Feasible Two-Way Circuit Partitioning with Complex Resource Constraints

Hsun-Cheng Lee and Ting-Chi Wang
Department of Information and Computer Engineering
Chung Yuan Christian University
Chungli, Taiwan

Abstract — We study in this paper the feasibility problem for two-way circuit partitioning subject to complex resource constraints. We first prove that the problem is in general NP-complete. We then consider two special cases of the problem, and present polynomial-time algorithms for them. Finally we give a backtracking algorithm to solve the general case. To reduce the run time and the storage space of the backtracking algorithm, an incremental flow computation technique is employed. For each algorithm presented in this paper, the corresponding experimental results are also given to support its efficiency. To the best of our knowledge, this paper is the first one to address the feasibility problem for two-way circuit partitioning with complex resource constraints.

1 Introduction

Circuit partitioning has played an important role at different stages of VLSI design. It is to partition a circuit into the required number of sub-circuits such that each given constraint is satisfied and the given cost function is optimized [3]. Depending on the number of sub-circuits to be generated, circuit partitioning can be divided into two categories: two-way and multi-way. Since multi-way partitioning can be implemented by two-way partitioning in a recursive manner, most of the previous work on circuit partitioning was focused on two-way partitioning.

Traditionally, two-way circuit partitioning is formulated as the problem of *min-cut* partitioning with *size* constraints. The objective of the problem is to partition the circuit into two disjoint sub-circuits such that the number of cut nets between the two sub-circuits is as small as possible while the constraints imposed on the area of each sub-circuit are satisfied. Normally, the area of a circuit is measured in terms of the total number of gates, or the total area of the gates, in the circuit. Many heuristics have been reported in the literature for the problem of two-way min-cut circuit partitioning with size constraints. The heuristics include the iterative improvement based methods [4,5,6,8], the simulated annealing based method [9], and the network flow based method [11], etc.

Recently, Liu *et al*. proposed a new two-way min-cut circuit partitioning problem in which the *complex resource* (instead of the area) constraints are imposed on the two sub-circuits [10]. Two target resource sets are given in the problem. Each resource set consists of a set of heterogeneous resource types, and each resource type is associated with a capacity that denotes the available amount of that resource type. In addition, each node in the circuit is decomposed into a set of basic cells each of which in turn is implemented by one of certain resource types. This implies that each node in the circuit may be implemented in more than one possible way. (For example, the Actel ES6500 FPGA family contains LUT2 and LUT3 [1,2], and hence a 2-input gate can be implemented with either a LUT2 or a LUT3.) As a result, the complex resource constraints are naturally generated, and cannot be captured by the traditional size constraints at all. Given a circuit and two resource sets, the objective of the problem is to partition the set of nodes in the circuit into two subsets such that the partitioning solution is feasible, and the number of cut nets is minimized. A partitioning solution is said to be *feasible* if each subset can be implemented by a different resource set. As pointed out by Liu *et al.*, the problem finds its applications in the partitioning based placement methodology for a single large hierarchical FPGA (e.g., the Actel ES6500 family [1,2]) and in multi-FPGA partitioning for logic emulation. Since the problem is also proven to be NP-hard [10], Liu *et al*. presented a modified FM-type heuristic [6] called FFC-fm for it. The core of the FFC-fm algorithm is to apply a network flow based method to check whether moving a node (with maximum gain) from one subset to another subset will result in a feasible solution. Besides, an incremental flow computation technique is also employed to speed up the feasibility checking process.

Since the FFC-fm algorithm is an iterative improvement based algorithm, it needs a feasible initial solution to begin with. Liu *et al*. briefly mentioned in [10] that a feasible initial solution can be found by randomly distributing the set of nodes in the circuit to the two resource sets based on the information on resource capacity and the information on implementation relationship between basic cells and resources. However, the random distribution method was not described in any detail, and hence how the method finds a feasible initial solution (if one exists) and how much time the method spends are both not clear in [10]. This motivates us to study in this paper the problem of finding a feasible initial solution subject to the given complex resource constraints. First we prove that the feasibility problem is NP-complete in general. We then consider two special cases of the problem, and present polynomial-time algorithms for them. Finally we give a backtracking algorithm to solve the general case. To reduce the run time and the storage space of the backtracking algorithm, an incremental flow computation technique is employed. For each algorithm presented in this paper, the corresponding experimental results are also given to support its efficiency.

The rest of this paper is organized as follows. In Section 2, the formulation of the feasibility problem is given. The NP-complete result is presented in Section 3. In Section 4, two special cases are addressed, and the polynomial-time algorithms for solving them are described. The backtracking algorithm for the general case is given in Section 5, and the experimental results are reported in Section 6.

2 Preliminaries and Problem Description

Let $G=(V,E)$ denote a circuit where V is the node set, and E is the net set. Let $C=\{c_1,c_2,\ldots,c_{|C|}\}$ denote a set of different types of basic cells. Each type of basic cell corresponds to a logic function that is different from any other type of basic cell. In the circuit partitioning problem with complex resource constraints, each node v in V is decomposed into (and hence represented by) a set of *basic cells*, denoted by $decomp(v)=\{(c_{i1},n_{i1}),(c_{i2}, n_{i2}),\ldots,(c_{ip}, n_{ip})\}$, where each c_{ij} is in C, and n_{ij} denotes the *multiplicity* of c_{ij} in v. For example, $decomp(v)=\{(c_1,4),(c_2,2),(c_3,3)\}$ means that v is decomposed into 9 basic cells, and among them, 4 cells are of type c_1, 2 cells are of type c_2, and 3 cells are of type c_3. Besides, we also have a set of different types of resources, denoted by $R=\{r_1,r_2,\ldots,r_{|R|}\}$. Each type of basic cell has at least one type of resource to implement it, and meanwhile each type of resource implements at least one type of basic cell. In addition, a basic cell is implemented exactly by a resource (of proper type), and a resource can implement exactly a basic cell (of proper type). For each basic cell type c in C, let $r\_map(c)=\{r_{i1},r_{i2},\ldots,r_{iq}\} \subseteq R$ denote the set of different resource types each of which can implement c.

