Journal of Algorithms **37,** 505–521 (2000)
doi:10.1006/jagm.1999.1122, available online at http://www.idealibrary.com on IDEAL®

Finding Skew Partitions Efficiently[1]

Celina M. H. de Figueiredo and Sulamita Klein

Instituto de Matemática and COPPE, Universidade Federal do Rio de Janeiro, Caixa Postal 68530, 21945-970 Rio de Janeiro, RJ, Brazil
E-mail: celina@cos.ufrj.br, sula@cos.ufrj.br

Yoshiharu Kohayakawa

Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508-900 São Paulo, SP, Brazil
E-mail: yoshi@ime.usp.br

and

Bruce A. Reed

CNRS, Institut Blaise Pascal, Université Pierre et Marie Curie, case 189 - Combinatoire, 4 Place Jussieu, 75252 Paris Cedex 5, France
E-mail: reed@ecp6.jussieu.fr

Received January 31, 2000

A skew partition as defined by Chvátal is a partition of the vertex set of a graph into four nonempty parts A, B, C, D such that there are all possible edges between A and B and no edges between C and D. We present a polynomial-time algorithm for testing whether a graph admits a skew partition. Our algorithm solves the more general list skew partition problem, where the input contains, for each vertex, a list containing some of the labels A, B, C, D of the four parts. Our polynomial-time algorithm settles the complexity of the original partition problem proposed by Chvátal in 1985 and answers a recent question of Feder, Hell, Klein, and Motwani.

[1]Research partially supported by CNPq, MCT/FINEP PRONEX Project 107/97, CAPES(Brazil)/COFECUB(France) Project 213/97, FAPERJ, and FAPESP Proc. 96/04505-2.

0196-6774/00 $35.00

1. INTRODUCTION

A *skew partition* is a partition of the vertex set of a graph into four nonempty parts A, B, C, D such that there are all possible edges between A and B and no edges between C and D. We present a polynomial-time algorithm for testing whether a graph admits a skew partition, as well as for the more general *list skew partition problem*, where the input contains, for each vertex, a list containing some of the four parts.

Many combinatorial problems can be described as finding a partition of the vertices of a given graph into subsets satisfying certain properties *internally* (some parts may be required to be independent or sparse in some other sense, while others may conversely be required to be complete or dense) and *externally* (some pairs of parts may be required to be completely nonadjacent, others completely adjacent). Feder *et al.* [10] defined a parameterized family of graph problems of this type. The basic family of problems they considered is as follows: partition the vertex set of a graph into k parts $A_1, A_2, \ldots, A_k$ with a fixed *pattern* of requirements in which A_i are independent or complete and in which pairs A_i, A_j are completely nonadjacent or completely adjacent. These requirements may be conveniently encoded by a symmetric $k \times k$ matrix M in which the diagonal entry $M_{i,i}$ is 0 if A_i is required to be independent, 2 if A_i is required to be a clique, and 1 otherwise (no restriction). Similarly, the off-diagonal entry $M_{i,j}$ is 0, 1, or 2, if A_i and A_j are required to be completely nonadjacent, have arbitrary connections, or are required to be completely adjacent, respectively. Feder *et al.* [10] call such a partition an *M-partition*.

Many combinatorial problems just ask for an M-partition. For instance, a k-coloring is an M-partition where M is the adjacency matrix of the complete k-graph. More generally, an H-coloring (homomorphism to a fixed graph H [12]) is an M-partition where M is the adjacency matrix of H. It is known that H-coloring is polynomial-time solvable when H is bipartite and NP-complete otherwise [12]. When M is the adjacency matrix of H plus twice the identity matrix (all diagonal elements are 2), M-partitions reduce to the so-called (H, K)-partitions, which were studied by MacGillivray and Yu [15]. When H is triangle-free, the (H, K)-partition is polynomial-time solvable, otherwise it is NP-complete.

Other well-known problems ask for M-partitions in which all parts are restricted to being nonempty (e.g., skew partitions, clique cutsets, stable cutsets). In yet other problems there are additional constraints, such as those in the definition of a homogeneous set (requiring one of the parts to have at least 2 and at most $n - 1$ vertices). For instance, Winkler asked for the complexity of deciding the existence of an M-partition, where M has the rows 1101, 1110, 0111, and 1011, such that all parts A, B, C, D are nonempty and there is at least one edge between parts A and B, B and C,

C and D, and D and A. This has recently been shown to be NP-complete by Vikas [17].

The most convenient way to express these additional constraints is to allow specifying (as part of the input) for each vertex a *list* of parts in which the vertex is allowed. Specifically, the *list-M-partition problem* asks for an M-partition of the input graph in which each vertex is placed in a part which is in its list. Both the basic M-partition problem ("Does the input graph admit an M-partition?") and the problem of existence of an M-partition with all parts nonempty admit polynomial-time reductions to the list-M-partition problem, as do all of the above problems with the "additional" constraints. List partitions generalize list-colorings, which have proved very fruitful in the study of graph colorings [1, 11]. They also generalize list-homomorphisms, which were studied earlier [8, 9].

Feder *et al.* [10] were the first to introduce and investigate the list version of these problems. It turned out to be a useful generalization, since list problems recurse more conveniently. This enabled them to classify the complexity (as polynomial-time solvable or NP-complete) of list-M-partition problems for all 3×3 matrices M and some 4×4 matrices M. For other 4×4 matrices M, they were able to produce subexponential algorithms—including one for the skew partition problem described below. This was the first sub-exponential algorithm for the problem and an indication that the problem was not likely to be NP-complete. Motivated by their approach, we show in this paper that in fact one can use the mechanism of list partitions to give a *polynomial-time* algorithm for the problem.

