VLSI Design
1994, Vol. 1, No. 3, pp. 181–192
Reprints available directly from the publisher
Photocopying permitted by license only

Cluster Partitioning Techniques for Data Path Synthesis

CHIEN-IN HENRY CHEN
Dept. of Electrical Engineering, Wright State University, Dayton, OH

GERALD SOBELMAN
Dept. of Electrical Engineering, University of Minnesota, Minneapolis, MN

(Received July 17, 1989, Revised July 10, 1990)

New, efficient algorithms for the automated synthesis of buses in data path design are presented. Modifications to the technique of Generalized Clique Partitioning (GCP) are discussed which lead to better designs and reduced computation time in large synthesis problems. The new approach, Weighted Cluster Partitioning (WCP), eliminates the need for backtracking. The algorithm guides the process of bus formation by assigning a higher weight to those interconnection units that should be combined first. The operation of the WCP algorithm is clearly demonstrated using a detailed example.

A modified priority ordering in the selection of the candidate pair is also discussed which can improve the performance of GCP and WCP. We demonstrate that GCP II performs better than GCP, while WCP II consistently produces the best results of all these algorithms on a set of large synthesis examples.

Key Words: *Data path; Synthesis; Clique partitioning; Resource allocation*

INTRODUCTION

High-level data path synthesis is concerned with the automatic generation and allocation of registers, data operators and buses. Many approaches to automated data path synthesis have been proposed in the literature [1]–[10]. A particularly interesting and powerful approach to all phases of data path synthesis has been proposed by Tseng and Siewiorek [1]. In their approach, the problems of register, operator and bus synthesis are cast into the "Clique Partitioning" (CP) problem in which it is necessary to partition the nodes of a compatibility graph into a set of disjoint clusters. Their objective is to partition the nodes of this graph in such a way that the minimal number of disjoint clusters is obtained. In terms of the corresponding data path element, this implies that the minimal hardware cost for a given degree of system concurrency is achieved.

As further discussed in [1], the use of CP alone is insufficient to obtain the best design possible. This is due to the fact that, under this simple algorithm, it is not possible to distinguish among many designs that are not all equivalent when further considerations are taken into account. "Generalized Clique Partitioning" (GCP) was introduced by these authors to provide for a more precise selection of design alternatives. While the GCP algorithm results in considerably better designs than simple CP, the time complexity of the algorithm is significantly greater and involves following multiple paths with some backtracking.

Another heuristic solution to the clique problem was proposed in [11] in which a suboptimal covering procedure was implemented. The simulations show that the computation is quite excessive when the algorithm generates a large number of cliques which produces enormous covering tables. In [12], a further heuristic algorithm which generates a sub-optimal solution without enumerating all of the cliques from the partitioned graph was proposed. This approach has a smaller run time, however, it may result in a locally minimum solution, i.e., in some worse cases, the algorithm may generate the solutions which have 15% over the optimal solutions.

In this paper, we propose a more efficient algorithm than GCP for application to the data path synthesis problem. Our algorithm, Weighted Cluster

Partitioning (WCP), yields the same or better results as GCP in a considerably reduced computation time. We will illustrate the technique by focusing on the problem of bus system synthesis. However, with suitable modifications, WCP can also be applied to the problems of register and operator synthesis. We will present large synthesis examples which show that results superior to those of GCP can be obtained in a shorter time using our procedures.

The remainder of this paper is organized as follows. The Weighted Cluster Partitioning algorithm will be introduced and developed for the case of the bus synthesis problem. The steps of the algorithm will be illustrated through a detailed example. Implementation aspects will be discussed, and the results of applying the algorithm to several large synthesis problems will be given. Following this, a modification to the WCP and GCP algorithms will be presented which leads to even better results. The modified WCP algorithm, "WCP II," is seen to produce the best results for the set of large examples considered. The results of our work are summarized and illustrations of applying the "WCP" algorithm of the synthesis example of Ref. [1] are presented in the Appendix.

WEIGHTED CLUSTER PARTITIONING

The goal of bus system synthesis is to provide the required communications paths between the registers and data operators in such a way that the number of interconnections and gating elements (i.e., multiplexers) that are required is minimized. Common buses are an extremely efficient structure for achieving area minimization so that the bus-oriented interconnection of registers and data operators plays an important role in data path synthesis. Two interconnection units can be grouped together into a common bus if they are not required to be used simultaneously. Frequently, the benefit attained by grouping some set of interconnection units into a bus will be greater than that obtained by grouping another set of interconnection units into a bus. Furthermore, the grouping of one set of interconnection units into a bus may preclude the subsequent grouping of another set. Hence, it is important to select those groupings that will result in the most benefit or "profit" in the final design.

In order to describe the WCP algorithm, we must first introduce some terminology from graph theory [14]. A graph is called *complete* if all of its vertices are connected to all of the other vertices in the graph. A *clique* is a maximal complete subgraph of a graph. In other words, a clique is a complete subgraph that is not contained within any other subgraph that is also complete. The search for cliques in a graph has been proved to be NP-complete; related research can be found in [15–19]. The heuristic CP algorithm proposed in [1] uses the "neighborhood property" among vertices to partition a graph into the minimum number of disjoint clusters. It should be pointed out that the resulting clusters obtained are not necessarily all cliques, as the cliques of a graph are not required to be disjoint. (See for example the cliques of the Moon-Moser graphs as given in [13].) We prefer the term "cluster partitioning" for this reason. The time complexity of this algorithm is a polynominal function of the number of nodes and edges in the graph. However, direct use of CP is insufficient for bus system design as more than one suboptimal solution will be found and there is no way to distinguish between them and an optimal solution.

In the WCP algorithm, a weight is assigned to each edge in the compatibility graph. The weight is a measure of the benefit of "profit" to be gained by combining the two interconnection units into a common bus. The value of the weight is used to select the best choice among a set of otherwise equally viable candidates. The weights of edges are updated as the algorithm is executed to properly reflect the changing relative values of combining the remaining interconnection units into buses. In this way, WCP is able to avoid the sub-optimal solutions obtained using CP. Note that whereas GCP maintains a separate graph for those interconnection units having a common source or destination, only a single weighted graph G is required in WCP.

We begin with the definition of the weighted graph G:

Definition 1: G is a *weighted graph* $< V, E >$, where V is a set of vertices and E is a set of weighted edges $E = \{(v_1, v_2, w) \mid v_1, v_2 \in V$ and w is a weight$\}$.

In the following discussion, we use the code sequence of Ref. [1], shown in Table I, as a running example. The indices assigned to the different interconnection units for the weighted graph G of the code sequence are listed in Table II.