Let $R' \subseteq R$ denote a resource set. For each resource type r in R', let $r\_cap(R',r)$ denote the capacity (i.e., the available amount) of r in R'. A node set V is said to be *feasible in* R' if each basic cell in V

is implemented by a resource (of proper type) in R' and each resource in R' implements at most one basic cell (of proper type) in V. (Note that each basic cell of type c can be implemented only by a resource whose type is in $r\_map(c)$.)

In this paper we consider the *feasibility* problem for two-way circuit partitioning with complex resource constraints. The input to an instance of the problem consists of the following:

(1) A set C of basic cell types, and a set R of resource types.
(2) A set V of circuit nodes, and the decomposition information, i.e., $decomp(v)$ for each node v in V.
(3) Two resource sets $R_1 \subseteq R$, $R_2 \subseteq R$, and the resource capacity information, i.e., $r\_cap(R_1,r)$ for each resource type r in R_1, and $r\_cap(R_2,r')$ for each resource type r' in R_2.
(4) The implementation information $r\_map(c)$ for each cell type c.

As a result, we will use an 8-tuple $<C,R,V,decomp,R_1,R_2,r\_cap,r\_map>$ to denote an instance of the feasibility problem. The output to an instance of the problem is to answer whether there exists a *feasible* solution that partitions V into two disjoint subsets V_1 and V_2 such that the following three conditions are satisfied:

(C1) $V_1 \cup V_2 = V$,
(C2) For each node v in V, the basic cells in $decomp(v)$ are all in the same subset (i.e., either in V_1 or V_2, but not in both),
(C3) V_1 is feasible in R_1, and V_2 is feasible in R_2.

Since each node v in V denotes a logic function, it is preferable to put all of its basic cells into the same subset. As a result, the condition (C2) is imposed for the feasibility problem. The feasibility problem defined here is the same as that defined in [10] except that no cost function (i.e., the min-cut objective) is given. Therefore, the net set of the circuit is excluded from the input.

One example, as shown in Figures 1, is used to illustrate the feasibility problem. Figure 1(a) gives an instance. Figure 1(b) shows a feasible solution, while Figure 1(c) shows an infeasible solution that violates condition (C2) since the basic cells c_1 and c_2 that belong to the same node, v_1, are now in different subsets, and so are c_3 and c_4.

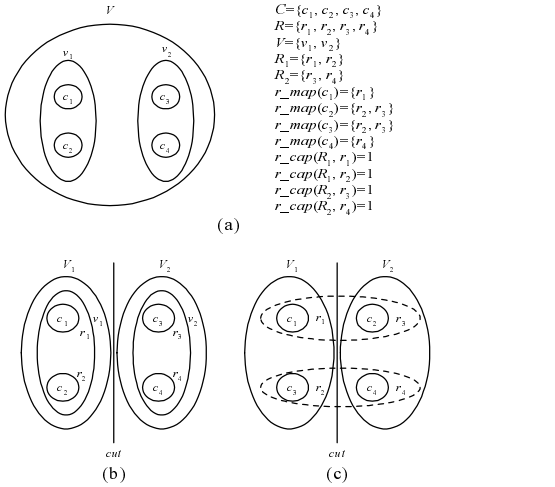

Figure 1: Illustration of the feasibility problem.

3 An NP-complete Result

In this section we prove that the partition problem, that is known to be NP-complete, is polynomial-time reducible to the feasibility problem for two-way circuit partitioning with complex resource constraints. The partition problem is described as follows [7]: Given a set $A=\{a_1,a_2,\ldots,a_n\}$ where each a_i is a positive integer, is there a partition of A two disjoint subsets A_1 and A_2 such that the sum of the elements in A_1 equals the sum of the elements in A_2?

Theorem 1: *The feasibility problem for two-way circuit partitioning with complex resource constraints is NP-complete.*

Proof: First the feasibility problem is in NP because we can guess a partitioning solution (V_1,V_2) and check whether it is a feasible solution in polynomial time using the network flow based method as proposed by Liu *et al.* [10]. The network flow based feasibility checking method is described as follows. For each subset V_i, i=1,2, let $C_i \subseteq C$ denote the set of different basic cell types obtained by decomposing the nodes in V_i. For each given resource set R_i, i=1,2, construct a flow network $F_i=(U_i,E_i)$ such that

(1) $U_i = \{s_i\} \cup C_i \cup R_i \cup \{t_i\}$ denotes the vertex set, where s_i and t_i are the source and the sink of F_i.
(2) $\forall c \in C_i$, add a directed edge from s_i to c with capacity $c\_cap(V_i,c)$, where $c\_cap(V_i,c)$ is the total number of basic cells of type c obtained by decomposing the nodes in V_i.
(3) $\forall c \in C_i$ and $\forall r \in r\_map(c)$, add a directed edge from c to r with capacity ∞.
(4) $\forall r \in R_i$, add a directed edge from r to t_{ii} with capacity $r\_cap(R_i,r)$, where $r\_cap(R_i,r)$ is the total number of resources of type r in R_i.
(5) For each edge in E_i, the initial flow is 0.

