes and the elapsed CPU-time for finding an optimal solution as well as the final solution are listed. The last entry represents the latency of the final solution.

The minimal latency without resource constraints is $L_{min} = 53$ time steps.

In Table 6.1, one can recognize that for the resource constraints $\alpha = (1, 2, 1)$ and $\alpha = (4, 4, 4)$ the solver always found the optimal solution without having to do even one branching step using the ALPS-model. The reason is that in these cases, the resource constraints never become active when solving a relaxation step for latency minimization.

Now, we modify the example by multiplying the execution times of nodes by a factor of ten. By doing this, we would like to show that in this case, the solution time

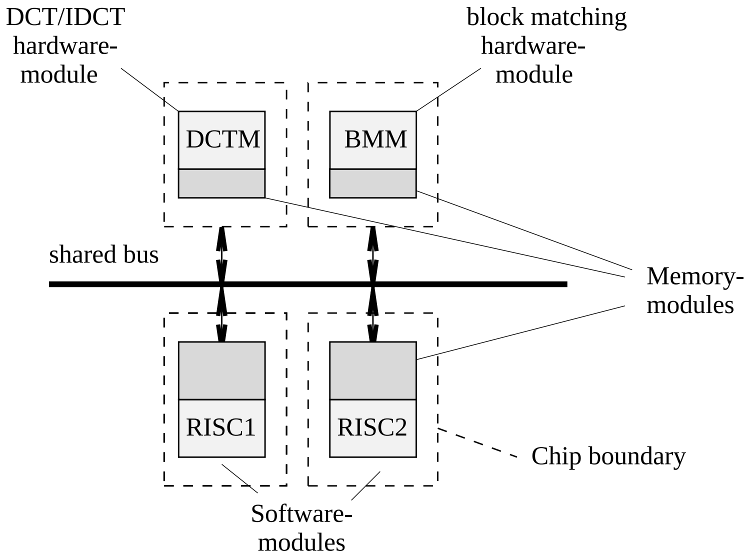

Figure 6.5: Architecture of the video-codec.

increases drastically, because in the ALPS-model, the number of variables is proportional to the latency. The results of this series of measurements are summarized in Table 6.2.

6.2.2 Resource minimization video-codec in the ALPS-model

In Tables 6.3 and 6.4, the given latency bound $\bar{L}$, the number of computed LP-relaxations (nodes) for finding a first valid (integral) solution, the number of nodes and elapsed CPU-time for finding the final solution, and the final solution are shown. The last entry presents the found optimal allocation α (order: BMM, RISC, DCTM). In the objective function, the costs $c(BMM) = c(RISC) = c(DCTM) = 1$ have been chosen.

Again, the example is modified by increasing all computation times of nodes by a factor of 10. The corresponding results are shown in Table 6.4.

6.2.3 Latency minimization video-codec in the flow-model

The same experiments are carried out now using the flow-model.

In Tables 6.5 and 6.6 again, the given latency bound, the number of computed LP-relaxations (nodes) for finding a first valid (integral) solution, the number of nodes and the elapsed CPU-time for finding the optimal solution and the final solution are shown. The last entry presents the latency L of the final solution.

Table 6.5 shows that for an allocation of $\alpha = (1, 1, 1)$ no solution could be found in an acceptable amount of CPU-time (here: 1h).

