Fast Parallel Algorithms for the Modular Decomposition

Mark B. Novick*

TR 89-1016
June 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This work was supported by NSF grant CCS-8806979.

Fast Parallel Algorithms for the Modular Decomposition

Mark B. Novick *

Abstract

A module in a graph is like a black box: all the vertices in the module look the same to vertices not in the module. This paper gives the first NC algorithm for finding the modular decomposition of a graph. The algorithm runs in $O(\log n)$ time using $O(n^3)$ processors on a CRCW PRAM. This decomposition is used to obtain fast sequential and parallel algorithms for solving graph problems on graphs of bounded module size, e.g. the class of cographs, where each module with more than one vertex is either disconnected or its complement is disconnected. These graph problems include minimum coloring, maximum clique, matching, Hamiltonian circuit, and maximum cut. Many of these problems can be solved with $O(n)$ processors in $O(\log n)$ time. All of them can be solved in NC. Our modular decomposition algorithm can be used to obtain more efficient algorithms for recognizing and orienting comparability graphs.

1 Background

The modular decomposition of an undirected graph G repeatedly breaks the vertex set of G into modules, subsets of vertices which are adjacent to the same vertices outside the module. Many researchers have used a modular decomposition to solve graph problems such as maximum clique, finding longest paths in a directed acyclic graph, and solving scheduling problems. Two good surveys are [10] and [9]. Several sequential algorithms have been given for finding the modular decomposition, with Muller and Spinrad's $O(n^2)$ time algorithm [11] being the fastest known. These algorithms tend to be inherently sequential. Kozen *et al.* [7] gave an NC algorithm to find the Gallai decomposition [4], a modular decomposition for the class of comparability graphs, the graphs which can be transitively oriented. We improve their result in two ways: we generalize it so that it works for graphs other than comparability graphs, and also give a more efficient implementation of the algorithm. Our algorithm runs in $O(\log n)$ time on a concurrent read-concurrent write PRAM using $O(n^3)$ processors. The corresponding sequential version is as fast as nearly all other sequential algorithms for the problem except for Muller and Spinrad's.

We also give some classes of modularly composed graphs for which it is easy to solve the problems of maximum clique size, chromatic number, maximum cut, and matching. The first three of these problems are *NP-complete* for general graphs, but we obtain fast polynomial time algorithms for our restricted versions. Furthermore, all of these problems are hard to solve in

*Computer Science Department, Cornell University, Ithaca, NY 14853-7501. Supported by NSF grant CCS-8806979.

parallel for general graphs, yet when restricted to suitable modularly composed graphs we get efficient NC algorithms.

Our modular graph classes are generalizations of cographs. Cographs can be decomposed so that no module has more than two maximal submodules. We generalize this notion by examining graphs whose modules can have more than two maximal submodules as long as the number of maximal submodules is bounded. This should often happen in practical situations since many modularly designed systems do not have large modules, but instead are composed of many smaller modules. (For example, a modularly designed program would consist of many small procedures, none of which is more than a page long.)

Our algorithmic approach resembles one that has been used for other graph classes with "nice" decomposition trees, such as trees and series-parallel graphs. The standard dynamic programming methods (see [2]) break the graph into two or more pieces, solve the problems on the individual pieces, and then combine this information to get the answer for the entire graph. We can do this because modules can be homomorphically equivalent with respect to a graph problem. It does not matter which module is used among modules that are homomorphic. All of them when used will give the same final answer. He and Yesha [5] have shown how to parallelize the algorithms for trees by using Miller and Reif's [8] technique of parallel tree contraction. This technique can be used to give parallel algorithms for problems on modularly composed graphs.