After each flow network F_i is built, a maximum flow on F_i is computed. Whether V_i is feasible in R_i or not can be then checked by the theorem stating that V_i is feasible in R_i if and only if all the out-going edges from s_i is saturated. (An edge is saturated if its capacity is equal to its flow.) The proof of the theorem can be found in [10]. Let p denote the maximum number of different types of basic cells that a node in V can be decomposed. Clearly $p=O(|C|)$. The time for constructing each flow network F_i is $O(|C_i||R_i|+p|V_i|)$ since there are $O(|C_i|+|R_i|)$ vertices and $O(|C_i||R_i|)$ edges in F_i, and computing all $c\_cap(V_i,c)$'s takes $O(p|V_i|)$ time. The time for computing a maximum flow on each flow network F_i is polynomial in $|C_i|$ and $|R_i|$ [12]. The time for checking whether each out-going edge from s_i is saturated or not is $O(|C_i|)$. Totally, the time for checking whether (V_1,V_2) is a feasible solution or not is polynomial in $|C|$, $|R|$, p, and $|V|$, which implies the feasibility problem belongs to NP since those numbers constitute the problem size for the feasibility problem.

We next show that the partition problem is polynomial-time reducible to the feasibility problem. Given an instance $<A>$ of the partition problem, we construct an instance $<C,R,V,decomp,R_1,R_2,r\_cap,r\_map>$ of the feasibility problem such that $<A>$ has a "yes" answer if and only if $<C,R,V,decomp,R_1,R_2,r\_cap,r\_map>$ has a "yes" answer. Let $A=\{a_1,a_2,\ldots,a_n\}$. Without loss of generality, we assume $\sum_{a_i \in A} a_i$ is an even number. The corresponding instance of the feasibility problem is constructed as follows. First we construct $C=\{c_1\}$, $R=\{r_1\}$, and $r\_map(c_1)=\{r_1\}$. Then for each a_i, $1 \le i \le n$, we construct a node v_{ii} in V, and let $decomp(v_i)=\{(c_1,a_i)\}$. Finally we construct $R_1=R_2=\{r_1\}$, and let $r\_cap(R_1,r_1)=r\_cap(R_2,r_1)=B$, where $B=(1/2)\sum_{a_i \in A} a_i$. Clearly, the time for constructing the instance $<C,R,V,decomp,R_1,R_2,r\_cap,r\_map>$ is polynomial in the size of A. We now prove that there exists a partition of A into two disjoint subsets A_1 and A_2 such that $\sum_{a_i \in A_1} a_i = \sum_{a_j \in A_2} a_j = B$ if and only if V has a feasible two-way partitioning solution (V_1,V_2).

($\Rightarrow$) Suppose that there exists a partition of A into two disjoint

subsets A_1 and A_2 such that $\sum_{a_i \in A_1} a_i = \sum_{a_j \in A_2} a_j = B$. Let $V_1 = \{v_i \mid a_i \in A_1\}$ and $V_2 = V - V_1 = \{v_j \mid a_j \in A_2\}$. It is not hard to see that V_1 and V_2 each are decomposed into B basic cells of type c_1. Each c_1 can be implemented by a r_1, and R_1 and R_2 each have B r_1's, so $V_1$and V_2 each are feasible in R_1 and R_2, respectively. Therefore, (V_1, V_2) is a feasible solution to V.

($\Leftarrow$) Suppose that (V_1, V_2) is a feasible solution to V. It is not hard to see that V_1 and V_2 each must be decomposed into B basic cells of type c_1 since each c_1 is implemented by a r_1, R_1 and R_2 each have B r_1's, and the total number of basic cells of type c_1 in V is $2 \times B$. Let $A_1 = \{a_i \mid v_i \in V_1\}$ and $A_2 = A - A_1 = \{a_j \mid v_j \in V_2\}$. Since V_1 and V_2 each are decomposed into B basic cells of type c_1, and the basic cells contained in each node of V are all in the same subset, we have $\sum_{a_i \in A_1} a_i = \sum_{a_j \in A_2} a_j = B$. Therefore, (A_1, A_2) is a feasible partition to A. This completes our proof.

4 Algorithms for Two Special Cases

In this section we discuss two special cases of the feasibility problem, and present a polynomial-time algorithm for each of them.

4.1 The first special case

The first special case is that every circuit node is decomposed into exactly one type of basic cell with multiplicity 1. In the following, we describe a network flow based algorithm, called FP-NF (standing for Feasible Partitioning based on Network Flow), for this special case.

For each basic cell of type c_i, let $v(c_i)$ denote the set of nodes in V each of which is decomposed into one c_i, and let $v\_cap(c_i)$ denote the total number of elements (i.e., the cardinality) in $v(c_i)$. To solve this special case, the FP-NF algorithm first scans each node in V once to get the value of each $v\_cap(c_i)$. Then the following flow network $F = (V', E')$ is constructed:

(1) Create the source s and sink t of the flow network.

(2) For each $c_i \in C$, create three nodes v_{c_i}, v_{1,c_i}, and v_{2,c_i}.

(3) For each $r_i \in R$, create nodes r_{1,r_i}, and r_{2,r_i}.

(4) For each $c_i \in C$, add an edge $s \rightarrow v_{c_i}$ with capacity $v\_cap(c_i)$.

