



ELSEVIER

Discrete Applied Mathematics 88 (1998) 7–28

DISCRETE  
APPLIED  
MATHEMATICS

## On testing consecutive-ones property in parallel

Fred Annexstein<sup>a,\*</sup>, Ram Swaminathan<sup>b,2</sup>

<sup>a</sup>Department of ECECS, University of Cincinnati, Cincinnati, OH 45221-0030, USA

<sup>b</sup>Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974-0636, USA

Received 20 September 1996; received in revised form 20 June 1997; accepted 25 November 1997

---

### Abstract

The consecutive-ones property problem has many important applications in the field of discrete algorithms, including the physical mapping problem in computational molecular biology. A  $(0, 1)$ -matrix is said to satisfy the *consecutive-ones property* if there is a permutation of the rows of the matrix such that in each column all non-zero entries are adjacent. The problem of determining such a permutation, if one exists, is the *consecutive-ones property problem*. The classic algorithm for solving this problem is a linear time sequential algorithm of Booth and Lueker (1976) which is known to be based on the *PQ*-tree data structure. In this paper we present a new algorithm for this problem using a divide-and-conquer method that employs a graph-theoretic data structure known as Tutte decomposition, i.e., decomposition of graphs into 3-connected components. Our algorithm enjoys the property that it efficiently parallelizes using the standard PRAM parallel computational model, while avoiding the complex implementations associated with *PQ*-trees. Our algorithm is more work efficient than previous parallel solutions, improving on the known processor bounds. © 1998 Elsevier Science B.V. All rights reserved.

---

### 1. Introduction

#### 1.1. Motivation from computational biology

A physical mapping of some DNA target, say a chromosome or entire genome, is obtained by specifying the linear order of a set of landmark sites. One popular method is based on clone maps that are constructed using a clone library consisting of a large number of overlapping fragments. These clones may be fingerprinted using a set of unique probes called sequence tagged sites (STSs). STSs are very short DNA sequences

---

\* Corresponding author. E-mail: fred.annexstein@uc.edu.

<sup>1</sup> Supported in part by NSF Grant CCR-93-09470.

<sup>2</sup> Supported in part by NSF Grant CCR-93-09470 and ONR Grant N00014-94-1-0382. This work was done while the author was with the University of Cincinnati, and RUTCOR and DIMACS Center, Rutgers University.

that appear in a single position in the genome. The fingerprint of a clone is simply the set of STSs contained within it. A key algorithmic question for physical mapping is the following: given the fingerprint data for a clone library, how best to identify the linear order of the STSs in the DNA target?

We can view the final laboratory data as a large  $(0, 1)$ -matrix  $A = [a_{i,j}]$ , with  $a_{i,j} = 1$ , if and only if, the  $i$ th clone contains the  $j$ th STS. Typically, the data associated with a mapping problem for the human genome is quite large. Reported experiments employ about 18 000–25 000 clones and about 9000–15 000 STSs [1, 15]. A proposed ordering of the STSs is *consistent* with the data if and only if on each clone the STSs in the fingerprint are consecutively ordered (with no gaps). If a consistent ordering exists, then we say the matrix  $A$  has the *consecutive-ones property*, since permuting the columns (i.e., the linear ordering of STSs) of  $A$  according to this ordering results in each row (i.e., fingerprint data for a clone) possessing a unique consecutive block of ones. We denote by *CIP* the problem of determining whether a given matrix has the consecutive-ones property.

In practice, a serious problem arises from the fact that algorithms for CIP are not, in general, adaptable to errors in the data set. Experimental data is likely to contain numerous errors, including false positives, false negatives, and other abnormalities, such as chimerisms [15]. Optimization problems associated with maximum likelihood maps are often computationally difficult, e.g., NP-hard or worse. Previous work on heuristic strategies for such optimization problems include parallel simulated annealing [3] and local search methods [1]. The best approach to dealing with error-prone data is not yet clear. Other strategies could make use of known CIP algorithms and data structures in subroutines. Hence, there is motivation for deriving a variety of competitive algorithms and data structures for CIP solutions in applications associated with physical mapping problems.

## 1.2. Our results

The classic algorithm for solving CIP is a linear-time sequential algorithm of Booth and Lueker [6]. The Booth–Lueker algorithm is known to be based on *PQ*-trees, a data structure with a complicated implementation. In this paper we present a new algorithm for CIP using a divide-and-conquer method using an alternative data structure. Our algorithm enjoys the property that it efficiently parallelizes using the standard PRAM parallel computational model, while avoiding implementations of the *PQ*-tree data structure. We employ as a primary data structure the decomposition of graphs into 3-connected components, called the *Tutte decomposition*. The implementation of this graph-theoretic data structure may appeal to some readers as conceptually simpler than implementations of *PQ*-trees. In a sequential computation model, we can apply the linear-time algorithm for Tutte decomposition of Hopcroft and Tarjan [12] to achieve an overall time complexity  $O(n \log n)$ . Fussell et al. [10] showed that Tutte decomposition can be computed on a PRAM in  $O(\log n)$  time using a sublinear number of processors. Using this and other standard parallel algorithmic techniques, we show our

algorithm can run on a PRAM with a time complexity of  $O(\log^2 n)$  using a sublinear number of processors.

### 1.3. Comparisons with previous work

Previously, Bixby and Wagner [3] have presented a solution to a generalized CIP problem, using Tutte decomposition in conjunction with disjoint-set union, that achieves nearly linear time complexity. However, their strategy is not parallelizable since rows of the matrix are processed sequentially. Our algorithm, using a divide-and-conquer method, is highly parallelizable and more work efficient than all previous parallel solutions.

In previous work on parallel solutions for CIP, Klein and Reif [14] presented an algorithm based on parallel implementations of complex  $PQ$ -tree manipulations. The work complexity of this algorithm was later improved by Klein [13] yielding a PRAM solution with time complexity  $O(\log^2 n)$  using linearly many processors. Chen and Yesha [7] using other methods described a PRAM algorithm (specified by an  $n \times m$  matrix) that runs in time  $O(\log m + \log^2 n)$  using  $O(n^2m + n^3)$  processors.

### 1.4. Other motivations

Apart from physical mapping of genomes, consecutive-ones property has many other applications. In the area of database theory, the problem has been studied under the name *consecutive-retrieval property* [11]. The problem is also associated with efficient solutions to certain instances of NP-complete problems. For example, the *gate-matrix layout problem* is NP-Complete for arbitrary  $(0,1)$ -matrices. However, the problem is solvable in linear time when restricted to the class of  $(0,1)$ -matrices having the consecutive-ones property; this result was shown by Deo et al. [9]. Also, it is well known that matrices which satisfy the consecutive-ones property are totally unimodular, and thus give rise to polytime solutions to certain integer programming problems [2]. The recognition problem for interval graphs can also be reduced to the CIP problem [6].

There is yet another interesting feature of our algorithm described in this paper. Truemper [18] had asked whether the problem of testing a given  $(0,1)$ -matrix for graphicness (in the matroidal sense) admits a divide-and-conquer algorithm, where the divided partitions are submatrices of the input matrix. Truemper [18] remarked that finding such an algorithm would be difficult. However, for testing consecutive-ones property, which is a special case of testing for graphicness, our algorithm answers his question in the affirmative.

The rest of the paper is organized as follows. Section 2 gives the preliminary definitions and results. Section 3 gives the justification for a divide-and-conquer solution to the CIP problem, and describes the steps of our main algorithm. Section 4 provides some technical details of certain steps of the main algorithm. Section 5 describes the sequential and parallel complexity analysis of our algorithm.

## 2. Preliminaries

We begin with definitions and related graph-theoretic terminology; for undefined terms we refer the reader to the standard reference by Bondy and Murty [5].

We pose the consecutive-ones property (C1P) problem in terms of sets as follows: define an *ensemble*  $(A, \mathcal{C})$  as a set  $A$  of *atoms*, along with a collection  $\mathcal{C}$  of *columns*, where each column is a subset of  $A$ . The C1P problem is to find a linear layout of the atoms of  $A$  such that all of the atoms comprising each column are contiguous in the layout. We say the ensemble  $(A, \mathcal{C})$  is *path graphic* if there exists a path  $P$  of  $|A|$  edges such that the edges of  $P$  are indexed on the set  $A$  and the columns of  $\mathcal{C}$  each correspond (as an edge set) to a connected subpath of  $P$ . If such a  $P$  exists, then  $P$  is a *path realization* for  $(A, \mathcal{C})$ , and represents a solution to the C1P problem for the instance  $(A, \mathcal{C})$ .

