Project No. E-24-618 (R5969-0A0)

Project Director: C. A. Tovey

Sponsor: National Science Foundation

Type Agreement: Grant ECS-8451032

Award Period: From 7/1/85 To 12/31/85 (Performance) 3/31/87 (Reports)

Sponsor Amount:

- Estimated: $
- Funded: $

This Change

Total to Date

- $ 125,000 **
- $ 25,000

Cost Sharing Amount: $

Cost Sharing No: E-24-323

Title: Presidential Young Investigator Award

ADMINISTRATIVE DATA

1) Sponsor Technical Contact:

Michael P. Polis
National Science Foundation
ENG/ECSE
Washington, DC 20550
202/357-9618

Defense Priority Rating: N/A

2) Sponsor Admin/Contractual Matters:

Joe Carrabino
National Science Foundation
EGC/ENG
Washington, DC 20550
202/357-9602

Military Security Classification: N/A

(or Company/Industrial Proprietary: N/A)

REQUIREMENTS

See Attached Supplemental Information Sheet for Additional Requirements.

Travel: Foreign travel must have prior approval -- Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

Equipment: Title vests with

COMMENTS:

*Includes a six month unfunded flexibility period.

No funds may be expended after 12/31/86.

**NSF expects to continue support of this project for an additional 4 years.

SPONSOR'S I.D. NO. 02.107.000.85.1226

COPIES TO:

- Research Administrative Network
- Research Security Services
- Research Property Management
- Accounting

GTRC
Library
Project File
Other A. Jones
GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

---

Closeout Notice Date 09/30/91

Project No. E-24-618

Center No. R5969-0A0

Project Director TOVEY C A

School/Lab ISYE

Sponsor NATL SCIENCE FOUNDATION/GENERAL

Contract/Grant No. DDM-8451032

Contract Entity GTRC

Prime Contract No.

Title PYI AWARD: COMPUTATIONAL COMPLEXITY AND RESCHEDULING ALGORITHMS

Effective Completion Date 910630 (Performance) 910930 (Reports)

---

Closeout Actions Required:

<table>
<thead>
<tr>
<th>Action</th>
<th>Y/N</th>
<th>Date Submitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Invoice or Copy of Final Invoice</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Final Report of Inventions and/or Subcontracts</td>
<td>Y</td>
<td>910920</td>
</tr>
<tr>
<td>Government Property Inventory &amp; Related Certificate</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Classified Material Certificate</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Release and Assignment</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Comments: BILLING VIA NSF LINE OF CREDIT. 98A SATISFIES REQUIREMENT FOR PATENT.

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

<table>
<thead>
<tr>
<th>Distribution Required</th>
<th>Y/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Director</td>
<td>Y</td>
</tr>
<tr>
<td>Administrative Network Representative</td>
<td>Y</td>
</tr>
<tr>
<td>GTRI Accounting/Grants and Contracts</td>
<td>Y</td>
</tr>
<tr>
<td>Procurement/Supply Services</td>
<td>Y</td>
</tr>
<tr>
<td>Research Property Management</td>
<td>Y</td>
</tr>
<tr>
<td>Research Security Services</td>
<td>N</td>
</tr>
<tr>
<td>Reports Coordinator (OCA)</td>
<td>Y</td>
</tr>
<tr>
<td>GTRC</td>
<td>Y</td>
</tr>
<tr>
<td>Project File</td>
<td>Y</td>
</tr>
<tr>
<td>Other</td>
<td>N</td>
</tr>
</tbody>
</table>

NOTE: Final Patent Questionnaire sent to PDPI.

---
GEORGIA INSTITUTE OF TECHNOLOGY  
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT (SUBPROJECTS)

Closeout Notice Date 09/30/91

<table>
<thead>
<tr>
<th>Project No.</th>
<th>Sponsor</th>
<th>School/Lab</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-24-618</td>
<td>NATL SCIENCE FOUNDATION/GENERAL</td>
<td>ISYE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project #</th>
<th>PD</th>
<th>Unit</th>
<th>MOD#</th>
<th>Code</th>
<th>Start</th>
<th>End</th>
<th>Funded</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-37-625</td>
<td>KERTZ R P</td>
<td>02.010.144</td>
<td>04</td>
<td>MATH *</td>
<td>850701</td>
<td>881231</td>
<td>6,316.00</td>
</tr>
</tbody>
</table>

LEGEND
1. * indicates the project is a subproject.
2. I indicates the project is active and being updated.
3. A indicates the project is currently active.
4. T indicates the project has been terminated.
5. R indicates a terminated project that is being modified.
PROGRESS REPORT      JUNE, 1987

CRAIG A. TOVEY

PRESIDENTIAL YOUNG INVESTIGATOR AWARD

Grant No. ECS-8451032
My research in the past year falls into three categories: (1) algorithms; (2) scheduling in automated manufacturing systems; (3) complexity analysis.