(5) For each $c_i \in C$, add edges $v_{c_i} \rightarrow v_{1,c_i}$ and $v_{c_i} \rightarrow v_{2,c_i}$ each with capacity ∞.

(6) For each $c_i \in C$, add edges $v_{1,c_i} \rightarrow r_{1,r_j}$ and $v_{2,c_i} \rightarrow r_{2,r_j}$ each with capacity ∞ for each $r_j \in r\_map(c_i)$.

(7) For each $r_i \in R$, add edges $r_{1,r_i} \rightarrow t$ and $r_{2,r_i} \rightarrow t$ each with capacity $r\_cap(R_1, r_i)$ and $r\_cap(R_2, r_i)$, respectively.

(8) For each edge in F, the initial flow is 0.

After the above flow network is built, the FP-NF algorithm proceeds to compute a maximum flow on F. If every out-going edge from s is saturated, then a feasible partitioning solution exists. Such a feasible solution can be found as follows. For each set $v(c_i)$, the algorithm assigns $m(c_i)$ nodes in it to V_1, and assigns the remaining nodes to V_2, where $m(c_i)$ is the flow on edge $v_{c_i} \rightarrow v_{1,c_i}$. By doing this, (V_1, V_2) can be proven to be a feasible solution to V. (See Theorem 2.) On the other hand, if at least one out-going edge from s is not saturated, then there exists no feasible solution to V. The correctness of the FP-NF algorithm is stated in Theorem 2.

Theorem 2: *There exists a feasible two-way partitioning solution to V if and only if after computing the maximum flow on F, all the out-going edges from s (i.e., $s \rightarrow v_{c_i}$, for $1 \le i \le |C|$) are saturated, i.e., $cap(s, v_{c_i}) = flow(s, v_{c_i})$, where $cap(s, v_{c_i})$ and $flow(s, v_{c_i})$ denote the capacity and flow of edge $s \rightarrow v_{c_i}$.*

Proof: ($\Rightarrow$) Suppose that there exists a feasible two-way partitioning solution (V_1,V_2) to V. Then each node in V is assigned to an available resource to implement since each node is decomposed into one basic cell. For each node in V_j (j = 1 or 2), if it is decomposed into c_i and is implemented by a resource of type r_k in R_j, then we can add a flow of value 1 on the path: $s \rightarrow v_{c_i} \rightarrow v_{j,c_i} \rightarrow r_{j,r_k} \rightarrow t$, where $r_k \in r\_map(c_i)$. Thus after having added all the flows, we have for each edge $s \rightarrow v_{c_i}$, $flow(s, v_{c_i}) = v\_cap(c_i)$. Since $v\_cap(c_i) = cap(s, v_{c_i})$, we have $cap(s, v_{c_i}) = flow(s, v_{c_i})$ for each edge $s \rightarrow v_{c_i}$.

($\Leftarrow$) Suppose that after the maximum flow computation, every edge $s \rightarrow v_{c_i}$ is saturated. Each flow of value 1 on the path: $s \rightarrow v_{c_i} \rightarrow v_{j,c_i} \rightarrow r_{j,r_k} \rightarrow t$, corresponds to that a node $v \in v(c_i)$ can be assigned to V_j since there is a resource r_k in R_j to implement it, where j = 1 or 2, and $r_k \in r\_map(c_i)$. Since each edge $s \rightarrow v_{c_i}$ is saturated, and the capacity on the edge is the total number of nodes in V that are decomposed into c_i, each node in V can be always assigned to a resources to implement. Besides, for each cell type of c_i, $flow(s, v_{c_i}) = flow(v_{c_i}, v_{1,c_i}) + flow(v_{c_i}, v_{2,c_i})$, and hence for the nodes that are decomposed into c_i, among them, we can assign $m(c_i)$ and $n(c_i)$ nodes to V_1 and V_2, respectively, where $m(c_i) = flow(v_{c_i}, v_{1,c_i})$, and $n(c_i) = flow(v_{c_i}, v_{2,c_i})$. This implies that (V_1, V_2) is a feasible solution to V. This completes our proof.

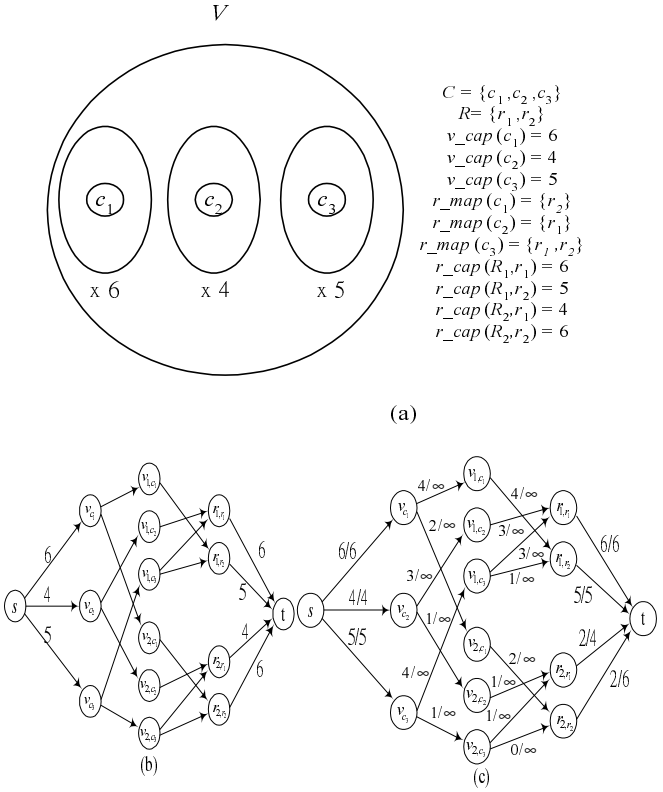

Figure 2: Illustration of the FP-NF algorithm.

