INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 10 (2003) 1–11

An improved direct labeling method for the max-flow min-cut computation in large hypergraphs and applications

Joachim Pistorius and [a]Michel Minoux

Altera Corporation, MS 2301, 101 Innovation Drive, San Jose, CA 95134, USA and [a]Université Pierre et Marie Curie, Laboratoire d'Informatique de Paris 6, 4, place Jusseiu, F-75252, Paris Cedex – 05, France
E-mail: Joachim_Pistorius@altera.com [Pistorius]; Michel.Minoux@lip6.fr [Minoux]

Received 2 April 2002; accepted 1 July 2002

Abstract

Algorithms described so far to solve the maximum flow problem on hypergraphs first necessitate the transformation of these hypergraphs into directed graphs. The resulting maximum flow problem is then solved by standard algorithms. This paper describes a new method that solves the maximum flow problem directly on hypergraphs, leading to both reduced run time and lower memory requirements. We compare our approach with a state-of-the-art algorithm that uses a transformation of the hypergraph into a directed graph and an augmenting path algorithm to compute the maximum flow on this directed graph: the run-time complexity as well as the memory space complexity are reduced by a constant factor. Experimental results on large hypergraphs from VLSI applications show that the run time is reduced, on average, by a factor approximately 2, while memory occupation is reduced, on average, by a factor of 10. This improvement is particularly interesting for very large instances, to be solved in practical applications.

Keywords: Graph theory, optimization, minimum cut, maximum flow problem, hypergraph

1. Introduction

The maximum flow problem has applications in numerous fields (Ahuja, Magnanti and Orlin, 1993) such as operations research (Rardin, 1997), VLSI CAD (partitioning, placement, routing, floor-planning) (Hu and Moerder, 1985; Kuo and Cheng, 1997; Liu and Wong, 1998; Yang and Wong, 1996), or transportation problems (transportation of fluids, of commodities, of information in a communication network) (Beasley, 2000; Hu, 1963). All these applications can be modeled in terms of capacitated directed graphs.

Let $G = (V, E)$ be a directed graph with vertex set V containing $|V|$ nodes, and arc set E containing $|E|$ arcs. Each arc $e \in E$ is an ordered pair of nodes $\forall e \in E$, $e = (v, w)$, $v \neq w$ which is directed from v to w. We consider two distinguished nodes s and t called source and sink, and a positive capacity $c(e) = c(v, w) > 0$ for each arc $e = (v, w) \in E$. A flow on G is a function $f : E \rightarrow IR$ satisfying the following constraints:

$$\forall v \in V \backslash \{s, t\}, \quad \sum_{e \in \omega^+(v)} f(e) = \sum_{e \in \omega^-(v)} f(e) \quad \text{(flow conservation)}$$
$$\forall e \in E, \quad 0 \leqslant f(e) \leqslant c(e) \quad \text{(capacity limit)}$$

where $\omega^+(v)$ denotes the set of arcs originating at v and $\omega^-(v)$ denotes the set of arcs terminating at v.

The value of a flow $val(f)$ is the sum of the flows coming out of the source node or going into the sink node:

$$val(f) = \sum_{e \in \omega^+(s)} f(e) = \sum_{e \in \omega^-(t)} f(e)$$

A maximum flow is a flow of maximum value. The maximum flow problem is to find such a flow in the network. An *s-t* cut is a bipartition $(X, \overline{X})$ of the vertex set V such that $X \cup \overline{X} = V$, $X \cap \overline{X} = \varnothing$, $s \in X$, and $t \in \overline{X}$. The capacity of such a cut is

$$c(X, \overline{X}) = \sum_{e \in \omega^+(X)} c(e)$$

A cut is called minimum if its capacity is minimum. Ford and Fulkerson (1962) first exhibited the relationship between minimum cut and maximum flow, which is obtained by means of the concept of augmenting paths in the flow network. An augmenting path from node v to node w in G is a simple path from v to w which can be used to push additional flow from v to w.

Theorem 1.1 *Max-flow min-cut theorem (Ford and Fulkerson, 1962):*
Let f be a maximum flow f of G. Let $X = \{v \in V | \exists$ *augmenting path from s to* v *in* $G\}$, *and let* $\overline{X} = V \backslash X$. *Then* $(X, \overline{X})$ *is a cut of minimum capacity, and f saturates all arcs from X to* $\overline{X}$.