(1) I have extended the simulated annealing method in a rigorous way to nonlinear continuous problems (this is joint work with Robert Kertz). We are now writing up the convergence proofs and doing the computational testing. Also in this category, I have developed algorithms and lower bound analysis for the problem of finding a local optimum in an arbitrary discrete structure. The results from this work are to appear in the American Mathematical Monthly, and submitted to Discrete Applied Mathematics, (see appendices 1 and 2).

(2) I completed the deterministic analysis of the station routing problem, which arises in printed circuit board automated assembly, and more generally in jobshops using process planning (see appendix 3). I am now working on a harder related problem in multiple fixture assembly, and on the probabilistic analysis of the first problem.

(3) I have tried to resolve the complexity of characterization of the matrix classes K and Q in linear complementarity theory. These two characterization problems are generally considered to be the major open questions in LCP theory. So far I have identified possible candidates for the appropriate completeness level, and found some classes of matrices that are in Q except for an arbitrarily small region. These classes could be useful in constructing polynomial transformations. This complexity analysis continues as an important goal.

I have also worked on applying complexity theory to social choice methods. It turns out that some of the Arrow-type impossibility theorems can be circumvented by use of computational complexity (see appendices 4, 5, submitted to Social Choice and Welfare).
LOCAL OPTIMIZATION ON GRAPHS

Donna Crystal Llewellyn*
Craig Tovey**
Michael Trick

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0205

The complexity of finding local optima is an open problem for many neighborhood structures. We show how to derive close lower and upper bounds on the minimum number of function evaluations needed to find a local optimum in an arbitrary graph. When these bounding techniques are applied to the hypercube, the results give insights into the class PLS and the gap between the average and worst case behavior of local search.

*Supported by NSF Postdoctoral Fellowship
**Research supported by NSF Grant No. ECS 8451032
1. Introduction

Finding local optima is at the heart of most heuristic algorithms for difficult combinatorial optimization problems. Hence, the efficiency of these heuristics is heavily dependent upon the effectiveness of the local optimization procedures. However, the complexity of finding local optima remains an open problem for most neighborhood structures. A classical example of this anomaly occurs with the traveling salesman problem. The "2-opt" neighborhood structure is very popular, but, as [JPY] point out, the complexity of finding a "2-optimal" tour is not known. Further, the most widely used local optimization method, local improvement, has been shown in certain cases to have exponential worst case behavior [L].

In this paper we study the complexity of finding a local optimum of an arbitrary function, \( f \), over an arbitrary neighborhood graph. As in [HK,NW] our computational model employs a (not necessarily compactly representable) oracle to compute the values of \( f \). We consider several simple structures — a path, a grid and a hypercube. The analysis of these structures is nontrivial and the results are often counterintuitive. For example, we show that the problem of finding an entry in a square matrix that is minimum in its row and column requires, asymptotically, the examination of between \( 1/2 \) and \( 2/3 \) of the matrix entries. Further, for a \( d \)-dimensional cube, at least \( 2^{d/2\sqrt{d}} \) vertices must be examined.

We find it interesting that the problem of determining the complexity of local optimization proves so rich even for such regular structures. Moreover, the analysis here yields insight into the class
PLS (Polynomial time local search -- the class of local optimization problems, roughly speaking, where the first two iterations of the local improvement algorithm are guaranteed to be polynomial time (see [JPY])) and the relationship between worst and average case performance of local improvement algorithms. This wealth of problems arising from "simple" graphs leads us to study the general question of discrete local optimization in some detail.

Our analysis uses various techniques. We show how to derive close lower and upper bounds on the minimum number of function evaluations needed to find a local optimum in a graph. The upper bounds arise from a general divide and conquer algorithm while the lower bounds result from an adversarial argument that uses a structure similar to a spanning tree. Determining each of these bounds is ultimately tied to computing the value of a separation game on the graph of neighbors. Unfortunately, we also prove that this value is NP-hard to compute exactly. We discuss different ways to get effective estimates of this value in certain cases.

The paper is organized as follows: in Section 2 we derive the divide and conquer algorithm and the adversarial argument, using the path as a motivating example. Then we define the separation game. In Section 3 we analyze this game for a matrix. Section 4 contains the proof that determining the value of the game in general is NP-hard as well as a discussion of our methods for determining estimates of this value. In Section 5 we analyze the special case of the d-dimensional hypercube and investigate the implications and relations of this analysis to the class PLS, the expected performance of local improvement algorithms, and other results. We conclude with some remarks in Section 6.
2. Adversarial Divide and Conquer

2.1 Divide and Conquer Algorithm

In a typical combinatorial optimization problem, each instance has a finite set of feasible solutions and each solution is assigned a subset of these as its "neighbors." A (strict) local minimum has an objective function value which is less than or equal to (strictly less than) the values of each of its neighbors. We represent the solutions as vertices in a graph, \( G = (V, E) \), and place edges between each vertex and its assigned neighbors, and define a local minimum to be a vertex \( v \) such that \((v,w) \in E \implies f(v) \leq f(w)\), where \( f : V \to \mathbb{R} \) is the objective function to be minimized. Given a function evaluation oracle for function \( f \) we are interested in determining the number of calls to the oracle needed to find a local optimum of \( f \) in \( G \). We will assume, without loss of generality, that we are looking for a local minimum.