TABLE I
Example Code Sequence of Ref. [1]

S1: V3 = V1 $+_1$ V2; V12 = V1
S2: V5 = V3 $-_1$ V4; V2 = V3 x_1 V6
S3: V3 = V3 $+_2$ V5; V2 = V1 $+_3$ V2; V5 = V10 $/_1$ V5
S4: V1 = V3 AND_1 V5; V2 = V12 OR_1 V2

ALU1: { $+_1$, x_1, $+_3$, OR_1 }
ALU2: { $-_1$, $+_2$, AND_1 }
ALU3: {$/_1$ }

TABLE II
Interconnection Units of the Example Code Sequence

Source	Destination	Index No.	Executed Cycle
V1	V12	1	1
V1	ALU1.IN1	2	1, 3
V2	ALU1.IN2	3	1, 3, 4
V3	ALU1.IN1	4	2
V3	ALU2.IN1	5	2, 3, 4
V4	ALU2.IN2	6	2
V5	ALU2.IN2	7	3, 4
V5	ALU3.IN2	8	3
V6	ALU1.IN2	9	2
V10	ALU3.IN1	10	3
V12	ALU1.IN1	11	4
ALU1.OUT	V2	12	2, 3, 4
ALU1.OUT	V3	13	1
ALU2.OUT	V1	14	4
ALU2.OUT	V3	15	3
ALU2.OUT	V5	16	2
ALU3.OUT	V5	17	3

Three steps are required for establishing the weighted graph for the interconnection units. In the process, we must construct two intermediate sets of weighted graphs. These three steps are summarized below.

Step 1: *Establish the weighted graph G1 in which two interconnection units form a weight-0 edge if they represent register-to-register, register-to-operator, or operator-to-register data transfers that do not occur during the same clock cycle. Remove any edges that correspond to a pair of interconnection units having a different source and different inputs of the same operator as their destination.*

The weight-0 edges of G1 of the example sequence in Table I are as follows:

(1,4) (1,5) (1,6) (1,7) (1,8) (1,9) (1,10) (1,11) (1,12) (1,14) (1,15) (1,16) (1,17)
(2,4) (2,6) (2,11) (2,14) (2,16)
(3,6) (3,9) (3,16)
(4,7) (4,8) (4,10) (4,11) (4,13) (4,14) (4,15) (4,17)
(5,13)
(6,7) (6,8) (6,10) (6,11) (6,13) (6,14) (6,15) (6,17)
(7,9) (7,13) (7,16)
(8,9) (8,11) (8,13) (8,14) (8,16)
(9,10) (9,13) (9,14) (9,15) (9,17)
(10,11) (10,13) (10,14) (10,16)
(11,13) (11,15) (11,16) (11,17)
(12,13)
(13,14) (13,15) (13,16) (13,17)
(14,15) (14,16) (14,17)
(15,16)
(16,17)

Note that the three edges (2,9), (3,4) and (9,11) have been removed since they correspond to cases in which the pair has a different source and a different input of the same ALU as their destination. There is no benefit to be gained in combining such interconnection units into a bus.

Step 2: *Establish the weighted graph G2 in which two interconnection units can form a weight-1 edge if either of the following two criteria are satisfied:*

(1) Both interconnection units have the same source.

(2) Both interconnection units have the same destination, provided that they are not both active during the same cycle.

Note that in the first case where the two interconnection units have the same source, we do *not* insist that the two interconnection units be active during different cycles. The weight-1 edge of G2 are listed as follows:

(1,2) (2,4) (2,11) (3,9) (4,5) (4,11) (6,7) (7,8) (12,13) (13,15) (14,15) (14,16) (15,16) (16,17)

Step 3: *Establish the final weighted graph G as follows: G consists of the union of the weighted edges in the two graphs G1 and G2. If an edge appears in both G1 and in G2, it becomes a weight-1 edge in the composite graph G.*

The final weighted graph G is shown in Figure 1.

The second part of the syntehsis process is to apply the WCP algorithm to the weighted graph G obtained by the above 3-step process. First, however, we must define some additional terms:

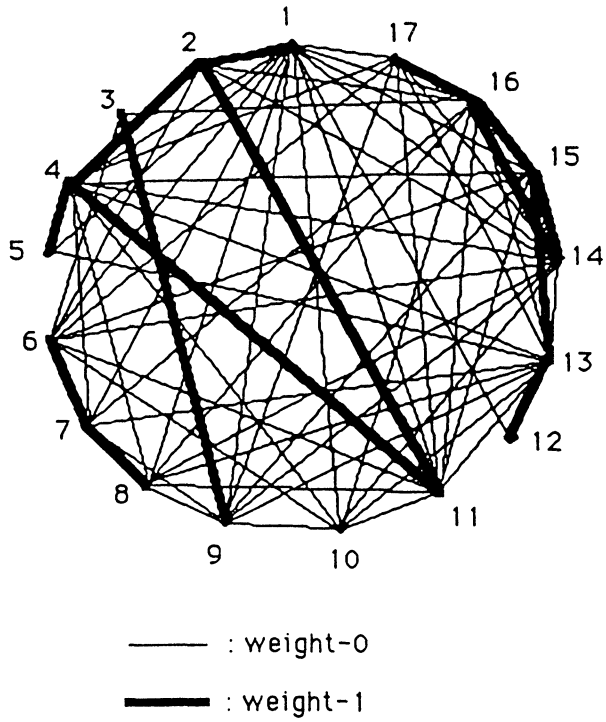

FIGURE 1 Weighted Graph *G* for the code sequence of Table I.

Definition 2: A node is said to be a *composite node* (i, j) if node i and node j are partitioned into a common cluster.

Definition 3: A node k is said to be a *common neighbor* of an edge (i, j), $i < j$ if k is connected to both nodes i and j.

Definition 4: A *deleted edge* of a composite node (i, j) can arise in either of the following three ways:

(1) A node k which is not a common neighbor of i and j will no longer be connected to the composite node (i, j). Thus, the edge (i, k) or (j, k) has to be deleted, as appropriate.

(2) A node k which is a common neighbor of i and j will still be connected to the composite node (i, j). However, only one of the original two edges needs to be retained. Thus, one of the edges (i, k) or (j, k) has to be deleted. In our algorithm, the edge (j, k) will be deleted if $j > i$. Otherwise, the edge (i, k) will be deleted. When one of these two edges is deleted, the weight of the retained edge is updated in the following way. Its new weight is the sum of its previous weight plus the weight of the edge (i, j) and the weight of the deleted edge.

(3) The edge (i, j) itself must, of course, be deleted.

In the discussion of the WCP operation, the above three situations in which edges may be deleted will be referred to as "case-1 deletions," "case-2 deletions" and "case-3 deletions," respectively.

Definition 5: A *candidate pair* (p, q) for the weighted graph G is determined by the following criteria (listed in order of priority):

(1) (p, q) has the maximum number of common neighbors.

(2) (p, q) has the minimum number of deleted edges.

(3) (p, q) has the maximum weight.