Let  $P$  be a path realization of a path-graphic ensemble  $(A, \mathcal{C})$ . Then, by definition, every column of  $\mathcal{C}$  corresponds to a subpath of  $P$ . Starting with the path  $P$ , construct a graph  $G$  by adding an edge between the ends of each such subpath. We call these edges the *non-path edges* of  $G$ , whereas the edges of  $P$  are the *path edges* of  $G$ . The pair  $(G, P)$  completely specifies  $(A, \mathcal{C})$ , and is called a *gp-realization* of  $(A, \mathcal{C})$ . Thus, to determine whether  $(A, \mathcal{C})$  is path graphic, it suffices to determine whether it has a gp-realization. In general, if  $G$  is a connected graph which contains a Hamiltonian path  $P$ , then we call the ordered pair  $(G, P)$  a *gp-pair*.

By analogy, we say an ensemble  $(A, \mathcal{C})$  is *cycle graphic* if there exists a cycle  $O$  such that the edges of  $O$  are indexed on the set  $A$  and the columns of  $\mathcal{C}$  are edge sets of subpaths of  $O$ . If such a cycle  $O$  exists, then  $O$  is a *cycle realization* for  $(A, \mathcal{C})$ , and represents a solution to the *circular-ones property problem*. As before, we may construct a graph  $G$  by adding an edge between the ends of each column subpath. The pair  $(G, O)$  completely specifies  $(A, \mathcal{C})$ , and is called a *gc-realization* of  $(A, \mathcal{C})$ . Thus, to determine whether  $(A, \mathcal{C})$  is cycle graphic, it suffices to determine whether it has a gc-realization. In general, if  $G$  is a connected graph which contains a Hamiltonian cycle  $O$ , then we call the ordered pair  $(G, O)$  a *gc-pair*.

Our divide-and-conquer approach to the C1P problem involves the decomposition of the ensemble into a pair of subensembles (defined in Section 3), recursively computing the gp-realizations of the subensembles, and properly merging them together by manipulating the resulting graphs. To accomplish this merging we require an operation which aligns certain edges of the graphs while preserving path-graphicness. The operation that supports this alignment is called a *Whitney switch*. To compute the required set of Whitney switches we need to first decompose the graph using *Tutte decomposition* into 3-connected components. We provide a formal description of the Whitney switch operation and Tutte decomposition in the next two subsections.

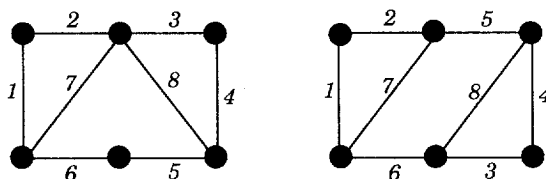


Fig. 1. 2-isomorphic graphs that are not isomorphic.

### 2.1. Whitney switches

We begin with some necessary definitions. A connected graph is *2-connected* if it has no cut vertex. A *2-separation* of a 2-connected graph  $G$  is a partition  $\{E_1, E_2\}$  of the edge set  $E(G)$  such that  $|E_1|, |E_2| \geq 2$  and  $|V(G[E_1]) \cap V(G[E_2])| = 2$ , i.e., the number of distinct vertices appearing in both edge-induced subgraphs is exactly two. A *2-connected* graph is *3-connected* if it has no 2-separation.

Let  $G = (V, E)$  be a 2-connected graph, and let  $\{E_1, E_2\}$  be a 2-separation of  $G$ . Let  $u$  and  $v$  be the vertices common to the edge-induced subgraphs  $G[E_1]$  and  $G[E_2]$ . Let  $G'$  be the graph obtained from  $G$  by interchanging, or switching, the incidences of vertices  $u$  and  $v$  in  $G[E_1]$ . Then,  $G'$  is said to be obtained from  $G$  by *switching*  $u$  and  $v$  in  $G[E_1]$ , we call such an operation of *Whitney switch*. A graph  $H$  is *2-isomorphic* to  $G$  if  $H$  is obtained from  $G$  by a sequence of such switches.

The reader should note that we equate trees and cycles of a graph with their respective edge sets. It is easy to see that a pair of *2-isomorphic* graphs that are edge labeled have the same set of cycles. The following theorem shows that the converse also holds.

**Theorem 1** (Whitney [21]). *Let  $G$  and  $G'$  be 2-connected graphs on the same edge set. Then,  $G$  and  $G'$  have the same set of cycles if and only if they are 2-isomorphic.*

Fig. 1 illustrates that 2-isomorphism is a generalization of isomorphism between graphs. Note that either graph can be “switched” to the other by using the 2-separation  $\{1, 2, 6, 7\}$  and  $\{3, 4, 5, 8\}$ .

As the following proposition shows, Whitney switches may be applied to gp-realizations of connected ensembles for they are always 2-connected.

**Proposition 1.** *Let  $(G, P)$  be a gp-realization of a connected ensemble  $(A, \mathcal{C})$ . Then, the graph  $G$  is 2-connected.*

**Proof.** If  $G$  is not 2-connected, then there exists a partition  $\{E_1, E_2\}$  of  $P$  such that no fundamental cycle (with respect to  $P$ ) of  $(G, P)$  has non-empty intersection with both  $E_1$  and  $E_2$ . This implies that the bipartite graph associated with  $(A, \mathcal{C})$  is disconnected, a contradiction.  $\square$

We say that a gp-pair  $(G, P)$  is *2-isomorphic* to a gp-pair  $(G', P')$  if  $G'$  is 2-isomorphic to  $G$ , and  $P$  and  $P'$  are equivalent as edge sets. The following proposition gives a relationship between any two gp-realizations of a path-graphic ensemble.

**Proposition 2.** *Let the ensemble  $(A, \mathcal{C})$  be connected and path-graphic, and let  $(G, P)$  and  $(G', P')$  be gp-realizations of  $(A, \mathcal{C})$ . Then,  $G$  and  $G'$  are 2-isomorphic.*

**Proof.** Since  $(G, P)$  and  $(G', P')$  are gp-realizations of  $(A, \mathcal{C})$ , they have the same set of fundamental cycles with respect to the path. By a standard result in graph theory, this implies that  $G$  and  $G'$  have the same set of cycles. By Proposition 1,  $G$  and  $G'$  are 2-connected. Thus, by Theorem 1,  $G$  and  $G'$  are 2-isomorphic.  $\square$

## 2.2. Tutte decomposition

To solve the CIP problem, we will need to compute a set of Whitney switches of gp-realizations of certain subensembles. These switches can be found via a decomposition of the gp-realizations into 3-connected components. This graph decomposition, called a *Tutte decomposition*, was first introduced by Tutte [20], and studied further by Cunningham and Edmonds [8], and Hopcroft and Tarjan [12]. For our purposes, we use Tutte decompositions since they permit an explicit representation of all possible Whitney switches, and therefore all possible gp-realizations of an ensemble.

A *Tutte decomposition* of a graph  $G$ , denoted  $\mathcal{D}(G)$ , or simply  $\mathcal{D}$ , if  $G$  is clear from context, is the set of graphs constructed recursively starting with  $\{G\}$  as follows. If some graph  $H$  in the current set of graphs has a 2-separation, then  $H$  is replaced by the members of simple decomposition of  $H$ , where a pair of *marker* edges are introduced (i.e., they are not edges of any graph in the current set) between the vertices of the 2-separation. Each member of the set of graphs thus constructed is either a *bond* (i.e., a connected, loopless graph on two vertices), a *polygon* (i.e., a cycle having at least three edges), or a 3-connected graph on at least four vertices. Finally, if any two bonds or two polygons share a marker edge, then the two graphs are “merged” by identifying the two copies of the marker edge and deleting it. A tree  $\mathcal{T}$  can be associated with  $\mathcal{D}$  so that every vertex of  $\mathcal{T}$  is a unique member of  $\mathcal{D}$  and two vertices of  $\mathcal{T}$  are adjacent if the corresponding members of  $\mathcal{D}$  share a common marker edge. A connected subset  $\mathcal{D}' \subseteq \mathcal{D}$  (with respect to the underlying tree) of a Tutte decomposition of a graph  $G = (V, E)$  is called a *minimal decomposition with respect to  $E' \subseteq E$* , if every edge of  $E'$  is in some member of  $\mathcal{D}'$ , and every leaf member of  $\mathcal{D}'$  has an edge from  $E'$ .