We use Figure 2 to illustrate the FP-NF algorithm. Figure 2(a) shows that the given circuit has 6, 4, and 5 nodes that are decomposed into basic cells of types c_1, c_2, and c_3, respectively.

Figure 2(b) gives the initial flow network, where each edge is labeled by its capacity. If an edge is not labeled, that means its capacity is ∞. Figure 2(c) shows a maximum flow on the network where the first label on each edge denotes the flow. Based on Figure 2(c), a feasible partitioning solution (V_1,V_2) can be found where V_1 contains 4, 3, and 4 nodes that are decomposed into basic cells of types c_1, c_2, and c_3, respectively, and V_2 has the remaining nodes.

We now analyze the complexity of the FP-NF algorithm. It takes $O(|V|)$ time to compute all $v\_cap(c_i)$'s. Constructing the initial flow network takes $O(|C||R|)$ time since the number of vertices is $3|C| + 2|R| + 2$ and the number of edges is $O(|C| + |C||R|+|R|)$. Computing a maximum flow can be done in time polynomial in the size of the flow network. Finally, checking whether each out-going edge from the source is saturated or not, and finding a feasible solution (if one exists) both can done in $O(|C|)$ time. Totally, the complexity of the FP-NF is polynomial in $|V|$, $|C|$, and $|R|$.

In fact the size of the flow network may be further reduced as pointed out in [10]. Because some basic cells may be implemented by the same set of the resource types, (i.e., it may be possible that $\exists c_i,c_j \in C$, $r\_map(c_i)= r\_map(c_j)$), we can treat those equivalent cells as one type of cell. (Note that two types of basic cells c_i and c_j are said to be equivalent if $r\_map(c_i) = r\_map(c_j)$). If we let C' be the new set of types of basic cells, then we can replace C by C' when building the flow network. Note that each cell in C' uniquely maps to one subset of resource. For $|R|$ resource types, the number of non-empty subset is at most $2^{|R|}$-1, so the number of cells in C' is at most $\min(2^{|R|}\text{-}1, |C|)$. For example, if $|R|$ is 5, then $|C'|$ is at most 31. Therefore the size of the flow network may be further reduced after C is replaced by C'.

4.2 The second special case

As mentioned in Section 1, the problem of min-cut circuit partitioning with complex resource constraints has an application in modeling the partitioning based placement problem for a single large hierarchical FPGA. For this particular application, an FPGA vendor may provide a cell library that also consists of hard, and soft cells in addition to basic cells. A typical example is the Actel ES6500 family which has about 300 library cells as mentioned in [10]. Each hard or soft cell can be in turn decomposed into a set of basic cells. With the cell library, the circuit can be optimized by technology mapping such that each node corresponds to a hard, or soft, or basic cell. In practice, the cell library usually remains fixed for the FPGAs in the same family. This motivate us to study another special case of the feasibility problem in which the following assumptions are made:

(1) The cell library that consists of hard, soft, and basic cells is fixed.
(2) The set C of different types of basic cells and the set R of different types of resources both are fixed. This implies that $r\_map(c)$ is also fixed for each type of basic cell in C.
(3) Each node in a circuit corresponds to a hard, or soft, or basic cell. Each hard or soft cell is in turn decomposed into a set of basic cells. This implies that the decomposition information for each library cell is also fixed.

Under the above assumptions, C, R, r_map and $decomp$ are treated as constants while only the circuit node set V, the two target resource sets R_1 and R_2, and the resource capacity information r_cap on the two target resource sets are treated as variables. However, the size of each R_1,R_2, and r_cap is bounded above by $O(|R|)$. *Hence the problem size for this special case only involves* $|V|$. In the following we present an integer linear programming (ILP) based algorithm, called FP-ILP (standing for Feasible Partitioning based on Integer Linear Programming), to solve this special case. Since the goal of the feasibility problem is to find a feasible solution (if one exists), there will be no objective function in our ILP formulation.

Let $N=\{n_1,n_2,\ldots,n_{|N|}\}$ denote the fixed set of different types of cells in the library. A circuit node is said to be of type n_i if it corresponds to a library cell of type n_i. The FP-ILP algorithm first scans each node in V to determine the value of each $v\_cap(n_i)$, where $v\_cap(n_i)$ denotes the number of nodes of type n_i that are in V. We next define the non-negative integer variables that will be used in the ILP formulation. Let X_i and Y_i denote the numbers of nodes of type n_i to be placed in V_1 and V_2, respectively. Let $Z_{1,j,k}$ and $Z_{2,i,k}$ denote the numbers of basic cells of type c_i in V_1 and V_2 such that each basic cell is implemented by a resource of type r_k, respectively. Then the set of linear constraints in the ILP formulation can be divided into three categories: for nodes, for cells, and for resources. Since each node in V must be assigned to either V_1 or V_2, the number of nodes of type n_i in V_1 plus the number of nodes of type n_i in V_2 must be equal to the number of nodes of type n_i in V. Hence we have the following linear constraints for nodes:

$$X_i + Y_i = v\_cap(n_i),\ \forall\ n_i \in N. \quad (1)$$

For each basic cell that is obtained by decomposing a node in V_1 (V_2), there must be a corresponding resource in R_1 (R_2) to implement it. For each basic cell of type c_i, let $decomp^{-1}(c_i)$ denote the set of different types of nodes that contain c_i. We have the following linear constraints for basic cells:

$$\sum_{n_j \in decomp^{-1}(c_i)} X_j = \sum_{r_k \in r\_map(c_i)} Z_{1,i,k}\ ,\ \forall\ c_i \in C, \text{ and}$$
$$\sum_{n_j \in decomp^{-1}(c_i)} Y_j = \sum_{r_k \in r\_map(c_i)} Z_{2,i,k}\ \ \forall c_i \in C. \quad (2)$$

