



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Search Algorithms for Primal Graphs

Sara M. A. Stairs

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University

Montréal, Québec, Canada

December 1988

© Sara M. A. Stairs, 1988



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-49069-1

ABSTRACT

Search Algorithms for Primal Graphs

Sara M. A. Stairs

We say that a *factorization* of a simple graph G is a non-isomorphic edge decomposition of G , that is, a partition of the edges of G into subgraphs G_1, G_2, \dots, G_k which are mutually isomorphic. A *primal graph* is one whose only factorization is the trivial one into G itself. The primal graphs were completely known hitherto only for the graphs with no more than seven vertices; several infinite families of primal graphs are also known. Using computer methods, we have exhaustively searched several classes of graphs with 8 to 19 vertices to find other primal graphs. This search shows that the only primal graphs with 8 or 9 vertices belong to known infinite families. We have also found 22 primal graphs with 14 edges and 14 to 18 vertices, 17 of which were produced by the program. Of these 22 graphs, two belong to known primal families and four have never been published.

Acknowledgements

I wish to express my deep gratitude to Dr. Eric Regener for the assistance, advice and guidance given me throughout the last few years.

I would also like to thank a very close friend, Bill Hayden, to whom this work is dedicated.

Table of Contents

1. Introduction	1
2. Theoretical Results	9
3. Algorithms to Test Graphs for Primality	14
Algorithm 1: Primality, first version	15
Lexical Ordering of Primal Graphs	17
Algorithm 2: Primality, recursive version	20
Algorithm 3: Primality, final version	24
4. Search Procedures	30
Search Procedure Algorithms	32
Algorithm SunSearch	32
Algorithm CrabSearch	33
Algorithm RSearch	34
Algorithm QSearch	35
Algorithm YSearch	36
Star Search Procedure	37
Algorithm StarSearch	38
K2N Search Procedure	39
Algorithm K2NSearch	40
5. Implementation	41
Generation of Input Files	41
Performance of the Program	42
6. Conclusions	46
7. Appendix: Proof of Theorem 1	50
8. Bibliography	52

List of Tables

Table 1.	Primal graphs on 14 edges which contain a cycle of length 7.	4
Table 2.	Primal graphs on 14 edges which contain a cycle of length 8.	5
Table 3.	A primal graph on 14 edges which contains a cycle of length 9.	5
Table 4.	Primal graphs on 14 edges which contain a cycle of length 6.	6
Table 5.	Primal graphs on 14 edges which contain a cycle of length 6 and a cycle of length 3.	7
Table 6a.	Files Tested for Primality.	42
Table 6b.	Files Tested for Primality.	43
Table 7.	Summary of Computer Time spent on Vax 8500.	44
Table 8a.	Summary of Computer Time spent on Micro-Vax.	45
Table 8b.	Summary of Computer Time spent on Micro-Vax.	45
Table 9.	Run Times for Different Lexical Orders for a file consisting of 40161 graphs on 9 vertices.	46
Table 10.	Estimate of time required to complete some test cases.	48

Section 1: Introduction

The idea of "primal graphs" was introduced in 1970 by Dewdney [1], who considered an analogy between certain factorings of graphs and bases for vector spaces.

Finite simple graphs, like vectors, can be added. Does there exist a set of graphs which behaves with respect to other graphs like the basis of a vector space? If certain abstract properties of a vector space basis are considered in isolation regarding graphs, the existence of such a basis (here called 'Primal') set can be proved. This set also turns out to be unique.

Primal Graphs

The graph G is said to be the sum of the subgraphs H_1, H_2, \dots, H_n if every edge of G lies in exactly one H_i . In this case, $G = H_1 + H_2 + \dots + H_n$ is a factorization of G with factors H_1, H_2, \dots, H_n . A trivial factorization of G is one in which only a single H_i contains edges, or equivalently, G has only one factor, namely itself.

A set of graphs Γ is said to be primal relative to the set of graphs Ω if:

- i) $\Gamma \subseteq \Omega$,
- ii) each graph in Ω factors into non-isomorphic elements of Γ , and
- iii) graphs in Γ have only a trivial such factorization.

If H is a subgraph of G , then $G - H$ denotes a graph obtained from G by deleting the edges of H and then deleting any resulting isolated vertices. There may be several non-isomorphic graphs $G - H$, for any given G and H .

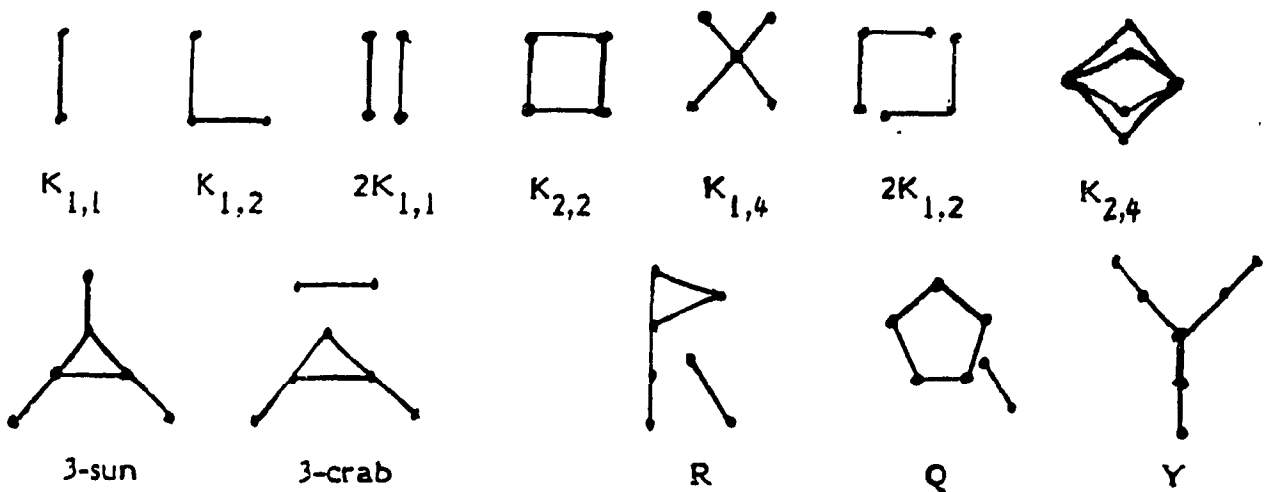
Throughout this work, all graphs considered are finite, without loops or multiple edges. We assume each graph has no isolated vertices and at least one edge. All factorizations considered in the following work are assumed to have non-isomorphic factors.

Dewdney [1] proves the following:

Theorem 1: Let Γ_n denote the set of all graphs with at most n vertices. Then there is a unique set Π_n of graphs that is primal relative to Γ_n . see appendix

Chinn et al. [2], continued the work in 1984, determining all graphs on at most 7 vertices along with several infinite families of primal graphs. They also provided several conjectures and problems concerning primal graphs.

Let Π denote the set of graphs primal relative to the set of all graphs. A graph G is said to be primal if $G \in \Pi$. The set Π_7 of primal graphs on at most seven vertices consists of the following graphs:



Chinn et al. [2], also show that the graphs $\alpha K_{n,\beta}$, consisting of α copies of the complete bipartite graph $K_{n,\beta}$, are primal for the following values of α , n , and β :

- $n = 1$, $\alpha = 2^r$, $\beta = 2^s$, for all $r, s \geq 0$. These graphs are known as stars.
- $n = 2$, $\alpha = 2^r$, $\beta = 2$, for all $r \geq 0$.
- $n = 2$, $\alpha = 1, 2$, $\beta = 4$.
- $n = 2$, $\alpha = 1$, $\beta = 2^s$, for all $s \geq 3$.

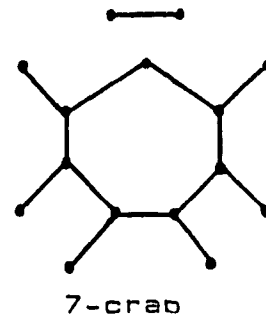
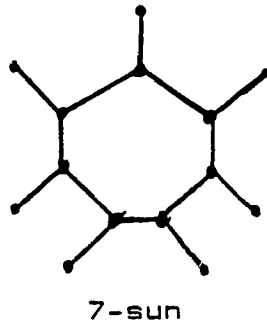
Graphs of this form where $n = 2$ will be referred to as K2N's.

Π_7 contains the five stars $K_{1,1}$, $K_{1,2}$, $2K_{1,1}$, $K_{1,4}$ and $2K_{1,2}$, and the two K2N's $K_{2,2}$ and $K_{2,4}$.

The 3-sun and 3-crab belong to two families known as suns and crabs, respectively.

A k -sun consists of a cycle of length k with one edge extending from each vertex of the cycle. A k -crab consists of a cycle of length k with one edge extending from all but one vertex of the cycle, plus a $K_{1,1}$.

It is shown in [2] that the 7-sun and 7-crab are primal.



Definitions

The content of $K_{1,1}$ in the graph G , written $m_1(G)$, is the maximum n such that $G \supseteq nK_{1,1}$.

Similarly, the content of $K_{1,2}$ in G , written $m_2(G)$, is the maximum n such that $G \supseteq nK_{1,2}$.

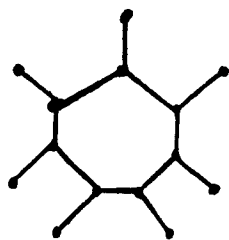
Conjecture 1 [2]: The k -sun and k -crab are primal if and only if $k=2^i-1$ for $i \geq 2$.

The graphs R , Q , and Y of Π_7 do not belong to known primal families, and are referred to as sporadic primal graphs.

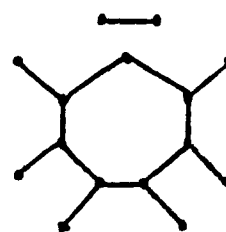
The problem of finding the complete set of primal graphs on 8 and 9 vertices by hand is too time-consuming. The purpose of this research is to determine primal graphs by use of computer search.

This paper explains what is currently known about primal graphs and describes the computer program designed to find primal graphs.