The WCP algorithm can now be stated as follows:

Step 1: *Traverse the list of edges in G. For each edge (i, j), $i < j$, compute the number of common neighbors, the number of deleted edges and record its weight.*

Step 2: *Find the candidate pair (p, q). Choose the smaller of p and q to be the head of a new cluster. Remove the deleted edges of the composite node (p, q). Update the list of edges accordingly and recompute the number of common neighbors and the number of deleted edges for all remaining edges. Update the weight of any remaining edges that were affected by any "case-2 deletions." If the list of edges is empty, then WCP is complete.*

Step 3: *Assume that p is the head of the current cluster. Find a candidate pair which joins node p and another node r. Choose the smaller of p and r to be the head of the current cluster. Remove the deleted edges of the composite node (p, r) or (r, p). Update the list of edges accordingly and recompute the number of common neighbors, the number of deleted edges and the weight of each remaining edge. If the list of edges is empty, then WCP is complete. Otherwise, if node p (or r if $r < p$) no longer appears in the updated list of edges, then go to step 2 and start to form another cluster. Otherwise, repeat step 3 and continue to find other nodes to add to the current cluster.*

WCP partitions the interconnection units into eight clusters (i.e. eight buses). This partitioning is the same as the partitioning obtained using the GCP algorithm of Ref [1]. These partitions are listed below, and the data path design is shown in Figure 2.

1: (1,2,4,11), (3,9), (5), (6,7,8), (10), (12), (13,14,15,16), (17)

An illustration of the WCP algorithm applied to the weighted graph G of the same example code sequence to find the best design possible is given in the Appendix. The steps of the WCP algorithm for G are shown in Tables IV–XII. However, using CP, seven additional partitionings into eight clusters are possible:

2: (1,2,4,11), (3), (5), (6,13,14,15), (7,8,9), (10), (12), (16,17)

3: (1,2,4,11), (3,9), (5), (6,7,8), (10,13,14,16), (12), (15), (17)

4: (1,2,4,11), (3), (5), (6,7,8), (9,10), (12), (13,14,16,17), (15)

5: (1,2,4,11), (3,6), (5), (7,8,9), (10), (12), (13,14,16,17), (15)

6: (1,2,4,11), (3,10), (5), (6), (7,8,9), (12), (13,14,16,17), (15)

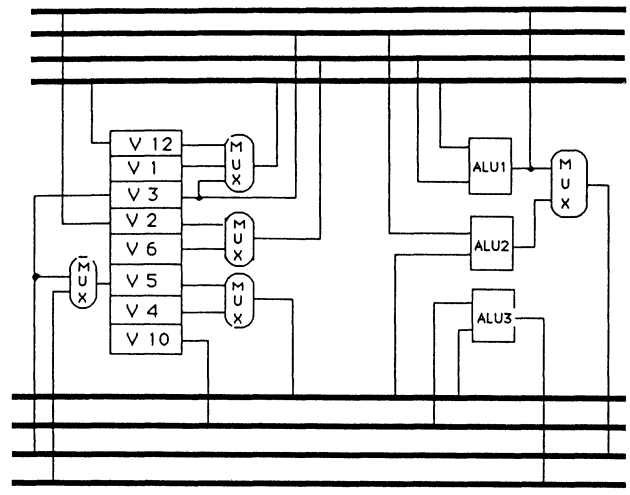

FIGURE 2 Data path synthesized by WCP.

TABLE III
The Results Obtained for P = 0.3 to 0.7

Nodes in G	Edges in G	Weight—1 Edges in G	Number of Clusters			
			GCP	GCPII	WCP	WCPII
		305	7	7	6	5
		394	7	7	6	5
50	982	496	7	6	6	5
		583	7	7	6	5
		686	8	6	6	5
		836	18	16	15	14
		1123	17	15	15	14
100	2938	1436	17	16	15	14
		1752	17	16	15	14
		2068	17	16	15	14
		2377	18	17	15	14
		3173	18	17	15	15
150	7948	3963	17	16	15	14
		4786	16	16	15	14
		5586	16	15	15	14
		3486	28	28	25	23
		4694	28	28	25	23
150	11872	5877	27	27	25	23
		7111	28	28	25	23
		8348	26	25	25	23
		5587	35	32	30	28
		7468	31	31	30	28
200	18630	9319	31	31	30	28
		11243	34	31	29	28
		13127	31	29	29	28

7: (1,2,4,11), (3), (5), (6,7,8) (9,10,13,14), (12), (15), (16,17)
8: (1,2,4,11), (3), (5), (6,7,8) (9,13,14,17), (10), (12), (15,16)

TABLE IV
Step 1 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
	(1,2)	5	14	0	
CP→	(1,4)	9	15	0	←WCP
	(1,5)	1	15	0	
	(1,6)	8	16	0	
	(1,7)	5	15	0	
	(1,8)	7	15	0	
	(1,9)	6	16	0	
	(1,10)	6	15	0	
	(1,11)	8	15	0	
	(1,12)	0	15	0	
	(1,14)	9	15	0	
	(1,15)	6	15	0	
	(1,16)	8	16	0	
	(1,17)	6	15	0	
	(2,4)	3	13	1	
	(2,6)	3	13	0	
	(2,11)	4	11	1	
	(2,14)	4	12	0	
		.			
		.			
		.			
	(14,17)	6	12	0	
	(15,16)	4	14	1	
	(16,17)	4	14	1	

TABLE V
Step 2 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
	(1,2)	2	11	2	
	(1,5)	0	10	1	
	(1,7)	1	13	0	
	(1,8)	3	13	0	
	(1,10)	2	13	0	
CP→	(1,11)	5	12	1	←WCP
	(1,14)	5	13	0	
	(1,15)	2	13	0	
	(1,17)	2	13	0	
	(2,6)	2	12	0	
	(2,11)	3	10	1	
	(2,14)	3	11	0	
		.			
		.			
		.			
	(14,17)	5	11	0	
	(15,16)	3	13	1	
	(16,17)	3	13	1	

Though these eight different partitionings have the same number (8) of buses, they lead to different data paths respectively. The difference between them is in the number and sizes of the multiplexers that are required in each case. Cluster 1's data path will require 5 multiplexers, consisting of four 2x1-MUX's and one 3×1-MUX. However, the other seven data paths generated by CP will require six or more multiplexers.

From the above example, we have seen how WCP improves upon CP by partitioning the interconnection units to form "better" clusters. The hardware cost (ie, multiplexers) and chip area are thereby reduced. While the above example serves to illustrate the operation of the WCP algorithm, it is too small to fully demonstrate the capabilities of the WCP algorithm. These capabilities are shown in the next section.