Ford and Fulkerson's algorithm (1962) is based on the iterative search of augmenting paths between source and sink nodes. Edmonds and Karp (1972) showed that using a breadth-first search algorithm to find the shortest augmenting path between source and sink at each iteration, leads to a polynomial time algorithm $O(|V||E|^2)$. Dinic (1970) proposed a three phase algorithm with time complexity $O(|V|^2|E|)$. Goldberg and Tarjan (1988) further reduced the time complexity to $O(|V||E|\log(|V|^2/|E|))$ by an algorithm which is based on a preflow concept. Goldberg and Rao (1997) finally presented an algorithm that introduced the concept of arc lengths that are based on the residual flow values and the residual arc capacities. This algorithm has an $O(\min(|V|^{2/3}, |E|^{1/2})|E|\log(|V|^2/|E|)\log(U))$ time bound, where the arc capacities are in the range $[1, \ldots, U]$. For more details on algorithms that solve the maximum flow problem and their respective time complexity we refer to Goldberg and Rao, (1997).

All these algorithms work on directed graphs. Some applications, such as partitioning problems (Alpert, 1996) however, are more accurately modeled by their more general representation as a hypergraph. Cambini, Gallo and Scutella (1997) presented the minimum cost flow problem[1] on

[1] Given a directed graph with a capacity and a cost factor associated with each arc. Then the minimum cost flow problem consists of finding a way of supplying the sink nodes from the source nodes at minimum cost. The maximum flow problem is a variation of the minimum cost flow problem as it aims at maximizing the flow from source to sink without considering a cost associated with the arcs (Beasley, 2000).

directed hypergraphs. They defined spanning hypertrees and showed that, like in the standard and in the generalized minimum cost flow problems, there is a correspondence between bases and spanning hypertrees. They also showed that, like for the network simplex algorithms for the standard and for the generalized minimum cost flow problems, most of the computations performed at each pivot operation have direct hypergraph interpretations. Finally, Cambini, Gallo and Scutella (1997) presented the outline of a specialized simplex-type algorithm for minimum cost hyperflow computations and tested it on random hypergraphs.

In this paper, we focus on the maximum flow problem on undirected hypergraphs. Such a hypergraph $H = (V, E)$ is defined by a vertex set V and a set of hyperedges E. Each hyperedge $e \in E$ is defined as a subset of the vertex set $e \subseteq V$ containing at least two nodes.[2] With each hyperedge $e \in E$ there is an associated capacity $c(e) \geqslant 0$. A cut of a hypergraph is a bipartition $(X, \overline{X})$ of the vertex set V such that $X \cup \overline{X} = V$, $X \cap \overline{X} = \emptyset$. The capacity of this cut is the sum of the capacities of the hyperedges having nodes in both X and $\overline{X}$.

$$c(X, \overline{X}) = \sum_{\{e \in E | e \cap X \neq \emptyset, e \cap \overline{X} \neq \emptyset\}} c(e)$$

The concepts of a flow, maximum flow, and its relation to minimum cut can be extended to hypergraphs.[3] Thus minimum cuts in hypergraphs can be found by means of maximum flow computations after transformation of the given hypergraph into a directed graph. Several transformation techniques have been presented so far. The most common transformation is the substitution of each hyperedge by a clique. Each edge of the clique is weighted, depending on the number of nodes in the hyperedge. However, Ihler, Wagner and Wagner (1993) showed that there is no possible weight that will always realize the same capacity of a minimum cut in the graph and in the corresponding hypergraph separating the same sets of nodes.

Another basic transformation, the star model (see Fig. 1), introduces a dummy node for each hyperedge, and connects this dummy node to each node in the corresponding hyperedge (Hu and Moerder, 1985). The graph obtained by this transformation is undirected. Therefore, it has to be transformed into a directed graph. This is done by substituting each edge between two nodes v and w by two arcs, (v, w) and (w, v). Both arcs have the same capacity as the original edge.