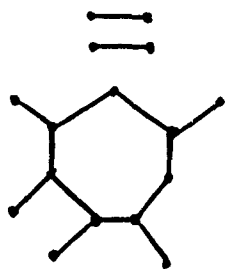
By searching all graphs with 8 and 9 vertices, we have established that the only primal graphs in these sets are $2K_{2,2}$, $4K_{1,1}$, and $K_{1,8}$. We have also found 22 primal graphs on 14 edges and 14 to 17 vertices, of which only two, the 7-sun and 7-crab, belong to known families. Five of these graphs were determined primal by hand. These are the graphs V6-8 to V6-12, inclusive. They were not shown to be primal by use of the program since they were not contained in any of the input files used for testing primality. Of all these, the graphs V6-5, V6-6, V6-9, V6-10 are hitherto unpublished.



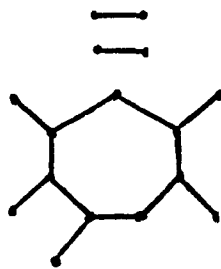
V7-1: 7-sun



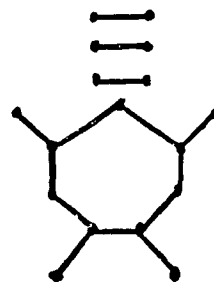
V7-2: 7-crab



V7-3

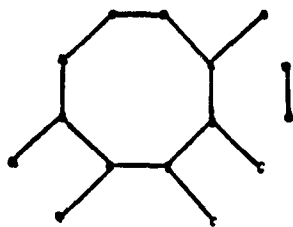


V7-4

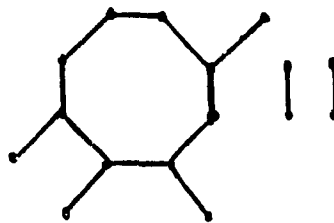


V7-5

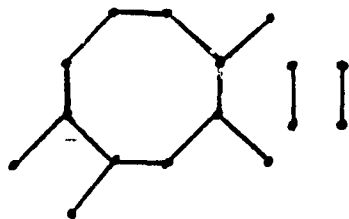
Table 1 - Primal graphs on 14 edges which contain a cycle of length 7.



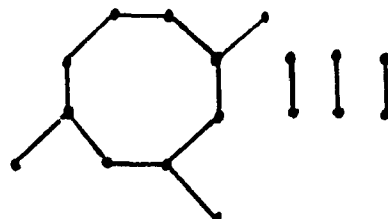
V8-1



V8-2



V8-3



V8-4

Table 2 - Primal graphs on 14 edges which contain a cycle of length 8.



V9-1

Table 3 - A primal graph on 14 edges which contains a cycle of length 9.

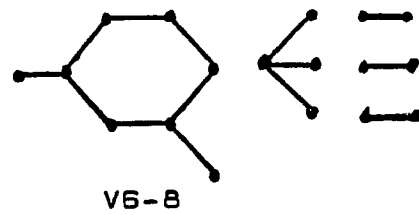
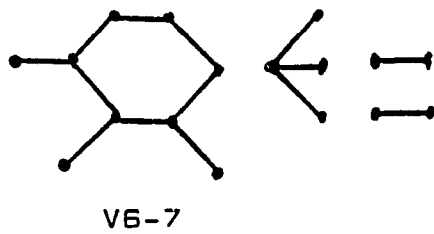
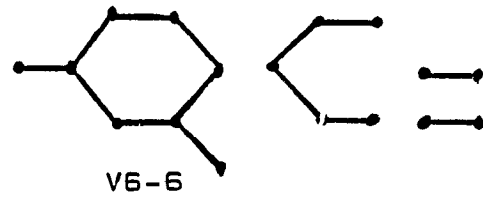
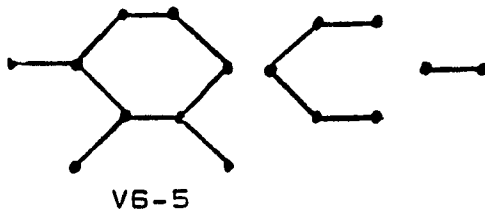
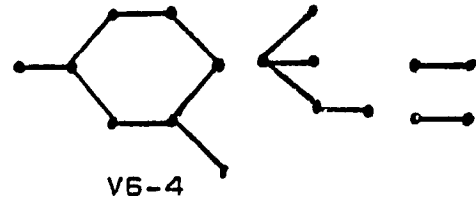
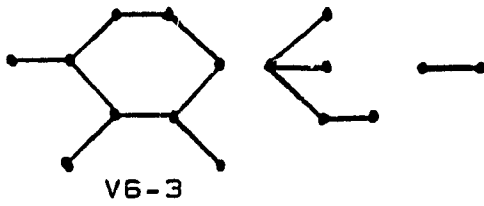
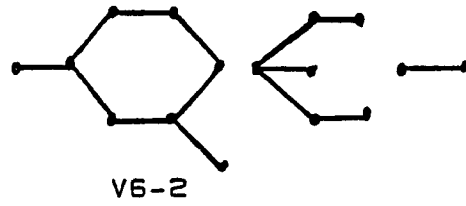
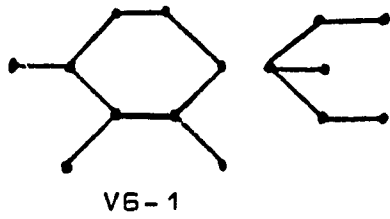


Table 4 - Primal graphs on 14 edges which contain a cycle of length 6.

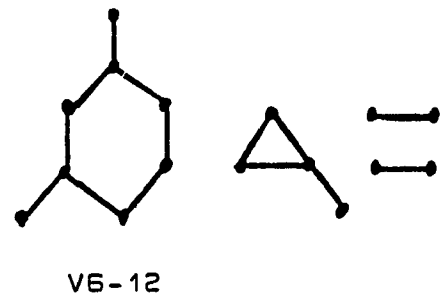
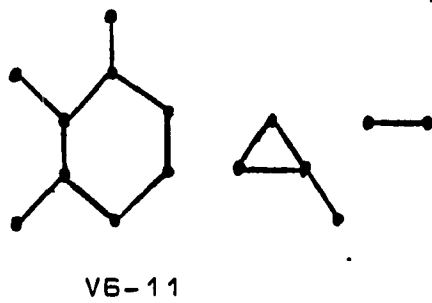
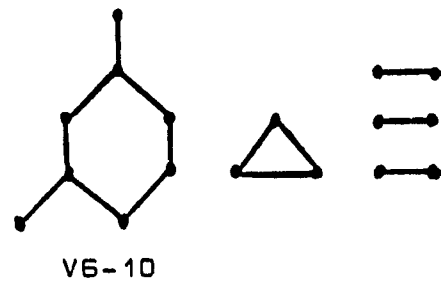
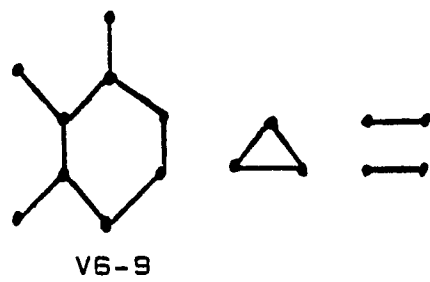


Table 5 - Primal graphs on 14 edges which contain a cycle of length 6 and a cycle of length 3.

Notation

Throughout this work the following notation will be used:

$V(G)$: vertices of the graph G .

$E(G)$: the set of edges of the graph G .

$\deg(v)$: degree of the vertex v .

$\mathcal{F}(G)$: complete factorization of the graph G .

$\mathcal{F}(G)$: partial factorization of the graph G .

Φ : an empty graph (no edges).

$P \simeq M$: P and M are isomorphic graphs.

Section 2: Theoretical Results

Let Π_0 be the set of primal graphs in Π_7 together with the infinite families of stars and K_2N 's. The problem of finding primal graphs on more than 7 vertices requires searching each graph for a factorization involving graphs in Π_0 . If an exhaustive search for a factorization of a graph G is unsuccessful, then G must either be primal or contain a primal graph not in Π_0 . There are two observations which can be made to reduce the searching time for factorizations of graphs:

1) Odd number of edges

Theorem 2(Chinn et al. [2]) : Each graph in $\Pi - \{K_{1,1}\}$ has an even number of edges.

Proof: Order Π by increasing number of edges:

$$P_1, P_2, P_3, \dots, P_i, \dots$$

$$\text{i.e. } P_1 = K_{1,1}, P_2 = 2K_{1,1}, P_3 = K_{1,2}, \text{ etc...}$$

Assume P_i is the first primal graph after $K_{1,1}$ with an odd number of edges. Let $G = P_i - K_{1,1}$. Obviously, G has an even number of edges.

Case 1. If G is primal, then P_i has the factorization $\{K_{1,1}, G\}$, hence P_i is not primal.

Case 2. If G is not primal, then since $|E(G)|$ is even and $K_{1,1}$ is the only primal graph P with odd $|E(P)|$ and $|E(P)| < |E(G)|$, G must have a factorization involving only primal graphs P with even $|E(P)|$. Thus, P_i is not primal, it has the factorization $\{K_{1,1}, F(G)\}$.

2) Containment

Lemma 1(Chinn et al. [2]) : If there is a $P \in \Pi$ such that $P \subseteq G$ but $P \not\subseteq G - P$, then G is not primal.

Proof: Case 1: If $G - P$ is primal, then G is not primal, it has the

factorization $\{G-P, P\}$.

Case 2: If $G - P$ is not primal then $G = F(G - P) + P$. Since $P \not\subseteq G - P$, $F(G - P)$ cannot contain P , and G has the factorization $\{F(G-P), P\}$.

Lemma 2 : If G is primal and $P \subseteq G$ then $P \subseteq G - P$, for any primal graph P .

Proof: If $P \not\subseteq G - P$, then by Lemma 1 G is not primal.

Corollary 1:

Let $\mathcal{P}(G) = \{P_1, P_2, \dots, P_r\}$ be a partial factorization of G into primal graphs, that is $G = \mathcal{P}(G) + H$, $P_i \in \Pi$ for all i , and $P_i \not\subseteq P_j$ for $i \neq j$. If $\forall P_i \in \mathcal{P}(G)$, $P_i \not\subseteq H$, then G factors into elements of Π .