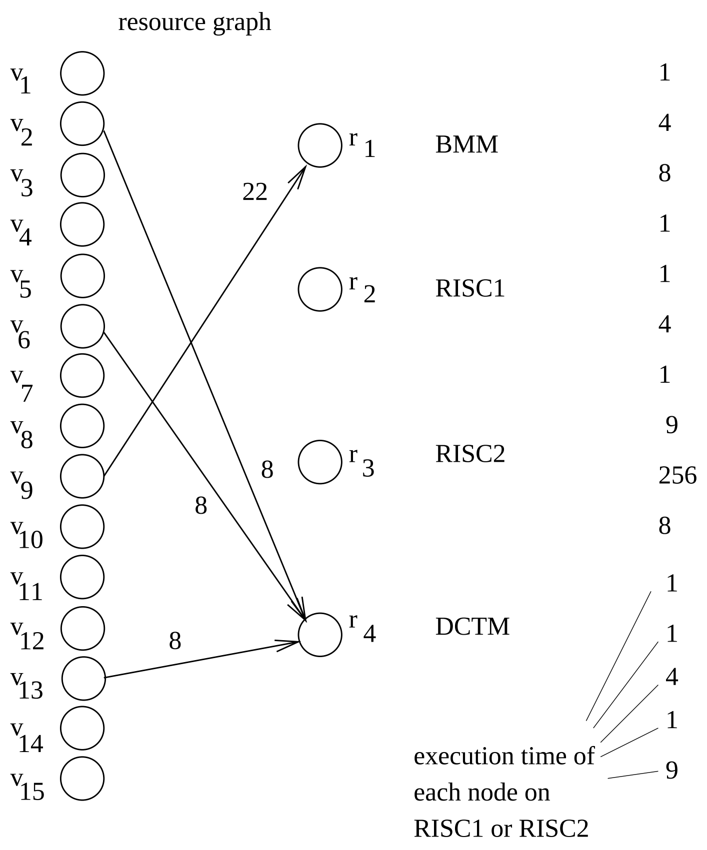

Figure 6.6: Resource graph of the video-codec

For $\alpha = (1, 2, 1)$ and $\alpha = (4, 4, 4)$, however, the solution time of the flow-model is shorter then for the ALPS-model.

Now again, we modify the example by increasing all computation times of nodes by a factor of 10. We will see that, contrary to the ALPS-model, the solution time does not increase. This result can be depicted in Table 6.6.

6.2.4 Resource minimization video-codec in the flow-model

A number of measurements has been made for different latency bounds $\bar{L}$.

In Tables 6.7 and 6.8, the given latency bound $\bar{L}$, the number of computed LP-relaxations (nodes) for finding the first valid (integral) solution, the number of nodes and elapsed CPU-time for finding the optimal solution, and the final solution are shown. The last entry presents the found optimal allocation α (order: BMM, RISC, DCTM). The cost parameters in the objective functions are $c(BMM) = c(RISC) = c(DCTM) = 1$.

The results of resource minimization using the flow-model are shown in Table 6.7 for the video-codec.

Again, the example is modified by increasing all computation times by a factor

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
55	0 / 0.46s	0 / 0.46s	0 / 0.46s	55
60	8 / 1.52s	8 / 1.52s	8 / 1.52s	55
65	8 / 0.68s	8 / 0.68s	8 / 0.68s	55
70	53 / 3.63s	53 / 3.63s	53 / 3.63s	55
100	6 / 2.00s	6 / 2.00s	6 / 2.00s	55
200	11 / 5.35s	11 / 5.35s	11 / 5.35s	55

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
53	0 / 0.23s	0 / 0.23s	0 / 0.23s	53
54	0 / 0.27s	0 / 0.27s	0 / 0.27s	53
60	0 / 0.40s	0 / 0.40s	0 / 0.40s	53
100	0 / 0.86s	0 / 0.86s	0 / 0.86s	53
200	0 / 2.29s	0 / 2.29s	0 / 2.29s	53

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
53	0 / 0.23s	0 / 0.23s	0 / 0.23s	53
54	0 / 0.27s	0 / 0.27s	0 / 0.27s	53
60	0 / 0.40s	0 / 0.40s	0 / 0.40s	53
100	0 / 0.86s	0 / 0.86s	0 / 0.86s	53
200	0 / 2.29s	0 / 2.29s	0 / 2.29s	53

Table 6.1: latency minimization video-codec in the ALPS-model with resource constraints $\alpha = (1,1,1), (1,2,1)$ and $(4,4,4)$.

of 10. The results are summarized in Table 6.8.

6.3 Elliptical wave filter

As another benchmark problem from the domain of high-level synthesis, we consider the dependence graph of an elliptical wave filter shown in Fig. 6.7 (acyclic version). The figure also shows a schedule optimized using the flow model for resource minimization under latency constraints ($\bar{L} = 21$). One multiplier and two adders are used.

In the following, we compare the solution times for the different ILP models for latency- and resource minimization and different constraints.