The outline of the remainder of the paper is as follows: After giving the basic definitions, in section 3 we show how to find the modular decomposition of a graph efficiently in parallel. Next we give NC algorithms for maximum clique size, chromatic number, maximum matching size, and maximum cut on generalizations of cographs. One larger class consists of graphs whose modules are composed of a bounded number of maximal submodules. We solve these problems with brute-force efficiently since there are few maximal submodules. The other more general class consists of those graphs whose modules are trees. Previous results about tree algorithms scale up to this larger graph class. We conclude by suggesting some ways of extending our work.

2 Definitions

We let $G = (V, E)$ denote a graph in the usual way with $n = |V|$ and $m = |E|$. E. The complementary graph of G, G^c, contains as edges E^c all edges not present in G. Many definitions and results below are symmetric in E and E^c; in the statement of such results, we use the letter F to represent either E or E^c, and F^c to represent the other.

Definition 2.1 *A module is a set A of vertices that are indistinguishable by the vertices outside the set; i.e., any vertex in $V - A$ is either adjacent to all vertices of A or to none of them.*

Modules are formed from smaller submodules in three ways: parallel, series, and neighborhood modules.

Definition 2.2 *Parallel modules are the disjoint union of a collection of submodules. Series modules are the disjoint union of a collection of submodules, plus edges between every pair of vertices in different modules is joined by an edge. In a neighborhood module, the graph induced by the submodules is both connected and complement-connected.*

Definition 2.3 *A modular decomposition of G is a tree whose nodes are modules of G. Every vertex of G is a leaf in this decomposition tree. A parent node in this tree contains all of its children as submodules. The children of a parallel module form a maximal set of connected submodules. The submodules of a series module form a maximal set of complement-connected graphs. The children of a neighborhood module are maximal proper submodules of the neighborhood module.*

Cographs are a special type of modularly composed graph that has been examined in the literature. For example, Corneil *et al.* [3] gave efficient algorithms for solving the Hamiltonian circuit and coloring problems on cographs.

Definition 2.4 *Cographs are those graphs which do not have any neighborhood modules in their modular decomposition.*

3 Finding the modular decomposition

We give an algorithm that can find the modular decomposition of a graph on a concurrent-read concurrent-write parallel random-access machine (CRCW PRAM with arbitrary resolution of write conflicts) in $O(\log n)$ time using $O(n^3)$ processors. Several researchers (see [11]) have shown that a graph has a unique modular decomposition.

In describing the algorithm we will find it helpful to introduce some terms used in the study of comparability graphs.

Definition 3.1 *A transitive orientation of G is a subset T of E such that*

1. *for each $uv \in E$, either $uv \in T$ or $vu \in T$ but not both;*
2. *if $uv, vw \in T$, then $uw \in T$.*

The graph G is called a comparability graph if there exists a transitive orientation of it.

In the following definition, directing uv in one direction forces us to direct $u'v'$ in the "same" direction if we are to obtain a transitive ordering of G.

Definition 3.2 *For $uv, u'v' \in F$, define $uv \angle u'v'$ if either*

1. *$u = u'$ and $vv' \in F^c$, or*
2. *$v = v'$ and $uu' \in F^c$.*

*Let $\angle^*$ be the reflexive transitive closure of $\angle$. The $\angle^*$-*class *of uv is denoted $[uv]$ and is called the implication class of uv. The set $[uv] \cup [vu]$ is called the color class of uv. The set of vertices touched by edges of $[uv]$ is denoted by $V([uv])$. Note that $\angle$ and $\angle^*$ are defined on edges in E^c as well as those in E.*

Definition 3.3 *For $uv, u'v' \in F$, define $uv \sim u'v'$ if either*

1. *$u = u'$ and $vv' \in [vu] \cup [uv']$, or*

2. $v = v'$ and $uu' \notin [uv] \cup [vu']$.

Let Γ^* be the reflexive transitive closure of Γ. The Γ^*-class of uv is denoted by $[\![uv]\!]$. The set of vertices touched by edges of $[\![uv]\!]$ is denoted $V[\![uv]\!]$.