The most widely applied solution method is local search, using either a best or better neighbor selection rule (see, e.g., [PS.T2]). As an example, let the graph \( G = (V,E) \) be a simple path; that is, \( V = \{v_1, \ldots, v_n\} \) and \( E = \{(v_i, v_{i+1}) | 1 \leq i < n\} \). It is clear that any variation of local search can require asking the function value of \( O(n) \) vertices. However, a divide and conquer strategy can find a local optimum with only \( O(\log n) \) queries. To see this, first ask the function value of \( v_{\lfloor n/2 \rfloor} \), followed by its neighbors, \( v_{\lfloor n/2 \rfloor - 1} \) and \( v_{\lfloor n/2 \rfloor + 1} \). If \( f(v_{\lfloor n/2 \rfloor}) \) is the minimum of these three values, then we may stop. Otherwise, a local minimum is located within the half of the path which contains the smaller valued neighbor. Iterate this procedure on this (smaller) graph. Clearly, this will require at most \( 3\log n \) queries.
We now generalize the above divide and conquer scheme to all graphs. Whereas in the path example a single vertex sufficed to split the graph into smaller cases, in general we will need a more complex separator set. This is the only change in the procedure below.

Algorithm Divide and Conquer

Input: Graph $G = (V,E)$ and function $f: V \rightarrow \mathbb{R}$.
Output: $v^* \in V$ such that $f(v^*) \leq f(w)$ for all $(v^*,w) \in E$.

Step 0: Let $i = 0$, $G_0 = G$.

Step 1: Select vertices one at a time to be submitted to the oracle until this collection of vertices separates $G_1$. Call these vertices $S$.

Step 2: Find $v' \in S$ that minimizes $f(v)$ for all $v \in S$, breaking ties arbitrarily.

Step 3: Query the neighbors of $v'$. Call these vertices $N$.

Step 4: If $f(v') \leq f(w)$ for all $w \in N$, return $v^* = v'$ and stop.

Step 5: Let $w \in N/S$ be such that $f(w) < f(v')$. Let $G_{i+1}$ be the connected component of $(G_1 \backslash S)$ containing $w$ and let $i = i + 1$.

Step 6: Go to 1.

Lemma 2.1: Algorithm Divide and Conquer finds a local minimum of $f$ in $G$.

Proof: If local search were carried out starting with $w$ (chosen in Step 5) then it could never select a vertex in $S$ because $f(w) < f(v') \leq f(u)$ for all $u \in S$. Hence the algorithm could never leave $G_i$ and so must find a local minimum in $G_i$. Thus there exists a local minimum in the component chosen in Step 5 and the desired result now follows immediately. 

Let the number of queries required by our divide and conquer algorithm for graph G and function f be denoted $dc(G, f)$. It is clear that $dc(G, f)$ provides an upper bound on the number of queries needed by any method to solve this problem. Given function f, let $L(G, f, A)$ be the minimum number of queries that algorithm A requires to find a local optimum of f in graph G. Then let $L(G, A) = \max_f L(G, f, A)$, and

$L(G) = \min_A L(G, A)$. Similarly, let $dc(G) = \max_f dc(G, f)$. It turns out that in general, $dc(G)$ provides a very close upper bound on $L(G)$. To see this it is first necessary to turn to the problem of determining a lower bound on $L(G)$. We will show that this lower bound is usually close to $dc(G)$ and hence it must follow that in these cases $dc(G)$ is close to $L(G)$.

2.2 An Adversarial Argument

Suppose the function evaluation oracle is an adversary who attempts to stymie the optimizer. This adversary will provide function values that cause the person trying to find a local minimum to continue to query for a long time. If the oracle can force the minimizer to ask for $g(n)$ values, then $g(n)$ must be a lower bound on the worst case number of queries for the graph. We give below a specific adversarial strategy that we will apply to get a lower bound.

First we show how the adversary would work on the path example. When the optimizer queries a vertex, v, this vertex automatically splits the path into two subpaths, say $P_a$ and $P_b$, each containing v. The oracle then selects one of these, say subpath $P_a$, as the part of the graph where the local optimum will be located. In choosing $P_a$, she has decided not to have any local minima of f within $P_b$. To insure this, she sets
function values for each of the nodes in $P_b$ that decrease along the path to $v$. Now whenever the optimizer queries a node in $P_b$, the oracle simply responds with the preset value; thus the optimizer is wasting time with any queries within $P_b$. For any queries in $P_a$, the oracle can now repeat this process since any selected vertex in $P_a$ breaks this path into two subpaths.