Neither transformation appears to be well-suited to the maximum flow problem, because they do not permit the calculation of the exact minimum cut value of the cut on the hypergraph. A better approach has been proposed by Hu and Moerder (1985) who transformed an undirected graph obtained by a star transformation into a directed graph using an approach first suggested by Lawler (1976). In this approach, the resulting graph has $|V| = 2|V| + 2|E|$ nodes and $|E'| = 2|V| + 2(p + 2)|E|$ arcs where p is the average cardinality of the hyperedges (see Fig. 1). Yang and Wong (1996) further improved this transformation by only duplicating the dummy node. The transformation from a hypergraph $H = (V, E)$ to a directed graph $G' = (V', E')$ proceeds as follows:

- for each node $v_i \in V$, create a node $v_i' \in V'$;

[2] If each hyperedge $e \in E$ contains exactly two vertices $v \in V$, then the hypergraph is said to be a graph.

[3] The definition of flows on hypergraphs is the same as for flows on directed graphs (flow conservation, capacity limit, flow value, and cut capacity) with the difference that the meaning of e has changed. e is an arc in the directed graph and a hyperedge in the hypergraph.

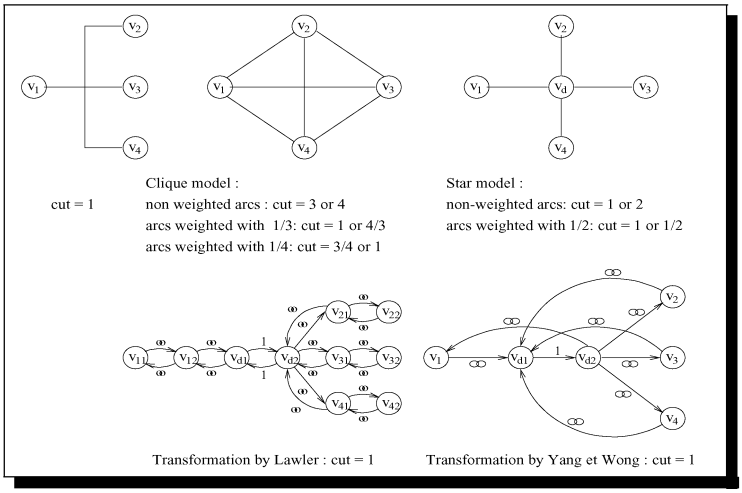

Fig. 1. Transformation of a hypergraph into a graph.

- for each hyperedge $e_i \in E$, create two dummy nodes v_{di1} and v_{di2} in V' and one arc of unit capacity (v_{di1}, v_{di2}) in E';
- for each node $v_j \in e_i$, create two arcs of infinite capacity between (v_j, v_{di1}) and (v_{di2}, v_j) in E'.

This leads to $|V'| = |V| + 2|E|$ nodes and $|E'| = (2p + 1)|E|$ edges.

The inconvenience of all the above techniques is that the transformation of a hypergraph into a graph leads to a significant increase in the number of nodes and edges. Practical applications, e.g. in CAD of VLSI circuits, now require treatment of very large hypergraphs containing up to hundreds of thousands of nodes and edges. The main drawback of using intermediate graph representations of these hypergraphs is the huge memory requirements.

In this paper, we present an improved labeling algorithm for determining a maximum flow via an augmenting path approach which does not use an intermediate graph representation and *works directly on the hypergraph*. As will be shown in Section 3, this greatly reduces the memory space requirements, and according to our experiments (see Section 4) simultaneously reduces the run time as compared with the labeling of the directed graph. The rest of this paper is organized as follows: In the next section, we introduce an algorithm that computes the maximum flow directly on a hypergraph and determines the minimum cut corresponding to the maximum flow obtained. Section 3 presents an analysis of both time and space complexity of the algorithm and compares it to the complexity of the method proposed by Yang and Wong (1996). In the fourth section, we give some experimental results, followed, in the last section, by a conclusion.

2. Maximum flow computation by direct labeling on the hypergraph

Let $H = (V, E)$ be a hypergraph with $|V|$ nodes and $|E|$ hyperedges. The labeling rules given below may be viewed as implicit labeling rules for the underlying directed graph according to Yang and Wong's representation (1996). The labeling however is carried out explicitly on the hypergraph.

The labeled nodes are organized according to a FIFO (first-in, first-out) list in order to find the shortest augmenting paths in the hypergraph. If an augmenting path exists, the flow on the hyperedges that constitute the augmenting path are updated. Each hyperedge $e_i \in E$ has two labels, one label giving the pointer to the node where the flow comes from (flowFrom(e_i)), and one label giving the pointer to the node it goes to (flowTo(e_i)). The capacity of each hyperedge is equal to one ($c(e_i) = 1$).[4] Furthermore, we limit the flow on each hyperedge to be either 0 (no flow) or 1 (there is a flow of value 1 on the hyperedge).