Proof: Let $F(H)$ be any factorization of H . Since $P_i \not\subseteq H$, $P_i \notin F(H)$. Hence, G has the factorization $\mathcal{P}(G) + F(H)$.

Corollary 2:

Let P be a primal graph contained in G , and $G = P + H$. If there exists a factorization $F(H)$ of H such that $P \not\subseteq F(H)$, then G is not primal.

Theorem 3: If G is primal then $3G$ is not primal.

Proof:

Case 1: If $2G$ is primal, then $3G$ has the factorization $\{G, 2G\}$.

Case 2: If $2G$ is not primal, then $F(2G)$ cannot have G as one of its factors since $2G = G + G$. By Corollary 2, since $G \not\subseteq F(2G)$, $3G$ has the factorization $\{G, F(2G)\}$.

Theorem 4

If a graph, G , on s^{k-2} edges has $K_{1,1}$'s and $K_{1,2}$'s as its only primal factors and $m_1(G) \leq 2^{k-1}-1$, $m_2(G) \leq 2^{k-2}-1$, then G is primal.

Proof

The primal graphs available as factors of G are:

$$2K_{1,1}, 4K_{1,1}, \dots, 2^{k-2}K_{1,1}, K_{1,2}, 2K_{1,2}, \dots, 2^{k-3}K_{1,2}.$$

The total number of edges in these graphs is :

$$2(2 + 4 + \dots + 2^{k-2}) = 4(1 + 2 + 4 + \dots + 2^{k-3}) = 4(2^{k-2} - 1) = 2^k - 4 < |E(G)| = 2^k - 2.$$

Since G can have no non-trivial factorization into primal graphs, G must be primal.

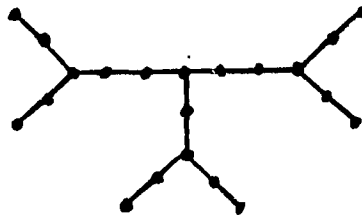
Richter [4] has proved the following:

Theorem 5

Let G be a graph with $m_1(G) + 2m_2(G) < |E(G)|$. Then $\{H; \exists n, H \subseteq nG\}$ contains infinitely many primal graphs not of the form $2^i K_{1,1}$ or $2^i K_{1,2}$.

For example: the graph $7C_5 + K_{1,1} \subseteq 8C_5$ is primal (C_5 is the 5-cycle): for this graph $m_1 = 15$, $m_2 = 7$, $m_1 + 2m_2 = 29 < |E| = 36$.

The graph consisting of 25 disjoint copies of



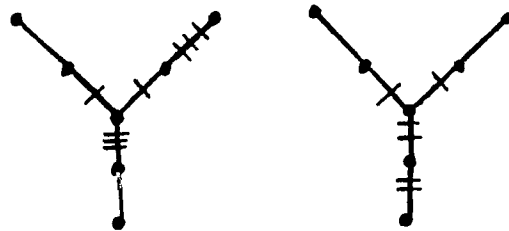
contains at least 1 unknown primal graph (Richter). For this graph, $m_1 = 10$, $m_2 = 4$, $|E| = 21 > 18 = m_1 + 2m_2$.

The graphs on 14 edges given in Tables 1 to 5 all have $m_1 = 7$, $m_2 = 3$ (as is easily verified), so that $m_1 + 2m_2 = 13 < 14 = |E|$. The only primal graphs in Π_7 which the graphs in Tables 1 to 5 may contain are the stars of degree < 4 . These are $2K_{1,1}$, $K_{1,2}$, $4K_{1,1}$ and $2K_{1,2}$, having a total of 12 edges.

Examples of Factorizations

In general it is not true that if P is primal, $2P$ is also primal, e.g.:

A Factorization of $2Y$

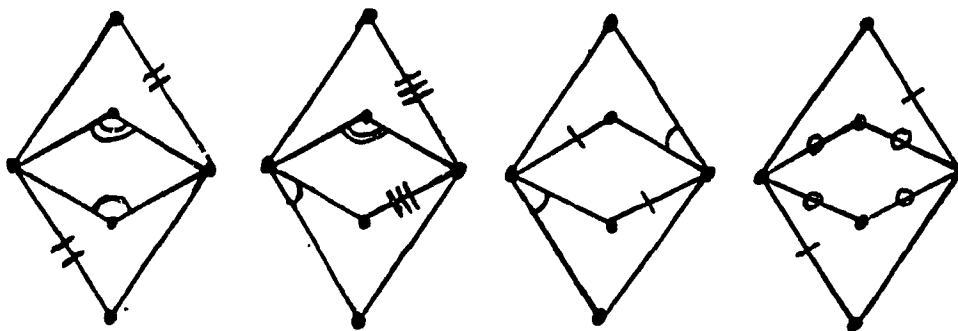


$$2Y = 2K_{1,2} + K_{1,2} + 4K_{1,1} + 2K_{1,1}$$

— = ≡

As another example, the graph $2K_{2,4}$ is primal although $4K_{2,4}$ and $2K_{2,8}$ are not.

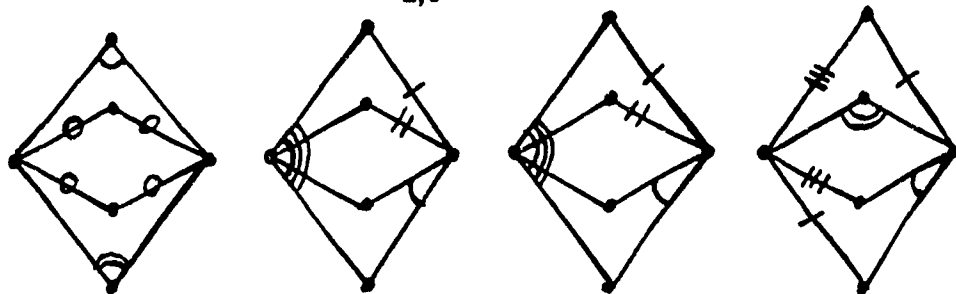
A Factorization of $4K_{2,4}$



$$4K_{2,4} = K_{2,2} + 4K_{1,2} + 2K_{1,2} + K_{1,2} + 8K_{1,1} + 4K_{1,1} + 2K_{1,1}$$

o) .)) ≡ — =

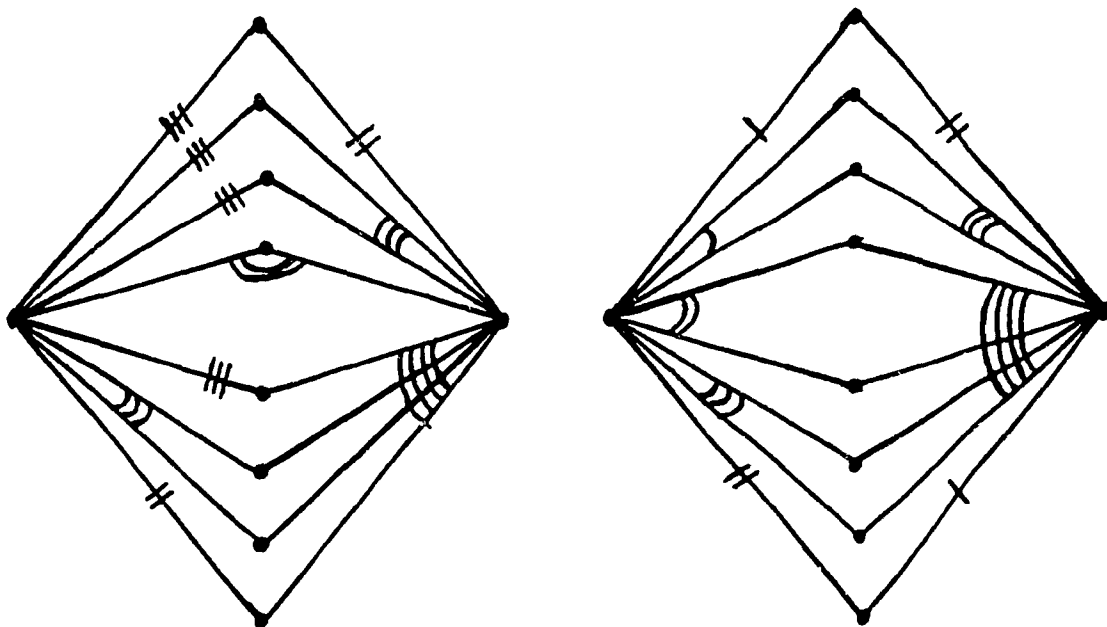
A Second Factorization of $4K_{2,4}$



$$4K_{2,4} = K_{2,2} + 2K_{1,4} + 4K_{1,2} + 2K_{1,2} + K_{1,2} + 4K_{1,1} + 2K_{1,1}$$

$\square \quad \quad \quad))) \quad \quad \quad) \quad \quad \quad)) \quad \quad \quad \equiv \quad \quad \quad - \quad \quad \quad =$

Factorization of $2K_{2,8}$



$$2K_{2,8} = 2K_{1,4} + K_{1,4} + 4K_{1,2} + 2K_{1,2} + K_{1,2} + 4K_{1,1} + 2K_{1,1}$$

$)))) \quad \quad \quad \equiv \quad \quad \quad))) \quad \quad \quad)) \quad \quad \quad) \quad \quad \quad = \quad \quad \quad -$

Section 3: Algorithms to Test Graphs for Primality

In order to determine primality of a graph efficiently by computer, we must base our algorithm as strongly as possible on theoretical results. Theorem 2 states that all graphs other than $K_{1,1}$ with an odd number of edges are not primal. Thus, when searching for primal graphs, we need not concern ourselves with any graphs with an odd number of edges. Lemma 1 states that for a graph, G , and a primal graph, P , such that $P \subseteq G$, if we can find a $G - P$ such that $P \not\subseteq G - P$, then G is not primal. This Lemma, along with Corollaries 1 and 2, may be used for certain graphs G to prove non-primality without factorizing G completely.

We develop the algorithm in three steps. The first algorithm determines if a graph, G , has a factorization involving a primal graph, P . The second determines if G has a factorization involving any of the primal graphs in Π_0 . Algorithm 3 is an expansion of algorithm 2. We include the procedures which search for containment of primal graphs and we disregard any input graph with one or more isolated vertices. Algorithm 3 is written so that it may be easily implemented on the computer and is the algorithm upon which the program is based.