In this example, it is a simple exercise to see that the adversarial strategy gives a bound of $\lg n$ queries for any choice of first vertex. This follows from the observation that the oracle will pick $P_a$ so that it is at least as long as $P_b$. Hence, $P_a$ will have at least $\lceil n/2 \rceil$ vertices. Iterating this process clearly gives a lower bound of $\lceil \lg n \rceil$ queries.

To generalize the path example, note that as long as the graph is connected, the oracle can form a "valley" whose deepest point is at the chosen minimum. When the graph becomes disconnected, this valley must be generalized to a network of deepening ravines. This is the core of our adversarial strategy given below. We need the following definition. A bare tree of $G$ is an acyclic connected subgraph, $T$, of $G$ such that all vertices of $G$ are in or adjacent to $T$. (A bare tree is a spanning tree without its leaves, hence the name.) Then our strategy follows.

Adversarial Strategy

Step 0: Let $i = 0$ and $G_0 = G$. Let $w_0$ be any integer.

Step 1: Give the default value $w_i$ to any vertex asked until there is no bare tree of $G_i$ using only unqueried vertices.

Let the last vertex queried be $v_i$. 
Let the newly formed components of the subgraph induced by the unqueried vertices be denoted \( C_{1i}, \ldots, C_{ik} \). In \( G_i \), \( v^i \) is adjacent to each of these components.

**Step 2:** Choose some \( j, 1 \leq j \leq k \).

In \( C_{1r} \), \( r \neq j \), choose a bare tree \( T_r \).

For vertex \( u \notin C_{ij}, u \notin T_r \) for any \( r \), and \( u \) not queried, let
\[
f(u) = w^i_r.
\]

Along each branch of each \( T_r \), pick function values less than \( w^i_r \) decreasing toward \( v^i_r \).

Let \( f(v^i_r) = \min\{f(u) | u \in C_{ik} \} - 1 \).

Let \( w^i_r = f(v^i_r) - 1 \).

**Step 3:** Let \( i = i + 1 \). Let \( G_{i} = C_{i-1,j} \).

**Step 4:** Go to Step 1.

**Lemma 2.2:** If the oracle responds according to the strategy above, then as long as \( G \) has more than one vertex, it cannot be determined where \( G \) has a local minimum.

**Proof:** This follows directly from our comment above that as long as a graph is connected, the oracle can devise a network of deepening ravines toward any selected vertex. \( \Box \)

**2.3 The Separation Game**

In order to obtain a good lower bound on \( L(G) \) from Lemma 2.2 we need to analyze the adversarial strategy. It is interesting that analyzing this strategy is very similar to analyzing the divide and conquer algorithm. This is because each procedure has the following two person game embedded within its structure.
Separation Game

Input: Graph $G = (V, E)$

Two Players: minimizer I
          maximizer II

Description: Player I removes vertices from $C$ until $C$ is disconnected and then passes play to II. Player II chooses one of the newly created components to call $G$, discards the other components and passes $G$ back to I. The game ends when $|V| \leq 1$.

Step 1: $i = 0$, $V^0 = V$; score $(C) = 0$.
Step 2: if $|V^i| \leq 1$ go to Step 6.
Step 3: Player I chooses $S^i \subseteq V^i$ such that $G^i/S^i$ is not connected or is the empty graph; score $(C) = score (G) + |S^i|$.
Step 4: Player II chooses $G^{i+1}$, a connected component of $G^i/S^i$; $i = i + 1$.
Step 5: Go to Step 2.
Step 6: STOP.

We define the value of the separation game on a graph $G$, denoted $v(G)$, to be score $(C)$ when each player plays optimally. I to minimize score $(G)$ and II to maximize.

Lemma 2.3: The value of the separation game provides a lower bound on the amount of work required by any algorithm to find a local minimum on graph $G$; while the amount of work required by algorithm divide and conquer provides an upper bound.
**Proof**: This lemma is equivalent to claiming that \( v(G) \leq L(G) \leq dc(G) \).

The second inequality follows from Lemma 2.1 and the definitions. Further, in any application of the adversarial strategy described in Section 2.2, the oracle can select the component of \( G \) that is chosen in the optimal play of the separation game. Hence, \( v(G) \) provides a lower bound on the number of function queries forced by the oracle in the adversarial strategy, which is less than \( L(G) \) by Lemma 2.2.

This lemma shows that if the gap between \( v(G) \) and \( dc(G) \) is small then either one will provide a good estimate of \( L(G) \). In the path example, we saw that one implementation of the divide and conquer algorithm (choosing the midpoint of the path at each iteration) gives a \( 3\log n \) upper bound on \( dc(G) \) and that the adversarial strategy gives a lower bound of \( \log n \). The difference between these two can be viewed in two ways. First, the upper bound is three times the lower bound; second, the gap is logarithmically small. In general, the latter view is the correct one. In particular, the gap is equal to the maximum degree of a vertex \( (2) \) times the number of times Step one is needed (\( \log n \)). Actually, it is foolish for the adversary to select either subpath \( P_a \) or \( P_b \) immediately. Rather, she should wait for the next query and then select the subpath that does not contain this queried vertex. Hence the optimal optimizer strategy is not the naive one of picking the midpoint, but rather to slightly unbalance the two subpaths by doing Fibonacci search. It is straightforward to prove that this gives the optimal value of the separation game to be \( \log_\phi n + 2 \); where \( \phi \) is the golden mean \( (\sqrt{5} - 1)/2 \) and the two arises from the extra queries to test the local optimality of the final selection.
In the next section we investigate another example where we are able to establish fairly tight bounds on \( L(G) \). We will use the following result.