Definition 3.4 *A subset $A \subseteq V$ is called normal if for all $u \notin A$ and $v, v' \in A$, $[uv] = [u'v']$.*

Normal sets and modules are related by the following lemma.

Lemma 3.1 *Every normal set of G also forms a module of G, though not necessarily a series, parallel, or neighborhood module.*

Two lemmas of Kozen *et al.* are useful in characterizing normal sets.

Lemma 3.2 *$V[\![uv]\!]$ is the smallest normal set containing u and v.*

Lemma 3.3 *If A, B are normal, then either $A \subseteq B$, $B \subseteq A$, or $A \cap B = \emptyset$.*

By using these lemmas, they had characterized the normal sets of a graph, and shown when a graph has no nontrivial normal subsets.

Theorem 3.1 *The following are all the normal subsets of G:*

1. *singletons $\{u\}$, $u \in V$;*
2. *$V[\![uv]\!]$, $u, v \in V$.*

Theorem 3.2 *A graph G has no nontrivial normal subsets if and only if one of the following conditions holds:*

1. *G is a clique;*
2. *G is totally disconnected;*
3. *G contains at least four vertices and has exactly two color classes, one each in E and E^c.*

These three cases correspond to series, parallel, and neighborhood modules respectively. Kozen, Vazirani, and Vazirani gave the following algorithm as part of an algorithm to orient comparability graphs. In fact their algorithm can be used to compute the modular decomposition of any graph. The steps of the algorithm are:

1. Determine all classes $[uv]$ and $[\![uv]\!]$ for all pairs of vertices u and v.
2. Find all normal sets.
3. Create the tree of normal sets, ordered by inclusion.

We give a more efficient implementation of their algorithm.

Theorem 3.3 *The modular decomposition of a graph can be found in $O(\log n)$ time on a CRCW-PRAM with $O(n^3)$ processors.*

Proof: The forcing relations $\angle$ and [illegible] are symmetric, so we find their transitive closure by graph connectivity algorithms [13]. This graph has $O(n^3)$ edges because each uv is only adjacent to vertices which contain u and v. We can check two normal sets for inclusion by checking if they contain a vertex in common, and which set has larger size. By a previous lemma we know that two normal sets cannot just partially intersect. The bottleneck in the algorithm is the first step, which uses $O(n^3)$ processors and $O(\log n)$ time. ▮

The process of finding the modular decomposition of a comparability graph is the dominant one in the parallel algorithms for orienting a comparability graph. Applying our algorithm also gives a more efficient algorithm for orienting a graph than the one given by Helmbold and Mayr [6] did.

4 Algorithms on special modular graphs

We can obtain fast algorithms for solving our problems on cographs because each module can be represented as the composition of only two smaller modules. More generally, when the neighborhood modules have small size, *i.e.* small number of maximal submodules, we can apply brute force enumeration to solve the problems on the bounded size modules. This gives us asymptotically efficient algorithms, albeit with a large constant factor behind the big-Oh. Also we show that if the neighborhood modules are trees, then we can efficiently use the modular decomposition to solve graph problems. The approach we use there is similar to the one used by Bern *et al.* [2] for problems on graph classes like trees and series-parallel graphs. In the remainder of this section we show how to extend our cograph algorithms to more general classes of modularly composed graphs.

4.1 Maximum weight clique

The first, and easiest, problem we examine is finding the maximum clique of a graph.

Theorem 4.1 *Given the modular decomposition of a graph with bounded module size, we can solve the maximum weight clique problem in $O(n)$ time. A parallel version of this algorithm runs in $O(\log n)$ time using $O(n)$ processors.*

Proof: Since the submodules look like black boxes to the other modules in the graph, we want to find the maximum weight clique in each submodule in order to find the maximum weight clique of the entire graph. We give each module a weight, namely the weight of its maximum clique. A maximum weight clique in the resulting graph corresponds to a maximum weight clique in the original graph. There are less than 2^k cliques in a graph with k vertices. If no neighborhood module has more than k maximal submodules, then in constant time we can quickly find its maximum weight clique when k is bounded. Doing this for each of the $O(n)$ nodes of the decomposition tree gives a linear-time algorithm.