Each node $v_i \in V$ has two flags. The first flag (label(v_i)) indicates whether the node has been labeled and stored in the FIFO, while the second one (onEdge(v_i)) points to the hyperedge from which the node has been labeled. The algorithm is composed of three phases and proceeds according to the following steps:

Hypergraph labeling algorithm (HLA)

1) Initialization:
   For all hyperedges e_i:
   $f(e_i) = 0$, flowFrom(e_i) = NULL, flowTo(e_i) = NULL
   For all nodes v_i:
   label(v_i) = 0, onEdge(v_i) = NULL
   Label source node s: label(s) = 1
   Push s into the FIFO
2) Iteration:
   While there are nodes in the FIFO and sink node t is unlabeled (label(t) = 0):
   Pop v_i from the FIFO
   Label all non labeled neighbor nodes v_k of node v_i by applying procedure SCAN (see Section 2.1)
   Push v_k into the FIFO
3) If label(t)! = 0:
   For all hyperedges e_j on the augmenting path from s to t:
   Update the flow on hyperedge e_j by applying procedure UPDATE (see Section 2.2)
   For all nodes v_i:
   label(v_i) = 0, onEdge(v_i) = NULL
   Return to 2)
   Else:
   The flow is maximum.
   End of the algorithm.

2.1. Labeling a hyperedge

During the labeling phase for the neighboring nodes of a given node, there are four distinct cases which may arise (presented in Fig. 2).

[4] When there is a hyperedge e_j with weight $w(e_j) > 1$, it can be modeled by w hyperedges with unit weight. Hence, without loss of generality, we can assume that each hyperedge has a capacity of one.

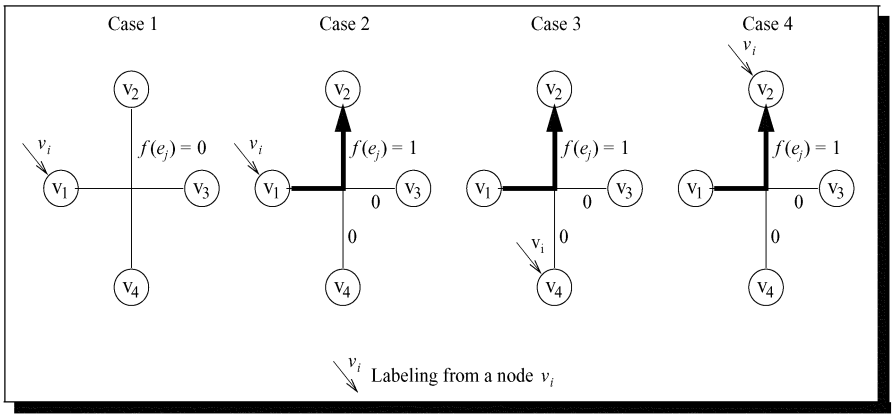

Fig. 2. The four different cases of hyperedge labeling.

The labeling algorithm proceeds according to the following steps:

Procedure SCAN: Scanning a node and adjacent hyperedges

Let v_i be the node extracted from the FIFO list.
For all hyperedges e_j containing node v_i ($v_i \in e_j$):

- Case 1: If there is no flow on hyperedge e_j ($f(e_j) = 0$):
  $\forall v_k \in e_j$, $v_k \neq v_i$, label$(v_k) = 0$, set:
  label$(v_k) = 1$, fromVertex$(v_k) = v_i$, and onEdge$(v_k) = e_j$
- Case 2: If $f(e_j) = 1$ and flowFrom$(e_j) = v_i$:
  Do nothing.
- Case 3: If $f(e_j) = 1$ and flowFrom$(e_j) \neq v_i$ and flowTo$(e_j) \neq v_i$, set:
  $\forall v_k \in e_j$, $v_k =$ flowFrom(e_j), $v_k \neq v_i$, label$(v_k) = 0$:
  label$(v_k) = 1$, fromVertex$(v_k) = v_i$, and onEdge$(v_k) = e_j$
- Case 4: If $f(e_j) = 1$ and flowFrom$(e_j) \neq v_i$ and flowTo$(e_j) = v_i$:
  $\forall v_k \in e_j$, $v_k \neq v_i$, label$(v_k) = 0$, set:
  label$(v_k) = 1$, fromVertex$(v_k) = v_i$, and onEdge$(v_k) = e_j$

2.2. Updating the flow

The updating algorithm is executed, when an augmenting path has been found (i.e. when the sink node t has been reached). The four possible cases which may arise and the updating result in each case are illustrated in Fig. 3.