**Proposition 2.4:** In the separation game \( S^i \) can always be taken to be a minimal separating set of \( G \). That is, for any \( S \subseteq S^i \), \( G^i/S \) is connected.

**Proof:** The removal of any nodes not in a minimal separating set can be delayed one iteration, at which time they can be removed (if in the component chosen by player II) keeping score \( (G) \) the same, or ignored (otherwise) decreasing score \( (G) \).

\[ \square \]

3. **Finding a row-column minimum in a matrix**

We now apply the ideas of Section 2 to a specific local optimization problem. Suppose \( A \) is an \( nxn \) matrix: how hard is it to find an entry that is less than or equal to every other entry in its row and column? It is clear that one can play the adversary against a local improvement algorithm in such a way to force every entry of the matrix to be queried. We show that asymptotically, one needs only to query between one half and two thirds of the matrix, and we conjecture that the upper bound of \( 2/3 \) is asymptotically tight.

Following our development in Section 2, the graph of neighborliness, \( G \), is formed by defining a node for each matrix entry with an edge between two nodes if and only if they are either in the same row or column. Using Lemma 2.3, we will attempt to find \( v(G) \) in order to obtain a good estimate of \( L(G) \). However, \( G \) is dense and difficult to visualize. Therefore, we move to an auxiliary graph. Consider the complete bipartite graph \( H = (R \cup C, E) \), where \( R = \{1, \ldots, n\} \) and \( C = \{1, \ldots, n\} \).
In this graph we have one node for each row and for each column, with an edge connecting each row and column pair. Here, each edge corresponds to an element in the matrix, and so we move to an edge version of our separation game where player I removes edges from the auxiliary graph rather than deleting vertices from the original graph. Other than this change, the game is played in the same manner as the original (vertex) separation game.

**Proposition 3.1:** The value of the (vertex) separation game played on the original graph is the same as the value of the edge separation game played on the auxiliary graph.

**Proof:** We noted above that edges in the auxiliary graph correspond to vertices of the original graph. This implies that a separating set of vertices in $G$ is a disconnecting collection of edges in $H$. Further a disconnecting set of edges in $H$ must correspond to a separating set of vertices in $G$ unless one of the resulting components in $H$ is an isolated vertex. However, this case will never arise in our application since player II would never choose such a component before the last iteration of the game.

We now analyze this game. For ease of presentation, we will relax the assumption that $|R| = |C|$: but will always assume that $m = |R| \leq |C| = n$. Suppose that player I chooses subset $F$ of the edges to separate graph $H = (R \cup C, E)$. By Prop. 2.1, we can assume that $F$ is a minimal separating set. Then clearly, since $F$ is a set of edges, it must disconnect $H$ into exactly two components, each a complete bipartite graph. The components can be represented by $(J,K)$ and $(m-J,n-K)$, where
in each case the ordered pair corresponds to the number of row vertices and the number of column vertices in that subgraph. Clearly, player II will return to player I the component which requires more work to finish disconnecting. Let \( f(m,n) \) be the value of the edge separation game on the complete bipartite graph \( H \). Where for general \( r \) and \( s \) \( f(r,s) \) is to read as \( f(\min(r,s),\max(r,s)) \). The discussion above implies

\[
\begin{align*}
f(m,n) &= \min_{1 \leq J \leq m} \min_{1 \leq K \leq n} \left\{ \left[ \max(f(J,K),f(m-J,n-K)) \right] + (n-K) + K(m-J) \right\} \\
&= \min_{1 \leq J \leq m} \min_{K(J,m) \leq K \leq n} \left\{ f(J,K) + J(n-K) + K(m-J) \right\}.
\end{align*}
\]

To simplify this, let \( K(J,m) \) be the smallest integer such that

\[ f(J,K(J,m)) \geq f(m-J,n-K(J,m)). \]

Thus, we can assume without loss of generality that \( F \) is chosen in such a way that \( K \geq K(J,m) \) and hence that player II returns the \((J,K)\) component. This implies

\[
\begin{align*}
f(m,n) &= \min_{1 \leq J \leq m} \min_{K(J,m) \leq K \leq n} \left\{ f(J,K) + J(n-K) + K(m-J) \right\} \\
&= \min_{1 \leq J \leq \left\lfloor \frac{m}{2} \right\rfloor} \min_{K(J,m) \leq K \leq n} \left\{ f(J,K) + Jn + K(m-2J) \right\}.
\end{align*}
\]