A *skew partition* is an M-partition where M has the rows 1211, 2111, 1110, and 1101, such that all parts are nonempty. The list skew partition problem (LSP) is simply the list-M-partition problem for this M. We can solve skew partition by solving at most n^4 LSP problems such that $v_i \in A_i$ for $1 \leq i \leq 4$, for all possible quadruples $\{v_1, v_2, v_3, v_4\}$ of vertices of the input graph.

The skew partition problem has interesting links to perfect graphs and is one of the main problems in the area. Before presenting our two algorithms we discuss perfect graphs and their link to skew partition.

2. SKEW CUTSETS AND THE STRONG PERFECT GRAPH CONJECTURE

A graph is *perfect* if each induced subgraph admits a vertex coloring and a clique of the same size. A graph is *minimal imperfect* if it is not perfect but all its properly induced subgraphs are perfect. Perfect graphs were defined in 1960 by Berge [2], who proposed the *strong perfect graph conjecture*: the only minimal imperfect graphs are the odd chordless cycles of length at

least 5 and their complements. Since then researchers have enumerated a list of properties of minimal imperfect graphs. The strong perfect graph conjecture remains open and is considered a central problem in graph theory. The recognition of perfect graphs is a famous open problem [14].

In 1985, Chvátal [4] proved that no minimal imperfect graph contains a *star cutset*: a vertex cutset consisting of a vertex and some of its neighbors. Chvátal exhibited a polynomial-time recognition algorithm for graphs with a star cutset. He also defined the following generalization of star cutsets: A *skew partition* is a partition of the vertex set of a graph into four nonempty parts A, B, C, D such that there are all possible edges between A and B and no edges between C and D. We call each of the four nonempty parts A, B, C, D *a skew partition set* and we say that $A \cup B$ is a *skew cutset*. Chvátal [4] proposed the *skew partition conjecture*: no minimal imperfect graph contains a skew partition. This conjecture and the complexity of testing for the existence of a skew cutset attracted some attention in recent years [6, 5, 10, 13, 16]. In particular, Feder *et al.* [10] described a quasi-polynomial algorithm for testing whether a graph admits a skew partition, which strongly suggested that this problem was not NP-complete. In this paper, we present a polynomial-time recognition algorithm for testing whether a graph admits a skew partition.

Hoàng [13] proved that no minimal imperfect graph contains a *three-pair*: a pair of nonadjacent vertices such that all chordless paths joining them have exactly three edges. On the way to establishing this property of minimal imperfect graphs, Hoàng obtained several partial results on the skew partition conjecture.

Cornuéjols and Reed [5] proved that no minimal imperfect graph contains a skew partition in which A and B are both stable sets. Actually, they proved the following more general result. Let a *complete multipartite* graph be one whose vertex set can be partitioned into stable sets $S_1, \ldots, S_k$, such that there are all possible edges between S_i and S_j, for $i \neq j$. They proved that no minimal imperfect graph contains a skew cutset that induces a complete multipartite graph. Their work raised questions about the complexity of testing for the existence of either a complete bipartite cutset or a complete multipartite cutset in a graph.

Subsequently, Klein and de Figueiredo [16] showed how to use a result of Chvátal [3] on matching cutsets in order to establish the NP-completeness of recognizing graphs with a stable cutset. In addition, they established the NP-completeness of recognizing graphs with a complete multipartite cutset. In particular, their proof showed that it is NP-complete to test for the existence of a complete bipartite cutset, even if the cutset induces a $K_{1,p}$.

As shown by Chvátal [4], to test for the existence of a star cutset is in P, whereas to test for the existence of the special star cutset $K_{1,p}$ is NP-complete [16]. The polynomial-time algorithm described in this paper

offers an analogous complexity situation: to test for the existence of a skew cutset is in P, whereas to test for the existence of a complete bipartite cutset is NP-complete [16].

3. OVERVIEW

The goal of this paper is to present a polynomial-time algorithm for the following decision problem:

Skew Partition Problem

Input: a graph $G = (V, E)$.

Question: Is there a skew partition A, B, C, D of G?

We actually consider LSP problems, stated as decision problems, as follows:

List Skew Partition Problem

Input: a graph $G = (V, E)$ and, for each vertex $v \in V$, a subset L_v of $\{A, B, C, D\}$.

Question: Is there a skew partition A, B, C, D of G such that each v is contained in some element of the corresponding L_v?

Throughout the algorithm, we have a partition of V into at most 15 sets S_L, indexed by the nonempty subsets L of $\{A, B, C, D\}$, such that Property 1 below is always satisfied. Note that the relevant inputs for LSP have S_A, S_B, S_C, and S_D nonempty.

Property 1. If the algorithm returns a skew partition, then if v is in S_L, then the returned skew partition set containing v is in L.

Initially, we set $S_L = \{v : L_v = L\}$, for each $L \subseteq \{A, B, C, D\}$.

We also restrict our attention to LSP instances that satisfy the following property:

Property 2. If $v \in S_L$, for some L with $A \in L$, then v is adjacent to every vertex of S_B. If $v \in S_L$, for some L with $B \in L$, then v is adjacent to every vertex of S_A. If $v \in S_L$, for some L with $C \in L$, then v is nonadjacent to every vertex of S_D. If $v \in S_L$, for some L with $D \in L$, then v is nonadjacent to every vertex of S_C.

Both Properties 1 and 2 hold throughout the algorithm.

Remark 1. Since S_B must be contained in B, we know that if v is to be in A for some solution to the problem, then v must be adjacent to all of S_B. Thus if some $v \in S_A$ is not adjacent to a vertex of S_B, then there is no solution to the problem and we need not continue. If there is some L with A properly contained in L and a vertex v in S_L which is not adjacent to a vertex of S_B, then we know that in any solution to the problem v must be contained in some element of $L \setminus A$. So we can reduce to a new problem where we replace S_L by $S_L \setminus v$, we replace $S_{L \setminus A}$ by $S_{L \setminus A} + v$, and all other S_L are as before. Such a reduction reduces $\sum_L |S_L||L|$ by 1. Since this sum is at most $4n$, after $O(n)$ similar reductions we must obtain an LSP problem satisfying Property 2 (or halt because the original problem has no solution).