Let v_i be the current node, $e_j =$ onEdge(v_i) the hyperedge, and $v_k =$ fromVertex(v_i) the node from which the node v_i has been labeled.

Procedure UPDATE: Flow update

- Case 1: If $f(e_j) = 0$:
  Label $f(e_j) = 1$, flowFrom$(e_j) = v_k$, and flowTo$(e_j) = v_i$.
- Case 2: If $f(e_j) = 1$, flowFrom$(e_j) = v_i$, and flowTo$(e_j) = v_k$:

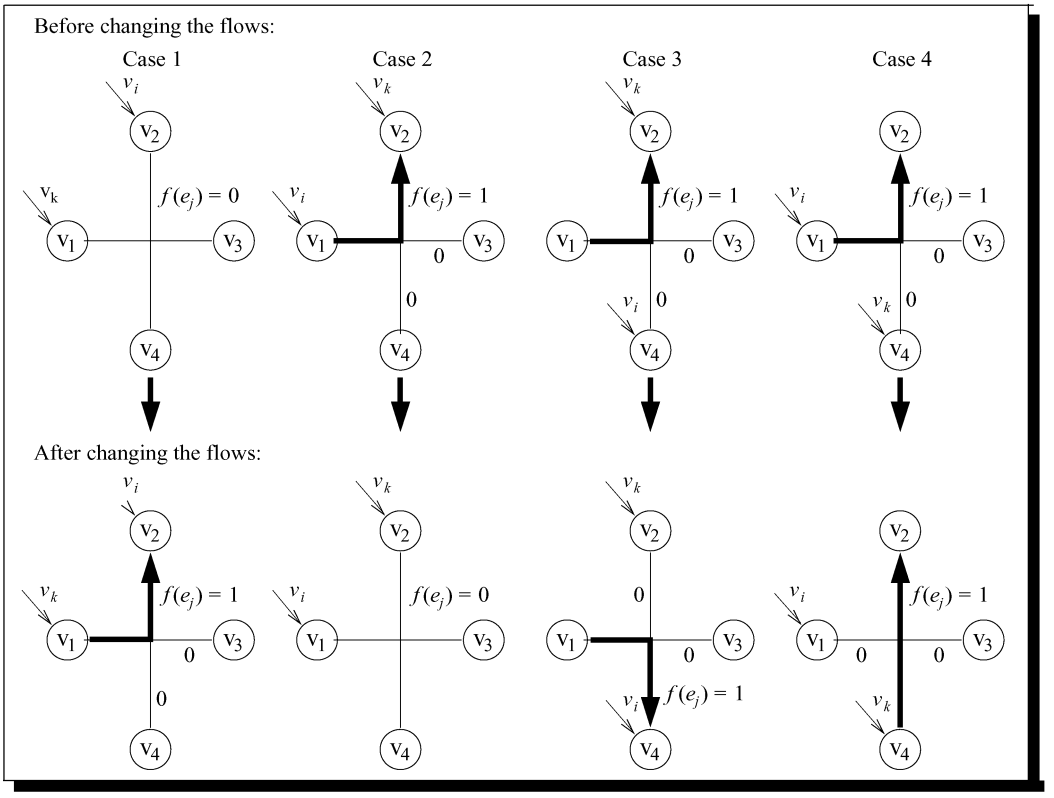

Fig. 3. The four distinct cases of changing the flow on a hyperedge.

Label $f(e_j) = 0$, flowTo$(e_j) =$ NULL, and flowTo$(e_j) =$ NULL.

- Case 3: If $f(e_j) = 1$, flowFrom$(e_j) = v_d$ ($v_d \neq v_i$, $v_d \neq v_k$), and flowTo$(e_j) = v_k$: Label flowTo$(e_j) = v_i$.
- Case 4: If $f(e_j) = 1$, flowTo$(e_j) = v_d$ ($v_d \neq v_i$, $v_d \neq v_k$), and flowFrom$(e_j) = v_i$: Label flowFrom$(e_j) = v_k$.