Initially, we give a simple algorithm to determine by exhaustive search if a given graph G is factorable using a given primal graph P , by using Lemma 1 and Corollary 2.

Two outcomes are possible:

- G has a factorization involving P and thus is not primal.
- G cannot be factored using P .

Notation:

/* comment */

Config(G-P) : the set of all non-isomorphic graphs G-P.

status : a variable with the following possible values:

not-primal : G has a factorization involving P.

not-proved : G cannot be factored using P.

Algorithm 1: Primality, first version

/* Search for a factorization of G involving the primal graph P. */

algorithm Primall (G,P)

P1. $\mathcal{N} \leftarrow \text{Config}(G-P)$

P2. status \leftarrow not-proved

P3. while $\mathcal{N} \neq \emptyset$ and status = not-proved do

P4. H \leftarrow any member of \mathcal{N}

P5. $\mathcal{N} \leftarrow \mathcal{N} - \{ H \}$

P6. if $P \not\subseteq H$ then status \leftarrow not-primal

P7. elseif $\exists F(H), P \not\subseteq F(H)$ then status \leftarrow not-primal

endif

endwhile

P8. return

Algorithm 1 searches for a factorization of a graph, G , involving a particular primal graph, P . To obtain the set $\text{Config}(G-P)$ we delete the edges of P and any resulting isolated vertices from G . This is done for each possible way that P is contained in G . For each $H \in \text{Config}(G-P)$ and while we have not found a solution we do the following. If $P \not\subseteq H$ then G is not primal by Lemma 1. If a complete factorization of H not involving P can be found, then G has the factorization $\{P, F(H)\}$. If all configurations have been tested and we have not arrived at a solution then we conclude that G does not have a factorization involving P .

Algorithm 1 asks "Does G have a factorization involving the primal graph P ?". This is not sufficient to determine whether G is or is not primal. Algorithm 2 will address the question "Does G have a factorization involving any of the graphs in a given set, Π , of primal graphs?". Algorithm 2 will also provide an answer to the question in step P7, "Does H have a factorization not involving P ?".

In order to determine if G has a factorization involving any of the primal graphs in Π , we simply need to invoke the algorithm once for each P in Π such that $|V(P)| \leq |V(G)|$. This can be accomplished by lexically ordering the primal graphs and searching for the primal graphs one by one. If G does not have a factorization involving any of the graphs in Π , then we may conclude that G has a factorization involving a primal graph $\notin \Pi$. To determine whether H has a factorization not involving P , we may call the algorithm recursively, searching for a factorization involving graphs in $\{\Pi - P\}$. Initially we call the algorithm with $\Pi = \Pi_0$, the set of all primal graphs in Π_7 along with the families of stars and K_2 's.

Lexical Ordering of Primal Graphs

We may look at a graph, G , and predict what a possible factorization may be. Several factorizations may be tested in an *ad hoc* manner until a factorization is found, all possible factorizations have been tried, or we simply give up. For smaller graphs, with 7 vertices or less, it is not terribly difficult or time-consuming for a person to perform the search. For larger graphs, if a possible factorization is not immediate to the eye, the search for a factorization may be lengthy. Since the computer cannot make intuitive assumptions as to which primal graphs may or may not be factors, there must be a systematic approach to the search. This necessitates lexical ordering of the primal graphs. The computer may then keep track of which factorizations have been attempted and know which factorizations are to be tried next.

We now define a lexical ordering of the graphs in Π_0 .

The precedence operator, $<$, is used to indicate priority, i.e., $P < M$ indicates that P is lexically prior to M .

To facilitate the ordering of the primal graphs, they are first divided into classes:

C_1 : STARS - the graphs $\alpha K_{1,\beta}$ where $\alpha = 2^r$, $\beta = 2^s$, $r, s \geq 0$

C_2 : K2N'S - the graphs $\alpha K_{2,\beta}$ where $\alpha = 2^r$, $\beta = 2^s$, $r, s \geq 0$;

for all r, s such that the resulting graph is primal,

i.e., for $s = 1$, $r \geq 0$; for $s = 2$, $r = 1, 2$; for $s > 2$, $r = 1$.

C_3 : SUN - the 3-sun

C_4 : CRAB - the 3-crab

C_5 : RSPORADIC - the sporadic graph R .

C_6 : QSPORADIC - the sporadic graph Q .

C_7 : YSPORADIC - the sporadic graph Y .

For classes containing several graphs, we order the graphs within their classes:

STARS and K2N's:

The stars, $\alpha K_{1,\beta}$, and K2N's, $\alpha K_{2,\beta}$, are ordered such that the larger the degree β , the greater the priority, and for graphs with the same degree, the larger the number of copies α , the larger the priority.

Formally, if $G, H \in C_1$, $G = \alpha K_{1,\beta}$, $H = \alpha' K_{1,\beta'}$,

or $G, H \in C_2$, $G = \alpha K_{2,\beta}$, $H = \alpha' K_{2,\beta'}$,

then $G < H$ if $\beta > \beta'$, or $\beta = \beta'$ and $\alpha > \alpha'$.

The ordering of the classes has precedence over the ordering of the graphs within a given class. i.e. If $\text{Kind}(P) < \text{Kind}(M)$, then $P < M$, where $\text{Kind}(P)$ indicates to which class P belongs. The lexical ordering is structured so that it may be modified. The ordering of the graphs within their classes cannot be modified easily but the ordering of the classes can be adjusted to suit the needs of the program.

The following notation will be used throughout the remainder of the text:

$\text{First}(\Pi_0)$: The graph in Π_0 which comes first in the lexical order.

$\text{Last}(\Pi_0)$: The graph in Π_0 which is last in the lexical order.

$\text{Next}(P)$: The graph in Π_0 which follows P in the lexical order.

EMPTY: A "dummy" graph which precedes the first graph in the lexical order.

i.e. $\text{First}(\Pi_0) = \text{Next}(\text{EMPTY})$

FULL: A "dummy" graph which follows the last graph in the lexical order.

i.e. $\text{FULL} = \text{Next}(\text{Last}(\Pi_0))$

We now present Algorithm 2 which determines whether or not a graph, G , has a factorization involving any of the primal graphs in Π , a given set of primal graphs. The algorithm is divided into two parts. The first part, Primal2, determines if G_0 has an odd number of edges, if so, G_0 is the primal graph $K_{1,1}$ or it is not primal by Theorem 2. Primal2 invokes the second part of the algorithm, Factor, if G_0 has an even number of edges. Factor is the recursive portion of the algorithm and performs the search for a factorization.

The modified algorithm returns one of the following results:

- G is a primal graph $\in \Pi_0$.
- G has been proved not primal.
- G is primal or has a primal factor $\notin \Pi_0$.

Notation for algorithm 2.

G_0 : the graph to be tested for primality.

\mathcal{F}_0 : a partial factorization of G_0 .

status : a variable which may have one of the following values :

notknown: the search is incomplete.

notprimal: G_0 has been proved not primal.

knownprimal: G_0 is a known primal graph.

unsolved: G_0 is either primal or has a factorization involving a
primal graph $\notin \Pi_0$.

To determine whether or not the graph G_0 is primal, initialize $\mathcal{F}_0 \leftarrow \{\}$ and call Primal(G_0).

Algorithm 2: Primality, recursive versionalgorithm Primal2(G_0)P1. $\text{status} \leftarrow \text{notknown}$ P2. if $|E(G_0)| = 1$ then $\text{status} \leftarrow \text{knownprimal}$ P3. elseif $|E(G_0)|$ is odd then $\text{status} \leftarrow \text{notprimal}$ P4. elseP5. $P \leftarrow \text{FIRST}$ P6. while P is not FULL and status is notknown doP7. Factor($G_0, \text{Next}(P), \emptyset$) endwhile endifP8. if $\text{status} = \text{notknown}$ then $\text{status} \leftarrow \text{unsolved}$ P9. return (status)

algorithm Factor(G, P, \mathcal{F}_0)

```

F1.  if  $P \subseteq G$  then
F2.      for each  $G' = G - P$ 
F3.      and while status = notknown do
F4.           $\mathcal{F}_0 \leftarrow \mathcal{F}_0 + P$ 
F5.          if  $G' = \emptyset$ 
              then
F6.              if  $\mathcal{F}_0 = \{P\}$  then status  $\leftarrow$  knownprimal      /*  $G = P$  */
F7.              else status  $\leftarrow$  notprimal
              endif
F8.          elseif  $\forall F \in \mathcal{F}_0, F \not\subseteq G'$  then status  $\leftarrow$  notprimal
F9.          elseif  $\forall F \in \mathcal{F}_0, F \neq G'$  then Factor( $G', \text{Next}(P), \mathcal{F}_0$ )
              endif
F10.         if status = notknown then  $\mathcal{F}_0 \leftarrow \mathcal{F}_0 - P$ 
              endwhile
              endfor
              endif
F11.  return (status)

```

If G_0 is the primal graph $K_{1,1}$, it is recognized in step P2. Other graphs with an odd number of edges are not primal (Theorem 2) and are recognized as such in step P3. Thus the factorizing routine, Factor, is invoked only for graphs with an even number of edges. Factor is invoked once for each primal graph in the lexical order until a solution is reached or until all primal graphs have been tested.

Primal graphs are added to the list of factors removed from G_0 as they are removed from the subgraph G . The case in which G_0 is a primal graph in Π_0 is recognized in step F6. The case in which G_0 has a factorization involving P may be solved in one of two places. Step F7 recognizes the situation in which G_0 has been completely factored. Step F8 recognizes that G_0 is not primal before a complete factorization occurs, using Corollary 1.

Step F9 is included to avoid needless searching. If the subgraph G equals any of the factors removed from G_0 then we need not continue the search, since our current factorization involves two isomorphic graphs.

The algorithm for the function Next(P) must consider a finite subset, Π' , of Π , which has infinite size. We insure that Π' is finite by checking for $|V(P)| \leq |V(G)|$. Also we make sure that for each vertex, v_1 , in P such that $\deg(v_1) > 1$, there is a corresponding vertex, v_2 , in G such that $\deg(v_2) \geq \deg(v_1)$. For example, if P is the 3-crab and the 3-sun is the immediate successor of the 3-crab in the lexical order, then Factor will be called with Next(P) = 3-sun only if G contains at least 6 vertices, 3 of which have degree ≥ 3 .