In our discussion we often create new LSP instances and whenever we do so, we always perform this procedure to reduce to an LSP problem satisfying Property 2.

For an instance I of LSP we have $\{S_L(I) : L \subseteq \{A, B, C, D\}\}$, but we drop the (I) when it is not needed for clarity.

We consider a number of restricted versions of the LSP problems:

- MAX-3-LSP: an LSP problem satisfying Property 2 such that $S_{ABCD} = \emptyset$;
- MAX-2-LSP: an LSP problem satisfying Property 2 such that if $|L| > 2$, then $S_L = \emptyset$;
- AC-TRIV LSP: an LSP problem satisfying Property 2 such that $S_{AC} = \emptyset$;

Remark 2. Recall that the relevant inputs for LSP have S_A, S_B, S_C, and S_D nonempty. It is easy to obtain a solution to an instance of AC-TRIV-LSP as follows: $A = S_A$, $B = \bigcup_{B \in L} S_L$, $C = S_C$, and $D = \bigcup_{D \in L,\, B \notin L} S_L$. By Property 2 this is indeed a skew partition.

- BD-TRIV LSP, AD-TRIV LSP, BC-TRIV-LSP; these problems are defined and solved similar to AC-TRIV LSP.

Our algorithm for solving LSP requires four subalgorithms which replace an instance of LSP by a polynomial number of instances of more restricted versions of LSP.

ALGORITHM 1. This algorithm takes an instance of LSP and returns in polynomial time a list $\mathcal{L}$ of a polynomial number of instances of MAX-3-LSP such that

(i) a solution to any problem in $\mathcal{L}$ is a solution of the original problem, and

(ii) if none of the problems in $\mathcal{L}$ have a solution, then the original problem has no solution.

ALGORITHM 2. This algorithm takes an instance of MAX-3-LSP and returns in polynomial time a list $\mathcal{L}$ of a polynomial number of instances of MAX-3-LSP such that

(i) and (ii) of Algorithm 1 hold, and

(iii) for each problem in $\mathcal{L}$, either $S_{ABC} = \emptyset$ or $S_{ABD} = \emptyset$.

ALGORITHM 3. This algorithm takes an instance of MAX-3-LSP and returns in polynomial time a list $\mathcal{L}$ of a polynomial number of instances of MAX-3-LSP such that

(i) and (ii) of Algorithm 1 hold, and

(iii) for each problem in $\mathcal{L}$, either $S_{BCD} = \emptyset$ or $S_{ACD} = \emptyset$.

ALGORITHM 4. This algorithm takes an instance of MAX-3-LSP such that

(a) either S_{ABC} or S_{ABD} is empty, and

(b) either S_{BCD} or S_{ACD} is empty

and returns in polynomial time a list $\mathcal{L}$ of a polynomial number of problems each of which is an instance of one of MAX-2-LSP, AC-TRIV-LSP, AD-TRIV-LSP, BC-TRIV-LSP, or BD-TRIV-LSP such that (i) and (ii) of Algorithm 1 hold.

We also need two additional algorithms for dealing with the most basic instances of LSP.

ALGORITHM 5. This algorithm takes an instance of MAX-2-LSP and returns either

(i) a solution to this instance of MAX-2-LSP, or

(ii) the information that this problem instance has no solution.

Remark 3. Algorithm 5 simply applies 2-SAT as discussed in [7]; we omit the details.

ALGORITHM 6. This algorithm takes an instance of AC-TRIV LSP, AD-TRIV LSP, BC-TRIV LSP, or BD-TRIV LSP and returns a solution using the partitions discussed in Remark 2.

To solve an instance of LSP, we first apply Algorithm 1 to obtain a list L_1 of instances of MAX-3-LSP. For each problem instance I on L_1, we apply Algorithm 2 and let L_I be the output list of problem I. We let L_2 be the concatenation of the lists $\{L_I : I \in L_1\}$. For each I in L_2, we apply Algorithm 3. Let L_3 be the concatenation of the lists $\{L_I : I \in L_2\}$. For each problem instance I on L_3, we apply Algorithm 4. Let L_4 be the concatenation of the lists $\{L_I : I \in L_3\}$. Each element of L_4 can be solved in

polynomial time using either Algorithm 5 or Algorithm 6. If any of these problems has a solution S, then by the specifications of the algorithms, S is a solution to the original problem. Otherwise, by the specifications of the algorithms, there is no solution to the original problem. Clearly, the whole algorithm runs in polynomial time.

4. SOME RECURSIVE PROCEDURES

Algorithm 1 recursively applies Procedure 1, which runs in polynomial time.

Procedure 1.

Input: An instance I of LSP.

Output: Four instances I_1, I_2, I_3, I_4 of LSP such that, for $1 \leq j \leq 4$, we have $|S_{ABCD}(I_j)| \leq \frac{9}{10}|S_{ABCD}(I)|$.

It is easy to prove inductively that applying Procedure 1 recursively yields a polynomial-time implementation of Algorithm 1 which when applied to an input graph with n vertices creates as output a list $\mathscr{L}$ of instances of LSP such that $|\mathscr{L}| \leq 4^{\log_{10/9} n} = n^{\log_{10/9} 4} \leq n^{14}$.

Algorithm 2 recursively applies Procedure 2, which runs in polynomial time.

Procedure 2.

Input: An instance I of MAX-3-LSP.