We can parallelize this method by parallel tree contraction. We construct a computation tree corresponding to the decomposition tree. Each computation tree node corresponding to a neighborhood module of size k contains k children. When all but one of the children has been evaluated the node becomes half-evaluated, and it can later be compressed. Suppose the unevaluated submodule has maximum clique size ω', the maximum clique size of the rest of the module is a, and the maximum clique size of the neighboring submodules is b. Then $\omega = \max(a, \omega' + b)$. Functions of this form can be composed quickly. If we compose it with a function of the form $\omega'' = \max(c, \omega + d)$, then $\omega'' = \max(\max(b + c, d), \omega' + (a + c))$. The tree contraction takes $O(\log^2 n)$ time using $O(n)$ processors, because half-evaluating a node requires $O(\log n)$ time. ∎

We can also easily handle the case when the neighborhood module is a tree. We just find the maximum weight clique in the corresponding weighted tree, which can be done by standard methods. The parallel version uses tree contraction and composes functions of the form seen in the previous paragraph. The running times of the algorithm for these graphs is asymptotically the same as that for bounded module size graphs.

4.2 Maximum cut

Suppose the largest module of G is a neighborhood module with k maximal submodules and edge set N representing which submodules are adjacent.

Theorem 4.2 *The maximum cut problem can be solved in $O(n^k)$ time for graphs with maximum module size k. The corresponding parallel algorithm uses $O(n^{k+1} + n^4)$ in $O(\log^2 n)$ time.*

Proof: Let $c_j[i_j]$ denote the size of the maximum cut of submodule j that puts i_j vertices on one side of the cut. Then $c[i]$ is defined by:

$$c[i] = \max_{i_1 + \ldots + i_k = i} \left(\sum_{j=1}^{k} c_j[i_j]\right) + \sum_{a,b \in N} i_a(n_b - i_b) + (n_a - i_a)i_b.$$

This follows from trying all the different ways of cutting G into two pieces, one of which contains exactly i vertices.

Let $T(n)$ denote the worst-case running time of this algorithm on an n vertex graph. It obeys

$$T(n) = \max_{n_1 + \ldots + n_k = n} T(n_1) + \ldots + T(n_k) + n_1 n_2 \ldots n_k.$$

Because of convexity, this recurrence is maximized when the graph is divided into submodules of equal size. Solving the resulting recurrence gives $T(n) = O(n^k)$. The algorithm can be parallelized to run in NC by using parallel tree contraction where the compress step is a $(\max, +)$ matrix multiplication. Forming the matrices used requires $O(n^k)$ per matrix, while $O(n)$ are used per step of the algorithm. If we also include the processors require for the matrix multiplications, we find the entire algorithm uses $O(n^{k+1} + n^4)$ processors. ∎

We can make changes to the algorithm so that it works when the neighborhood module is a tree. We use the table-driven approach of Bern *et al.* among others.

Theorem 4.3 *Given a modular decomposition, the maximum cut problem can be solved in $O(n^3)$ time for graphs whose neighborhood modules are trees. This algorithm can be converted to an NC algorithm.*

Proof: Root the tree, and then break it into two subtrees, one of which is rooted at the old root, and the other is rooted at one of the old root's children. Let n_p and r_p denote the number of vertices in the modules of the parent tree and the number of vertices in the root module of the parent tree, respectively. For $0 \le i_p \le n_p$, $0 \le j_p \le r_p$, let $c_p[i_p, j_p]$ denote the size of the largest cut which leaves exactly n_p vertices, including r_p vertices in the root module, on one side of the cut. Let n_c, r_c and c_c denote the corresponding values for the child tree, while n, r and c are the ones for the entire tree. Then c can be computed by the formula:

$$c[i, j] = \max_{i_p + i_c = i} (c_p[i_p, j] + c_c[i_c, j_c] + j(i_c - j_c) + (i_p - j)j_c).$$