2.3. Correctness of the algorithm

Yang and Wong (1996) showed that a hypergraph can be transformed into a directed graph and that the minimum cut separating the two distinguished nodes source and sink in this directed graph determined by computing the maximum flow between them corresponds to the minimum cut of the underlying hypergraph separating the same sets of nodes. Edmonds and Karp (1972) used a breadth-first search algorithm to find the shortest augmenting path between source and sink at each iteration in order to determine the maximum flow in a directed graph.

The algorithm presented in this paper corresponds to the algorithm of Edmonds and Karp applied to a hypergraph. Therefore, we need to show that the labeling and flow updating operations carried out on the hypergraph correspond to the labeling and flow updating operations on the directed graph obtained by Yang and Wong's transformation. It can easily be verified that the hyperedge labeling algorithm presented

in Section 2.1 and the flow updating algorithm presented in Section 2.2 label the nodes and update the flow in each of the four cases equivalent to Yang and Wong's algorithm on their overlying directed graph. Therefore, Yang and Wong's algorithm and the algorithm presented above determine both the same maximum flow leading to the same minimum s-t cut separating the same sets of nodes when determining the minimum s-t cut separating the two distinguished nodes source and sink in a hypergraph.

2.4. Determining the minimum cut

The minimum cut of the hypergraph is a by-product of the labeling process. The algorithm which determines the minimum cut of the hypergraph starts from the source node s and labels all nodes on augmenting paths as being part of the same partition. For this, it uses the result of the first labeling that does not find an augmenting path from the source node s to the sink node t.

3. Time and memory space complexity analysis

The maximum flow algorithm searches shortest (i.e. minimum cardinality) augmenting paths between the source and the sink node. The search of one augmenting path is done by breadth-first search (BFS) which has a run time of $O(|E| + |V|)$. Since the capacity of each hyperedge is one, the flow value cannot exceed $|E|$.

Because each augmenting path increases the flow value by at least one, the number of iterations is at most $|E|$. The worst case running time of the maximum flow algorithm is therefore $O(|E|^2)$ based on the hypothesis that $|V| < |E|$. Therefore, our algorithm has the same time complexity as the algorithm of Yang and Wong (1996) (this will be confirmed by our experiments in Section 4 which show that computation time is reduced by a constant factor). Note that in their paper, Yang and Wong observed a complexity of $O(|V||E|)$ based on the hypothesis that $|E| < |V|$. Yang and Wong examine circuit bipartitioning. The set of nodes $v \in V$ consists of cells (look-up tables or gates) and input/output pins. In general every cell and every input pin drives exactly one net $e \in E$, whereas output pins do not. Therefore, Yang and Wong assume $|E| < |V|$. However, when partitioning into multiple circuits on a board, when dealing with memory blocks, or more generally, as soon as the circuit (hypergraph) contains blocks (nodes) that drive multiple nets (hyperedges), we have $|V| < |E|$ (or even $|V| \ll |E|$), which leads to a complexity $O(|E|^2)$ for Yang and Wong's algorithm.

The memory space occupied by our algorithm essentially corresponds to the memory required by the hypergraph which is of $|E|$ hyperedges and $|V|$ nodes having each on average $p * |E|/|V|$ pointers to the hyperedge structure, where p is the average cardinality of hyperedges In the worst case, each hyperedge is connected to $p = |V|$ nodes, thus leading to a memory space occupation of $O(|V||E|)$.

The algorithm of Yang and Wong needs additional memory space for the directed graph $G' = (V', E')$ with $|E'| = (2p + 1)|E|$ edges and $|V'| = |V| + 2|E|$ nodes. Each edge of the directed graph is pointed by its two adjacent nodes which leads to $2 * |E'|$ pointers to the struct edge. Therefore, the memory space occupied by this algorithm[5] corresponds to $|E| + (2p + 1)|E| = (2p + 2)|E|$

[5] We reasonably assume, that a hyperedge takes approximately the same amount of memory space as an edge and that a node in the hypergraph the same amount of memory as a node in the hypergraph.

hyperedges, $|V| + |V| + 2|E| = 2|V| + 2|E|$ nodes, and $p|E| + 2 * (2p + 1)|E| = (5p + 2)|E|$ pointers to the struct hyperedge. In the worst case, each hyperedge is connected to $p = |V|$ nodes, leading to a memory space occupation of $O(|V||E|)$.