TABLE VI
Step 3 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
CP→	(1,2)	1	8	0	←WCP
	(1,8)	1	11	0	
	(1,10)	1	10	0	
	(1,15)	1	10	0	
	(1,17)	1	10	0	
	(2,6)	0	11	1	
	(2,14)	0	10	2	
		.			
		.			
		.			
	(14,17)	4	10	0	
	(15,16)	2	12	1	
	(16,17)	2	12	1	

TABLE VII
Step 4 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
	(3,6)	0	10	0	
	(3,9)	0	10	1	
	(3,16)	0	10	0	
	(5,13)	0	11	0	
	.				
	.				
	.				
	(10,16)	2	10	0	
	(12,13)	0	11	1	
CP→	(13,14)	7	11	0	←WCP
	(13,15)	4	11	1	
	(13,16)	6	12	0	
	(13,17)	4	11	0	
	(14,15)	4	8	1	
	(14,16)	5	10	1	
	(14,17)	4	8	0	
	(15,16)	2	10	1	
	(16,17)	2	10	1	

IMPLEMENTATION ASPECTS AND RESULTS: DATA PATH SYNTHESIS

The WCP algorithm described in the previous section has been implemented in C code. *Nodelist* is a two-dimensional linked list data structure which is used to store the nodes of the weighted graph. Each hor-

TABLE VIII
Step 5 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
	(3,6)	0	9	0	
	(3,9)	0	9	1	
	(3,16)	0	9	0	
	(6,7)	1	9	1	
	(6,8)	2	9	0	
	(6,10)	1	9	0	
CP→	(6,13)	4	9	0	
	(6,15)	1	9	0	
	(6,17)	1	9	0	
	(7,8)	3	5	1	
	(7,9)	1	9	0	
	(7,16)	1	9	0	
	(8,9)	2	9	0	
	(8,13)	3	8	0	
	(8,16)	2	9	0	
	(9,10)	1	9	0	
CP→	(9,13)	4	9	0	
	(9,15)	1	9	0	
	(9,17)	1	9	0	
	(10,13)	3	7	0	
	(10,16)	1	9	0	
	(13,15)	3	7	2	
CP→	(13,16)	4	9	1	←WCP
	(13,17)	3	7	0	
	(15,16)	1	9	1	
	(16,17)	1	9	1	

TABLE IX
Step 6 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
	(3,6)	0	7	0	
	(3,9)	0	7	1	
	(6,7)	1	7	1	
	(6,8)	1	8	0	
	(6,10)	0	8	0	
	(6,15)	0	8	0	
	(6,17)	0	8	0	
	(7,8)	2	4	1	
	(7,9)	1	7	0	
	(8,9)	1	8	0	
	(8,13)	0	7	1	
	(9,10)	0	8	0	
	(9,15)	0	8	0	
	(9,17)	0	8	0	
CP→	(10,13)	0	6	1	
CP→	(13,15)	0	6	4	←WCP
CP→	(13,17)	0	6	2	

TABLE X
Step 7 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
	(3,6)	0	6	0	
	(3,9)	0	6	1	
	(6,7)	1	6	1	
	(6,8)	1	6	0	
	(6,10)	0	6	0	
	(6,17)	0	6	0	
CP→	(7,8)	2	3	1	←WCP
	(7,9)	1	6	0	
	(8,9)	1	6	0	
	(9,10)	0	6	0	
	(9,17)	0	6	0	

TABLE XI
Step 8 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
	(3,6)	0	5	0	
	(3,9)	0	5	1	
CP→	(6,7)	0	5	2	←WCP
	(6,10)	0	5	0	
	(6,17)	0	5	0	
CP→	(7,9)	0	5	1	
	(9,10)	0	5	0	
	(9,17)	0	5	0	

TABLE XII
Step 9 of WCP in G

	Edges	Common Neighbors	Deleted Edges	Weight	
CP→	(3,9)	0	3	1	←WCP
CP→	(9,10)	0	3	0	
CP→	(9,17)	0	3	0	

izontal list contains the nodes that have been coalesced into a cluster. The head node of the list has a field *wt* to record the total weight of the current cluster. Initially, each node of the weighted graph G occupies a horizontal list having total weight 0. When two nodes are to be grouped into a cluster, they are coalesced into a horizontal list. The total weight of the edges in the cluster is updated in the head node's *wt* field. The set of distinct clusters are linked by the vertical list. *Edgelist* is a two-dimensional array of records which are used to store the edges in the graph. Each edge's record contains five fields. The field *flag* is a Boolean variable which is true whenever the edge is in the current partitioned weighted graph. Otherwise, it is false. The field *cnbr* stores the number of common neighbors of an edge. The field *excl* store the number of deleted edges. The field *wt* stores the weight of current edge. Finally, the field *mark* is a Boolean variable which is used in computing the number of deleted edges of every pair of nodes in the current weighted graph. Initially, every edge's *mark* field is set to false. While computing the number of deleted edges of a composite node (i, j) $i < j$, a node k which is connected to both i and j is still connected to the composite (i, j). However, only one edge (i, k) is retained. The other edge (j, k) has to be deleted. Therefore, we change the *mark* field of the edge (i, k) to be true so that (i, k) will not be counted in the number of deleted edges of (i, j). Moreover, by identifying the *mark* field, the edge (i, k) will be retained after the deleted edges of the candidate pair (i, j) have been removed.

The GCP and WCP algorithms were executed for several test examples on a SUN 3/160 workstation. The large graphs considered here were generated from a random graph generator program. In this program, edges having the same source or the same destination are generated with a probability P. Figures 3–7 show the execution times for GCP and WCP with probability P = 0.3, 0.4, 0.5, 0.6 and 0.7, respectively. (The meaning of the results shown for "WCP II" and "GCP II" will be explained in the following section.) These figures show that the execution time for WCP is less than that for GCP in all cases.

Table III shows the number of clusters generated for each of these cases. In this table, for each vlaue of the number of nodes and edges in G, there are