Output: Four instances I_1, I_2, I_3, I_4 of MAX-3-LSP such that, for $1 \leq j \leq 4$, we have $|S_{ABC}(I_j)||S_{ABD}(I_j)| \leq \frac{9}{10}|S_{ABC}(I)||S_{ABD}(I)|$.

It is easy to prove that applying Procedure 2 recursively to an input graph with n vertices yields a polynomial-time implementation of Algorithm 2 which creates an output list $\mathscr{L}$ with at most $4^{\log_{10/9} n^2} \leq n^{28}$ elements.

Algorithm 3 recursively applies Procedure 3, which runs in polynomial time.

Procedure 3.

Input: An instance I of MAX-3-LSP.

Output: Four instances I_1, I_2, I_3, I_4 of MAX-3-LSP such that, for $1 \leq j \leq 4$, we have $|S_{BCD}(I_j)||S_{ACD}(I_j)| \leq \frac{9}{10}|S_{BCD}(I)||S_{ACD}(I)|$.

It is easy to prove that applying Procedure 3 recursively to an input graph with n vertices yields a polynomial-time implementation of Algorithm 3 which creates an output list $\mathscr{L}$ with at most $4^{\log_{10/9} n^2} \leq n^{28}$ elements.

Algorithm 4 recursively applies Procedure 4, which runs in polynomial time.

Procedure 4.

Input: An instance I of MAX-3-LSP such that S_{ABD} is empty and S_{ACD} is empty.

Output: Four instances I_1, I_2, I_3, I_4 of MAX-3-LSP such that, for $1 \leq j \leq 4$, we have $|S_{ABC}(I_j)||S_{AD}(I_j)| \leq \frac{9}{10}|S_{ABC}(I)||S_{AD}(I)|$.

Algorithm 4 also recursively applies three procedures whose definitions are similar. We will not need to give formal specifications for them. It is easy to see that recursively applying Procedure 4 or one of its variants, as appropriate, yields a polynomial-time implementation of Algorithm 4 which when applied to an input graph with n vertices yields an output list with at most n^{28} elements. It remains only to describe the four procedures.

5. THE DETAILS

Procedure 1.

Let $n = |S_{ABCD}(I)|$. For a skew partition $\{A, B, C, D\}$, let $A' = A \cap S_{ABCD}(I)$, $B' = B \cap S_{ABCD}(I)$, $C' = C \cap S_{ABCD}(I)$, and $D' = D \cap S_{ABCD}(I)$.

Case 1: There exists a vertex v in S_{ABCD} such that $\frac{n}{10} \leq |S_{ABCD} \cap N(v)| \leq \frac{9n}{10}$.

Branch according to whether $v \in A$, $v \in B$, $v \in C$, or $v \in D$ with instances I_A, I_B, I_C, I_D, respectively. Define I_A by initially setting $S_A(I_A) = v + S_A(I)$ and reducing so that Property 2 holds. Define I_B, I_C, I_D similarly. Note that by Property 2, if $v \in C$, then $D \cap N(v) = \emptyset$. So, $S_{ABCD}(I_C) \subset S_{ABCD}(I) \backslash N(v)$. Because there are at least $\frac{n}{10}$ vertices in $S_{ABCD} \cap N(v)$, this means $|S_{ABCD}(I_C)| \leq \frac{9n}{10}$. Symmetrically, $|S_{ABCD}(I_D)| \leq \frac{9n}{10}$.

Similarly, by Property 2, $S_{ABCD}(I_A) \subset S_{ABCD}(I) \cap N(v)$, so $|S_{ABCD}(I_A)| \leq \frac{9n}{10}$. Symmetrically $|S_{ABCD}(I_B)| \leq \frac{9n}{10}$.

Let $W = \{v \in S_{ABCD} : |S_{ABCD} \cap N(v)| > \frac{9n}{10}\}$ and $X = \{v \in S_{ABCD} : |S_{ABCD} \cap N(v)| < \frac{n}{10}\}$.

Case 2: There are at least $\frac{n}{10}$ vertices in X and there are at least $\frac{n}{10}$ vertices in W.

Branch according to $|A'| \geq \frac{n}{10}$, $|B'| \geq \frac{n}{10}$, $|C'| \geq \frac{n}{10}$, or $|D'| \geq \frac{n}{10}$ with corresponding instances $I_{A'}, I_{B'}, I_{C'}$, and $I_{D'}$. Each of these choices forces either all the vertices in W or all the vertices in X to have smaller label sets, as follows. If $|A'| \geq \frac{n}{10}$, then every vertex in B has $\frac{n}{10}$ neighbors in $S_{ABCD}(I)$, so $B \cap X = \emptyset$. Thus, $S_{ABCD}(I_{A'}) = S_{ABCD}(I) \backslash X$, and $|S_{ABCD}(I_{A'})| \leq \frac{9n}{10}$. If $|B'| \geq \frac{n}{10}$, then a symmetrical argument shows that $X \cap A = \emptyset$. Thus, $S_{ABCD}(I_{B'}) = S_{ABCD}(I) \backslash X$ and $|S_{ABCD}(I_{B'})| \leq \frac{9n}{10}$. If $|C'| \geq \frac{n}{10}$, then every

vertex in D has at least $\frac{n}{10}$ nonneighbors in $S_{ABCD}(I)$. Hence $W \cap D = \emptyset$, $S_{ABCD}(I_{C'}) = S_{ABCD}(I) \backslash W$, and so $|S_{ABCD}(I_{C'})| \leq \frac{9n}{10}$. If $|D'| \geq \frac{n}{10}$, then a symmetrical argument shows that $|S_{ABCD}(I_{D'})| \leq \frac{9n}{10}$.

Case 3: There are more than $\frac{9n}{10}$ vertices in W.