However, if we begin with \( m \leq n \) then in order to maintain this condition of no more row vertices than column vertices, we must assume that

\[ J \leq \left\lfloor \frac{m}{2} \right\rfloor. \]

Now it is clear that for \( J > \left\lfloor \frac{m}{2} \right\rfloor \), switching the roles of \( R \) and \( C \) will give a symmetric argument. Hence we now assume \( J \leq \left\lfloor \frac{m}{2} \right\rfloor \), giving

\[
\begin{align*}
f(m,n) &= \min_{1 \leq J \leq \left\lfloor \frac{m}{2} \right\rfloor} \min_{K(J,m) \leq K \leq n} \left\{ f(J,K) + Jn + K(m-2J) \right\}.
\end{align*}
\]
Further, it follows from the definition that for $J \leq \left\lfloor \frac{m}{2} \right\rfloor$, $f(J,K)$ is nondecreasing in $K$. Hence,

$$f(m,n) = \min_{1 \leq J \leq \left\lfloor \frac{m}{2} \right\rfloor} \{f(J,K(J,m))+Jn+K(J,m)(m-2J)\}$$

We now need to determine some effective bounds on $K(J,m)$.

**Prop. 3.2:** If $m \leq m'$ and $n \leq n'$ then $f(m,n) \leq f(m',n')$.

**Proof:** This is clear by definition. □

**Prop. 3.3:** $K\left(\left\lfloor \frac{m}{2} \right\rfloor, m \right) \geq \left\lfloor \frac{n}{2} \right\rfloor$

**Proof:** By definition, $f(\left\lfloor \frac{m}{2} \right\rfloor, k(\left\lfloor \frac{m}{2} \right\rfloor, m) \geq f(\left\lfloor \frac{m}{2} \right\rfloor, n-k(\left\lfloor \frac{m}{2} \right\rfloor, m))$. There are two cases.

1. $\left\lfloor \frac{m}{2} \right\rfloor \leq n-K(\left\lfloor \frac{m}{2} \right\rfloor, m)$
   - Now, $\left\lfloor \frac{m}{2} \right\rfloor \leq \left\lfloor \frac{n}{2} \right\rfloor$ by Prop. 3.2 that $K(\left\lfloor \frac{m}{2} \right\rfloor, m) \geq n-K(\left\lfloor \frac{m}{2} \right\rfloor, m)$
   - $\Rightarrow K(\left\lfloor \frac{m}{2} \right\rfloor, m) \geq \frac{n}{2}$.

2. $\left\lfloor \frac{m}{2} \right\rfloor \geq n-K(\left\lfloor \frac{m}{2} \right\rfloor, m)$
   - Then, by definition $f(\left\lfloor \frac{m}{2} \right\rfloor, K(\left\lfloor \frac{m}{2} \right\rfloor, m) \geq f(n-K(\left\lfloor \frac{m}{2} \right\rfloor, m), \left\lfloor \frac{m}{2} \right\rfloor)$
   - a) $n-K(\left\lfloor \frac{m}{2} \right\rfloor, m) \leq \left\lfloor \frac{m}{2} \right\rfloor \Rightarrow K(\left\lfloor \frac{m}{2} \right\rfloor, m) \geq n-\left\lfloor \frac{m}{2} \right\rfloor \geq \frac{n}{2}$
   - b) $n-K(\left\lfloor \frac{m}{2} \right\rfloor, m) > \left\lfloor \frac{m}{2} \right\rfloor$. Then, by Prop. 3.2, $K(\left\lfloor \frac{m}{2} \right\rfloor, m) \geq \left\lfloor \frac{m}{2} \right\rfloor \Rightarrow n-K(\left\lfloor \frac{m}{2} \right\rfloor, m) < \left\lfloor \frac{m}{2} \right\rfloor$ which is a contradiction.

In each of these cases, the integrality of $K(J,m)$ gives the desired result. □

Using a similar argument, one can also prove the following upper bound.

**Prop. 3.4:** $K\left(\left\lfloor \frac{m}{2} \right\rfloor, m \right) \leq \left\lfloor \frac{n}{2} \right\rfloor + 1$ □
Corollary 3.5: If $m$ is even then $K\left(\frac{m}{2}, m\right) = K\left(\frac{m}{2}, m\right) = \left\lceil \frac{n}{2} \right\rceil$.

Proof: Using Prop. 3.2 and Prop. 3.3, clearly $\frac{n}{2} \leq K\left(\frac{m}{2}, m\right) \leq \left\lceil \frac{n}{2} \right\rceil + 1$.

Consider the case when $n$ is odd. This gives $\frac{n}{2} \leq K\left(\frac{m}{2}, m\right) \leq \left\lceil \frac{n}{2} \right\rceil$ and we are done.