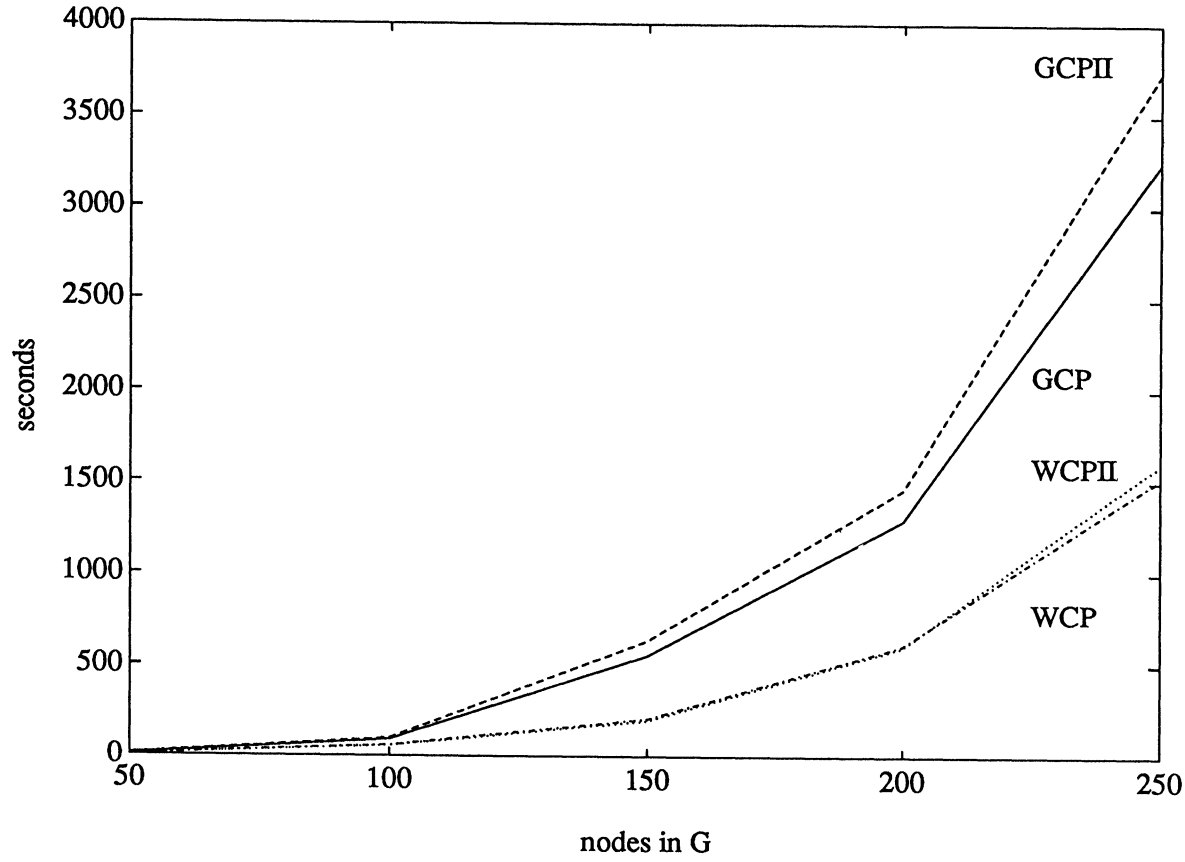

FIGURE 3 Execution times for P = 0.3.

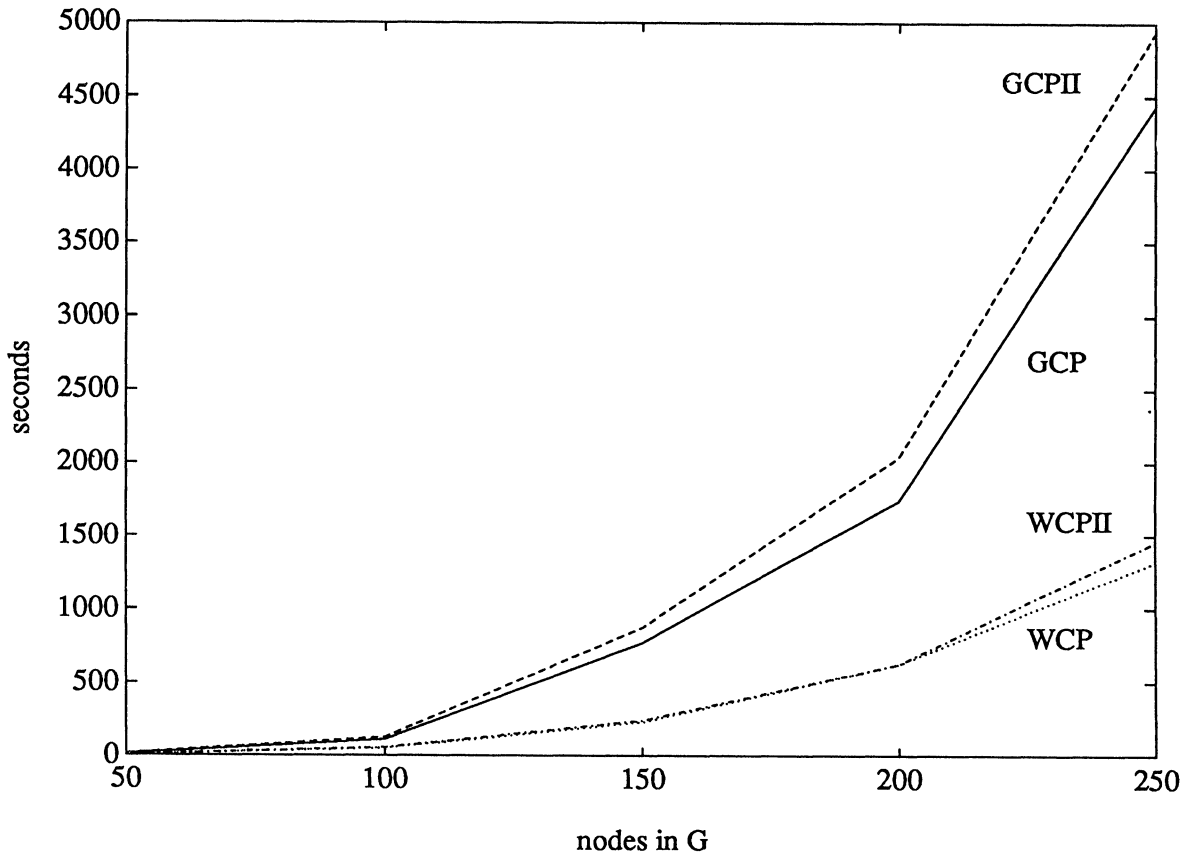

FIGURE 4 Execution times for P = 0.4.

five separate lines of numerical data. Each of these five lines corresponds to the results for P = 0.3, 0.4, 0.5, 0.6 and 0.7, respectively. Note that in all cases, WCP produces a fewer number of clusters than GCP. Therefore, designs synthesized using WCP would require a fewer number of buses than those that were synthesized using GCP.

Finally, we consider the computational complexity of our procedure. Let the number of nodes and edges of a graph be represented by N and E, respectively. A simple analysis of the operations described in Steps 1–3 of the WCP algorithm shows that the computational complexity is $O(NE) + O(N^2)$. This compares favorably with the computational complexity of GCP, which has been shown to be $O(N^3) + O(N^2E) + O(NE^2)$ [20].

GCP II AND WCP II: ALGORITHM AND EXAMPLES

In the previous sections, we have illustrated through several large examples that WCP is more efficient and can generate fewer buses than GCP. In this section, we discuss a simple modification of the GCP and WCP algorithms that has been found to produce even better results in many cases.

Recall that in both the GCP and the WCP algorithms, the highest priority for the selection of the candidate pair is the edge (p, q) which has the maximum number of common neighbors. This has the effect of including the largest possible number of edges into the current cluster, a desirable result. On the other hand, having the minimum number of deleted edges as a lower priority may jeopardize the possibilities for forming other large clusters at a later stage of the process. Our overall objective is to find the minimum number of disjoint clusters that will cover all the nodes of the initial weighted graph. Thus, one may expect that the local optimizations produced by the GCP and WCP algorithms will act to limit the degree of global optimization that can be achieved.