We will repeatedly apply the following procedure to various $W' \subseteq W$ with $|W'| \geq \frac{8n}{10}$. We recursively define a partition of W' into three sets O, T, and NT such that:

- There are all edges between O and T;
- For every w in NT, there exists v in O such that w is not adjacent to v;
- The complement of O is connected.

Start by choosing v_1 in W' and setting $O = \{v_1\}$, $T = N(v_1) \cap W'$, and $NT = W' \backslash (N(v_1) \cup \{v_1\})$. Note that for each vertex v of W', since v is not adjacent to at most $\frac{n}{10}$ vertices of S_{ABCD}, we have $|N(v) \cap W'| > |W'| - \frac{n}{10}$. So $|NT| = |W' \backslash (N(v_1) \cup \{v_1\})| < \frac{n}{10}$. Increase O by moving an arbitrary vertex v from NT to O and by moving $T \backslash N(v)$ from T to NT, until:

(i) $|O| + |NT| \geq \frac{n}{10}$; or
(ii) $NT = \emptyset$.

If the growth process stops with condition (i), i.e., $|O| + |NT| \geq \frac{n}{10}$, and v_i was the last vertex added to O, then adding v_i to O increased $|NT|$ by at most $|W' \backslash (N(v_i) \cup \{v_i\})| < \frac{n}{10}$. Thus, $|O| + |NT| < \frac{n}{10} + \frac{n}{10} = \frac{2n}{10}$. So, $|T| \geq \frac{8n}{10} - \frac{2n}{10} = \frac{6n}{10} \geq \frac{n}{10}$.

Our first application of the procedure is with $W' = W$. If we stop because (i) holds, then we define four new instances of LSP according to the intersection of the skew partition sets A, B, C, or D with O, as follows:

(a) $I_1 : C \cap O \neq \emptyset$,
(b) $I_2 : C \cap O = \emptyset$, $D \cap O \neq \emptyset$,
(c) $I_3 : O \subseteq A$,
(d) $I_4 : O \subseteq B$.

Recall that the complement of O is connected, which implies that if $O \cap (C \cup D) = \emptyset$, then either $O \subseteq A$ or $O \subseteq B$. If $O \subseteq A$, then $NT \cap B = \emptyset$ since for every $w \in NT$ there is a vertex $v \in O$ such that $vw \notin E$. Thus, $(O \cup NT) \cap S_{ABCD}(I_3) = \emptyset$. Hence $|S_{ABCD}(I_3)| \leq \frac{9n}{10}$. A symmetrical argument shows that $|S_{ABCD}(I_4)| \leq \frac{9n}{10}$.

If $C \cap O \neq \emptyset$, then $D \cap T = \emptyset$. Thus, $T \cap S_{ABCD}(I_1) = \emptyset$, which implies $|S_{ABCD}(I_1)| \leq \frac{9n}{10}$. A symmetrical argument shows that $|S_{ABCD}(I_2)| \leq \frac{9n}{10}$. Thus, if our application of the above procedure halts with output (i), then we have found the four desired output instances of LSP.

Otherwise, the growth process stops with condition (ii), i.e., $NT = \varnothing$ and $|O| < \frac{n}{10}$. Set $O_1 = O$ and reapply the algorithm to $W' = W \backslash O_1$ to obtain O_2. More generally, having constructed disjoint sets $O_1, \ldots, O_i$ with $|\bigcup_{j=1}^{i} O_j| < \frac{n}{10}$, we construct O_{i+1} by applying the algorithm to $W_i = W \backslash \bigcup_{j=1}^{i} O_j$. Note $|W_i| > \frac{8n}{10}$.

We continue until $|\bigcup_{j=1}^{i} O_j| \geq \frac{n}{10}$ or condition (i) occurs. If condition (i) ever occurs, then we proceed as above. Otherwise, we stop after some iteration i^* such that $|\bigcup_{i<i^*} O_i| < \frac{n}{10}$ and $|\bigcup_{i \leq i^*} O_i| \geq \frac{n}{10}$. Since $|O_{i^*}| < \frac{n}{10}$, we have that $|\bigcup_{i \leq i^*} O_i| \leq \frac{2n}{10}$. Also, all the edges between sets $Z = \bigcup_{j=1}^{i} O_j$ and $Y = W \backslash \bigcup_{j=1}^{i} O_j$ exist, which implies that $C \cap Z = \varnothing$ or $D \cap Y = \varnothing$.

We now define two new instances of LSP according to the intersection of skew partition sets C or D with Z, as follows:

(a) I_1: $C \cap Z = \varnothing$,

(b) I_2: $D \cap Y = \varnothing$.

In either output instance I_i, $|S_{ABCD}(I_i)| \leq \frac{9n}{10}$.

The case $|X| > \frac{9n}{10}$ is symmetric to Case 3 (consider $\overline{G}$) and is omitted. This ends Procedure 1.

Procedure 2.

Let $S_1 = S_{ABC}(I)$ and $S_2 = S_{ABD}(I)$. Let $n_1 = |S_1|$ and $n_2 = |S_2|$. Given $v \in S_1 \cup S_2$, let $d_i(v) = |N(v) \cap S_i|$, $i = 1, 2$.

Case 1: There exists a vertex $v \in S_1$ with $n_2/10 \leq d_2(v) \leq 9n_2/10$.

This case is analogous to Case 1 of Procedure 1: we create three new instances I_A, I_B, I_C according to whether $v \in A$, $v \in B$, or $v \in C$. $S_{ABD}(I_A) \subseteq S_2 \cap N(v)$ and hence $|S_{ABD}(I_A)| \leq 9n_2/10$. The other two cases are handled similarly.

The case "there exists a vertex $v \in S_2$ with $n_1/10 \leq d_1(v) \leq 9n_1/10$", symmetric to Case 1, is omitted.