Associated with any Tutte decomposition  $\mathcal{D}$  is a *composition*. Let  $G_1$  and  $G_2$  be members of  $\mathcal{D}$  that have a marker edge  $e$  in common. Consider a one-to-one mapping

from the set of ends of  $e$  in  $G_1$  to its set of ends in  $G_2$ . Define  $H$  to be the graph obtained from  $G_1$  and  $G_2$  by first deleting  $e$  from each graph, and then by identifying the ends of  $e$  in  $G_1$  with their respective images in  $G_2$ . Now let  $\mathcal{D}' := (\mathcal{D} - \{G_1, G_2\}) \cup \{H\}$ . Then,  $\mathcal{D}'$  is a decomposition with one fewer member than  $\mathcal{D}$ . Repeating this *merging* process an additional  $(|\mathcal{D}| - 2)$  times results in a decomposition containing a single 2-connected graph. Given  $\mathcal{D}$  together with a collection  $M$  of mappings (as defined above), the resulting graph is uniquely determined up to the names of the vertices obtained by identification. The resulting graph is denoted  $m(\mathcal{D}, M)$ . For convenience, this notation is shortened to  $m(\mathcal{D})$  with the interpretation that some  $M$  has been specified.

Cunningham and Edmonds (Theorem 1 of [8]) proved that for a 2-connected graph  $G$ , there exists a unique Tutte decomposition  $\mathcal{D}$  such that  $m(\mathcal{D}) = G$ . (Here uniqueness means unique up to the names of the marker edges and their ends.) Hopcroft and Tarjan [12] independently proved this result, and gave an  $O(|E(G)|)$ -time algorithm for computing the Tutte decomposition of  $G$ . Fussell et al. [10] showed that a Tutte decomposition can be computed on a (CRCW) PRAM in  $O(\log n)$  time using  $(m + n) \log \log n / \log n$  processors, where  $n = |V(G)|$  and  $m = |E(G)|$ .

The following theorem describes the relationship between the Tutte Decompositions of a pair of 2-isomorphic graphs  $G_1$  and  $G_2$ . From this theorem it follows that each Whitney switch needed to transform  $G_1$  and  $G_2$  can be expressed as either a relinking (i.e., permuting the edges) of one of the polygons in the decomposition, or as a relabeling the ends of a marker edge.

**Theorem 2.** *Let  $G_1$  and  $G_2$  be a pair of 2-connected graphs whose edges are indexed on the same set. Assume  $G_1$  is 2-isomorphic to  $G_2$ , and let  $\mathcal{D}_1$  and  $\mathcal{D}_2$  be their respective Tutte decompositions. Then the associated trees of these decompositions are isomorphic; moreover, associated pairs (via the isomorphism) of 3-connected pieces and bonds are (edge labeled) isomorphic up to the labeling of the marker edges, and each pair of associated polygons are (edge labeled) 2-isomorphic.*

**Proof.** Since  $G_1$  and  $G_2$  are 2-isomorphic, by definition there is a finite sequence of Whitney switches that will transform  $G_1$  to  $G_2$ , and vice versa. By the definition of Tutte decomposition and Theorem 1 of [8],  $\mathcal{D}_i$  is the unique representation of all possible 2-separations, and hence Whitney switches, of  $G_i$  for  $i \in \{1, 2\}$ . These switches are characterized by the 2-isomorphisms of the polygons in the decomposition and the possible orientations of the marker edges. Since the pair of Tutte decompositions  $\mathcal{D}_1$  and  $\mathcal{D}_2$  must exhibit the set of switches that transform  $G_1$  to  $G_2$ , the theorem follows.  $\square$

For the following pair of propositions we assume  $(G, P)$  is a gp-pair, and  $G$  has a non-path edge  $e$  between the end vertices of  $P$ , and  $G$  has no parallel non-path edges. Note that we allow a single non-path edge to be parallel with a single path edge. View the resulting Tutte decomposition  $\mathcal{D} = \mathcal{D}(G)$  as a rooted tree with the member

containing  $e$  as the root. Define a *complete child* of a member  $H$  of  $\mathcal{D}$  as the graph obtained by merging together a child of  $H$  with all of the child's descendants.

**Proposition 3.** *If  $J$  denotes a complete child of a member  $H$  of  $\mathcal{D}$ , then  $P$  restricted to  $J$  is a spanning path of  $J$ .*

**Proof.** Note that in every 2-separation of  $G$  both sets of edges contain at least one path edge. If not, then the induced subgraph of one of the sets must be a bond containing at least two non-path edges. This contradicts the assumption that we have removed all parallel non-path edges of  $G$ . Consider the 2-separation  $\{S_1, S_2\}$  of  $G$  defined by the unique parent marker of  $J$ . The edges of the Hamiltonian cycle defined by  $P \cup \{e\}$  when restricted to each induced subgraph  $G[S_i]$  forms a Hamiltonian path in  $G[S_i]$ , since both contain at least one path edge. Assume, without loss of generality, that the edge  $e$  is contained in  $G[S_2]$ . Then, it follows that the edges of  $P$  restricted to  $G[S_1] = J$  is a Hamiltonian path of  $J$ .  $\square$

**Proposition 4.** *If  $H$  denotes a polygon of  $\mathcal{D}$ , then  $H$  contains no non-path edge, except for distinguished edge  $e$ .*

**Proof.** Let  $H$  be a polygon of the decomposition  $\mathcal{D}$ . By Proposition 3,  $P \cup \{e\}$  when restricted to each complete child of  $H$  forms a Hamiltonian path. Hence, the set of edges consisting of the marker-edges of  $H$ , the path-edges of  $H$ , and the edge  $e$  must form a Hamiltonian cycle of  $H$ .  $\square$

### 3. Divide-and-conquer solution to CIP

Our divide-and-conquer approach to the CIP problem involves the decomposition of an ensemble into a pair of subensembles, recursively computing the gp-realizations of the subensembles, and finally merging them together by manipulating the resulting graphs. To accomplish this merging we require an operation which aligns certain edges of the graphs while preserving path-graphicness. The operation that supports this alignment is based on Whitney switches (defined in Section 2). In this section we prove theorems that demonstrate the validity of our approach. Essentially, these theorems say that there is a gp-realization for an ensemble if and only if there exist special gp-realizations for a pair of subensembles created by a partition of the original set of atoms (edges).

Given an ensemble  $(A, \mathcal{C})$  define the *associated bipartite graph*  $B$  as follows. The bipartitioned vertex set of  $B$  is  $A \cup \mathcal{C}$ . A vertex  $a \in A$  is adjacent to a vertex  $C \in \mathcal{C}$  iff  $a \in C$ . An ensemble  $(A', \mathcal{C}')$  is a *subensemble* of  $(A, \mathcal{C})$  if  $A' \subset A$  and each  $C' \in \mathcal{C}'$  is the restriction of column  $C \in \mathcal{C}$  to the set  $A'$ . Observe that the vertex set of a component of  $B$  induces a unique subensemble, called a *component* of  $(A, \mathcal{C})$ .



Given an ensemble  $(A, \mathcal{C})$ , our algorithm proceeds by partitioning  $A$  into two sets  $\{A_1, A_2\}$  with the following three properties: (i) the partition is *balanced*, i.e., the cardinality of each set is at least  $|A|/3$ , (ii) the subensemble  $(A_1, \mathcal{C}_1)$  induced by  $A_1$  is *connected*, i.e., the associated bipartite graph has exactly one component, and (iii) the atoms of  $A_1$  form a *segment* of  $(A, \mathcal{C})$ , i.e., in every gp-realization of  $(A, \mathcal{C})$  the path edges corresponding to  $A_1$  are contiguous.

In the next section we prove theorems that provide necessary and sufficient conditions for an ensemble  $(A, \mathcal{C})$  to be path graphic. These conditions are defined in terms of incidences of certain edges contained in gp-realizations of the induced subensembles  $(A_1, \mathcal{C}_1)$  and  $(A_2, \mathcal{C}_2)$ .

### 3.1. The alignment conditions

Given an ensemble  $(A, \mathcal{C})$  and a partition  $\{A_1, A_2\}$  of  $A$ , define a *crossing column* of  $\mathcal{C}$  as a column with non-empty intersection with both  $A_1$  and  $A_2$ . We define a *type-a* column to be a crossing column  $C \in \mathcal{C}$  such that  $A_1 \subset C$ ; all other crossing columns are *type-b* columns; columns that are non-crossing, we call *type-c*. Suppose the subensemble  $(A_1, \mathcal{C}_1)$  has a gp-realization  $(G_1, P_1)$ , and the subensemble  $(A_2, \mathcal{C}_2)$  has a gp-realization  $(G_2, P_2)$ . Each crossing column (type-a or type-b) is associated with a pair of non-path *crossing edges*, one in  $G_1$  and one in  $G_2$ . Each type-c column is associated with one non-path edge of either  $G_1$  or  $G_2$ .