To investigate this possibility, we have tested the same set of large examples that were considered in the previous section using modified procedures. Specifically, we define the algorithms for GCP II and WCP II to be the same as GCP and WCP, respec-

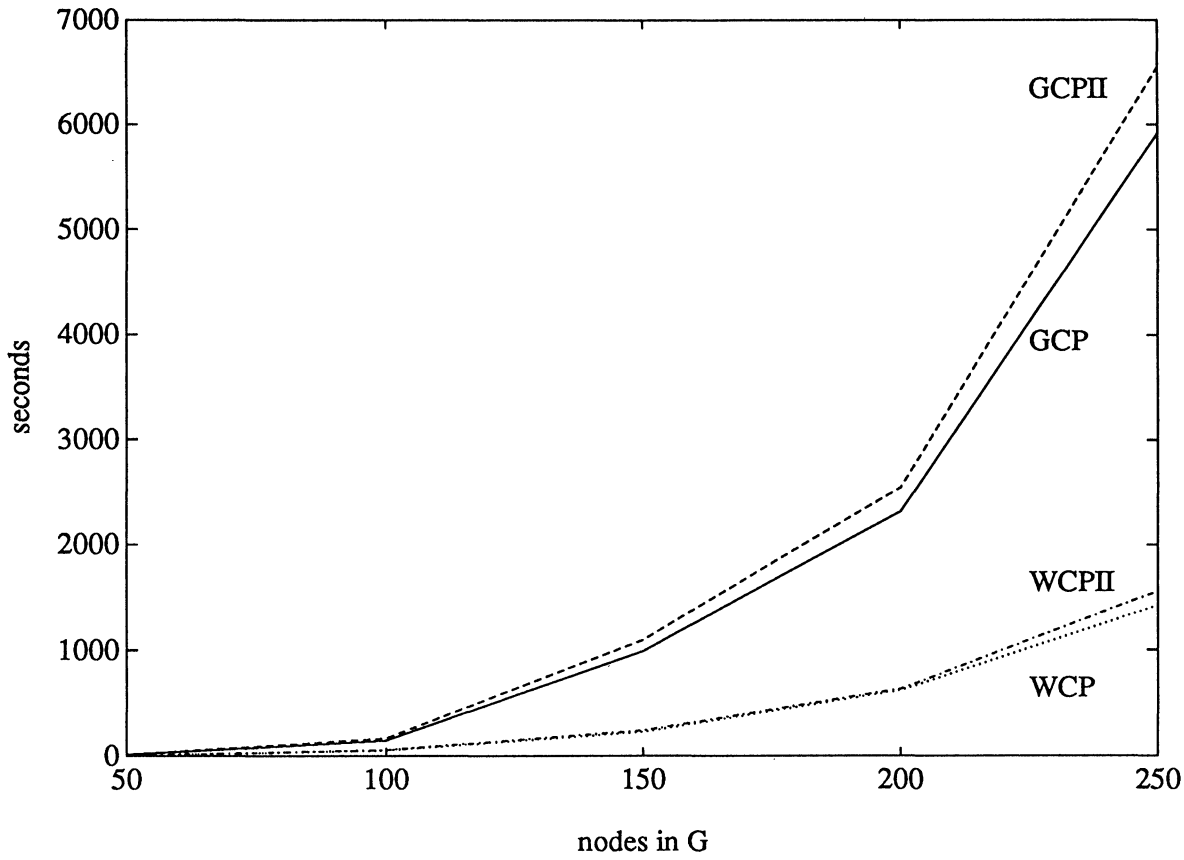

FIGURE 5 Execution times for P = 0.5.

tively, except that the first two priorities in the determination of the candidate pair are *switched.* Thus, the first priority is to select the edge (p, q) which has the minimum number of deleted edges, while the second priority is that it has the maximum number of common neighbors. (In the case of WCP II, the third priority remains that of having maximum weight.)

For completeness, we also give the results produced by GCP II and WCP II for the example considered in detail in Table I. Either of these procedures produces a total of 8 clusters, the same number as produced by CP, GCP and WCP. Both of the data path designs require the same number "5" of multiplexers.

The same set of large examples considered in Section 3 were also solved using GCP II and WCP II. First, re-examine Tables III. In all cases, GCP II produces the same or fewer number of clusters than GCP, and WCP II produces the same or fewer number of clusters than WCP. Moreover, WCP is as good or better than either GCP or GCP II. Of all four procedures, WCP II consistently produces the fewest number of clusters.

As far as computation time is concerned, it can be seen from Figures 3–7 that WCP and WCP II require nearly the same amount of time to execute. Moreover, they are both faster than GCP and GCP II, particularly for large values of the probability P.

SUMMARY

We have presented a simpler and more efficient algorithm for bus system synthesis than the previously proposed GCP algorithm. Like GCP, our approach takes account of factors not considered in simple CP in order to obtain a better design. However, WCP does not require backtracking, and therefore executes more rapidly than GCP. Moreover, the results obtained in our set of large synthesis examples indicates that WCP is able to generate designs having a fewer number of buses.

We have also presented a modification to GCP and WCP (GCP II and WCP II, respectively) which is seen to produce even better designs. By reversing the priorities in the selection of the candidate pair, these algorithms avoid certain locally optimally so-