Now consider when $n$ is even. In this case, the two components resulting from the removal of $F$ are $\left[\frac{m}{2}, K\left(\frac{m}{2}, m\right)\right]$ and $\left[\frac{m}{2}, n - K\left(\frac{m}{2}\right)\right]$. Clearly $\frac{n}{2}$ will suffice to make the first subgraph at least as hard to finish as the second and hence by minimality of $K(J, m)$ we must have $K\left(\frac{m}{2}, m\right) = \frac{n}{2} = \left\lceil \frac{n}{2} \right\rceil$.

Now, we can use these bounds to determine bounds on $f(m, n)$.

Prop. 3.6: The value of the edge separation game played on the complete $m \times n$ bipartite graph is greater than $\left\lceil \frac{mn}{2} \right\rceil$.

Proof: To prove this we will show that $f(m, n) > \left\lceil \frac{mn}{2} \right\rceil$. By definition,

$$f(m, n) = \min_{1 \leq J \leq \left\lfloor \frac{m}{2} \right\rfloor} \{f(J, K(J, m)) + J(n - K(J, m)) + K(J, m)(m - J)\}$$

$$= \min_{1 \leq J \leq \left\lfloor \frac{m}{2} \right\rfloor} \{f(J, K(J, m)) + Jn + K(J, m)(m - 2J)\}$$

Note that for all $J, K > 0$, $f(J, K) \geq 1$, hence,

$$f(m, n) \geq 1 + \min_{1 \leq J \leq \left\lfloor \frac{m}{2} \right\rfloor} \{K(J, m)(m - 2J) + Jn\}$$

It is clear that $K(J, m)$ is nonincreasing in $J$, hence.
\[ f(m,n) \geq 1 + \min_{1 \leq J \leq \left\lfloor \frac{m}{2} \right\rfloor} \left\{ K\left(\left\lfloor \frac{m}{2} \right\rfloor, m\right) \left(m-2J\right) + Jn \right\} . \]

Using Prop. 3.3 we get,

\[ f(m,n) \geq \min_{1 \leq J \leq \left\lfloor \frac{m}{2} \right\rfloor} \left\{ 1 + \left\lceil \frac{n}{2} \right\rceil \left(m-2J\right) + Jn \right\} \]

\[ \geq 1 + \frac{n}{2} m + 1(\text{for } n=2) \]

\[ \geq 1 + \frac{mn}{2} \]

as desired. \(\Box\)

Applying our inductive hypothesis, it is straightforward to obtain effective upper bounds on the value of the edge separation game in some special cases.

**Prop. 3.7** If \(m\) and \(n\) are powers of 2, with \(m \leq n\), then the value of the edge separation game played on the complete \(m \times n\) bipartite graph has an upper bound of \(\left\lfloor \frac{2}{3}mn + \frac{n}{3m} \right\rfloor\).

**Prop. 3.8:** The bound given in Proposition 3.3 does not necessarily hold if \(m\) or \(n\) is not a power of 2.

**Proof:** First, consider \(f(2,3)\). From Prop. 3.2, \(K(1,2) \geq \left\lfloor \frac{n}{2} \right\rfloor\). Hence here, \(K(1,2) \geq \left\lfloor \frac{3}{2} \right\rfloor = 2\). Clearly \(f(1,n) = n\) for all \(n \geq 1\). Thus,

\[ f(2,3) = f(1,2) + 1(3-2) + 2(2-1) \]

\[ = 2 + 1 + 2 = 5 \]
But, \[ \frac{2}{3}mn + \frac{n}{3m} = \frac{2}{3} \cdot 6 + \frac{3}{6} = 4 < 5. \]

Next, consider \( f(3,16) \). It is straightforward to prove that \( k(1,3) \geq \left\lceil \frac{3}{5} n \right\rceil \). Hence, here \( K(1,3) \geq 10 \).

Thus, \( f(3,16) \geq f(1,10) + 1(16-10) + 10(3-1) \)
\[ = 10 + 6 + 20 = 36. \]

But, \[ \frac{2}{3}mn + \frac{n}{3m} = \frac{2}{3} \cdot 48 + \frac{16}{9} = 33 < 36. \]

However, with careful application of divide and conquer, and induction, we can obtain an upper bound on the value for all \( m \) and \( n \).

The proof entails several cases. The nature of the proof in each individual case depends only on whether certain components (arising from the divide and conquer status) have an even number of vertices.

**Prop. 3.9:** For all \( m \) and \( n \), with \( m \leq n \), the value of the edge separation game played on the complete \( m \times n \) bipartite graph has an upper bound of \( \frac{2}{3}mn + \frac{n}{3} \).

**Proof:** We will show that \( f(m,n) \leq \frac{2}{3}mn + \frac{n}{3} \) and the result will follow by integrality of \( f \).