**Definition 1.** A pair of gp-realizations  $(G_1, P_1)$  for  $(A_1, \mathcal{C}_1)$  and  $(G_2, P_2)$  for  $(A_2, \mathcal{C}_2)$  is said to satisfy the *global-alignment for path (GAP) conditions* if the following three conditions on their non-path edges are met:

- (1) Each type-b edge of  $(G_1, P_1)$  is incident to exactly one of the two end vertices of  $P_1$ .
- (2) Each type-b edge of  $(G_2, P_2)$  is incident to a single vertex  $w$  of  $P_2$ , each type-a edge of  $(G_2, P_2)$  spans  $w$  (i.e., with respect to the path  $P_2$ , the end vertices of the edge are on distinct sides of  $w$ ) or is incident to  $w$ , and each type-c edge does not span  $w$ ; we call  $w$  the *split vertex*.
- (3) Two type-b edges are disjoint in  $(G_2, P_2)$  (i.e., they share no common path edge) if and only if the associated type-b edges of  $(G_1, P_1)$  are incident to opposite end vertices of  $P_1$ .

Fig. 2 gives a graphic example of the satisfaction of the GAP alignment conditions. As in this example, when gp-realizations for a pair of subensembles  $(A_1, \mathcal{C}_1)$  and  $(A_2, \mathcal{C}_2)$  satisfy the GAP conditions we can merge them to construct a gp-realization of the original ensemble, as the figure below illustrates. This sufficient condition for path-graphicness is formalized in the Theorem 3. The corresponding necessary condition is given by Theorem 4.

1	$a$	$b$	$c$	$d$	$e$	$f$	$g$	$(A_2, \mathcal{C}_2)$	type-a: $e, g$ type-b: $a, c, d, f$ type-c: $b$
2	1	0	0	0	1	0	0		
7	0	0	1	0	0	1	1	$(A_1, \mathcal{C}_1)$	
8	0	0	1	0	0	0	1		
3	1	0	0	1	1	0	1		
4	0	1	0	0	1	0	1		
5	0	1	1	0	1	0	1		
6	0	0	1	0	1	1	1		

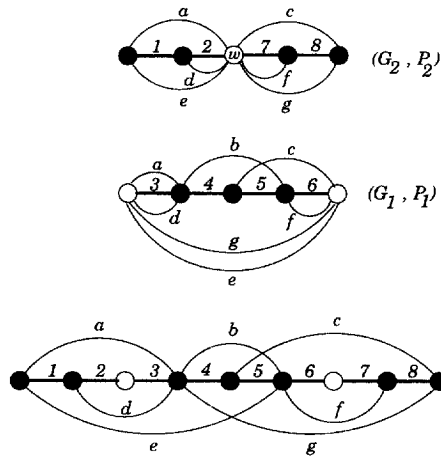


Fig. 2. A pair of gp-realizations that meet the GAP conditions, and how they are merged.

**Theorem 3.** *Let  $(A, \mathcal{C})$  be an ensemble, and let  $(A_1, \mathcal{C}_1)$  and  $(A_2, \mathcal{C}_2)$  be subensembles of  $(A, \mathcal{C})$ , where  $A = A_1 \cup A_2$  and  $A_1 \cap A_2 = \emptyset$ , such that some pair of gp-realizations of  $(A_1, \mathcal{C}_1)$  and  $(A_2, \mathcal{C}_2)$  satisfies the GAP conditions. Then,  $(A, \mathcal{C})$  is path graphic.*

**Proof.** Suppose we are given a pair of gp-realizations for subensembles that satisfy the three GAP conditions. We label the end vertices of  $P_1$  as  $u$  and  $v$ , and label the distinguished vertex of  $P_2$  as  $w$ ; note that when there are no type- $b$  edges it is possible that this split vertex  $w$  is not unique. By GAP-Condition-(3) it follows that two distinct type- $b$  edges are incident to the same vertex, either  $u$  or  $v$  in  $G_1$  if and only if they are incident to  $w$  as well as share at least one edge of  $P_2$  in  $G_2$ . Similarly, these edges are incident to different vertices in  $G_1$  if and only if they are incident to  $w$ , but otherwise disjoint in  $G_2$ . By GAP-Condition-(2) edges of type- $a$  span the vertex  $w$  in  $G_2$  if and only if they are incident to both end vertices of  $P_1$  in  $G_1$ . Hence, by splitting the vertex  $w$  and inserting the path  $P_1$  in between, and adjusting the type- $b$  edges appropriately (by GAP-Condition-(1) each column of  $\mathcal{C}$  corresponds to a fundamental cycle), we thus obtain a gp-realization for the given ensemble  $(A, \mathcal{C})$ .  $\square$

Let  $\{A_1, A_2\}$  be a partition of  $A$ . Recall that  $A_1$  forms a segment of  $(A, \mathcal{C})$  if in every gp-realizations of  $(A, \mathcal{C})$ , the path edges corresponding to  $A_1$  are contiguous. We have the following theorem.

**Theorem 4.** *If  $(A, \mathcal{C})$  is path-graphic and  $A_1$  is a segment of  $(A, \mathcal{C})$ , then the subensembles  $(A_1, \mathcal{C}_1)$  and  $(A_2, \mathcal{C}_2)$  have the property that some pair of gp-realizations of the subensembles satisfies the GAP conditions.*

**Proof.** Suppose that  $(A, \mathcal{C})$  is path graphic, and  $(G, P)$  is a gp-realization for  $(A, \mathcal{C})$ . Since  $A_1$  is a segment, the corresponding edges in  $P$  induce a subpath  $P_1$ . A gp-realization  $(G_2, P_2)$  for  $(A_2, \mathcal{C}_2)$  can be obtained from  $(G, P)$  by contracting  $P_1$  to a single vertex  $w$ . After the contraction, all of the non-path edges corresponding to type- $a$  edges span the vertex  $w$ , all of the non-path edges corresponding to type- $b$  are incident to  $w$ , and all type- $c$  edges do not span  $w$  (GAP-Condition-(2)). GAP-Condition-(3) simply follows from this contraction.

A gp-realization  $(G_1, P_1)$  for the subensemble  $(A_1, \mathcal{C}_1)$  can be obtained from  $(G, P)$  by contracting the path-edges corresponding to  $A_2$ . Since  $A_1$  is a segment the edges corresponding to  $A_2$  contract to either a single vertex or to two end vertices. After the contraction, all of the non-path edges corresponding to type- $a$  edges span  $P_1$  entirely. All of the type- $b$  edges are incident to one of the two end vertices of  $P_1$  (GAP-Condition-(1)).  $\square$

By analogy, the following theorems provide necessary and sufficient conditions for  $(A, \mathcal{C})$  to be cycle graphic in terms of incidences of the crossing edges in some gp-realization of the subensemble  $(A_1, \mathcal{C}_1)$  and gc-realization of the subensemble  $(A_2, \mathcal{C}_2)$ . The proofs are nearly identical to the proofs of Theorems 3 and 4, and so are omitted.

**Definition 2.** A pair of realizations, a gp-realization  $(G_1, P_1)$  for  $(A_1, \mathcal{C}_1)$  and a gc-realization  $(G_2, O_2)$  for  $(A_2, \mathcal{C}_2)$ , is said to satisfy the *global alignment for cycle (GAC) conditions* if the same three conditions on edges defined in Definition 1 are met, where the gp-pair  $(G_2, P_2)$  is replaced by the gc-pair  $(G_2, O_2)$ .

**Theorem 5.** *Let  $(A, \mathcal{C})$  be an ensemble, and let  $(A_1, \mathcal{C}_1)$  and  $(A_2, \mathcal{C}_2)$  be subensembles of  $(A, \mathcal{C})$ , where  $A = A_1 \cup A_2$  and  $A_1 \cap A_2 = \emptyset$ , such that some gp-realization of  $(A_1, \mathcal{C}_1)$  and gc-realization of  $(A_2, \mathcal{C}_2)$  satisfy the GAC conditions. Then,  $(A, \mathcal{C})$  is cycle graphic.*

**Theorem 6.** *Let  $(A, \mathcal{C})$  be a cycle-graphic ensemble. If  $A_1 \subseteq A$  is a segment of  $(A, \mathcal{C})$ , then the subensembles  $(A_1, \mathcal{C}_1)$  and  $(A_2, \mathcal{C}_2)$ , where  $A_2 = A - A_1$ , have the property that some gp-realization of  $(A_1, \mathcal{C}_1)$  and gc-realization of  $(A_2, \mathcal{C}_2)$  satisfy the GAC conditions.*

### 3.2. The partition

Recall from the introduction of Section 3 that our divide-and-conquer approach is based on finding a partition  $\{A_1, A_2\}$  of  $A$  that satisfies the three conditions (i) balanced sizes, (ii)  $(A_1, \mathcal{C}_1)$  is a connected subensemble, and (iii)  $A_1$  is a segment. We can always determine such a partition by considering the following two cases.