The problem of determining whether $P \subseteq G$ for a given primal graph $P \in \Pi_0$ and an arbitrary graph G is solved by writing a separate procedure for each class C_i of graphs in Π_0 which searches for containment of graphs of class C_i in G . One search procedure is written for each of the 7 classes of primal graphs in Π_0 .

In this research, we assume that all graphs considered will have no isolated vertices. To enforce this condition we disregard any input graph, G_0 , which has one or more isolated vertices.

Here is algorithm 3, which contains the above enhancements. It is the algorithm upon which the program is based, and contains five major parts:

$\text{Main}(G_0)$ - recognizes all graphs with isolated vertices and all graphs with an odd number of edges. Invokes Control for any graph with an even number of edges and no isolated vertices.

$\text{Control}(G, P, \mathcal{F}_0, \text{nfacs})$ - searches the list of primal graphs in lexical order beginning with the graph $\text{Next}(P)$, looking for a factorization involving any one of these graphs by invoking Search. \mathcal{F}_0 is the set of factors in the current partial factorization of G_0 , G is a configuration of $G_0 - \mathcal{F}_0$, and $\text{nfacs} = |\mathcal{F}_0|$.

$\text{Search}(G, P, \mathcal{F}_0, \text{nfacs})$ - Invokes a particular search procedure depending upon the class of the primal graph P .

$\text{GenericSearch}(G, P, \mathcal{F}_0, \text{nfacs})$ - determines if $P \subseteq G$; if not returns to Control, if so invokes ResumeSearch.

$\text{ResumeSearch}(G, P, \mathcal{F}_0, \text{nfacs})$ - adds P to the current partial factorization of G_0 and resumes the search for a factorization at the next level by invoking $\text{Control}(G - P, P, \mathcal{F}_0 + P, \text{nfacs} + 1)$.

In the implementation of the algorithm the routine GenericSearch is replaced by seven search procedures, one for each class of primal graphs in Π_0 . Each of these procedures will invoke the procedure ResumeSearch if $P \subseteq G$, which will continue the factorization at the next level. Each of the seven search procedures is different because each is searching for containment of a different primal graph. The algorithms for these procedures are not provided in this section in order to simplify the description of the entire algorithm. They are provided in the following section.

Notation

G_0 : the "original" input graph

status : a state variable with the following values:

notknown: status is unknown, the search is not yet complete.

notprimal: G_0 is not a primal graph.

knownprimal: G_0 is a known primal graph.

unsolved: G_0 is either an unknown primal graph or has a factorization

involving an unknown primal graph.

Here is the algorithm:

Algorithm 3: Primality, final version

algorithm Main(G_0)

M1. status \leftarrow notknown

M2. if $\exists v \mid \deg(v) = 0$ then status \leftarrow notprimal

M3. if $|E(G)| = 1$ then status \leftarrow knownprimal

M4. elseif $|E(G)|$ is odd then status \leftarrow notprimal

M5. else Control(G_0 , EMPTY, {}, 0)

endif

M6. if status = notknown then status \leftarrow unsolved

endif

M7. finished: return

algorithm Control($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

G : the graph which we are attempting to factorize.

P : the most recent primal graph considered to be a possible primal factor, this is
EMPTY on the first call.

\mathcal{F}_0 : the partial factorization of the input graph G_0 , this is $\{\}$ on the first call.

nfacs : the number of factors in the set \mathcal{F}_0 .

*/

C1. if $G = \Phi$

then

C2. if $\text{nfacs} = 1$ then $\text{status} \leftarrow \text{knownprimal}$

C3. else $\text{status} \leftarrow \text{notprimal}$

endif

C4. goto finished

C5. elseif $\forall F \in \mathcal{F}_0, F \not\subseteq G$ then $\text{status} \leftarrow \text{notprimal}$

C6. elseif $\forall F \in \mathcal{F}_0, F \neq G$

then

C7. while $P \neq \text{FULL}$ do

C8. Search ($G, \text{Next}(P), \mathcal{F}_0, \text{nfacs}$)

endwhile

endif

C9. return

algorithm Search($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

G : the graph which we are attempting to factorize.

P : the primal graph currently being considered as a possible primal factor.

\mathcal{F}_0 : the partial factorization of the input graph G_0 .

nfacs : the number of factors in the set \mathcal{F}_0 .

*/

S1. case kind(P) of:

S2. STAR: StarSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

S3. K2N: K2NSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

S4. SUN: SunSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

S5. CRAB: CrabSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

S6. RSPORADIC: RSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

S7. QSPORADIC: QSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

S8. YSPORADIC: YSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

end case

return

algorithm GenericSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

G : the graph now being factored.

P : the primal graph which is currently being considered as a possible factor.

\mathcal{F}_0 : the partial factorization of the input graph G_0 .

nfacs : the number of factors in the set \mathcal{F}_0 .

Given that the original input graph G_0 has the partial factorization $\mathcal{F}_0 + G$, where $\text{nfacs} = |\mathcal{F}_0|$, attempt to find a factorization of G using the primal graph P .

*/

G1. if $P \subseteq G$

G2. then ResumeSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

endif

return

algorithm ResumeSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

G : the graph now being factored.

P : the primal graph to be added to \mathcal{F}_0 , the partial factorization of the input graph G_0

nfacs : the number of factors in the set \mathcal{F}_0 .

*/

R1. $H \leftarrow G - P$

R2. $\text{nfacs} \leftarrow \text{nfacs} + 1$

R3. $\mathcal{F}_0 \leftarrow \mathcal{F}_0 + P$

R4. Control ($H, P, \mathcal{F}_0, \text{nfacs}$)

return

Proof of Algorithm 3

- Any input graph with one or more isolated vertices is disregarded in step M2.
- The graph $K_{1,1}$ is the only primal graph with an odd number of edges (Theorem 2). This is recognized in step M3.
- All graphs other than $K_{1,1}$ with an odd number of edges are not primal. Recognized in step M4.
- All graphs $\in \Pi_0$ are known to be primal, Step C2.
- Graphs which can be factored using the primal graphs in Π_0 are recognized by Steps C3 and C5. Graphs which are completely factored by the algorithm are recognized in Step C3, graphs which can be proved to be not primal by Corollary 1 are recognized in Step C5.
- All remaining graphs are those which cannot be factored using the graphs in Π_0 . These graphs are recognized as being "unsolved" in Step M6.

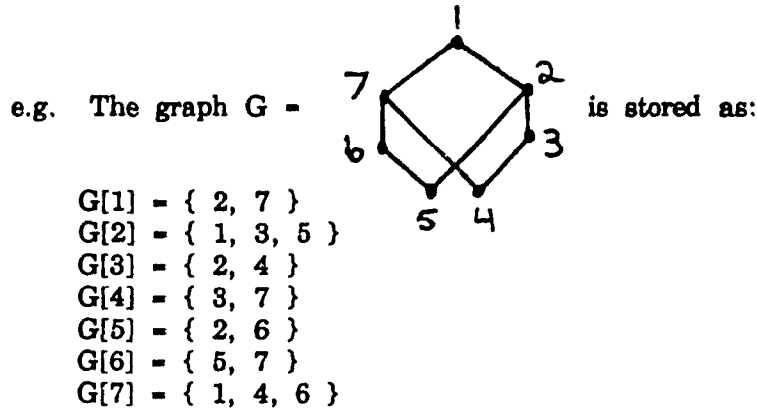
Procedure Control is first invoked with $P = \text{EMPTY}$. The search routine is then called with the first primal graph, P , which may be contained in G_0 (always using the lexical order as defined). The algorithm will either find a complete factorization of G_0 , or a partial factorization of G_0 sufficient to prove that G_0 is not primal by Corollary 1, or else it will find that G_0 does not have a factorization involving P . In this last case, a search is started for a factorization of G_0 involving the primal graph $\text{Next}(P)$. Note that a complete factorization results if G_0 is a primal graph in Π_0 . In this case the number of factors is one, and G_0 is correctly identified as being a primal graph in Π_0 (step C2).

Step C6 insures that time is not spent attempting a useless factorization. If G equals any of the factors in the partial factorization of G_0 , we have arrived at an isomorphic factorization. We must return to the previous level and try a different factorization.

The list of primal graphs becomes exhausted, $P = \text{FULL}$, at the topmost invocation of Control if G_0 does not have a factorization involving any of the primal graphs in Π_0 . The conclusion is that G_0 has a factorization involving a primal graph $\notin \Pi_0$, possibly itself.

Section 4: Search Procedures

When designing the search procedures, it is necessary to know how the graphs are to be represented in memory. The vertices of a graph, G , are labelled from 1 to $N = |V(G)|$. The graph G is stored as an array of N sets, where set $G[i]$ contains the neighbors of vertex v_i . We write i for v_i by abuse of notation. In the following, lower-case letters v_i , v , $a \dots g$ represent vertices.



Constraints are placed upon the search procedures to avoid duplication of searches. Before explaining the constraints used the following definition is provided:

Automorphism of a graph G :

A permutation α of $V(G)$ such that $\{v, w\} \in E(G) \leftrightarrow \{\alpha(v), \alpha(w)\} \in E(G)$.

The automorphisms form a group $\text{Aut}(G)$.

The automorphisms induce an equivalence relation ρ on $V(G)$ by:

$v \rho w$ iff there is an automorphism $\alpha \in \text{Aut}(G)$ such that $\alpha(v) = w$.

The equivalence class of a vertex v under ρ is called the orbit or transitivity class of v .

Following are two constraints which are used in the search procedure algorithms:

Equivalence Class Constraint

When searching for a primal graph, P , we wish to search only for one isomorph, not all. To accomplish this, we require that all vertices in an

equivalence class should follow each other in ascending order.

e.g. When searching for the 3-sun -



the equivalence class constraint requires that $a < b < c$.



When searching for the 3-crab -



the equivalence class constraint requires that $b < c$, and $f < g$.