Case 2: Every vertex v in S_1 satisfies $d_2(v) < n_2/10$ or $d_2(v) > 9n_2/10$. Every vertex v in S_2 satisfies $d_1(v) < n_1/10$ or $d_1(v) > 9n_1/10$.

Define four auxiliary sets, as follows:

$$X_1 = \left\{ v \in S_1 : d_2(v) < \frac{n_2}{10} \right\},$$

$$X_2 = \left\{ v \in S_2 : d_1(v) < \frac{n_1}{10} \right\},$$

$$W_1 = \left\{ v \in S_1 : d_2(v) > \frac{9n_2}{10} \right\},$$

$$W_2 = \left\{ v \in S_2 : d_1(v) > \frac{9n_1}{10} \right\}.$$

Note that Case 2 means that $S_1 = X_1 \cup W_1$ and $S_2 = X_2 \cup W_2$. We handle Case 2 according to the following three possibilities.

Case 2.1: $|X_1|, |W_1| \geq n_1/10$.

This case is analogous to Case 2 of Procedure 1. We create three new instances of LSP according to the size of skew partition sets, as follows:

(a) I_1: $|A \cap S_2| \geq n_2/10$,

(b) I_2: $|B \cap S_2| \geq n_2/10$,

(c) I_3: $|D \cap S_2| \geq n_2/10$.

If $|A \cap S_2| \geq n_2/10$, then as every vertex in B is adjacent to every vertex of A, we have $B \cap X_1 = \emptyset$. Thus, $S_{ABC}(I_1) \leq 9n_1/10$. Similarly, $S_{ABC}(I_2) \cap X_1 = \emptyset$ and $S_{ABC}(I_3) \cap W_1 = \emptyset$. So, for all three output instances, we have $|S_{ABC}| \leq 9n_1/10$, as required.

The case "otherwise $|X_2|, |W_2| \geq n_2/10$", symmetric to Case 2.1, is omitted.

Case 2.2: $|X_1| > 9n_1/10$.

Let $v \in X_1$. Recursively define three sets O, M, and NM such that:

- $O \subseteq X_1$, $S_2 = M \cup NM$;
- There are no edges between O and M;
- For every $u \in NM$, there is a $w \in O$ with $wu \in E$;

Start by setting $O = \{v\}$, $M = S_2 \backslash (N(v) \cup \{v\})$, and $NM = S_2 \cap N(v)$. Increase O in X_1 until:

(i) $|M| \leq n_2/2$; or

(ii) $|O| \geq 3n_1/10$.

If the growth process stops with condition (ii), i.e., $|O| \geq 3n_1/10$ and $|M| > n_2/2$, then define two new instances of LSP as follows:

(a) $I_1 : O \cap A = \emptyset$,

(b) $I_2 : O \cap A \neq \emptyset$.

Clearly, $S_{ABC}(I_1) \subseteq S_1 \backslash O$, and so $|S_{ABC}(I_1)| \leq 9n_1/10$. Further, if $O \cap A \neq \emptyset$, then $M \cap B = \emptyset$, so $|S_{ABD}(I_2)| \leq 9n_2/10$.

If the growing process stops with condition (i), i.e., $|M| \leq n_2/2$, then we define three new LSP instances:

(a) $I_1 : O \cap A \neq \emptyset$,

(b) $I_2 : O \cap B \neq \emptyset$,

(c) $I_3 : O \subseteq C$.

Because the last vertex added to O was in X_1, we have $n_2/2 - n_2/10 = 4n_2/10 < |M| \leq n_2/2$. In addition, note that $|NM| \geq n_2/2$.

If $O \cap A \neq \emptyset$, then $B \subseteq (S_2 \backslash M)$, so $|S_{ABD}(I_1)| \leq 9n_2/10$. Similarly $|S_{ABD}(I_2)| \leq 9n_2/10$. Finally, if $O \subseteq C$ then $NM \cap D = \emptyset$, so $|S_{ABD}(I_3)| \leq n_2/2$.

The case "otherwise $|X_2| > 9n_2/10$," symmetric to Case 2.2, is omitted.

Case 2.3: $|W_1| > \frac{9n_1}{10}$, and $|W_2| > \frac{9n_2}{10}$.

Let $W = W_1 \cup W_2$. We claim that each vertex $v \in W_1$ has $d_1(v) \geq 9n_1/10$. Suppose there exists $v \in W_1$ with $d_1(v) < 9n_1/10$. Consider instances I_A: $v \in A$, I_B: $v \in B$, and I_C: $v \in C$. If $v \in A$, then there are at least $n_1/10$ vertices in S_1 that are nonadjacent to v, which cannot be in B. So $|S_{ABC}(I_A)| \leq 9n_1/10$. Similarly $|S_{ABC}(I_B)| \leq 9n_1/10$. Finally, if $v \in C$, then there at least $n_2/10$ vertices adjacent to v in S_2 which cannot be in D so $|S_{ABD}(I_2)| \leq 9n_2/10$. Thus, if $d_1(v) < 9n_1/10$ for some $v \in S_1$, then we are done. Analogously, each vertex $v \in W_2$ has $d_2(v) \geq 9n_2/10$.

Recursively define a partition of W into three sets O, T, and NT such that:

- The complement of O is connected;
- There are all edges between O and T;
- For every $w \in NT$, there exists $u \in O$ such that $uw \notin E$.

Start by setting for some $v_1 \in W$: $O = \{v_1\}$, $T = N(v_1) \cap W$, and $NT = W \backslash (N(v_1) \cup \{v_1\})$. Note that $|NT \cap S_1| < n_1/10$ and $|NT \cap S_2| < n_2/10$. Increase O by moving an arbitrary vertex v from NT to O and by moving $T \backslash N(v)$ from T to NT until either:

(i) $|O \cap S_1| + |NT \cap S_1| \geq n_1/10$, or $|O \cap S_2| + |NT \cap S_2| \geq n_2/10$; or

(ii) $NT = \emptyset$.