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
550	no sol. after 1h			–
600				–
650				–
700				–
1000	CPLEX: *more than 240'000 coefficients*			–
2000	CPLEX: *more than 240'000 coefficients*			–

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
530	1 / 7.30s	1 / 7.30s	1 / 7.30s	530
540	0 / 7.53s	0 / 7.53s	0 / 7.53s	530
600	0 / 13.40s	0 / 13.40s	0 / 13.40s	530
1000	CPLEX: *more than 240'000 coefficients*			–
2000	CPLEX: *more than 240'000 coefficients*			–

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
530	1 / 7.10s	1 / 7.10s	1 / 7.10s	530
540	0 / 7.55s	0 / 7.55s	0 / 7.55s	530
600	0 / 13.40s	0 / 13.40s	0 / 13.40s	530
1000	CPLEX: *more than 240'000 coefficients*			–
2000	CPLEX: *more than 240'000 coefficients*			–

Table 6.2: Latency minimization video-codec in the ALPS-model with resource constraints $\alpha = (1,1,1), (1,2,1)$ and $(4,4,4)$ with nodes of 10 times increased computation times.

6.3.1 Latency minimization EWF in the ALPS-model

First, we present results for latency minimization in the ALPS- model. The considered allocations were

- 1 adder, 1 multiplier- module,
- 2 adder, 1 multiplier- module,
- 2 adder, 2 multiplier- modules,
- 4 adder, 4 multiplier- modules.

In each of the following tables, the given latency bound $\bar{L}$, the number of computed LP-relaxations (nodes) for finding the first valid (integral) solution, the number of nodes and elapsed CPU-time for finding an optimal solution and the final

$\bar{L}$	Solutions (#nodes/CPU- time)			allocation
	1. int. sol.	best int. sol.	final sol.	α
53	0 / 0.14s	0 / 0.14s	0 / 0.14s	1 2 0
54	0 / 0.15s	0 / 0.15s	0 / 0.15s	1 2 0
58	16 / 1.20s	16 / 1.20s	16 / 1.20s	1 2 0
59	4 / 1.97s	4 / 1.97s	4 / 1.97s	1 1 0
60	19 / –	34 / 4.95s	34 / 4.95s	1 1 0
100	No solution after 1h			–
200	no solution after 1h			–

Table 6.3: Resource minimization video-codec in the ALPS-model.

$\bar{L}$	Solutions (#nodes/CPU- time)			allocation
	1. int. sol.	best int. sol.	final sol.	α
530	0 / 4.16s	0 / 4.16s	0 / 4.16s	1 2 0
540	0 / 4.49s	0 / 4.49s	0 / 4.49s	1 2 0
590	1 / 145.3s	1 / 145.3s	1 / 145.3s	1 1 0
600	3 / 628.3s	3 / 628.3s	3 / 628.3s	1 1 0
1000	CPLEX: *more than 240'000 coefficients*			–
2000	CPLEX: *more than 240'000 coefficients*			–

Table 6.4: Resource minimization video-codec in the ALPS-model with node execution times increased by a factor of 10.

solution are displayed. The last entry presents the minimal latency L that is not necessarily optimal, if the solver has not terminated after 1 CPU h.

The minimal latency in case of no resource constraints is $L_{min} = 16$ time steps.

Obviously, the solver always found an optimal solution for the resource constraints $\alpha = (2, 2)$ and $\alpha = (4, 4)$, for $\alpha = (4, 4)$ even without having to carry out a single branching step.

6.3.2 Resource minimization EWF in the ALPS-model

We will present a measurement for different values of latency bounds $\bar{L}$. In Table 6.11, the latency bound $\bar{L}$, the number of computed LP-relaxations for finding the first valid (integral) solution, the number of nodes and elapsed CPU-time for finding an optimal solution and for the final solution are displayed. The last entry presents the found minimal allocation α (order: adder, multiplier). The cost weights in the objective function are $c(adder) = c(multiplier) = 1$.