Computing this table takes time $O(n_p n_c r_p r_c) = O(n^4)$. Since these tables are calculated at most n times, the entire algorithm takes $O(n^5)$ time. Again there is an $\mathcal{NC}$ solution based on tree contraction. ▮

4.3 Matching and coloring

The problems of matching and coloring a graph can both be formulated as integer programs. Suppose we have a module with k maximal submodules and edge set N. Once again let u_i denote the number of unmatched vertices in a maximum matching of submodule i, let χ_i denote the chromatic number of submodule i, and let m_{ij} denote the number of matched edges between vertices in module i and module j. Matching can be approached as a packing problem:

$$\max \sum_{(i,j)\in N} m_{ij} + \sum_i m_{ii}$$

subject to

$$2m_{ii} + \sum_{(i,j)\in N} m_{ij} \le n_i$$

for $i = 1, \ldots, k$

$$m_{ii} \le (n_i - u_i)/2$$

for $i = 1, \ldots, k$

$$m_{ij} \ge 0$$

and is integral for $1 \le i, j \le k$.

Coloring can be viewed as covering a graph by maximal independent sets. Let $\mathcal{I}$ denote the set of maximal independent sets of the neighborhood module graph. Then:

$$\min \sum_{I \in \mathcal{I}} x_I$$

subject to

$$\sum_{I: j \in I} x_I \ge \chi_j$$

for $j = 1, \ldots, k$

$$x_I \ge 0$$

and is integral for $1 \le j \le k$.

Each of these integer programs has at most $O(2^k)$ inequalities, where each inequality coefficient is either zero or one. This is sufficient to show that the number of extreme points of these integer polyhedra is at worst exponential in k, and independent of either the u or χ vectors. This is a consequence of the following result appearing as Theorem 17.4 in Schrijver [12]:

Lemma 4.1 *For each rational matrix A there exists an integral matrix M such that: for each column vector b there exists a column vector d such that*

$$\{x : Ax \le b\}_I = \{x : Mx \le d\},$$

where the subscript of I means the integer hull of the corresponding set. In fact, the rows of M can be chosen to be integer vectors w satisfying $\|w\|_\infty \le n^{2n}\Delta^n$.

Theorem 4.4 *The matching and coloring polytopes of a graph with k maximal submodules have an exponential number of extreme points, but this bound does not depend on u or χ.*

Proof: Consider the above integer programming formulations. The maximum subdeterminant is clearly $\le k^k$. Thus M can have at most $(k^{2k}\Delta^k)^k = k^{2k+kk}$ rows. If we check every possible basic solution for feasibility, then we examine at most $(k^{2k+kk})^k = k^{O(k^3)}$ possible extreme points. ∎

Theorem 4.5 *Matching and coloring can both be done in $O(n)$ time for graphs with a bounded size modular decomposition.*

Proof: By the previous theorem, the number of extreme points of the integer polyhedron only depends on k. Thus we can compute the matching or coloring number of a module by taking the maximum of our objective function at every basic feasible solution. This can be done in constant time for any fixed k. We do this for each of the $O(n)$ nodes of the decomposition tree, thereby we get an $O(n)$ time algorithm. ∎

We can still parallelize the matching algorithm by using tree contraction, but the proof that this is possible is non-trivial.

Theorem 4.6 *Suppose the maximal module of G contains k maximal submodules of sizes $n_1, \ldots, n_k$, all of which are known. Also suppose $u_2, \ldots, u_k$ are known, but u_1 is not. Then u can be represented as a function of u_1 of the form $\max(p, u_1 - q)$, for a suitable choice of constants p and q, namely $p = u(n_1 \bmod 2)$ and $q = n_1 - u(n_1)$.*

Proof: Clearly u_1 must have the same parity as n_1. For any x of the same parity as n_1 between 2 and n_1, it is clear that $u(x-2) \le u(x) \le u(x-2) + 2$. We will also show that u is convex, thereby implying it is piecewise linear.