Order Constraint

This constraint is a direct result of the equivalence class constraint. It is best explained by the use of an example. Take the above example of searching for a 3-sun. The equivalence class constraint requires that $a < b < c$. This results in the order constraint which requires that $a \leq N-2$, $b \leq N-1$, where N = the number of vertices in the graph G . Clearly, if we allow $a > N-2$, it is impossible to find b and c such that the equivalence class constraint will be satisfied.

Search Procedure Algorithmsalgorithm SunSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

Find a, b, c, d, e, f such that :

$$a < b < c$$

$$\{a, b, c\} = K_3$$

$$\{a, d\}, \{b, e\}, \{c, f\} \in E(G)$$

*/

for each $a \in \{v \in V(G) \mid \deg(v) \geq 3 \text{ and } v \leq N-2\}$ dofor each $b \in \{v \in V(G) \mid \deg(v) \geq 3 \text{ and } a < v \leq N-1 \text{ and } \{a, v\} \in E(G)\}$ dofor each $c \in \{v \in V(G) \mid \deg(v) \geq 3 \text{ and } b < v \leq N \text{ and } \{a, v\}, \{b, v\} \in E(G)\}$ dofor each $d \in \{v \in V(G) - \{a, b, c\} \mid \{a, v\} \in E(G)\}$ dofor each $e \in \{v \in V(G) - \{a, b, c, d\} \mid \{b, v\} \in E(G)\}$ dofor each $f \in \{v \in V(G) - \{a, b, c, d, e\} \mid \{c, v\} \in E(G)\}$ doResumeSearch($G, P, \mathcal{F}_0, \text{nfacs}$)return

algorithm CrabSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

Find a, b, c, d, e, f, g such that :

$b < c$

$\{a, b, c\} = K_3$

$\{b, d\}, \{c, e\}, \{f, g\} \in E(G)$

*/

for each $a \in \{v \in V(G) \mid \deg(v) \geq 2\}$ do

for each $b \in \{v \in V(G) \mid \deg(v) \geq 3 \text{ and } v \leq N-1 \text{ and } \{a, v\} \in E(G)\}$ do

for each $c \in \{v \in V(G) \mid \deg(v) \geq 3 \text{ and } b < v \leq N \text{ and } \{a, v\}, \{b, v\} \in E(G)\}$ do

for each $d \in \{v \in V(G) - \{a, b, c\} \mid \{b, v\} \in E(G)\}$ do

for each $e \in \{v \in V(G) - \{a, b, c, d\} \mid \{c, v\} \in E(G)\}$ do

if FindaK11($f, g, V(G) - \{a, b, c, d, e\}$) = FOUND

then

ResumeSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

return

algorithm FindaK11 (a, b, V)

/*

Find $a, b \in V$ such that $\{a, b\} = K_{1,1}$

*/

status \leftarrow NOT-FOUND

for each $a \in \{v \in V\}$ do

for each $b \in \{v \in V \mid \{v, b\} \in E(G)\}$ do

status \leftarrow FOUND

return

return

algorithm RSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

Find a, b, c, d, e, f, g such that :

$b < c$

$\{a, b, c\} = K_3$

$\{a, d\}, \{d, e\}, \{f, g\} \in E(G)$

*/

for each $a \in \{v \in V(G) \mid \deg(v) \geq 3\}$ do

for each $b \in \{v \in V(G) \mid \deg(v) \geq 2 \text{ and } \{a, v\} \in E(G)\}$ do

for each $c \in \{v \in V(G) \mid \deg(v) \geq 2 \text{ and } \{a, v\}, \{b, v\} \in E(G) \text{ and } b < v\}$ do

for each $d \in \{v \in V(G) - \{a, b, c\} \mid \deg(v) \geq 2 \text{ and } \{a, v\} \in E(G)\}$ do

for each $e \in \{v \in V(G) - \{a, b, c, d\} \mid \{d, v\} \in E(G)\}$ do

if FindaK11($f, g, V(G) - \{a, b, c, d, e\}$) = FOUND

then

ResumeSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

return

algorithm QSearch(G, P, \mathcal{F}_0 , nfacs)

/*

Find a,b,c,d,e,f,g such that :

a < b < c

a < d

a < e

{a,b,c,d,e} = C_5 (a pentagon)

*/

for each a $\in \{v \in V(G) \mid \deg(v) \geq 2 \text{ and } v \leq N-4 \}$ do

for each b $\in \{v \in V(G) \mid \deg(v) \geq 2 \text{ and } a < v \leq N-1 \text{ and } \{a,v\} \in E(G)\}$ do

for each c $\in \{v \in V(G) \mid \deg(v) \geq 2 \text{ and } b < v \leq N \text{ and } \{a,v\} \in E(G)\}$ do

for each d $\in \{v \in V(G) - \{a,b,c\} \mid \deg(v) \geq 2 \text{ and } a < v \leq N \text{ and } \{b,v\} \in E(G)\}$ do

for each e $\in \{v \in V(G) - \{a,b,c,d\} \mid \deg(v) \geq 2 \text{ and } a < v \leq N \text{ and } \{c,v\}, \{d,v\} \in E(G)\}$ do

if FindaK11(f, g, $V(G) - \{a,b,c,d,e\}$) = FOUND
then

ResumeSearch (G, P, \mathcal{F}_0 , nfacs)

return

algorithm YSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

Find a, b, c, d, e, f, g such that :

$b < d < f$

$\{a, b\}, \{a, d\}, \{a, f\}, \{b, c\}, \{d, e\}, \{f, g\} \in E(G)$

*/

for each $a \in \{v \in V(G) \mid \deg(v) \geq 3\}$ do

for each $b \in \{v \in V(G) - \{a\} \mid \deg(v) \geq 2 \text{ and } v \leq N-2 \text{ and } \{a, v\} \in E(G)\}$ do

for each $c \in \{v \in V(G) - \{a, b\} \mid \{b, v\} \in E(G)\}$ do

for each $d \in \{v \in V(G) - \{a, b, c\} \mid \deg(v) \geq 2 \text{ and } b < v \leq N-1 \text{ and } \{a, v\} \in E(G)\}$ do

for each $e \in \{v \in V(G) - \{a, b, c, d\} \mid \{d, v\} \in E(G)\}$ do

for each $f \in \{v \in V(G) - \{a, b, c, d, e\} \mid \deg(v) \geq 2 \text{ and } d < v \leq N \text{ and } \{a, v\} \in E(G)\}$ do

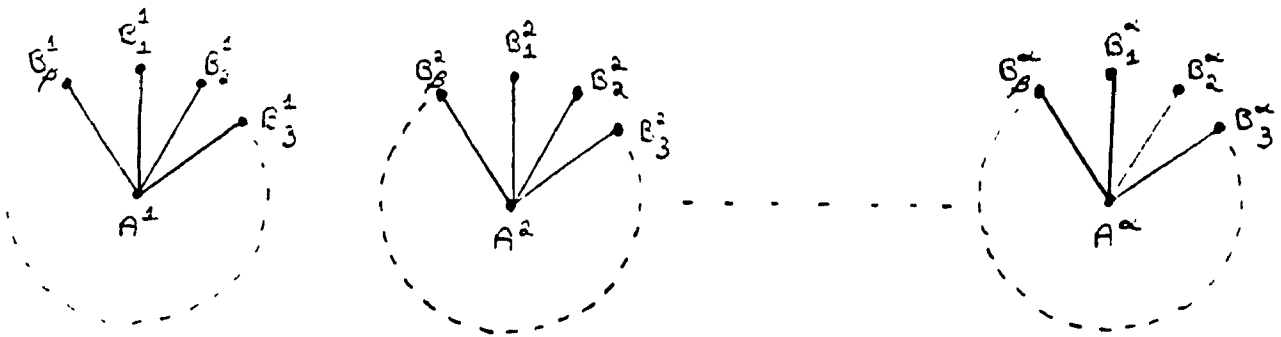
for each $g \in \{v \in V(G) - \{a, b, c, d, e, f\} \mid \{f, v\} \in E(G)\}$ do

ResumeSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

return

Star Search Procedure

The following represents the star $\alpha K_{1,\beta}$ where $\alpha = 2^r$, $\beta = 2^s$, $r, s \geq 0$; A^j is the central vertex of star j , and B_i^j is the i^{th} neighbor of A^j .



To insure that only one isomorphic copy of a particular star is found, we include the isomorph rejection constraints $A^1 < A^2 < \dots < A^\alpha$ and $B_1^i < B_2^i < \dots < B_\beta^i$ for $1 \leq i \leq \alpha$.

algorithm StarSearch($G, P, \mathcal{F}_0, \text{nfacs}$)

/*

Try to find in G a copy of the star $P = \alpha K_{1,\beta}$.

Variables:

α : the number of copies of the star

β : the degree of the star

center: the central vertex of one copy of the star

n : the number of copies of the star found so far

U : the set of vertices used so far in the search for $\alpha K_{1,\beta}$

*/

$\alpha \leftarrow$ the number of copies of the star P

$\beta \leftarrow$ the degree of the star P

$n \leftarrow 0$

center $\leftarrow 0$

$U \leftarrow \emptyset$

Findastar ($\alpha, \beta, \text{center}, n, U$)

return

algorithm Findastar ($\alpha, \beta, \text{center}, n, U$)

if $n = \alpha$

then ResumeSearch ($G, P, \mathcal{F}_0, \text{nfacs}$)

else

for each $a \in V(G) - U \mid \deg(a) \geq \beta$ and $\text{center} < a \leq N$ do

for each $b_1, b_2, \dots, b_\beta \in V(G) - U \mid \{a, b_i\} \in E(G) \forall i$ and
 $1 \leq b_1 \leq b_2 \leq \dots \leq b_\beta \leq N$ do