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
55	17548 /–	17548 / –	17548 / 203.55s	55
60	1267 / $L = 60$	No better sol. after 1h		60
65	1940 / $L = 65$	No better sol. after 1h		65
70	2490 / $L = 67$	No better sol. after 1h		67
100	187 / $L = 73$	No better sol. after 1h		73
200	25 / $L = 73$	No better sol. after 1h		73

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
53	0 / 0.10s	0 / 0.10s	0 / 0.10s	53
54	18 / –	33 / 0.28s	33 / 0.28s	53
60	170/ –	2261 / 19.56s	2261 / 19.56s	53
100	36 / –	433 / 3.09s	433 / 3.09s	53
200	52 / –	1610 / 12.51s	1610 / 12.51s	53

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
53	5 / 0.09s	5 / 0.09s	5 / 0.09s	53
54	0 / 0.08s	0 / 0.08s	0 / 0.08s	53
60	3 / –	9 / 0.12s	9 / 0.12s	53
100	0 / 0.09s	0 / 0.09s	0 / 0.09s	53
200	0 / 0.09s	0 / 0.09s	0 / 0.09s	53

Table 6.5: Latency minimization in the flow-model for the resource constraints $\alpha = (1,1,1), (1,2,1)$ and $(4,4,4)$.

6.3.3 Latency minimization EWF in the flow model

The results for the EWF using the flow-model are displayed in Table 6.10 for latency minimization.

6.3.4 Resource minimization EWF in the flow model

In Table 6.12, the given latency bound $\bar{L}$, the number of the computed LP-relaxations (nodes) for finding the first valid (integral) solution, the number of nodes and elapsed CPU-time for finding an optimal solution, and the final solution are displayed. The last entry in the table is the minimal allocation α found (order: adder, multiplier). The costs in the objective function have been chosen as $c(Addierer) = c(Multiplizierer) = 1$.

Table 6.12 shows that for small latency bounds, an optimal allocation was not found within 1 CPU h. However, for large latency bounds, the optimal solution $\alpha = (1,1)$ was found. Furthermore, a comparison with the ALPS-model (Table

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
550	17281 / 196.69s	17281 / 196.69s	17281 / 196.69s	550
600	13187 / $L = 560$	No better sol. after 1h		560
650	1953 / $L = 650$	No better sol. after 1h		560
700	1621 / $L = 620$	No better sol. after 1h		620
1000	50 / $L = 750$	No better sol. after 1h		750
2000	25 / $L = 730$	No better sol. after 1h		730

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
530	0 / 0.08s	0 / 0.08s	0 / 0.08s	530
540	76 / –	164 / 1.74s	164 / 1.74s	530
600	396 / –	768 / 6.59s	768 / 6.59s	530
1000	61 / –	11468 / 90.90s	11468 / 90.90s	530
2000	28 / –	2748 / 21.94s	2748 / 21.94	530

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
530	5 / 0.12s	5 / 0.12s	5 / 0.12s	530
540	0 / 0.08s	0 / 0.08s	0 / 0.08s	530
600	13 / –	18 / 0.19s	18 / 0.19s	530
1000	0 / 0.11s	0 / 0.11s	0 / 0.11s	530
2000	0 / 0.11s	0 / 0.11s	0 / 0.11s	530

Table 6.6: Latency minimization video-codec in the flow-model for resource constraints of $\alpha = (1,1,1), (1,2,1)$ and $(4,4,4)$ and node execution times increased by a factor of 10.

6.11) shows that with the ALPS-model, a good solution could be found for small latency bounds, however not for large latency bounds.

Finally, the assumption that the flow-model performs better for problems with larger execution times and larger latency bounds (e.g., task level) whereas the ALPS-model performs better in the other cases, hence is restricted to problems in high-level synthesis. There, the ALPS-model is unbeatable. Furthermore, we can recognize that in some special cases (i.e., latency minimization with hard resource constraints), ALPS also does not perform well even for problems of high-level synthesis.

The results for iterative scheduling problems not presented here confirm the so far presented results.