If the growth process stops with condition (i), say with no loss of generality that $|O \cap S_1| + |NT \cap S_1| \geq n_1/10$, and v_i was the last vertex added to O, then adding v_i to O increased $|NT \cap S_1|$ by less than $n_1/10$ and increased $|NT \cap S_2|$ by less than $n_2/10$. Thus, $|O \cap S_1| + |NT \cap S_1| < 2n_1/10$, and $|O \cap S_2| + |NT \cap S_2| < 2n_2/10$. On the other hand, $|T \cap S_1| \geq 8n_1/10 - 2n_1/10 \geq n_1/10$ and $|T \cap S_2| \geq 8n_2/10 - 2n_2/10 \geq n_2/10$.

Since the complement of O is connected, if $O \cap (C \cup D) = \emptyset$, then $O \subseteq A$ or $O \subseteq B$. So, we consider the following four LSP instances:

(a) I_1: $C \cap O \neq \emptyset$,
(b) I_2: $D \cap O \neq \emptyset$,
(c) I_3: $O \subseteq A$,
(d) I_4: $O \subseteq B$.

If $C \cap O \neq \emptyset$, then $(T \cap S_2) \cap D = \emptyset$, so $|S_{ABD}(I_1)| \leq 9n_2/10$. If $D \cap O \neq \emptyset$, then $(T \cap S_1) \cap C = \emptyset$, and analogously $|S_{ABC}(I_2)| \leq 9n_1/10$. If $O \subseteq A$, then $(NT \cap S_1) \cap B = \emptyset$. Thus, $|S_{ABC}(I_3)| \leq 9n_1/10$. If $O \subseteq B$, then a symmetrical argument shows that $|S_{ABC}(I_4)| \leq 9n_1/10$.

Now suppose the growing process stops with con dition (ii), i.e., $NT = \emptyset$ and both $|O \cap S_1| + |NT \cap S_1| < n_1/10$ and $|O \cap S_2| + |NT \cap S_2| < n_2/10$. Set $O_1 = O$ and reapply the algorithm to $W' = W \backslash O_1$ to obtain O_2 and continue until you reach situation (i), $|\bigcup_{j=1}^{i}(O_j \cap S_1)| \geq n_1/10$, or $|\bigcup_{j=1}^{i}(O_j \cap S_2)| \geq n_2/10$. If $n_1/10 \leq |\bigcup_{j=1}^{i}(O_j \cap S_1)| \leq 2n_1/10$, then we have all edges between sets $Z = (\bigcup O_i) \cap S_1$ and $Y = W \backslash \bigcup O_i$, which implies that $C \cap Z = \emptyset$ or $D \cap Y = \emptyset$.

We now define three new instances of LSP according to the intersection of skew partition sets C or D with Z, as follows:

(a) I_1: $C \cap Z \neq \emptyset$,
(b) I_2: $D \cap Z \neq \emptyset$,
(c) I_3: $Z \subseteq A \cup B$.

If $C \cap Z \neq \emptyset$, then $D \cap Y = \emptyset$. Thus, $(Y \cap S_2) \subseteq A \cup B$, which implies that $|S_{ABD}(I_1)| \leq 9n_2/10$. If $D \cap Z \neq \emptyset$, then an argument symmetric to $C \cap Z \neq \emptyset$ shows that $|S_{ABC}(I_2)| \leq 9n_1/10$. Otherwise, $Z \subseteq A \cup B$, which implies that $|S_{ABC}(I_3)| \leq 9n_1/10$.

This ends the description of Procedure 2.

Procedure 3 is a mirror image of Procedure 2 and is omitted.

Procedure 4.

Let $S_{ABC} = S_1$ with $|S_{ABC}| = n_1$ and $S_{AD} = S_2$ with $|S_{AD}| = n_2$. Given $v \in S_1 \cup S_2$, let $d_i(v) = |N(v) \cap S_i|$, $i = 1, 2$.

Case 1: There exists a vertex v in S_2 such that $n_1/10 \leq d_1(v) \leq 9n_1/10$.

This case is analogous to Case 1 of Procedure 1. Define two new instances of LSP, as follows:

(a) I_1: $v \in A$,
(b) I_2: $v \in D$.

If $v \in A$, then every vertex of S_1 that is nonadjacent to v cannot be placed in B, and so $|S_{ABC}(I_1)| \leq 9n_1/10$. If $v \in D$, then every vertex of S_1 that is adjacent to v cannot be placed in C, and so $|S_{ABC}(I_2)| \leq 9n_1/10$, as required.

Case 2: Every vertex v in S_2 satisfies either $d_1(v) < n_1/10$ or $d_1(v) > 9n_1/10$.

Let $W = \{v \in S_2 : |N(v) \cap S_1| > 9n_1/10\}$. We handle Case 2 according to the following two possibilities.

Case 2.1: $|W| > n_2/2$.

Let $v_1 \in W$. Recursively define three sets O, T, and NT such that:

- O is contained in W;
- $S_1 = T \cup NT$;
- There are all edges between O and T;
- For every w in NT, there exists v in O such that v is not adjacent to w.

Start by setting $O = \{v_1\}$, $T = N(v_1) \cap S_1$, and $NT = S_1 \backslash (N(v_1) \cup \{v_1\})$. Increase O inside W by adding an arbitrary vertex v of W to O and by moving $T \backslash N(v)$ from T to NT until:

(i) $|T| \leq 9n_1/10$; or

(ii) $|O| \geq n_2/10$.