*Case 1: There exists a proper size column.* Suppose there is a column  $C \in \mathcal{C}$  such that  $|A|/3 \leq |C| \leq 2|A|/3$ . In this case, we let  $A_1 = C$  and  $A_2 = A - A_1$ . It follows immediately that such a partition satisfies the three properties we require.

*Case 2: There exists no proper size column.* Suppose every column  $C \in \mathcal{C}$  has cardinality  $|C| < |A|/3$ , or  $|C| > 2|A|/3$ . In this case, we transform the original problem so that all columns have cardinality  $\leq |A|/3$ . To wit, add a new atom  $r$  to the set of  $A$ ; Let  $A' = A \cup \{r\}$ . All columns  $C$  with cardinality  $|C| < |A'|/3$  remain unchanged. However, if  $|C| \geq 2|A'|/3$  then replace  $C$  with its complement (i.e., replace  $C$  with  $A' - C$ ). Hence, the cardinality of each column in the transformed instance is at most  $|A'|/3$ . As the reader may easily verify (or see [19]), this transformed ensemble, call it  $(A', \mathcal{C}') = \mathbf{Transform}((A, \mathcal{C}))$ , has the circular-ones property if and only if the original problem instance  $(A, \mathcal{C})$  has the consecutive-ones property.

We say a set of columns is *connected* if the induced subgraph of the columns and the rows they span in the associated bipartite graph is connected. Choose a connected set of columns of  $\mathcal{C}$  with the property that the set of atoms  $A_1$  contained in this collection has cardinality such that  $|A'|/3 \leq |A_1| \leq 2|A'|/3$ . Since every column is of size at most  $|A'|/3$ , it should be clear that either such a collection can be found or all maximally connected collections of columns contain a total of fewer than  $|A'|/3$  atoms. (We describe in Section 5 how such maximally connected collections of columns can be identified efficiently in parallel.) Note that if such a proper-sized set  $A_1$  does not exist then the original problem trivially decomposes into a collection of independent problems, each of size smaller than  $|A'|/3$ . Henceforth, and in the algorithm below, we will assume that we can find such a proper-sized set  $A_1$ . Clearly,  $A_1$  is a segment of  $(A', \mathcal{C}')$ ; for if there existed a gp-realization of  $(A', \mathcal{C}')$  for which the path-edges corresponding to  $A_1$  were not contiguous, then it would follow that the edges corresponding to some column (that contributed to  $A_1$ ) would not be contiguous, a contradiction. Now let  $A_2 = A' - A_1$ . Due to the cardinality constraints of each column  $C \in \mathcal{C}'$ , a crossing set cannot completely contain all elements of  $A_1$ , nor can it contain all the elements of  $A_2$ . Hence, there are no type- $a$  columns in this transformed instance, and the conditions associated with the GAC conditions are simplified, i.e., we only need to compute a gp-realization of  $(A_2, \mathcal{C}_2)$  with alignment conditions met for type- $b$  edges.

### 3.3. The main algorithm

The main algorithm given in Fig. 3 can be described informally as follows. The algorithm begins by removing “unwanted” columns and adding a complete column if necessary (note that this insures connectivity and does not significantly increase the

**Procedure Path-Realization** ( $A, C$ )  
*Input:* An ensemble  $(A, C)$ .  
*Output:* A gp-realization of  $(A, C)$ , if it exists.

*Step 0:* If  $|A| \leq 2$  then **return** a gp-pair with  $|A|$  path edges and  $|C|$  non-path edges.

*Step 1:* Remove columns with at most one atom, and if necessary, add column  $C = A$  to  $C$  with distinguished label  $e$ .

// *The Divide Step and Recursive Calls*

*Step 2:* If there is  $C \in C$  such that  $|A|/3 \leq |C| \leq 2|A|/3$ .  
then (Case 1)  
let  $A_1 := C$  and  $A_2 := A - C$ .  
else (Case 2)  
let  $(A', C') := \text{Transform}((A, C))$ ;  
As described in Section 3.2, let  $A_1$  denote the proper sized column-set-union,  
let  $A_2 := A - A_1$ .  
Let  $(G_1, P_1) := \text{Path-Realization}(A_1, C_1)$ .  
Let  $(G_2, P_2) := \text{Path-Realization}(A_2, C_2)$ .

// *The Combine Steps*

*Step 3:* Compute Tutte decomposition on each  $(G_i, P_i)$ .  
*Step 4:* Identify all type- $a$ , type- $b$ , and type- $c$  edges.  
*Step 5:* Find the minimal decomposition (see Section 2.2) with respect to the edge set consisting of the distinguished edge  $e$ , all type- $a$ , and type- $b$  edges.  
*Step 6:* As described in Section 4.2:  
If (Case 1) compute switches to satisfy GAP conditions;  
If (Case 2) compute switches to satisfy GAC conditions;  
If such a sequence of switches does not exist then halt and report  $(A, C)$  is “not path-graphic”.  
*Step 7:* **Return** merge of  $(G_1, P_1)$  and  $(G_2, P_2)$  as follows:  
If (Case 1) then identify the split vertex  $w$  of  $G_2$ , insert  $G_1$ , and adjust type- $a$  and type- $b$  edges.  
If (Case 2) then properly match and identify the end vertices of  $P_1$  and  $P_2$ .  
Adjust type- $b$  edges, and then obtain gp-realization by deleting the path-edge associated with atom introduced by Step 2.

Fig. 3. Path-realization algorithm.

problem size). Next a partition of the atoms  $A$  is chosen as described in Section 3.2. Then recursive calls of the procedure are made on the pair of subensembles induced by the partition. Upon return from the recursive calls, Tutte decomposition is computed for each gp-realization. Next, the set of Whitney switches required to satisfy the alignment conditions (GAP or GAC, as described in Section 3.1) are computed. Finally, the algorithm returns the solution as the merge of the pair of realizations that satisfy the alignment conditions.

But for Step 6, the correctness of the algorithm follows directly from Theorems 2–6. The correctness of Step 6 is justified in detail in Section 4.

#### 4. Computing the Whitney switches

In this section, we develop a refinement of Step 6 of the procedure path-realization, showing how the required Whitney switches can be computed. For each  $i \in \{1, 2\}$ , let

$(G_i, P_i)$  denote a gp-realization of the subensemble  $(A_i, \mathcal{C}_i)$ . We assume that the minimal Tutte decomposition  $\mathcal{D}_i$  of  $(G_i, P_i)$  is given. Our goal is to reorder the path edges in each  $P_i$  via Whitney switches so that the GAP or GAC conditions are met, and thus permit a proper merge of the realizations to take place, if one exists. We note that by assuming that there exists a non-path edge  $e$  between the two end vertices of the path  $P_i$  in  $G_i$ , it follows that any Whitney switch preserves this Hamiltonian cycle  $P_i \cup \{e\}$ , and hence the integrity of the path  $P_i$  is maintained in every 2-isomorphic copy.

#### 4.1. Algorithms for aligning a pair of edges

To demonstrate how to align all the non-path edges as required by these conditions, we first show how to align single non-path edges and pairs of non-path edges. As we will show in Section 4.2, if these edges are chosen judiciously, then it will follow that all the required non-path edges will be properly aligned.

Consider a gp-pair  $(G, P)$ . Let  $e$  be a non-path edge of  $G$  between the end vertices of  $P$ , and let  $f$  and  $g$  be a pair of non-path edges of  $(G, P)$ . We next show algorithmically how to align  $f$  and  $g$  via Whitney switches, if such an alignment is possible. We provide three separate algorithms each corresponding to one of the following cases:

- (A) Align  $f$  so that it is incident to an end vertex of  $e$ .
- (B) Align  $f$  and  $g$  so that each is incident to a distinct end vertex of  $e$ .
- (C) Align  $f$  and  $g$  so that each is incident to the same, but arbitrary vertex.

In each of these three cases, our algorithms proceed by using the minimal Tutte decomposition  $\mathcal{D}$  of  $G$  with respect to a subset of the non-path edges  $\{e, f, g\}$ , where we assume that  $G$  has no parallel non-path edges. In this minimal decomposition we relabel marker edges that have no corresponding children in the decomposition as path edges.

The following three algorithms basically follow a case analysis involving a series of *check conditions*. If any check condition is not true, then it follows that no proper alignment is possible, and so we can halt and report failure. If all check conditions are true, then the algorithms applies a sequence of switches which induce a 2-isomorphic copy where the alignment conditions we seek are satisfied. Finally, we merge all the adjacent members (sharing a common marker edge) so that like-labeled vertices are identified. The resulting gp-pair is the desired 2-isomorphic copy.