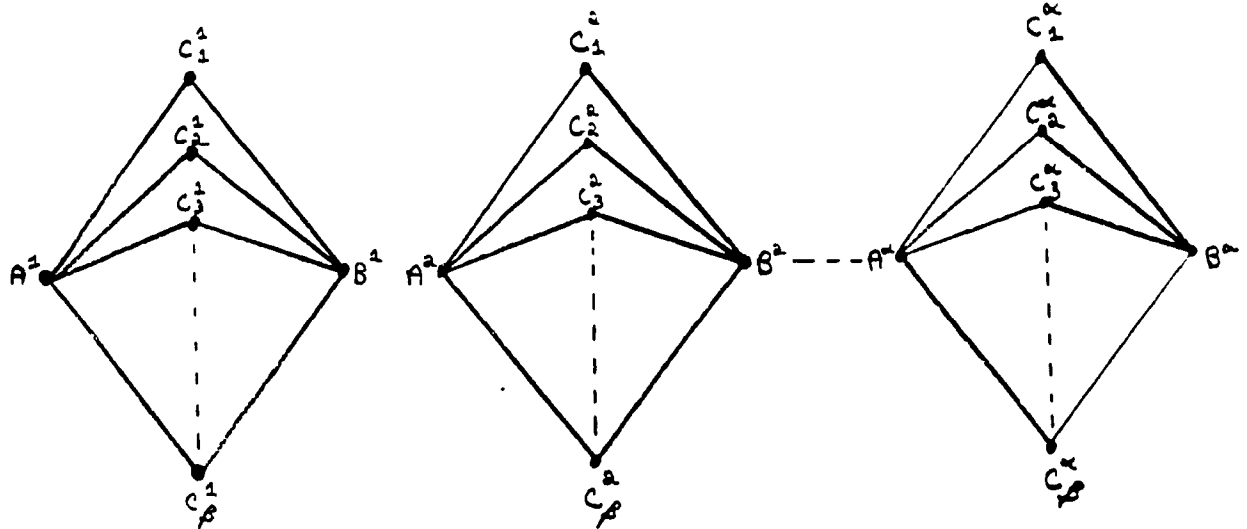
$U \leftarrow U + \{a, b_1, b_2, \dots, b_\beta\}$

Findastar ($\alpha, \beta, a, n+1, U$)

return

K2N Search Procedure

The following represents the K2N $\alpha K_{2,\beta}$ where $\alpha = 2^r$, $\beta = 2^s$, and for $s = 1$, $r \geq 0$; for $s = 2$, $r = 1, 2$; for $s > 2$, $r = 1$; A^j and B^j are the central vertices of the j^{th} K2N, and C_i^j is the i^{th} neighbor of both A^j and B^j .



To insure that only one isomorphic copy of a particular K2N is found, we include the isomorph rejection constraints $A^1 < B^1$ and $A^1 < A^2 < \dots < A^\alpha$ and $C_1^i < C_2^i < \dots < C_\beta^i$ for $1 \leq i \leq \alpha$.

algorithm K2NSearch(G, P, \mathcal{F}_0 , nfacs)

/*

Try to find in G a copy of the K2N $P = \alpha K_{2,\beta}$.

Variables:

α : the number of copies of the K2N

β : the degree of the K2N

center: the smallest of the two central vertices of one copy of the K2N

n: the number of copies of the K2N found so far

U: the set of vertices used so far in the search for $\alpha K_{2,\beta}$

*/

$\alpha \leftarrow$ the number of copies of the K2N P

$\beta \leftarrow$ the degree of the K2N P

$n \leftarrow 0$

center $\leftarrow 0$

$U \leftarrow \emptyset$

FindaK2N (α , β , center, n, U)

return

algorithm FindaK2N (α , β , center, n, U)

if $n = \alpha$

then ResumeSearch (G, P, \mathcal{F}_0 , nfacs)

else

for each $a, b \in V(G) - U \mid \deg(a), \deg(b) \geq \beta$ and center $< a < b \leq N$ do

for each $c_1, c_2, \dots, c_\beta \in V(G) - U \mid \{a, c_i\}, \{b, c_i\} \in E(G) \forall i$ and

$1 \leq c_1 \leq c_2 \leq \dots \leq c_\beta \leq N$ do

$U \leftarrow U + \{a, b, c_1, c_2, \dots, c_\beta\}$

FindaK2N (α , β , a, n+1, U)

return

Section 5: Implementation

The program was originally implemented in Pascal on the Vax 11/780 under VMS. The program was later transferred to and run on a Vax 8500 and a Micro-Vax.

Generation of Input Files

The graphs used for primality testing were generated using a program written by E. Regener, originally in connection with work in Ramsey theory [3]. This program generates for $n \geq 7$ vertices all non-isomorphic graphs not containing any of a certain set K of graphs on six vertices. The set K can be defined by the user to be any subset of the set G_6 of all 156 graphs on six vertices. The program generates graphs on $n \geq 7$ vertices from those on $n - 1$ vertices. The starting set $G_6 - K$ is generated by a separate program.

Three different classes of graphs were generated for primality testing. The first, called the "N-class", contains all graphs on n vertices without restriction: the set of forbidden graphs K_N is empty. The second, or "S-class", contains just those graphs which have no vertex of degree > 3 , and the third, or "T-class", contains those graphs from the S-class which have no cycle of length < 5 . In fact, we restricted the final runs to graphs in a "T-class", with no vertex of degree > 3 and no cycle of length < 7 .

Tables 6a and 6b give the number of graphs generated and tested in each class for each number of vertices ≥ 6 , together with the number of graphs in each category for which factorizations were not found. If the number generated is followed by * , then this is the total number of graphs in its class. If it is followed by $^+$, then this contains the total number of graphs in its class, but there may be some duplications (the graphs generated in each computer run are non-isomorphic, but on different runs there may be overlaps).

	C #Verts	Number Generated	Graphs in Π_0	Other graphs which are non-factorable
N	6	156*	3: $2K_{1,2}$, $K_{2,4}$, 3-sun	0
	7	1044*	4: 3-crab, R, Q, Y	0
	8	12346*	2: $2K_{2,2}$, $4K_{1,1}$	0
	9	274668*	1: $K_{1,8}$	0
	10	2982430 (35%)	2: $K_{2,8}$, $2K_{1,4}$	0
S	6	62*	-	-
	7	150*	-	-
	8	424*	-	-
	9	1165*	-	-
	10	3547*	0	0
	11	10946*	0	0
	12	36327*	1: $4K_{1,2}$	0
	13	124380*	0	0
	14	155858 (38%)	0	1: V7-1
	15	174292 (12%)	0	3: V7-2, V8-1, V6-1

Table 6a - Files Tested for Primality

Of all graphs tested, roughly half were eliminated immediately since they had an odd number of edges; in a given class G_n of graphs on n vertices, the set of graphs with at least one isolated point is just the set of graphs on $n - 1$ vertices, G_{n-1} , and these could also be eliminated with no further testing.

Performance of the Program

In practice, the program spent most of the time searching for stars. This is understandable considering that the primal graphs with the fewest edges are stars and can be removed from a graph in several ways. The program was modified so that if the time taken by a search exceeds a specified threshold value, the search

C #Verts	Number Generated	Graphs in Π_0	Other graphs which are non-factorable
T	6	20*	-
	7	38*	-
	8	83*	-
	9	183*	-
	10	461*	-
	11	1212*	-
	12	3578*	-
	13	11207*	-
	14	38208*	1: See S-class
	15	139375 ⁺	3: See S-class
	16	146459 (30%)	1: $8K_{1,1}$ 8: V7-3, V7-4, V8-2, V8-3, V9-1, V6-2, V6-3, V6-5
	17	532187 (30%)	0 5: V7-5, V8-4, V6-4 V6-6, V6-7
T'	16	32866*	See T-class 8: See T-class
	17	97286 ⁺	0 5: See T-class
	18	26801 (10%)	1: $2K_{1,8}$ 0
	19	84368 (10%)	0

Table 6b - Files Tested for Primality

is aborted and the graph written out on a file for later study.

Experimentation with the threshold value showed that most graphs took only a few tenths of a second to solve, while a very small minority took time ranging from 30 seconds to several minutes. Increasing the threshold time essentially increased the running time in proportion. Thus we evolved the following way of using the program: with a fairly low threshold value (about 1 minute), we separated the "difficult" graphs from the file. The program was run again on these graphs, using a different lexical ordering. These two runs were usually sufficient to

sift from the file all graphs except primal graphs and a few exceptionally difficult cases, which were easily solved by hand. It appears from the nature of the results that the backtracking part of the program is the least efficient by at least an order of magnitude, and if the program is to be run on very large cases this part must be rewritten.

Tables 7, 8a, and 8b show the computer time spent generating the input files and running the program to test for primality on these files. If the number of graphs tested for primality exceeds the number of graphs generated, then some graphs were tested more than once.

C	#Verts	Number of Graphs Generated	Time in seconds to Generate	Number of Graphs Tested for Primality	Time in seconds to Test
N	8	12346	254	12346	137
	9	274668	7828	515638	9466
S	7	150	6	0	0
	8	424	12	0	0
	9	1165	44	0	0
	10	3547	174	3547	34
	11	10946	770	10946	102
	12	36327	3439	36327	766
	13	124380	13553	124380	3984
	14	107232	14488	107232	2284
Totals:		571203	40518	810434	16773

Table 7: Summary of Computer Time spent on Vax 8500

C	#Verts	Number of Graphs Generated	Time in seconds to Generate	Number of Graphs Tested for Primality	Time in seconds to Test
N	8	12346	798	0	0
	9	105562	8802	0	0
	10	2982430	419655	2828064	121732
S	7	150	20	0	0
	8	424	38	0	0
	9	1165	140	0	0
	10	3547	576	0	0
	11	10946	2435	0	0
	12	36327	10815	0	0
	13	52369	24488	0	0
	14	48626	31941	48629	8068
	15	174292	151119	180773	29007
Totals:		3428184	650827	3057466	158807

Table 8a: Summary of Computer Time spent on Micro-Vax.