Although both algorithms have the same worst case memory space occupation, Yang and Wong's algorithm requires:

- $(2p + 2)$ times more hyperedges;
- $(2|E|/|V| + 2)$ times more nodes;
- $(5 + 2/p)$ times more pointers.

Typical values for these variables and their relation are:

- p is usually between 2 and 3 for sparse hypergraphs that represent netlists:
- $|E|/|V|$ varies between 1 and 100 (between 20 and 70 in the test cases used in Section 4);
- the number of bytes required to implement the data type hyperedge is equal to the number of bytes required to implement the data type node which is 10 to 100 times the number of bytes required to implement a pointer;
- $|V|$ is dependent on the problem size and might vary for today's applications between 10 and 100 000 (between 20 and 2200 in the test cases used in Section 4).

In the best case ($p = 2$, $|E|/|V| = 1$, $|V| = 10$, and a hyperedge and a node require 100 times more memory space than a pointer) Yang and Wong's algorithm *uses at least a constant factor of five times more memory space than our algorithm*.

4. Experimental results

We implemented both our algorithm (HLA) and the algorithm of Yang and Wong (1996). Both algorithms have been applied to large sparse hypergraphs with $|V|$ nodes, $|E|$ hyperedges, and an average cardinality p of each hyperedge. We compared the performances of executing one maximum flow computation in terms of run time (t) and memory requirements (m) on a set of test problems from VLSI applications involving hypergraphs with up to 2123 nodes and 98 915 hyperedges. The run time and the memory space requirements have been measured before and after maximum flow computation[6] by using the tool Quantify (Rational Software Corporation, 2000). The results displayed here are the difference between these measured values. The run time is displayed in units of 10^6 clock cycles, the memory occupied in KBytes. Note that both algorithms have the same source and sink node and found the same shortest augmenting paths in the hypergraph leading to the same minimum cut. The hypergraphs used for these experiments are from industrial VLSI designs (N7–H11) and from generated benchmarks presented in Pistorius, Legai and Minoux (1999) (H1–H6). H12 and H13 have been derived from H11 by randomly increasing the cardinality p of the hyperedges. All these designs have been implemented on programmable circuits or on boards containing multiple programmable circuits to be used in a hardware emulator. The results are presented in Table 1.

[6] The run time of Yang and Wong's algorithm includes the construction of the graph which takes less than 5% of the total run time.

Table 1
Results of the application of the maximum flow algorithm

Hypergraph				YW		HLA		Ratio	
Ident	$\|V\|$	$\|E\|$	p	t (10^6 cycles)	m (KB)	t (10^6 cycles)	m (KB)	t(HLA)/t(YW)	m(HLA)/m(YW)
H1	20	531	2.35	9.1	264	3.1	32	0.34	0.12
H2	51	3531	2.05	94.5	1632	30.0	176	0.32	0.11
H3	64	3750	2.61	124.4	1992	48.6	200	0.39	0.10
H4	192	4988	2.47	66.1	2584	32.1	264	0.49	0.10
H5	245	11417	2.55	65.7	6000	35.7	592	0.54	0.10
H6	265	14256	2.38	138.0	7160	68.5	720	0.50	0.10
H7	407	8302	2.87	66.7	4584	60.2	464	0.90	0.10
H8	419	9410	2.75	158.1	5000	99.1	512	0.63	0.10
H9	678	14913	2.84	53.1	8224	32.8	824	0.62	0.10
H10	1389	46016	2.78	1412.8	25504	274.2	2448	0.66	0.10
H11	2123	98915	2.55	1266.8	51808	887.3	5080	0.70	0.10
H12	2123	98915	5.45	486.9	77864	199.8	5888	0.41	0.08
H13	2123	98915	10.58	2694.3	126566	1089.0	7408	0.40	0.06
						Average ratio between HLA and YW		0.53	0.1

These results show that our algorithm not only outperforms the algorithm of Yang and Wong in terms of memory space, but is also practically more efficient in terms of computing time. On average, we use 0.53 of the CPU time needed by the algorithm of Yang and Wong (1996), and in only one case do we find a factor greater than 0.7. Concerning the memory space occupation, we use an average of ten times less memory space than the algorithm of Yang and Wong. Note that, consistently with our theoretical analysis, the improvement factor in memory space increases with the average cardinality of the hyperedges, and that this number is relatively small for the hypergraphs H1–H11. Thus, the examples displayed are not especially favorable for the comparison of our algorithm with the algorithm of Yang and Wong.