If the growth process stops with condition (i), i.e., $|T| \leq 9n_1/10$, then $|NT| \geq n_1/10$. In addition, if v_i was the last vertex added to O, then because $v_i \in W$, we have $d_1(v_i) > 9n_1/10$, and adding v_i to O caused us to remove fewer than $n_1/10$ vertices from T, which implies $|T| > 8n_1/10$.

Consider two new instances of LSP as follows:

(a) I_1: $D \cap O \neq \emptyset$,

(b) I_2: $O \subseteq A$.

If $D \cap O \neq \emptyset$, then $T \cap C = \emptyset$, which implies $|S_{ABC}(I_1)| \leq 9n_1/10$. Otherwise, $O \subseteq A$, and $NT \cap B = \emptyset$, since for every $w \in NT$ there is a vertex $v \in O$ such that $vw \notin E$. Thus $|S_{ABC}(I_2)| \leq 9n_1/10$.

Now suppose that the growing process stops with condition (ii), i.e., $|O| \geq n_2/10$ and $|T| > 9n_1/10$. Consider two new instances of LSP as follows.

(a) I_1: $C \cap T \neq \emptyset$,

(b) I_2: $T \subset (A \cup B)$.

Clearly, $T \cap S_{ABC}(I_2) = \emptyset$, and so $|S_{ABC}(I_2)| \leq 9n_1/10$. If $C \cap T \neq \emptyset$, then $O \cap D = \emptyset$, and so $|S_{AD}(I_2)| \leq 9n_1/10$.

Case 2.2: $|S_2 \backslash W| > n_2/2$.

Let $X = S_2 \backslash W$ and $v \in X$. Recursively define two sets O and M such that:

- $O \subseteq X$, $M \subseteq S_1$;
- There are no edges between M and O.

Start by setting $O = \{v\}$ and $M = S_1 \backslash (N(v) \cup \{v\})$. Increase O in X until either:

(i) $|M| \leq 9n_1/10$; or

(ii) $|O| \geq n_2/10$.

If the growth process stops with condition (i), i.e., $|M| \leq 9n_1/10$, and v_i was the last vertex added to O, then, as $v_i \in X$, adding v_i decreased $|M|$ by less than $n_1/10$, and we have $|M| > 8n_1/10$. Then, in either case (i) or (ii), we have $|M| > 8n_1/10$.

Define two new LSP instances as follows;

(a) I_1: $A \cap O \neq \emptyset$,

(b) I_2: $O \subseteq D$.

If $A \cap O \neq \emptyset$, then $M \cap B = \emptyset$. Hence because $|M| > 8n_1/10$, we have $|S_{ABC}(I_1)| < 2n_1/10 \leq 9n_1/10$. Otherwise, $O \subseteq D$. In case (i), $|M| \leq 9n_1/10$, which implies $|S_1 \backslash M| \geq n_1/10$, so we have $|S_{ABC}(I_2)| \leq 9n_1/10$. In case (ii), $|O| \geq n_2/10$, which implies $|S_{AD}(I_2)| \leq 9n_2/10$. So, for either output instance I_i, we have $|S_{ABC}(I_i)||S_{AD}(I_i)| \leq 9n_1n_2/10$, as required.

This ends the description of Procedure 4 and the paper.

ACKNOWLEDGMENTS

We thank Pavol Hell for discussions on skew partitions and the anonymous referee for carefully reading a previous version of this paper.

REFERENCES

1. N. Alon and M. Tarsi, Colorings and orientations of graphs, *Combinatorica* **12** (1992), 125–134.
2. C. Berge, Les problèmes de coloration en théorie des graphes, *Publ. Inst. Stat. Univ. Paris* **9** (1960), 123–160.
3. V. Chvátal, Recognizing decomposable graphs, *J. Graph Theory* **8** (1984) 51–53.
4. V. Chvátal, Star-cutsets and perfect graphs, *J. Combin. Theory Ser. B* **39** (1985), 189–199.
5. G. Cornuéjols and B. Reed, Complete multi-partite cutsets in minimal imperfect graphs, *J. Combin. Theory Ser. B* **59** (1993), 191–198.

6. Problem session on minimal imperfect graphs, in "DIMACS Workshop on Perfect Graphs," (V. Chvátal and L. Lovász, Organizers), Princeton University, NJ, June 1993.
7. H. Everett, S. Klein, and B. Reed, An optimal algorithm for finding clique-cross partitions, *Congr. Numer.* **135** (1998), 171–177.
8. T. Feder and P. Hell, List homomorphisms to reflexive graphs, *J. Combin. Theory Ser. B* **72** (1998), 236–250.
9. T. Feder, P. Hell, and J. Huang, List homomorphisms and circular arc graphs, *Combinatorica* **19** (1999), 487–505.
10. T. Feder, P. Hell, S. Klein, and R. Motwani, Complexity of graph partition problems, in "Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing," pp. 464–472, 1999.
11. H. Fleischner and M. Stiebitz, A solution of a coloring problem of P. Erdős, *Discrete Math.* **101** (1992), 39–48.
12. P. Hell and J. Nešetřil, On the complexity of H-coloring, *J. Combin. Theory Ser. B* **48** (1990), 92–110.
13. C. T. Hoàng, Some properties of minimal imperfect graphs, *Discrete Math.* **160** (1996), 165–175.
14. D. S. Johnson, The NP-completeness column: an ongoing guide, *J. Algorithms* **6** (1985), 434–451.
15. G. MacGillivray and M. L. Yu, Generalized partitions of graphs, *Discrete Appl. Math.* **91** (1999), 143–153.
16. S. Klein and C. M. H. de Figueiredo, The NP-completeness of multipartite cutset testing, *Congr. Numer.* **119** (1996), 216–222.
17. N. Vikas, Computational complexity of graph compaction, *in* "Proceedings of the Tenth ACM-SIAM Symposium on Discrete Algorithms," pp. 977–978, 1999.