##### Algorithm for Case (A)

Let  $\mathcal{D}$  be minimal with respect to  $\{e, f\}$ . View  $\mathcal{D}$  as a rooted decomposition tree with the member containing  $e$ , as the root  $R$ . Note that for every member  $Q$  of  $\mathcal{D}$ ,  $Q$  contains either  $f$  or exactly one child marker edge. Label both end vertices of  $e$  with  $u$ . Since  $G$  has no parallel non-path edges, and the edge  $e$  is incident to the two ends of the path, it follows that  $R$  is not a bond.

Suppose  $R$  is a polygon, then by Proposition 4,  $R$  has no crossing edge. Therefore,  $R$  has one child marker edge. Replace  $R$  with a 2-isomorphic copy of  $R$  in which the unique child marker edge is incident to an end vertex of  $e$ .

Suppose  $R$  is 3-connected. If  $f$  is in  $R$ , then check that  $f$  is incident to an end vertex of the path. If  $f$  is not in  $R$ , then check to see that the unique child marker edge of  $R$  is incident to an end vertex of  $e$ .

For every member  $Q$ , other than  $R$ , do the following:

- (A.1) If  $Q$  is a bond, then label one of its vertices with  $u$ .
- (A.2) If  $Q$  is a polygon, then it cannot contain  $f$  by Proposition 4. Permute edges so that the parent marker edge and the unique child marker edge are both incident to a vertex. Label that vertex  $u$ .
- (A.3) If  $Q$  is 3-connected, then check that the parent marker edge is either incident to a unique child marker or incident to  $f$ , if  $f$  is in  $Q$ . Label this vertex  $u$ .

### Algorithm for Case (B)

Let  $\mathcal{D}$  be minimal with respect to  $\{e, f, g\}$ . View  $\mathcal{D}$  as a rooted decomposition tree with the member containing  $e$ , as the root  $R$ . The root  $R$  has at most two child marker edges. Since  $G$  has no parallel non-path edges, and the edge  $e$  runs between the two ends of the path, it follows that  $R$  is not a bond. Label both end vertices of  $e$  with  $u$ .

Suppose  $R$  is a polygon, then recall that  $R$  has no crossing edge. Check that  $R$  has exactly two child marker edges. Replace  $R$  with a 2-isomorphic copy of  $R$  in which the two child marker edges are incident to distinct end vertices of  $e$ .

Suppose  $R$  is 3-connected. If  $f$  and  $g$  are in  $R$ , then check that  $f$  and  $g$  are incident to the two end vertices of  $e$ . If only one of  $f, g$  is in  $R$ , say  $f$ , then check that  $f$  and the child marker edge of  $R$  are incident to distinct end vertices of  $e$ . If neither  $f$  nor  $g$  is in  $R$ , then check that the two child marker edges of  $R$  are incident to distinct end vertices of  $e$ .

Note that for every member  $Q \neq R$  of  $\mathcal{D}$ ,  $Q$  contains one of either  $f$ , or  $g$ , or a single child marker edge. For every such member  $Q$  do the following:

- (B.1) If  $Q$  is a bond, then label one of its vertices with  $u$ .
- (B.2) If  $Q$  is a polygon, then check that there is exactly one child marker edge. Permute edges so that the parent marker edge and the unique child marker edge are both incident to a vertex. Label this vertex  $u$ .
- (B.3) If  $Q$  is 3-connected, then check that the parent marker edge, and one of either  $f$  (if  $f$  is in  $Q$ ), or  $g$  (if  $g$  is in  $Q$ ), or a single child marker edge are incident to a single vertex. Label this vertex  $u$ .

### Algorithm for Case (C)

Let  $\mathcal{D}$  be minimal with respect to  $\{f, g\}$ . View  $\mathcal{D}$  as a rooted decomposition tree with the member containing  $f$ , as the root  $R$ .  $R$  has at most one child marker edge.

If  $R$  is a bond, then label either end vertex with  $u$ . Since  $f$  is a non-path edge contained in  $R$ , by Proposition 4,  $R$  is not a polygon.

Suppose  $R$  is 3-connected. If  $g$  is in  $R$ , then check that  $f$  and  $g$  are incident. If  $g$  is not in  $R$ , then check that  $f$  and the child marker edge of  $R$  are incident.

Note that for every other member  $Q \neq R$  of  $\mathcal{D}$ ,  $Q$  contains either  $g$ , or a child marker edge. For every such member  $Q$  do the following:

- (C.1) If  $Q$  is a bond, then label one of its vertices with  $u$ .
- (C.2) If  $Q$  is a polygon then permute edges so that the parent and child marker edges are both incident to a vertex. Label this vertex  $u$ .
- (C.3) If  $Q$  is 3-connected, then check that the parent marker edge is incident to  $g$ , if  $g$  is in  $Q$ , or incident to the child marker edge, if  $g$  is not in  $Q$ . Label this vertex  $u$ .

**Proof of Correctness.** The proof of correctness of the above three algorithms follows from the fact that if an alignment exists then, per force, there is some 2-isomorphic copy that exhibits the alignment. The decomposition of this copy is related to the decomposition of the original graph as described in Theorem 2. Hence, it follows that the (local) alignment conditions, which each algorithm generates, will guarantee the (global) alignment of the edges we seek.  $\square$

#### 4.2. Algorithm to satisfy GAP conditions

We assume that we have two gp-realizations  $(G_1, P_1)$  for  $(A_1, \mathcal{C}_1)$  and  $(G_2, P_2)$  for  $(A_2, \mathcal{C}_2)$ , and their respective minimal decomposition tress  $\mathcal{D}_1$  and  $\mathcal{D}_2$  – minimal with respect to  $e$  and all the crossing edges. Our goal is to find 2-isomorphic copies  $(G'_1, P'_1)$  and  $(G'_2, P'_2)$ , respectively, which satisfy the GAP conditions. The satisfaction of the GAC conditions for  $(G'_1, P'_1)$  and  $(G'_2, P'_2)$  will follow similarly from applying the algorithm of Section 4.2.1.

Recall that to apply Proposition 3 we need to assume no parallel non-path edges. However removing parallel edges in  $G_2$  cannot be done arbitrarily, in particular those of type- $b$ . Hence, if one edge in a set of parallel edges is a type- $b$  edge, then keep it, and delete the rest. If there are no type- $b$  edges, then keep any one of them and delete the rest. Note that 2-connectivity of  $G_i$  is preserved even after this deletion.

Our goal is to identify a pair of edges (possibly non-crossing)  $f$  and  $g$ , such that aligning this pair implies the proper alignment of all edges.

##### 4.2.1. Satisfaction of Condition (1)

Recall that to meet GAP-Condition-(1), we must align all type- $b$  edges of  $(G_1, P_1)$  to one of the two end vertices of the path  $P_1$ .

Since  $\mathcal{D}_1$  is minimal, each leaf member of  $\mathcal{D}_1$  contains a type- $b$  edge. Check that  $D_1$  has at most two leaf members. If  $\mathcal{D}_1$  has exactly one leaf member, then pick any type- $b$  edge in that member and label it as  $f$ , and apply Algorithm for Case (A).

If  $\mathcal{D}_1$  has exactly two leaf members, then pick any type- $b$  edge in each of these members and label them as  $f$  and  $g$ , and apply Algorithm for Case (B). See Fig. 4 for an example.

**Theorem 7.** *If  $(A, \mathcal{C})$  is path graphic, then the above algorithm produces a sequence of Whitney switches that transforms  $(G_1, P_1)$  into  $(G'_1, P'_1)$  which satisfies GAP-Condition-(1).*