For each resource of type r_i in R_1 (R_2), the amount used to implement the corresponding basic cells in $V_{1|1}(V_2)$ must be not larger than $r\_cap(R_1,r_i)$ ($r\_cap(R_2,r_i)$). For each resource of type r_i, let $r\_map^{-1}(r_i)$ denote the set of different types of basic cells that can be implemented by r_i. We have the following linear constraints for resources:

$$\sum_{c_j \in r\_map^{-1}(r_i)} Z_{1,j,i} \le r\_cap(R_1, r_i)\ ,\ \forall\ r_i \in R, \text{ and}$$
$$\sum_{c_j \in r\_map^{-1}(r_i)} Z_{2,j,i} \le r\_cap(R_2, r_i)\ ,\forall r_i \in R. \quad (3)$$

Finally, we have the constraints that require that each of the variables X_j, Y_j, $Z_{1,i,k}$, and $Z_{2,i,k}$ must be a non-negative integer. We use Figure 3 to illustrate the above ILP formulation. Figure 3(a) gives an instance of this special case, and Figure 3(b) shows the linear constraints in the corresponding ILP formulation.

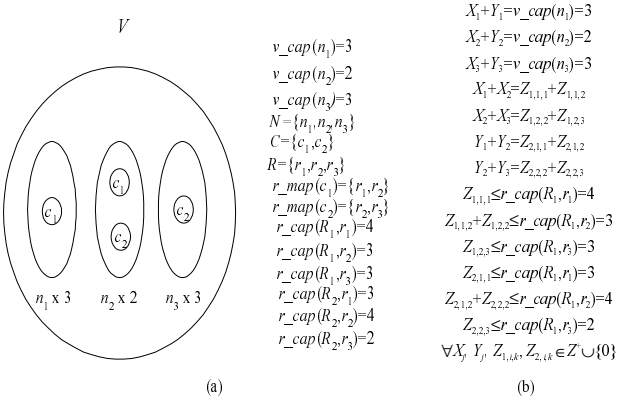

Figure 3: Illustration of the ILP formulation.

Once all the linear constraints have been constructed, the FP-ILP algorithm proceeds to apply any ILP solver to see whether there exists a set of feasible values for those integer variables. If such a set of feasible values exists, the corresponding feasible partition (V_1, V_2) can be obtained based on the values of X_i's and Y_i's.

The complexity of the FP-ILP algorithm is analyzed as follows. Scanning each node once to determine the value of each $v\_cap(n_i)$

is done in $O(|V|)$ time. In the ILP formulation, the number of integer variables is $O(|N|+\sum_{c_i \in C} r\_map(c_i))$, and the number of linear constraints is $O(|N|+|C|+|R|+\sum_{c_i \in C} r\_map(c_i))$. Since $decomp(n_i)$=O($|C|$) and $r\_map(c_i)$=$O(|R|)$, the ILP instance can be constructed in $O(|N|+|C||R|)$ time. Since $|N|$, $|C|$, and $|R|$ each are constants, the total construction time is also constant. It is known that the ILP problem in general is NP-hard [7], and hence its worst-case time complexity is exponential in both the number of variables and the number of linear constraints. However, the ILP formulation for our special case can be solved in constant time in practice because the number of variables and the number of linear constraints are both constants. Consequently, this special case can be solved by the FP-ILP algorithm in time linear in $|V|$.

5 An Algorithm for the General Case

In this section we present an algorithm, called FP-B (standing for Feasible Partitioning based on Backtracking), which is based on the backtracking technique to solve the general case of the feasibility problem. (The general case is different from the second special case in the sense that each input variable is not a fixed constant.) Clearly, the FP-B algorithm can be incorporated into the FFC-fm algorithm [10] to solve the min-cut partitioning problem since the FFC-fm algorithm needs a feasible initial solution to start with.

Before describing the FP-B algorithm, some necessary definitions are given first. A partitioning solution (V'_1, V'_2) is said to be a *candidate* solution if V'_1, $V'_2 \subseteq V$, $V'_1 \cap V'_2 = \phi$ and $V'_1 \cup V'_2 = V$. A solution (V'_1, V'_2) is said to be a *partial* solution if V'_1, $V'_2 \subseteq V$, $V'_1 \cap V'_2 = \phi$ and $V'_1 \cup V'_2 \subset V$. A partial solution (V'_1, V'_2) is said to be *promising* if V'_1 and V'_2 are feasible in R_1 and R_2, respectively.