Lemma 4.2 *If x has the same parity as n_1 and also satisfies $4 \le x \le n_1$, then we cannot have $u(x) = u(x-2) = u(x-4) + 2$.*

Proof: Our argument mimics the standard argument used to show that a graph has a maximum matching if and only if it has an augmenting path. The details will be given in the final version of the paper. ∎

We have shown that $u(u_1)$ is a piecewise linear function, consisting of a constant portion for low values of u_1 followed by a portion with slope 1 for high values of u_1. Thus $u(u_1)$ has the form $\max(p, u_1 - q)$. We choose $p = u(n_1 \bmod 2)$ so the formula is correct at the low end. For the high end, we choose q so that $n_1 - q = u(n_1)$. This means $q = n_1 - u(n_1)$. ▮

Theorem 4.7 *Given the modular decomposition of a bounded module size graph, we can find its maximum matching in $O(\log n)$ time using $O(n)$ processors.*

Proof: Use parallel tree contraction. Composing functions of the form $u(x) = \max(p, x - q)$ leaves another function of the same form which can be computed in constant time. Therefore, the entire tree contraction can be done efficiently. ▮

Coloring becomes harder when we try doing it in parallel. We do not know of an $O(n)$ processor $\mathcal{NC}$ algorithm for it. We mention in passing that matching and coloring problems remain easy when the neighborhood modules are trees. We omit the details and leave them to the reader.

We give a simple example of coloring a graph using the modular decomposition on a graph that is not perfect, *i.e.* it contains an induced subgraph for which the chromatic number and maximum clique size differ. Consider the case of C_5, the cycle with five vertices where v_i is adjacent to $v_{i+1 \bmod 5}$, and χ_i denotes the chromatic number of module i. Without loss of generality, assume that no module has smaller chromatic number than submodule one. Suppose x of the vertices in submodule one receive the same color as vertices in module four, and y of them receive the same color as vertices in module three. Then the minimum number of colors required to color G is:

$$a + \max(b + c - y, d + e - x, c + d - x - y)$$

subject to:

$$0 \le x \le d, 0 \le y \le c, x + y \le a.$$

Clearly, we minimize this when $x + y = a$. Thus the number of colors is:

$$\max_{0 \le x \le a} (b + c + x, a + d + e - x, c + d).$$

Solving this system we find that

$$\chi = \max(b + c, d + e, c + d, \lceil (a + b + c + d + e)/2 \rceil).$$

This function can be computed in $O(1)$ time given the chromatic numbers of the modules. Thus we can compute the chromatic number of a graph all of whose neighborhood modules are C_5's in $O(n)$ time.

5 Future directions

Our work can be extended in several ways. Here are a few suggestions. First, it would be desirable to obtain the modular decomposition in $\mathcal{NC}$ by using only $O(n^2)$ processors instead of $O(n^3)$ of them, thereby matching Muller and Spinrad's sequential bound. Second, we would like to be able to automatically generate programs for using the modular decomposition to solve graph problems. Similar work has been done on partial k-trees by Arnborg *et al.* [1], by using second-order logic. Perhaps the most important impact of our work will be in the area of practical applications. Many systems are designed in a modular fashion because that makes them easier to design and understand. Therefore, we believe it would be fruitful to try to design algorithms that take advantage of the modular structure of the systems. Our work helps start this effort.