5. Conclusions and perspectives

We have described an augmenting path algorithm that determines the maximum flow and the corresponding minimum cut directly on a hypergraph without transforming it into a directed graph. We have compared our approach in terms of both computing time and memory space requirements with the state-of-the-art approach of Yang and Wong (1996). Our experimental results confirm the theoretical analysis, as we obtain for all test cases better results with our approach in terms of both run time and memory space occupation.

However, any maximum flow algorithm, not restricted to augmenting path approaches can be applied to the directed graph stemmed from Yang and Wong's transformation. The algorithm proposed in this paper does not have this property. Nevertheless, as our algorithm implicitly uses the underlying directed graph from Yang and Wong (1996), we may expect that more efficient maximum flow algorithms (e.g.

the one described in Goldberg and Rao, 1997) could be implemented directly on hypergraphs in a way similar to what we have proposed here for the case of augmenting path methods. This remains an open area for future research.

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B., 1993. *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs.

Alpert, C.J., 1996. 'Multi-way Graph and Hypergraph Partitioning', Ph.D. Dissertation, University of California, Los Angeles.

Beasley, J.E., 2000. 'OR-Notes' http://www.ms.ic.ac.uk/jeb/or/netflow.html

Cambini, R., Gallo, G., Scutella, M.G., 1997. 'Flows on Hypergraphs', *Mathematical Programming* 78, 195–217.

Dinic, E.A., 1970. 'Algorithms for Solution of a Problem of Maximum Flow in Networks with Power Estimation', *Soviet Math. Doklady* 11, 1277–1280.

Edmonds, J., Karp, R.M., 1972. 'Theoretical improvements in algorithmic efficiency for network flow problems', *Journal of the Association for Computing Machinery* 19(2), 248–264.

Ford, L.R. Jr., Fulkerson, D.R., 1962. *Flows in Networks*, Princeton University Press, Princeton, NJ.

Goldberg, A.V., Tarjan, R.E., 1988. 'A New Approach to the Maximum-Flow Problem', *Journal of the Association for Computing Machinery*, 35(4), 921–940.

Goldberg, A.V., Rao, S., 1997. 'Beyond the Flow Decomposition Barrier', *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 2–11.

Gomory, R.E., Hu, T.C., 1961. 'Multi-Terminal Network Flows', *Journal of the SIAM* 9, 551–570.

Hu, T.C., 1963. 'Multi-Commodity Network Flows', *Operations Research* 11, 344–360.

Hu, T.C., Moerder, K., 1985. 'Multiterminal Flows in a Hypergraph', In: T.C. Hu and Ku (Eds), *VLSI Circuit Layout: Theory and Design* pp. 87–93. IEEE Press.

Ihler, E., Wagner, D., Wagner, F., 1993. 'Modeling Hypergraphs by Graphs with the Same Mincut Properties', *Information Processing Letters* 45(4); 171–175.

Kuo, M.-T., Cheng, C.-K., 1997. 'A Network Flow Approach for Hierarchical Tree Partitioning', *Proc. 34th ACM/IEEE Design Automation Conference*, 512–517.

Lawler, E.L., 1973. 'Cutsets and Partitions of Hypergraphs', *Networks* 3, 275–285.

Lawler, E.L., 1976. *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart & Winston.

Liu, H., Wong, D.F., 1998. 'Network Flow Based Multi-Way Partitioning with Area and Pin Constraints', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17(1), 50–59.

Pistorius, J., Legai, E., Minoux, M., 1999. 'Generation of Very Large Circuits to Benchmark and Partitioning of FPGAs', *Proc. ACM International Symposium on Physical Design*, 67–73.

Rational Software Corporation, 2000. 'Quantify User's Guide' Rational Software Corporation, http://www.rational.com

Rardin, R.L., 1997. *Optimization in Operations Research*, Prentice Hall, Englewood Cliffs.

Yang, H.H., Wong, D.F., 1996. 'Efficient Network Flow Based Min-Cut Balanced Partitioning', *IEEE Transactions on Computer-Aided Design* 15(12), 1533–1540.