$\bar{L}$	Solutions (#nodes/CPU- time)			allocation
	1. int. sol.	best int. sol.	final sol.	α
53	0 / 0.09s	0 / 0.09s	0 / 0.09s	1 2 0
54	0 / 0.27s	0 / 0.27s	0 / 0.27s	1 2 0
55	0 / 1 2 0	No better sol. after 1h		1 2 0
58	32 / 1 2 1	70 / 1 2 0	No better sol. after 1h	1 2 0
59	56 / 1 2 1	4785/ 117.32s	4785 / 117.32s	1 1 0
60	59 / 2 3 1	2316 / 55.09s	2316 / 55.09s	1 1 0
100	28 / 1 2 0	253 / 3.01s	253 / 3.01s	1 1 0
200	36 / 1 2 1	855 / 9.56s	855 / 9.56s	1 1 0

Table 6.7: Resource minimization video-codec in the flow model.

$\bar{L}$	Solutions (#nodes/CPU- time)			allocation
	1. int. sol.	best int. sol.	final sol.	α
530	0 / –	1221 / 12.91s	1221 / 12.91s	1 2 0
540	0 / 1 2 0	0 / 1 2 0	10000 / 187.22s	1 2 0
590	37/ 1 2 1	600 / 1 1 0	600 / 185.32s	1 1 0
600	54/ 2 2 1	No better solution after 1h		2 2 1
1000	58 / 1 2 0	2239 / 29.68s	2239 / 29.68s	1 1 0
2000	25 / 1 2 0	2036 / 24.13s	2036 / 24.13s	1 1 0

Table 6.8: Resource minimization video-codec in the flow-model with node execution times increased by a factor of 10.

$\bar{L}$	Solutions (#nodes/CPU- time)			latency L
	1. int. sol.	best int. sol.	final sol.	
28	No solution after 1 h			–
28	No solution after 1 h			–
34	129 / $L = 34$	No better sol. after 1h		34
50	75 / $L = 40$	No better sol. after 1h		40
100	72 / $L = 36$	No better sol. after 1h		36
200	78 / $L = 36$	No better sol. after 1h		36

$\bar{L}$	Solutions (#nodes/CPU- time)			latency L
	1. int. sol.	best int. sol.	final sol.	
21	23 / $L = 21$	23 / $L = 21$	19900 / 474.48s	21
22	30 / $L = 22$	No better sol. after 1h		22
27	56 / $L = 25$	No better sol. after 1h		25
50	53 / $L = 29$	No better sol. after 1h		29
100	34 / $L = 24$	No better sol. after 1h		24
200	92 / $L = 34$	No better sol. after 1h		34

$\bar{L}$	Solutions (#nodes/CPU- time)			latency L
	1. int. sol.	best int. sol.	final sol.	
18	27 / $L = 18$	27 / $L = 18$	27 / 2.95s	18
19	18 / $L = 19$	18 / $L = 19$	2335 / 47.95s	18
24	51 / $L = 23$	273 / $L = 18$	1500 / 38.0s	18
50	43 / $L = 21$	3298 / $L = 18$	3298 / 281.49s	18
100	42 / $L = 21$	952 / $L = 18$	2200 / 300.05s	18
200	50 / $L = 24$	549 / $L = 18$	2600 / 579.44s	18

$\bar{L}$	Solutions (#nodes/CPU- time)			latency L
	1. int. sol.	best int. sol.	final sol.	
16	0 / –	0 / –	0 / 0.05s	16
19	0 / –	0 / –	0 / 0.16s	16
24	0 / –	0 / –	0 / 0.25s	16
50	0 / –	0 / –	0 / 0.68s	16
100	0 / –	0 / –	0 / 1.59s	16
200	0 / –	0 / –	0 / 2.30s	16

Table 6.9: Latency minimization EWF in the ALPS-model for resource constraints $\alpha = (1,1), (2,1), (2,2)$ and $(4,4)$.