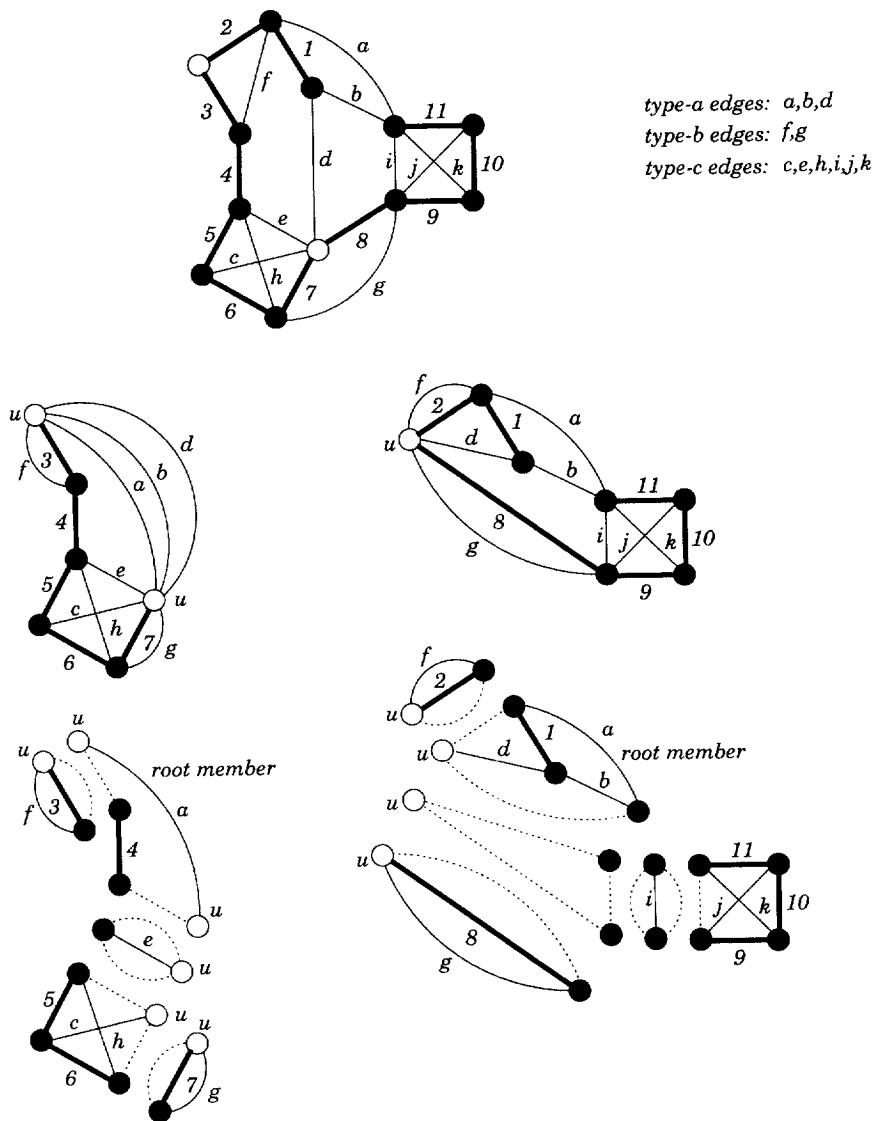


Fig. 4. An example for alignment algorithms. In the decomposition at the bottom left,  $f$  and  $g$  are aligned using algorithm for Case (B), thus satisfying GAP-Condition-(1). At the bottom right,  $f$  and  $g$  are aligned using algorithm for Case (C), thus satisfying GAP-Condition-(2). GAP-Condition-(3) is also satisfied, thus merging produces  $(G'_1, P'_1)$  and  $(G'_2, P'_2)$  (in the middle). Finally, at the top, we combine these two obtaining the desired gp-realization.

**Proof.** If  $(G'_1, P'_1)$  exists, then the set of type- $b$  edges can be partitioned into two disjoint maximal sets  $S_1, S_2$  (possibly empty) such that within each set the associated crossing columns are nested. (A set of crossing columns are *nested* if for every pair  $C_1, C_2$  of the set either  $C_1 \subset C_2$  or  $C_2 \subset C_1$ ). Therefore, if  $\mathcal{D}_1$  has three or more

leaf members, then by Proposition 3, their respective columns are pairwise disjoint, a contradiction. Moreover, due to this nesting of type- $b$  edges, it suffices to find a 2-isomorphic copy in which a unique pair of type- $b$  edges, one from  $S_1$  and one from  $S_2$ , whose corresponding crossing columns do not contain any other crossing column, are incident to the two end vertices of the path. We claim that aligning the type- $b$  edges  $f$  and  $g$  chosen in the above algorithm implies the satisfaction of this property. To see this, note that a leaf member  $L$  of  $\mathcal{D}_1$  cannot be a polygon due to Proposition 4. If  $L$  is a bond, then any type- $b$  edge of  $L$  can be chosen as  $f$  or  $g$ . Similarly, if  $L$  is 3-connected, then by Proposition 3,  $P_1$  restricted to  $L$  is a Hamiltonian path between its parent marker vertices. Therefore, all type- $b$  edge must be incident to one of the parent marker vertices (including the edge most deeply nested). Hence, we may choose any type- $b$  edge of  $L$  as  $f$  or  $g$ . The correctness now follows.  $\square$

#### 4.2.2. Satisfaction of Condition (2)

Recall that to meet GAP-Condition-(2) we must align all type- $b$  edges of  $(G_2, P_2)$  to a single distinguished vertex  $w$ , all type- $a$  edges of  $(G_2, P_2)$  must span or be incident to  $w$ , and all type- $c$  edges of  $(G_2, P_2)$  must not span  $w$ .

Since  $\mathcal{D}_2$  is minimal, each leaf member of  $\mathcal{D}_2$  contains a crossing edge (type- $a$  or type- $b$ ). Check that  $\mathcal{D}_2$  has at most two leaf members. If  $\mathcal{D}_2$  has exactly two leaf members, then pick any one crossing edge (of type- $a$  or type- $b$ ) in each of these members and label them as  $f$  and  $g$ . If  $\mathcal{D}_2$  has exactly one leaf member, then pick any crossing edge in that member and label it as  $f$ . To locate  $g$  we first identify the 3-connected or bond member nearest to the root which has a type- $b$  edge, or a type- $a$  non-spanning edge, or a spanning type- $c$  edge. (A non-path edge is considered a *spanning* edge if it connects two disjoint pieces of the Hamiltonian path restricted to that member.) Pick any one of these edges and label it as  $g$ , if it exists. If it exists then apply Algorithm for Case (C). If it does not exist, then no further alignment is needed.

**Theorem 8.** *If  $(A, \mathcal{C})$  is path graphic, then the above algorithm produces a sequence of Whitney switches that transforms  $(G_2, P_2)$  into  $(G'_2, P'_2)$  which satisfies GAP-Condition-(2).*

**Proof.** If  $(G'_2, P'_2)$  exists, then a subset of crossing edges can be partitioned into two disjoint maximal sets  $S_1, S_2$  (possibly empty) such that within each set the crossing columns are nested (as defined above). Since (by Proposition 3) columns in different members are pairwise disjoint, it follows that  $\mathcal{D}_2$  has at most two leaf members. Moreover, due to the nesting of crossing edges, it suffices to find a 2-isomorphic copy in which a pair of crossing edges whose corresponding crossing columns do not contain any other crossing column, are incident to the same vertex of the path. We claim that the edges  $f$  and  $g$  chosen in the above algorithm indeed satisfy this property.

A leaf member  $L$  of  $\mathcal{D}_2$  cannot be a polygon due to Proposition 4. If  $L$  is a bond, then any crossing edge of  $L$  can be chosen as  $f$  or  $g$ . Similarly, if  $L$  is 3-connected, then

by Proposition 3,  $P_2$  restricted to  $L$  is a Hamiltonian path between its parent marker vertices. Therefore, the crossing edge incident to one of the parent marker vertices whose corresponding column does not contain any other column can be chosen as  $f$  or  $g$ . If  $\mathcal{L}_1$  has exactly one leaf member, then  $f$  is chosen from this leaf member and  $g$  is chosen from a 3-connected member  $N$ . Again by Proposition 3, the unique parent and child marker edges of  $N$  with the edges of  $P_2$  restricted to  $N$  is a Hamiltonian cycle of  $N$ . Therefore, the corresponding columns of  $f$  and  $g$  do not intersect. If such an edge  $g$  does not exist, then either  $S_1$  or  $S_2$  is empty in which case the GAP conditions are easily seen to be satisfied by  $(G_2, P_2)$  itself. In this case there may be zero, one or more split vertices, and one can be found, if one exists, by computing the common intersection of all the crossing columns. (In parallel this can be done using a prefix scan). The correctness now follows.  $\square$

#### 4.2.3. Satisfaction of Condition (3)

Recall that to meet GAP-Condition-(3) we must align all type- $b$  edges so that any two type- $b$  edges are incident to an end vertex of  $P_1$  in  $(G_1, P_1)$  if and only if they are incident to a single distinguished vertex in  $(G_2, P_2)$ , and all type- $a$  edges span or be incident to this same vertex. Notice that no Whitney switch can alter the validity of this property for the gp-realizations  $(G_1, P_1)$  and  $(G_2, P_2)$ . Hence, this condition only needs to be checked after the algorithms of Sections 4.2.1 and 4.2.2 are completed.

#### 4.3. Algorithm to satisfy GAC conditions

The two gp-realization subproblems, associated with the cycle-realization algorithm, are both solved the same way as was done in the case of determining  $(G'_1, P'_1)$ , as discussed above. We leave the details to the reader.