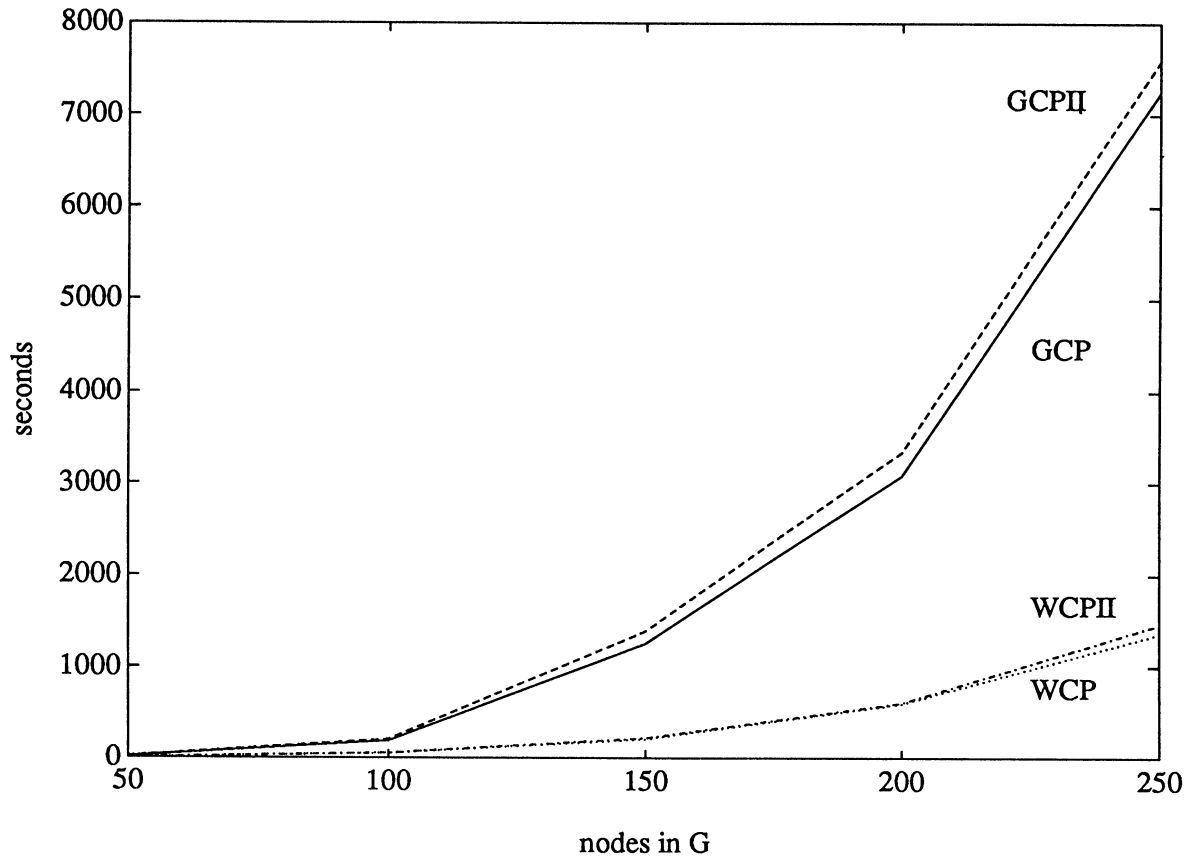

FIGURE 6 Execution times for P = 0.6.

lutions to achieve a more globally optimal solution. Our results indicate that GCP II is better than GCP and WCP II is better than WCP. Of all four algorithms considered, WCP II consistently performs the best on our set of large synthesis examples.

Finally, with suitable modifications, WCP (and/or WCP II) should also be applicable to the other phases of data path synthesis, namely the allocation of registers and data operators. These issues are currently under investigation.

Acknowledgment

This work was supported in part by the Graduate School of the University of Minnesota and the Center of Microelectronic and Information Sciences.

APPENDIX

The following is an illustration of the steps of the WCP algorithm as applied to the weighted graph G for the example code sequence. The list of edges in G, including the number of common neighbors, the number of deleted edges and the weight for each edge, is shown in Table IV. When this initial list is examined, it is found that the pair (1,4) has the maximum number (9) of common neighbors and it is thus selected as the first candidate pair. (For purposes of comparison, Tables IV–XII indicate which edge(s) would be selected using CP and which edge is selected using WCP.) Accordingly, node 1 becomes the head of the first cluster. Note that Table IV indicates that there are 15 deleted edges associated with this edge. Of these, there are five "case-1 deletions," nine "case-2 deletions," and one "case-3 deletion." To illustrate how the weight updating is computed, consider the edge (2,4), which is a "case-2 deletion". The retained edge in this case is the edge (1,2). Thus, the weight for the edge (1,2) becomes the sum of its previous weight and the weight of the deleted edge (2,4). As both of these weights were equal to 1, the weight of the retained edge (1,2) is updated to the value 2. The weights of other edges are updated in the same way.

Upon updating the remaining list of edges (see Table V), it is found that node 1 still appears in the list, so we continue to find another candidate pair

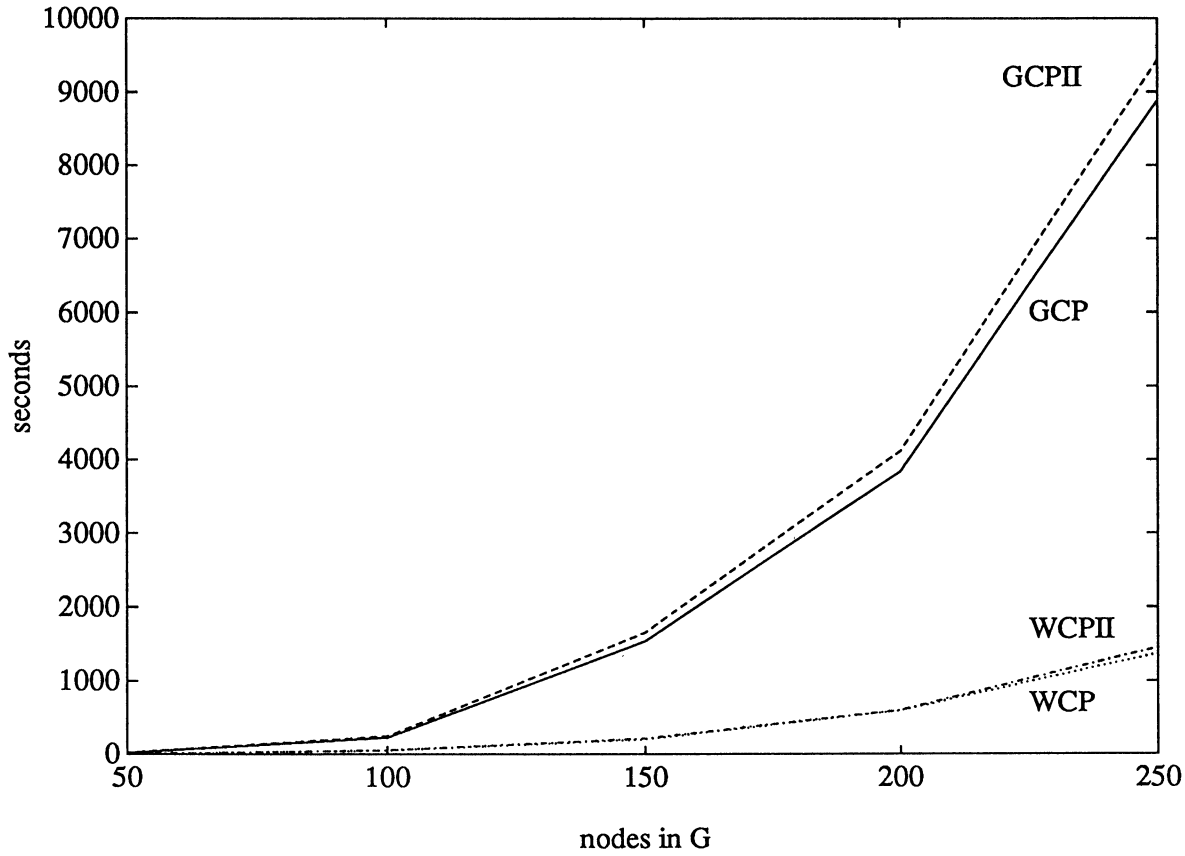

FIGURE 7 Execution times for P = 0.7.

which joins node 1 to another node. It is found that the two pairs (1,11) and (1,14) both have the maximum number (5) of common neighbors. Among these two edges, the pair (1,11) has the fewest number (12) of deleted edges, and it is thus selected as the next candidate pair.