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
28	No solution after 1h			–
29	No solution after 1h			–
34	No solution after 1h			–
50	No solution after 1h			–
100	No solution after 1h			–
200	No solution after 1h			–

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
21	No solution after 1h			–
22	No solution after 1h			–
27	No solution after 1h			–
50	103 / $L = 37$	No better sol. after 1h		37
100	2053 / $L = 36$	No better sol. after 1h		36
200	989 / $L = 33$	No better sol. after 1h		33

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
18	6233 / $L = 18$	6233 / $L = 18$	No better sol. after 1h	18
19	2508 / $L = 19$	No better sol. after 1h		19
24	43 / $L = 24$	No better sol. after 1h		24
50	164 / $L = 36$	No better sol. after 1h		36
100	91 / $L = 37$	No better sol. after 1h		37
200	1467 / $L = 34$	No better sol. after 1h		34

$\bar{L}$	Solutions (#nodes/CPU- time)			latency
	1. int. sol.	best int. sol.	final sol.	L
16	0 / $L = 16$	0 / $L = 16$	No better sol. after 1h	16
18	27 / $L = 18$	140 / $L = 16$	140 / 3.11s	16
24	34 / $L = 23$	561 / $L = 16$	561 / 14.33s	16
50	27 / $L = 30$	1678 / $L = 16$	1678 / 42.60s	16
100	51 / $L = 29$	2003 / $L = 16$	2003 / 50.33s	16
200	40 / $L = 34$	604 / $L = 16$	604 / 15.19s	16

Table 6.10: Latency minimization in the flow-model for the resource constraints $\alpha = (1,1), (2,1), (2,2)$ and $(4,4)$.

$\bar{L}$	Solutions (#nodes/CPU- time)			allocation
	1. int. sol.	best int. sol.	final sol.	α
16	1 / –	1 / –	1 / 0.08s	4 4
17	12 / 4 3	31 / 3 2	31 / 0.43s	3 2
18	17 / –	17 / –	17 / 0.53s	2 2
20	36 / 3 3	106 / 2 2	300 / 9.02s	2 2
21	40 / 2 2	34 / 4.95s	34 / 4.95s	1 1
27	39 / 3 3	No better solution after 1h		3 3
28	49 / 2 2	No better solution after 1h		2 2
50	44 / 2 1	No better solution after 1h		2 1
100	58 / 3 2	No better solution after 1h		3 2
200	47 / 2 2	No better solution after 1h		2 2

Table 6.11: Resource minimization EWF in the ALPS-model.

$\bar{L}$	Solutions (#nodes/CPU- time)			allocation
	1. int. sol.	best int. sol.	final sol.	α
16	0 / 5 4	71 / 4 4	No better solution after 1h	4 4
17	56 / 4 4	No better solution after 1h		4 4
18	56 / 4 2	No better solution after 1h		4 2
27	86 / 2 2	No better solution after 1h		2 2
28	114 / 2 2	No better solution after 1h		2 2
50	115 / 3 2	1111 / 1 1	1111 / 55.77s	1 1
100	71 / 3 2	2019 / 1 1	2019 / 100.23s	1 1
200	89 / 4 3	1753 / 1 1	1753 / 89.07s	1 1

Table 6.12: Resource minimization EWF in the flow-model.

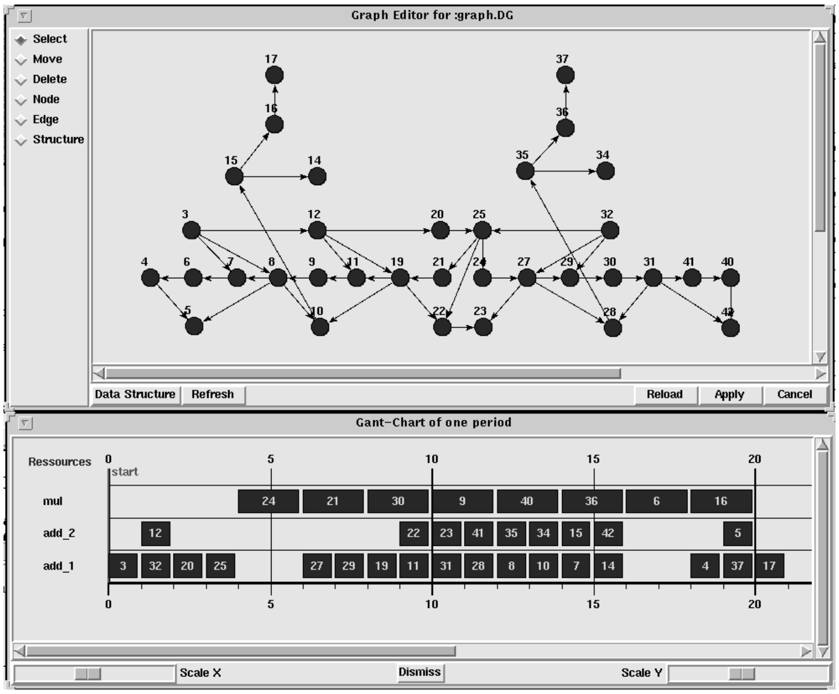

Figure 6.7: Problem graph and result of resource minimization of the EWF (acyclic case).