### 5. Complexity analysis

In this section we analyze the sequential and parallel complexity of the main algorithm described in Section 3.3. Recall that the input to the algorithm is an ensemble  $(A, \mathcal{C})$  where  $n = |A|$ ,  $m = |\mathcal{C}|$ , and  $p$  is the sum of the cardinalities of the columns of  $\mathcal{C}$ .

The sequential time complexity of our algorithm can be obtained immediately from the recursion tree which is of depth  $O(\log n)$ . The time complexity of each recursive call is dominated by the linear-time algorithm for Tutte decomposition [12]. Hence, we achieve an overall time complexity of  $O(p \log p)$ .

The parallel complexity is analyzed as follows. Consider the recursion tree obtained from application of the Path-Realization algorithm. We will argue that each level of the recursion tree can be scheduled to run in  $O(\log n)$  time using  $p \log \log n / \log n$  processors on a (CRCW) PRAM. Since the depth of the recursion tree is  $O(\log n)$

our claimed results will follow. Further we show that we can reduce the number of required processors to  $p/\log n$  for sufficiently dense problem instances.

Consider the execution corresponding to level-0, the root of the tree. Step 1, the removal of redundant columns and the addition, if necessary, of a complete column to insure connectivity, is easily done within the resource bounds, and does not increase the overall problem size in a significant way. Case 1 of Step 2 can be done easily in constant time. Case 2 of Step 2 requires the determination of the transformed ensemble which can be done in  $O(1)$  time using  $p$  processors. Then we need to find a connected set of columns meeting the size criterion. It is not difficult to see that by using a tree contraction algorithm [16] on the associated bipartite graph we can determine a connected set of columns of sufficient size. Hence, Step 2 can be done in time  $O(\log n)$  using  $(m + n + p)/\log n$  processors.

Step 3, computation of a Tutte decomposition, can be computed in  $O(\log n)$  time using  $(m + n) \log \log n / \log n$  processors [10].

Step 4, identifying the type of each edge, takes  $O(1)$  time using  $p$  processors, and hence, time  $O(\log n)$  using  $p/\log n$  processors.

Step 5, determination of the minimal decomposition, can be done within the stated resource bounds using standard Tree-Euler-tour techniques [17].

Step 6, computing the Whitney switches, is accomplished using the three alignment algorithms described in Section 4.1. All of the series of check conditions, the labeling of vertices, and possible reordering of edges of polygons can be done in  $O(1)$  time using  $n + m$  processors. Now consider the satisfaction algorithms described in Section 4.2. The series of checks described can all be done in  $O(1)$  time using  $n + m$  processors. The identification of distinguished edges used as input when calling algorithms of Section 4.1 also be done in  $O(1)$  time using  $n + m$  processors.

Step 7 may require a prefix scan (easily computed within resource bounds) to locate the split vertex, and the actual merge is a trivial step.

Hence, the root of the tree can be computed in  $O(\log n)$  time using  $p \log \log n / \log n$  processors. We now show that the same bounds are sufficient to compute each level of the recursion tree. First, for the sake of processor efficiency we modify the stopping conditions so that the recursion terminates when the number of ones  $p_i$  in a subproblem is at most  $\log n$ . For subproblems where  $p_i \leq \log n$  we can apply ours or any near linear time sequential algorithm [6, 4].

From this assumption it follows that the total number of (leaf-vertex) subproblems in the recursion tree is at most  $O(p/\log n)$ ; hence all leaf-subproblems can be computed in parallel in  $O(\log n)$  time using  $p/\log n$  processors.

Suppose at some fixed level  $d$  of the recursion tree there are  $k \leq n/\log n$  (non-leaf) subproblems with parameter sizes defined by the triples  $(p_i, n_i, m_i)$ , for  $1 \leq i \leq k$ . From the analysis above, we have that the  $i$ th subproblem can be solved in  $O(\log n)$  time using

$$p_i/\log n + (n_i + m_i) \log \log n_i / \log n$$

number of processors. It follows that  $\sum_i p_i \leq p$ , and that  $\sum_i n_i \leq n + dk \leq 2n$ , since each recursive call can introduce at most one new row (atom) to each subproblem, and thus there are at most  $dk$  additional rows at level  $d$ . Hence, the total number of processors required to compute this level of the recursion tree is at most

$$p/\log n + n \log \log n / \log n + \sum_{1 \leq i \leq k} m_i \log \log n_i / \log n.$$

We can bound this last term by observing that  $m_i \leq p_i$ ; hence the entire expression is (up to a constant) bounded by  $p \log \log n / \log n$ .

For sufficiently dense problems, we can obtain a slightly improved bound. Suppose that  $p = nm/f$ ,  $f$  being the density factor. Then the last term, involving the sum, is bounded by

$$\begin{aligned} \sum_{1 \leq i \leq k} m \log \log n / \log n &\leq nm \log \log n / \log^2 n \\ &\leq pf \log \log n / \log^2 n, \end{aligned}$$

since  $k \leq n/\log n$ . It follows that if  $f \leq \log n / \log \log n$ , then the processor complexity of the algorithm is  $p/\log n$ . Hence, we have proven the following theorem.

**Theorem 9.** *The algorithm Path-Realization( $A, \mathcal{C}$ ) of Section 3.3, with input size  $n = |A|$  and  $p$  the sum of the cardinalities of the columns of  $\mathcal{C}$ , solves the CIP problem on a RAM with sequential-time complexity  $O(p \log p)$ . Furthermore, the same algorithm solves the CIP problem on a PRAM with parallel-time complexity  $O(\log^2 n)$  using  $p \log \log n / \log n$  processors. The number of processors can be reduced by a factor  $\log \log n$  for all sufficiently dense problem instances.*

## References

- [1] F. Alizadeh, R.M. Karp, D.K. Weisser, G. Zweig, Physical mapping of chromosomes using unique probes, Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms 1994, pp. 489–500.
- [2] C. Berge, Balanced matrices, Mathematical Programming 2 (1972) 19–31.
- [3] S.M. Bhandarkar, S. Chirravuri, J. Arnold, Parallel computing of physical maps – a comparative study in SIMD and MIMD parallelism. J. Comput. Biol. 3 (4) (1996) 503–528.
- [4] R.E. Bixby, D.K. Wagner, An almost linear-time algorithm for graph realization, Math. Oper. Res. 13 (1998) 99–123.
- [5] J.A. Bondy, U.S.R. Murty, Graph Theory with Applications, North-Holland, New York, 1996.
- [6] K.S. Booth, G.S. Lueker, Testing for consecutive 1's property, interval graphs and graph planarity using PQ-tree algorithms, J. Comput. System Sci. 13 (1976) 335–379.
- [7] L. Chen, Y. Yesha, Parallel recognition of the consecutive-ones property with applications, J. Algorithms 12 (1991) 375–392.
- [8] W.H. Cunningham, J. Edmonds, A combinatorial decomposition theory, Canadian J. Math. 32 (1980) 734–765.
- [9] N. Deo, M.S. Krishnamoorthy, M.A. Langston, Exact and approximate solutions for the gate matrix problem, IEEE Trans. Computer-Aided Design 6 (1987) 79–84.
- [10] D. Fussell, V. Ramachandran, R. Thurimella, Finding triconnected components by local replacement, SIAM J. Computing 22 (3) (1993) 587–616.

- [11] S.P. Ghosh, File organization: The consecutive-retrieval property, *Comm. ACM* 15 (1972) 802–808.
- [12] J. Hopcroft, R.E. Tarjan, Dividing a graph into triconnected components, *SIAM J. Computing* 2 (1973) 135–158.
- [13] P.N. Klien, Efficient parallel algorithms for planar, chordal and interval graphs MIT/LCS/TR-426, Ph.D. Thesis, 1988.
- [14] P.N. Klien, J. Reif, An efficient parallel algorithms for planarity, *J. Comput. System Sci.* 37 (1988) 190–246.
- [15] E.S. Lander, Mapping heredity: using probabilistic models and algorithms to map genes and genomes, *Notices American Math. Soc.* 42 (8) (1995) 854–858.
- [16] G.L. Miller, J. Reif, Parallel tree contraction and its application, 26th Symposium on Foundations of Computer Science, 1985, pp. 478–489.
- [17] R.E. Tarjan, U. Vishkin, An efficient parallel biconnectivity algorithm, *SIAM J. Comput.* 14 (4) (1985) 862–874.
- [18] K. Truemper, *Matroid Decomposition*, Academic Press, New York, 1993.
- [19] A.C. Tucker, A structure theorem for the consecutive 1's property, *J. Combin. Theory Ser. B.* 12 (1972) 153–162.
- [20] W.T. Tutte, *Connectivity in Graphs*, University of Toronto Press, Toronto, 1966.
- [21] H. Whitney, Non-separable and planar graphs, *Trans. Amer. Math. Soc.* 34 (1932) 339–362.