It is not hard to see that any instance of the feasibility problem can be solved by assigning each circuit node in V into one of the subsets V_1 and V_2, and checking which assignment combinations give feasible solutions. Totally, there are $2^{|V|}$ candidate solutions. The set of all candidate solutions can be created by constructing a *state space tree* in which the possible assignments for the ith node in V are stored at level i. (Suppose that the root is at level 0.) Each pair of sibling nodes at level i corresponds to the possible assignments of the ith node v_i to V_1, and V_2, respectively. Hence, each node in the tree can be denoted by an ordered pair (i,j) meaning that v_i is assigned to V_j, where $1 \le i \le |V|$, and j=1 or 2. Figure 4 shows a state space tree for V=$\{v_1,v_2\}$. In a state space tree, any path from the root to a leaf corresponds to a candidate solution, and any path from the root to a non-leaf corresponds to a partial solution. A node in the tree is said to be *promising* if the corresponding partial solution is promising. On the other hand, a node in the tree is said to be *non-promising* if it is not a promising node.

The FP-B algorithm basically is to construct a state space tree in the *depth-first search* manner. However, to increase the efficiency, the algorithm adopts the pruning strategy that avoids constructing any subtree rooted at a non-promising node (since each node in that subtree is also non-promising). Hence, whenever a tree node is being constructed, whether it is promising or not will be checked immediately. The algorithm will terminate right after either a feasible solution is found, or no promising nodes in the tree can be generated any further.

The network flow based feasibility checking method [10], as described in the proof of Theorem 1, is applied by the FP-B algorithm to check whether a tree node is promising or not. In addition, to further reduce the run time, an incremental flow computation technique is also employed. Initially, two flow networks F_1, and F_2 corresponding to the two empty subsets induced by the root are constructed using the method described in the proof of Theorem 1. Note that no circuit node has been assigned to either subset yet at the beginning, and hence in each flow network, the capacity of each edge from the source to each cell type is 0, and the flow of each edge is also 0. Suppose that t is the node that is currently being considered for generating one child or for backtracking to its parent. In order to reduce the memory space for storing the state space tree, during any course of the execution of the algorithm, only one pair of flow networks (F_1,F_2) is maintained and is associated with the currently visited node, i.e., t. Depending on the role that t is playing, the algorithm proceeds in one of the following two ways.

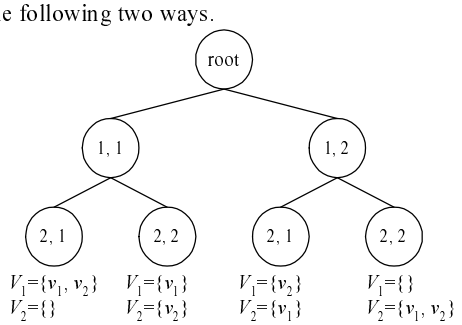

Figure 4: A state space tree.

Case 1: Suppose that t is at level i-1 and is currently being considered for generating one child, say t', at level i,. Let F= F_1if t' is a left child (that corresponds to assigning the ith circuit node v_i to V_1), and let F= F_2 if t' is a right child (that corresponds to assigning v_i to V_2). To check whether t' is promising, the *insertion* operation, as proposed by Liu *et al.* [10], is performed on F as follows. For each basic cell type c_j in $decomp(v_i)$, increase the capacity of the edge from the source to c_j by the corresponding multiplicity. Then additional flows are pushed from the source to the sink until the maximum flow in F is obtained. (Note that this is an incremental flow computation technique.) After the incremental flow computation is done, if the out-going edges from the source are all saturated, then t' is a promising node. In this case, t' becomes t, and the resulting pair of flow networks is maintained and associated with t'. On the other hand, if there exists one non-saturated out-going edge from the source, then t' is non-promising. In this case, the algorithm needs to restore the flow network F to its previous state before continuing to generate the other child of t or to backtrack to the parent of t. This is done by the *deletion* operation that was also proposed by Liu *et al* [10]. This operation correctly deletes the added capacity and flow on each modified edge. Please refer to [10] for the details of the insertion and the deletion operations.

Case 2: Suppose that t is at level i and is currently being considered for backtracking to its parent, say t', at level i-1. Let F= F_1if t is the left child of t', and let F= F_2 if t is a right child of t'. In this case, we need to perform the deletion operation on F using the information provided by $decomp(v_i)$ in order to restore F to its previous state that corresponds to t'.

As pointed out in [10], each insertion or deletion operation has the polynomial-time complexity in the worst case, but runs very efficiently in practice. Hence, the time complexity of the FP-B algorithm mainly depends on the number of tree nodes that are generated. For each tree node, it is not hard to verify that one insertion operation is required for generating the node, and one deletion operation is required for backtracking to the parent.

6 Experimental Results and Conclusions

The three algorithms, FP-NF, FP-ILP, and FP-B, have been all implemented in C++ language on a PC with an Intel Celeron 450MHz CPU. We first tested the FP-NF algorithm on 18

randomly generated circuits in which each circuit node is decomposed into one basic cell. The number of nodes in each of the circuits ranges from 10000 to 1000000. In addition, for each circuit, there are 6 different types of resources in R, and 300 different types of basic cells in C, and each type of basic cell can implemented by 1~4 types of resources. We used a parameter *rate* to control the capacity of each type of resource in each of R_1 and R_2. For example, if a circuit contains n basic cells of type c_1 which can be implemented by a resource of type r_1 or r_2, then the capacity of r_1 and the capacity of r_2 contributed by c_1 are both set to $(1/2)\times n\times rate$. Hence, based on the number of each type of basic cells in the circuit, the capacity of each type of resource is generated and then accumulated. Finally the capacity of each type of resource is evenly divided in the two target resource sets. Table 1 lists the experimental results. The data in the fourth column shows the run time, and the data in the fifth column indicates that whether there exists a feasible solution to each circuit. As can be seen in Table 1, the FP-NF algorithm spent at most 0.5 second on each test case, and hence ran very efficiently

We next tested the FP-ILP algorithm on another 18 randomly generated circuits. There are 500 different types of library cells, and each node can be decomposed into 1~4 basic cells. The statistics for the other input data are similar to those circuits used to test the FP-NF algorithm. In this set of experiments, only the circuit itself is treated as a variable. In the current implementation of the FP-ILP algorithm, the ILP formulation is solved by the package "LINGO". Table 2 gives the experimental results. As can be seen in Table 2, the run time on each test case is less than 10 seconds.