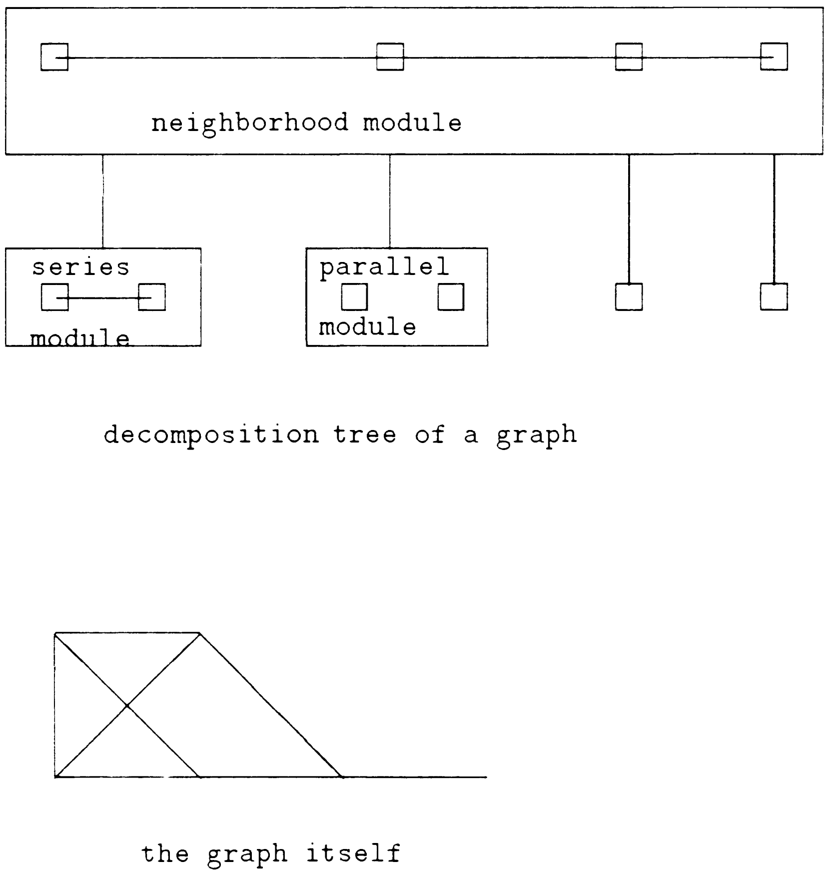
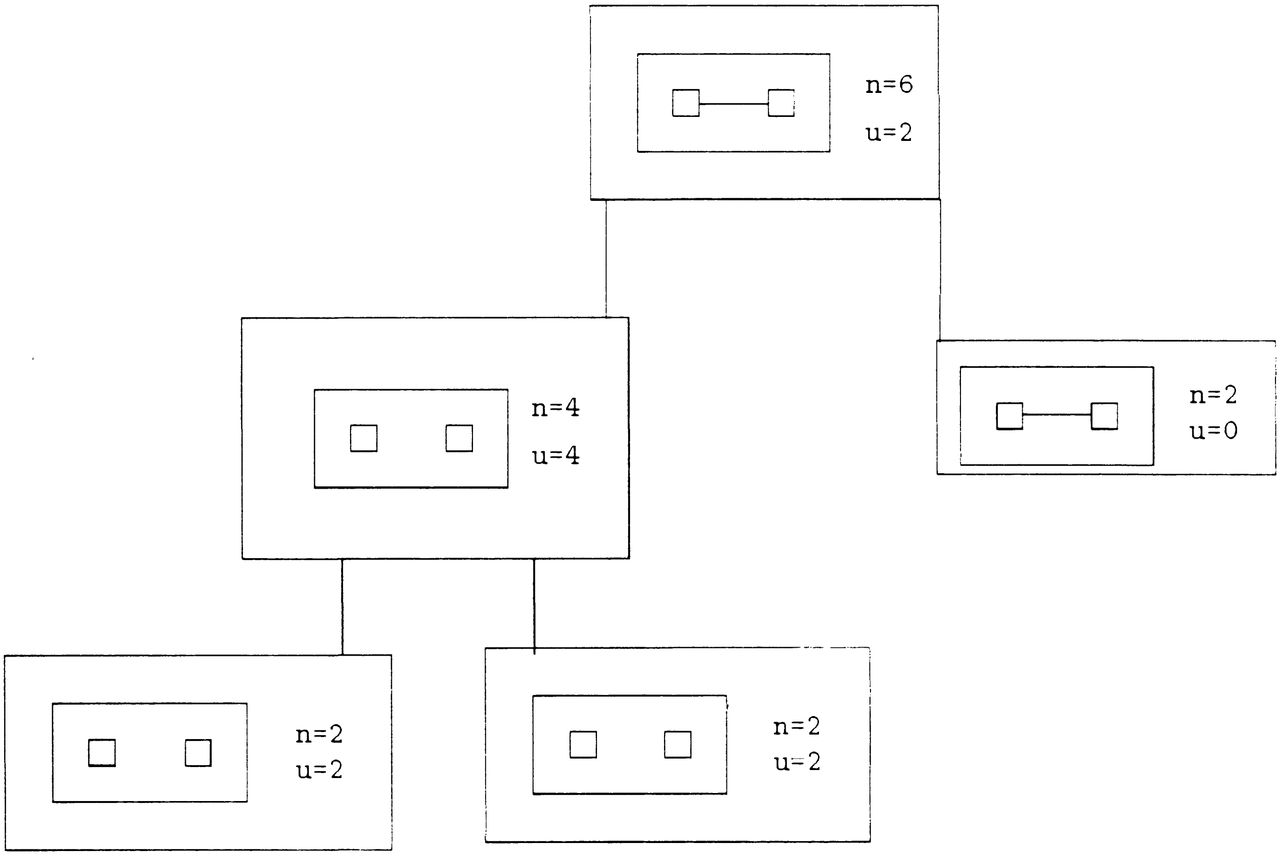