Chapter 7

Summary and Conclusions

It has been shown that the flow model is an interesting alternative formulation for resource-constrained scheduling problems, especially for task level problems.

In addition to introducing the flow model, we investigated the polyhedral structure of the flow polytope and gained interesting results on the integrality of certain variables. Furthermore, we presented some methods to optimize the structure of the constraints. It has also been shown that the LP-relaxation provides an equally good bound as the best Lagrange-relaxation of the sequentialization constraints.

The results obtained from experiments and comparisons sustained the basic advantage of the flow model as compared to other models like the ALPS or the OASIC model: the independence of the complexity of the latency and computation times of the problem. However, at the level of architectural synthesis problems with monocycle operations and small latencies, our flow model could not outperform the ALPS model.

We would like to remark that the flow model is also applicable in case of problems with rational computation times and non-integral start times of nodes. This does not hold in case of the ALPS model or the OASIC model.

Finally, an interesting relationship of resource-constrained scheduling and the „Traveling Salesman Problem“ (TSP) can be revealed:

Model TSP:$G(V, E)$ with $E = \emptyset$ and one resource type ($|V_T| = 1$) with $\alpha(r_k) = 1$ on which all nodes (cities) can be scheduled. Here, the computation times represent the distances between the cities with $w((v_i, v_j)) = w((v_j, v_i))$. A latency minimal solution of the scheduling problem gives on optimal tour of the TSP. Hence, TSP is a special case of the flow-based scheduling model. Using this analogy, it would be interesting to investigate methods applied for solving ILP formulations for the TSP, e.g., Branch- and Cut algorithms [JRT92]. In the same context, we would like to investigate methods for finding good initial solutions as well as different branching strategies.

Finally, some remarks on model extensions not shown here: Memory- and bus constraints [BST93] can be modeled by additional resources and can hence also be described in the flow model. Thus, also the minimization of memory by computation of lifetime of variables can easily be accomplished because the lifetimes do only depend on the data dependencies which are the same as in ALPS. Hence, the results described in [HLH91] and [BST93] may be directly applied.

Bibliography

[BST93] A. Bachmann, M. Schöbinger, and L. Thiele. Synthesis of domain specific multiprocessor systems including memory design. In *VLSI Signal Processing VI*, pages 417–425, IEEE Press, New York, 1993.

[CPL89] CPLEX Optimization Inc., Incline Village, NV, U.S.A. *Using the CPLEX Callable Library*, 1989.

[CWM94] S. Chaudhuri, R. A. Walker, and J. E. Mitchell. Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. *IEEE Trans. on VLSI Systems*, 2(4):456–471, December 1994.

[GE92] C. Gebotys and M. I. Elmasry. *Optimal VLSI Architectural Synthesis*. Kluwer Academic Publishers, Boston, MA, 1992.

[GE93] C. Gebotys and M. I. Elmasry. Global optimization approach for architectural synthesis. *IEEE Journal on CAD*, 12(9):1266–1278, September 1993.

[GJ76] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1976.

[HLH91] C.T. Hwang, J.H. Lee, and Y.C. Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design*, CAD-10(4):464–475, April 1991.

[JRT92] M. Jünger, G. Reinelt, and S. Thienel. Provably good solutions for the Traveling Salesman Problem. Technical Report 92-31, Interdisziplinäres Zentrum für wissenschaftliches Rechnen, Universität Heidelberg, Germany, 1992.

[NW88] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. John Wiley and Sons, New York, 1988.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization (Algorithms and Complexity)*. Prentice-Hall, 1982.