Finally we tested the FP-B algorithm on another 11 randomly generated circuits. The circuit size ranges from 1000 to 35000. The parameter *rate* was set to be 1.3. The statistics for the other input data are similar to those used to test the FP-ILP algorithm. The experimental results are given in Table 3. The third and fourth columns in the table list the number of tree nodes that were generated by the FP-B algorithm, and the approximately total number of tree nodes in each state space tree, respectively. For each circuit, the FP-B algorithm was always able to find a feasible solution. The last column in the table gives the run time of the FP-B algorithm. Clearly, the more tree nodes generated, the more run time the algorithm spent.

The experimental results shown in Tables 1, 2 and 3 indicate that our algorithms are all very efficient. Before closing this paper, we would like to make the following two remarks. Firstly, since no previous work has been done for the feasibility problem, we cannot provide any experimental comparison in this paper. Secondly, it is interesting to study whether there exist methods which are faster than the FP-B algorithm for solving the general case.

Acknowledgments

This work was partially supported by the National Science Council of Taiwan under grants NSC-88-2815-C-033-010-E and NSC-89-2215-E-033-006.

References

[1] Actel FPGA DATA Book and Design Guide, Actel Corporation, 1996.

[2] Actel's Reprogrammable SPGAs, Preliminary Advance Information, Actel Corporation, 1996.

[3] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey," *INTEGRATION, the VLSI Journal*, 1995, pp. 1-81.

[4] S. Dutt and W. Deng, "A Probability-Based Approach to VLSI Circuit Partitioning," *Proc. Design Automation Conf.,* 1996, pp. 100-105.

[5] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques," *Proc. International Conf. on Computer-Aided Design*, 1996, pp. 92-99.

[6] M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *Proc. Design Automation Conf.,* 1982, pp. 175-181.

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

[8] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning of Electrical Circuits," *Bell System Technical Journal*, Feb. 1970, pp. 291-307.

[9] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, 1983, pp. 671-780.

[10] H. Liu, K. Zhu and D. F. Wong, "Circuit Partitioning with Complex Resource Constraints in FPGAs," *Proc. International Symp. on Field Programmable Gate Arrays*, 1998, pp. 77-84.

[11] H. Yang and D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning," *Proc. International Conf. on Computer-Aided Design,* 1994, pp. 50-55.

[12] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *An Introduction to Algorithms*, the MIT Press, 1990.

Table 1: The experimental results of the FP-NF algorithm.

test case	# of circuit nodes	rate	time(sec)	state
1	10000	0.9	0.22	No
2	10000	1.1	0.22	Yes
3	50000	0.9	0.28	No
4	50000	1.1	0.28	Yes
5	90000	0.9	0.28	No
6	90000	1.1	0.28	Yes
7	100000	0.9	0.21	No
8	100000	1.1	0.27	Yes
9	300000	0.9	0.33	No
10	300000	1.1	0.33	Yes
11	500000	0.9	0.33	No
12	500000	1.1	0.32	Yes
13	700000	0.9	0.39	No
14	700000	1.1	0.39	Yes
15	900000	0.9	0.44	No
16	900000	1.1	0.38	Yes
17	1000000	0.9	0.38	No
18	1000000	1.1	0.5	Yes

Table 2: The experimental results of the FP-ILP algorithm.

test case	# of circuit nodes	rate	time(sec)	state
1	10000	0.9	9.22	No
2	10000	1.1	7.22	Yes
3	50000	0.9	9.22	No
4	50000	1.1	8.27	Yes
5	90000	0.9	9.22	No
6	90000	1.1	8.28	Yes
7	100000	0.9	9.22	No
8	100000	1.1	8.27	Yes
9	300000	0.9	9.33	No
10	300000	1.1	7.39	Yes
11	500000	0.9	9.38	No
12	500000	1.1	8.38	Yes
13	700000	0.9	9.44	No
14	700000	1.1	7.44	Yes
15	900000	0.9	9.5	No
16	900000	1.1	7.44	Yes
17	1000000	0.9	9.55	No
18	1000000	1.1	7.49	Yes

Table 3: The experimental results of the FP-B algorithm.

test case	# of circuit nodes	# of tree nodes generated	approx. total # of tree nodes	time (sec)
1	1000	1336	10^{301}	0.47
2	3000	4037	10^{903}	1.59
3	5000	6763	10^{1505}	2.75
4	7000	5408	10^{2107}	3.84
5	9000	6763	10^{2709}	4.89
6	10000	13502	10^{3010}	5.44
7	15000	18965	10^{4515}	7.2
8	20000	26970	10^{6021}	10.88
9	25000	31577	10^{7526}	12.14
10	30000	40459	10^{9031}	16.2
11	35000	44293	10^{10536}	16.97