Figure 1: Decomposition example



An example for graph matching

Figure 2: Finding maximum matching

References

[1] S. Arnborg, J. Lagergren, and D. Seese. Which problems are easy for tree-decomposable graphs? In *ICALP*, 1988.

[2] M. W. Bern, E. L. Lawler, and A. L. Wong. Why certain subgraph computations require only linear time. In *26th FOCS*, pages 117–125, 1985.

[3] D.G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Disc. Appl. Math*, 3:163–174, 1981.

[4] Tibor Gallai. Transitiv orientierbare graphen. *Acta Math. Acad. Sci. Hungar.*, 18:25–66, 1967.

[5] Xin He and Yaacov Yesha. Binary tree algebraic computation and parallel algorithms for simple graphs. *J. of Algorithms*, 9:92–113, 1988.

[6] David Helmbold and Ernst Mayr. Perfect graphs and parallel algorithms. In *1986 International Conference on Parallel Processing*, pages 853–860. IEEE, 1986.

[7] Dexter Kozen, Umesh V. Vazirani, and Vijay V. Vazirani. NC algorithms for comparability graphs, interval graphs, and unique perfect matchings. Technical Report TR 86-799, Department of Computer Science, Cornell University, December 1986.

[8] Gary L. Miller and John H. Reif. Parallel tree contraction and its application. In *26th FOCS*, pages 478–489, 1985.

[9] R. H. Möhring and F. J. Rademacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. In R. E. Burkhard, R. A. Cunninghame-Green, and U. Zimmermann, editors, *Algebraic and combinatorial methods in operations research*. North-Holland, 1984.

[10] Rolf H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In Ivan Rival, editor, *Graphs and Order*, pages 41–101. D. Reidel, 1984.

[11] John H. Muller and Jeremy Spinrad. Incremental modular decompostion. *J. of Assoc. Comput. Mach.*, 36:1–19, 1989.

[12] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1986.

[13] Y. Shiloach and U. Vishkin. An $o(\log n)$ parallel connectivity algorithm. *J. of Algorithms*, 3:57–63, 1982.