C	#Verts	Number of Graphs Generated	Time in seconds to Generate	Number of Graphs Tested for Primality	Time in seconds to Test
T and T'	7	38	6	0	0
	8	83	8	0	0
	9	183	24	0	0
	10	461	79	0	0
	11	1212	316	0	0
	12	3578	1205	0	0
	13	11207	5271	0	0
	14	38208	24634	38208	5390
	15	139375	120651	139375	9868
	16	146459	302488	183228	78692
	17	532187	722851	535053	328040
	18	26801	28578	26814	19241
	19	84368	113927	85937	106328
Totals:		984160	1320038	1008615	547559

Table 8b: Summary of Computer Time spent on Micro-Vax

Section 6: Conclusions

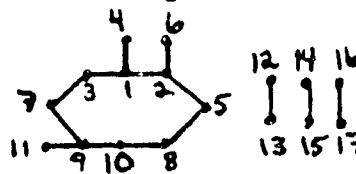
The algorithm is implemented in such a way that only a single line of code needs to be altered to change the ordering of the classes. This allows testing the various class orderings to see which is the most efficient. Table 9 shows some of the class orders tested on a file of 40161 graphs on 9 vertices with the time spent on each run. The timings of the different orderings suggest that it is best to order the graphs with priority given those graphs with the most edges, the least amount of symmetry, and no independent $K_{1,1}$'s. The extra $K_{1,1}$ in Q, R, and the 3-crab lengthens the search for these graphs since several $K_{1,1}$'s may be found for each occurrence of $P - K_{1,1}$, where P is the Q, R, or 3-crab. Corollary 1

Run Number	Class Order	Run Time in Seconds
008	3-sun, Q, R, Y, 3-crab, K2N's, Stars	459
010	3-sun, 3-crab, Q, R, Y, K2N's, Stars	565
005	Q, R, Y, 3-crab, 3-sun, K2N's, Stars	629
006	K2N's, Q, R, Y, 3-crab, 3-sun, Stars	907
002	R, Q, Y, 3-crab, 3-sun, K2N's, Stars	1098
009	Y, 3-sun, Q, R, 3-crab, K2N's, Stars	aborted after 600

Table 9: Run Times for Different Lexical Orders for a file consisting of 40161 graphs on 9 vertices.

states that for a graph G , $G \sim \mathcal{F}(G) + H$, where $\mathcal{F}(G)$ is a partial factorization of G , if $\forall P \in \mathcal{F}(G)$, $P \notin H$ then G factors into elements of Π . We were curious to see if the time spent deciding if all graphs in the partial factorization were not contained in H would be less than the time taken to ignore this step and simply continue the factorization. We disabled the code corresponding to Corollary 1 and re-tested run number 005, now called run 007. Run 007 required 1216 seconds, nearly twice the time required by run 005, 629 seconds. This indicates that

corollary 1, when incorporated into the algorithm provides a significant savings in search time.



The search for a factorization of the graph G - was aborted after 120.17 seconds. The lexical ordering of the primal graphs used for this search gives priority to K_{2N} 's over stars. Since all K_{2N} 's have priority over all stars, the program searches for a factorization involving both a $2K_{1,2}$ and a $K_{1,2}$. This graph cannot be factored using both a $2K_{1,2}$ and a $K_{1,2}$ as factors. If we were searching for primal graphs with more edges first, then we would search much sooner for $8K_{1,1}$ as a factor, whereupon the result would be immediate by Lemma 1.

Our program has delivered a total of 17 primal graphs with 14 edges, including the 7-sun and the 7-crab (see Tables 1-5). The graphs V6-8 to V6-12 inclusive were determined by hand. The graphs V6-9 to V6-12 could not be produced by the program in class T since they contain triangles. The graph V6-8 was not included in any of the runs. We produced these five graphs by hand based upon the results generated by the program, they are visually very similar. The graphs in Tables 1 to 5 include all primal graphs of 14 edges presently known, of which graphs V6-5, V6-6, V6-9, and V6-10 appear here for the first time. We have reason to believe that this list of primal graphs on 14 edges is complete.

Based on the tests run for this research, we can estimate the time needed to complete the search for primal graphs on a given number of vertices with certain restrictions. Table 10 shows our estimates in hours of computing on a Micro-Vax for certain classes of graphs (numbers of graphs are estimated to 3 figures).

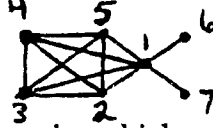
Possible Improvements

Symmetry of G_0 : Each of the search procedure algorithms recognize the symmetrical properties of the primal graphs by imposing constraints on the

Class of Graphs	Number of Vertices	Estimated Number of Graphs	Estimated Time in Hours
N-class	10	5558800 (65%)	279
S-class	14	254300 (62%)	59
	15	1278100 (88%)	367
T-class	16	341700 (70%)	247
	17	1241800 (70%)	681
T'-class	18	241200 (90%)	120
	19	759300 (90%)	551

Table 10 - Estimate of time required to complete some test cases.

searching techniques. The symmetry of the input graph, G_0 , is not recognized by the program and for some graphs this results in several redundant searches being performed. This inefficiency is best demonstrated by use of an example:

The graph $G =$  can be factored only using stars and K2N's. If the lexical ordering gives higher priority to the 3-sun than to the stars and K2N's then much search time is wasted attempting to factor G using the 3-sun. The 3-sun is contained in this graph but each configuration of $G - 3\text{-sun}$ yields another 3-sun. The program attempts to remove the 3-sun from the graph G 24 different ways before concluding that G does not have a factorization using the 3-sun. If the program recognized that vertices $\{2,3,4,5\}$ and $\{6,7\}$ each form a transitivity class of $V(G)$, then the conclusion that G does not have a factorization involving the 3-sun would be reached after the first attempt at such a factorization.

The efficiency of the algorithm depends heavily on the lexical ordering of the primal graphs. While the ordering of the classes is easily changed, the ordering of the graphs within the classes of stars and K2N's is not easily changed. It would be beneficial to change the lexical ordering so that it is independent of the classes. This would allow us to search for the larger graphs first. The order of the stars and K2N's would be interspersed, we would not need to search for all K2N's before

searching for any star.

The results and the nature of the problem suggest that the search for primal graphs may be best undertaken using files of graphs on a given number of edges rather than on a given number of vertices.

Appendix: Proof of Theorem 1

A set of graphs Γ is said to be primal relative to the set of graphs Ω if:

- i) $\Gamma \subseteq \Omega$,
- ii) each graph in Ω factors into non-isomorphic elements of Γ , and
- iii) graphs in Γ have only a trivial such factorization.

Theorem 1: (Dewdney [1])

Let Γ_n denote the set of all graphs with at most n vertices. Then there is a unique set Π_n of graphs that is primal relative to Γ_n .

Proof: (Dewdney [1])

The theorem is true in the case $n = 1$ for here $\Gamma_1 = \{\text{a single point}\} = \Pi_1$ and Π_1 is primal relative to Γ_1 since iii) holds and ii) holds trivially.

Suppose the theorem is true for $n = k - 1$. Since Γ_{k-1} contains at most a finite number of distinct graphs, let

$$\Pi_{k-1} = \{P_1, P_2, \dots, P_m\}$$

be the existing and unique set of graphs primal relative to Γ_{k-1} . If Π_k exists, it must contain a subset Π'_k consisting of all graphs in Π_k having one or more isolated points. If $G \in \Gamma_k$ and G has an isolated point, then G can be factored into distinct graphs in Π'_k . Furthermore Π'_k has property iii) with respect to the graphs in Γ_k which contain at least one isolated point. Thus if exactly one isolated point is removed from each graph in Π'_k , we obtain a set primal relative to Γ_{k-1} . By induction assumption, this set must be Π_{k-1} . If Π_k exists, we must include the graphs P'_1, P'_2, \dots, P'_m obtained from P_1, P_2, \dots, P_m respectively by adding exactly one isolated point to each. Let $\Pi'_k = \{P'_1, P'_2, \dots, P'_m\}$.

If every graph in Γ_k can be factored into distinct graphs in Π'_k , we have shown that $\Pi_k (= \Pi'_k)$ exists and is unique. Otherwise let $\Theta = \{H_1, H_2, \dots, H_r\}$ be the set of graphs in Γ_k which cannot be so factored. At least one of these graphs must have the property that any proper subgraph can be factored into distinct graphs in Π'_k . For if H_1 , say, does not have this property, it contains a proper subgraph K which cannot be so factored. Moreover $K \in \Theta$ by definition. Applying the same reasoning to K tells us either that K has the property in question or that it has some proper subgraph $\in \Theta$ which doesn't have the property. This process cannot continue indefinitely. Without loss of generality we assume that any proper subgraph of H_1 can be factored into distinct graphs in Π'_k .

If Π_k exists, it must contain H_1 , else H_1 can be factored non-trivially into distinct graphs of Π_k and each of these factors would be proper subgraphs of H_1 and therefore would lie in Π'_k . But H_1 cannot be so factored by its definition.

Let $\Pi''_k = \Pi'_k \cup \{H_1\}$ and let $\{H'_1, H'_2, \dots, H'_s\}$ be the set of graphs in Γ_k which cannot be factored into distinct graphs in Π''_k . (If none such exist, the argument below shows that Π'_k is the uniquely existing set primal relative to Γ_k). Arguing as before, we may assume that H'_1 has the property that any proper subgraph can be so factored. As before we can show that $H'_1 \in \Pi_k$, if Π_k exists.

Enlarge Π''_k to Π'''_k by adjoining the graph H'_1 and continue this process until a set $\bar{\Pi}_k$ is reached such that every graph in Γ_k can be factored into distinct graphs in $\bar{\Pi}_k$. $\bar{\Pi}_k$ has the property iii) relative to Γ_k since this property was preserved at each stage of the construction of $\bar{\Pi}_k$. Now $\bar{\Pi}_k$ is a set primal relative to Γ_k and is a set whose existence we have shown by construction. If Π_k contains a graph not in $\bar{\Pi}_k$, it can be factored non-trivially into graphs in $\bar{\Pi}_k \leq \Pi_k$ and thus into graphs in Π_k , which is a contradiction. Thus the set Π_k which is primal relative to Γ_k exists and is unique. This proves the theorem.

Bibliography

1. A. K. Dewdney, "Primal Graphs", *Aequationes Mathematicae* 4(1970), 326-328.
2. P. Z. Chinn, R. B. Richter, P. A. Thoelecke, "Families of Primal Graphs", Humboldt State University, Arcata, California, 95521.
3. E. Regener, C. W. H. Lam, J. Opatrny, "Invariants for exhaustive searching in Ramsey theory", Sixteenth S.E. Int. Conf. on Combinatorics, Graph Theory, and Computing, Feb. 1985. *Congressus Numerantium*, Vol. 49, pp. 147-160, 1985.
4. Robin J. Wilson, "Introduction to Graph Theory", Edinburgh: Oliver & Boyd, 1972.
5. R. B. Richter, private communication.