When the list of edges is now updated (see Table VI), it is seen that node 1 appears in five edges, all of which have the same number (0) of common neighbors. Among these five edges, the pair (1,2), with the minimum number (8) of deleted edges, is selected as the next candidate pair.

When the list of edges is updated again (see Table VII), node 1 no longer appears in any pair. Thus, we go back to step 2 of the WCP algorithm and begin to form the second cluster. The pair (13,14) with the maximum number (7) of common neighbors, will be selected as the first candidate pair for this new cluster, with node 13 as the head of the cluster. After updating the list of edges (see Table VIII), it is found that the three pairs (6,13), (9,13) and (13,16) all have the same maximum number (4) of common neighbors. Furthermore, these three edges also all have the same number (9) of deleted edges. Thus, in this case we must resort to the third priority, the weight, to determine the next candidate pair. Note that without using the weight property, as in simple CP, there are three equally valid choices. However, in WCP, the larger weight (1) of pair (13,16) will lead to its selection alone.

After choosing the candidate pair (13,16), the updated list of edges is shown in Table IX. The three pairs (10,13), (13,15) and (13,17) have the same number (0) of common neighbors and the same number (6) of deleted edges. The algorithm selects the edge (13,15), with the larger weight (4), as the next candidate pair.

Following this step, the node 13 no longer appears in the updated list of edges (see Table X). We then return to step 2 of the WCP algorithm to begin to form another cluster. In Table X, the pair (7,8) with the maximum number of common neighbors (2), will be the first candidate pair for this new cluster, with node 7 as the head of the cluster.

Upon updating (see Table XI), the pairs (6,7) and (7,9) which join the node 7 have the same number (0) of common neighbors and the same number (5) of deleted edges. Again, using only CP, both of these

edges would be equally valid choices. In WCP, however, the larger weight (2) of pair (6,7) means that it is the one that should be selected.

After having chosen the candidate pair (6,7), the updated list of edges is shown in Table XII. Since the (new) head node 6 no longer appears in the list, we go back to step 2 of the WCP algorithm to form another cluster. Note that all of the remaining edges in Table XI have the same number (0) of common neighbors and the same number (3) of deleted edges. WCP will therefore select the edge (3,9) as the next candidate pair because of its larger weight. Finally, after combining the pair (3,9) the list of edges is empty, and WCP is completed.

References

[1] C.J. Tseng and D.P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. on Computer-Aided Design,* CAD-5, pp. 379–395, 1986.

[2] H.C. Torng and N.C. Wilhelm, "The Optimal Interconnection of Circuit Modules in Microprocessor and Digital System Design," *IEEE Trans. on Computers,* C-26, pp. 450–457, 1977.

[3] C.Y. Hitchcock III and D.E. Thomas, "A Method of Automatic Data Path Synthesis," *Proceedings, 20th ACM/IEEE Design Automation Conference,* pp. 484–489, 1983.

[4] L.J. Hafer and A.C. Parker, "Automated Synthesis of Digital Hardware," *IEEE Trans. on Computers,* C-31, pp. 93–109, 1982.

[5] L.J. Hafer and A.C. Parker, "A Formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic," *IEEE Trans. on Computer-Aided Design,* CAD-2, pp. 4–18, 1983.

[6] A.C. Parker, F. Kurdahi, and M. Milnar, "A General Methodology for Synthesis and Verification of Register-Transfer Design," *Proceedings, 21st ACM/IEEE Design Automation Conference,* pp. 329–335, 1984.

[7] S.W. Director, A.C. Parker, D.P. Siewiorek, and Donald E. Thomas, "A Design Methodology and Computer Aids for Digital VLSI Systems," *IEEE Trans. on Circuits and Systems,* CAS-28, pp. 634–645, 1981.

[8] B.M. Pangrle and D.D. Gajski, "Design Tools for Intelligent Silicon Compilation," *IEEE Trans. on Computer-Aided Design,* CAD-6, pp. 1098–1112, 1987.

[9] P.G. Paulin and J.P. Knight "Force-Directed Scheduling in Automatic Datapath Synthesis," *Proceedings, 21st ACM/IEEE Design Automation Conf.,* pp. 195–202, 1987.

[10] S. Davadas and A.R. Newton, "Algorithms for Hardware Allocation in Data Path Synthesis," *IEEE Trans. on Computer-Aided Design,* CAD-7, pp. 768–781, 1989.

[11] G. Craig and C. Kime, "Determining Parrel Test Schedules for VLSI Built-In Test," *Tech. Rep. ECE-84-23,* Dept. of Elec. Comput. Eng., Univ. of Wisconsin-Madison, 1984.

[12] G. Craig, C. Kime and K.K. Saluja, "Test Scheduling and Control for VLSI Built-In Self-Test," *IEEE Trans. on Computers,* C-37, pp. 1099–1109, 1988.

[13] E. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice,* Prentice-Hall, pp. 353–354, 1977.

[14] S. Baase, *Computer Algorithms,* Addison-Wesley, 1978.

[15] J.G. Augustson and J. Minker, "An Analysis of Some Graph Theoretical Cluster Techniques," *J. ACM,* pp. 571–588, 1970.

[16] C. Bron and J. Kerbosch, "Finding All Clique of An Undirected Graph–Algorithm 457," *Commun. ACM,* pp. 575–577, 1973.

[17] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms,* Computer Science Press, 1978.

[18] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs,* Academic Press, 1980.

[19] G.D. Mulligan and D.G. Corneil, "Corrections to Bierstone's Algorithm for Generating Cliques," *J ACM,* pp. 244–247, 1972.

[20] C.J. Tseng, "Automated Synthesis of Data Paths in Digital Systems," *Ph.D. dissertation,* Dept. of Electrical and Computer Engineering, Carnegie-Mellon University, 1984.

Biographies

CHIEN-IN HENRY CHEN is an Assistant Professor of Electrical Engineering at Wright State University. He received the B.S. degree from the National Taiwan University, Taiwan, in 1981, the M.S. degree from the University of Iowa, Iowa City, in 1986, and the Ph.D. degree from the University of Minnesota, Minneapolis, in 1989, all in the Electrical Engineering. During the summer 1992, he worked as a Faculty Research Associate at Wright Laboratory, Wright-Patterson, Air Force Base, Dayton, Ohio. His current research interests are in areas of computer-aided design, simulation and testing of VLSI circuits, design for testability, and fault-tolerant computing.

GERALD E. SOBELMAN received a B.S. in physics from UCLA and an M.A. and Ph.D. in physics from Harvard University. He subsequently became a research associate at Rockefeller University, where he worked on computational problems in theoretical physics. He has held engineering positions at Sperry Corporation and Control Data Corporation, working on the development of CAD tools for VLSI design. Since 1986, he has been an Associate Professor of Electrical Engineering at the University of Minnesota. His current research interests are in the areas of CAD, VLSI design and digital signal processing.