We need to break this into several cases. The idea in each is the same. We try to decompose each vertex set of the bipartite graph into equal parts and analyze the total work involved. We will be getting an upper bound by analyzing the work needed to solve the problem using the following specific strategy: Let \( R = A \cup B \) and \( C = S \cup T \) (Hence \( |A| + |B| = m \) and \( |S| + |T| = n \)). Then we will first disconnect \( A \) from \( T \) which requires \( |A||T| \) edges removed, then we will disconnect \( B \) from \( S \) which requires \( |B||S| \) edges removed. Then we finish by choosing the component that requires the most work, among \((A,S)\) and \((B,T)\). The
total work needed then is $|A||T| + |B||S| + \max\{f(A,S),f(B,T)\}$. Using induction of $m$ and $n$ implies

$$f(m,n) \leq |A||T| + |B||S| + \max\left[\frac{2}{3}|A||S| + \frac{1}{3}|S|, \frac{2}{3}|B||T| + \frac{|T|}{3}\right]$$

Here, if one analyzes the cases $m$ and $n$ are both even, $m$ is odd and $n$ is even, $m$ is even and $n$ is odd, and $m$ and $n$ are both odd, using our results above, it is clear that the desired result will follow.

Putting together propositions 3.6 and 3.9, we have proved the following theorem.

**Theorem 3.10:** The value of the edge separation game played on a complete $m \times n$ bipartite graph has a lower bound of $\left\lfloor \frac{mn}{2} \right\rfloor$ and an upper bound of $(2 + o(1))mn$.

**Corollary 3.11:** The amount of work required to find an element that is smallest in its row and in its column in an $m \times n$ matrix, with $m \leq n$, is greater than $\left\lfloor \frac{mn}{2} \right\rfloor$. Further, our divide and conquer algorithm requires no more than $\frac{2mn + n}{3} + 2n\lg m$ work to solve this problem.

**Proof:** From Theorem 3.9, we see that $L(G) > \left\lfloor \frac{mn}{2} \right\rfloor$ for all $m \leq n$. The other result will follow from Theorem 4.7 in the next section.

**Conjecture 3.11:** $v(G) > \frac{2}{3}mn + \frac{n}{3m}$ for all $m \leq n$.

We have verified this conjecture for all $m \leq n \leq 64$.

4. **Computing the value of the Separation Game**

We have seen in Lemma 2.3 that the value of the separation game on $G$ is useful since it provides a lower bound on the number of function
evaluations required by any algorithm, and a fairly close estimate on the work required by the divide and conquer algorithm. However, we have also seen in Section 3 that analyzing even fairly simple graphs can be difficult. In this section, we establish the complexity of determining \( v(G) \) and we show how to obtain bounds on this value.

4.1 Complexity of Computation

Despite the apparent simplification provided by Prop. 2.4, it is not easy to determine Player I's optimal strategy. It seems plausible that Player I wishes to separate \( G^0 \) into roughly equal parts at minimum cost, for if \( G^0 \) were separated into unequal parts then Player II could simply choose the larger component. Unfortunately, the size of a component is a poor measure of the value of the game. For instance, the graphs \( P_n \) (a path on \( n \) vertices) and \( K_n \) (the complete graph on \( n \) vertices) have the same number of vertices yet the separation games on these graphs have very different values. To see this, from Section 2 we know that \( v(P_n) \leq \lg(n) \), while in contrast it is easy to establish that \( v(K_n) = n-1 \).

This complication is formalized by showing that calculating \( v(G) \) is NP-hard. To do this, we will show that to play optimally during the first two moves, Player I must solve an NP-hard problem. The precise objective for Player I's first move is to choose \( S^0 \) to minimize

\[ |S^0| + \max_i v(G^1_i) \]

where the \( G^1_i \) are the components of \( G^0 / S^0 \).

Notice that (assuming the complexity result is true) \( v(G^1_i) \) is hard to evaluate! This is one of those curious cases where the complexity of the problem fights against a proof of its complexity. To resolve this, we will find a structure such that the values of the graphs that result after the first few moves are known.
We need an intermediate result on a restricted version of vertex cover:

A tripartite graph is a graph $G = (V_1, V_2, V_3; E_{12}, E_{13}, E_{23})$ where $V_i$, $i=1,2,3$ are disjoint vertex sets and $(i,j) \in E_{ab} \Rightarrow i \in V_a$ and $j \in V_b$. That is, a tripartite graph is a graph with an explicit 3-coloring.

Recall that the vertex cover problem is to find a minimum cardinality subset of vertices of a graph which contains at least one endpoint of each edge.

**Theorem 4.1.** Vertex Cover restricted to tripartite graphs is NP-complete.

**Proof:** The reduction is a modification of the reduction of [GJ] from 3-SAT to Vertex Cover. Rather than 3-SAT, we will use another restricted form of SAT, proved NP-complete in [T1]:

SAT restricted by (1), (2), and (3) below is NP-complete:

1. Each clause has 2 or 3 literals;
2. Each variable appears in at most 3 clauses;
3. Each variable appears complemented exactly once.

If the reduction of [GJ] for 3-SAT to Vertex Cover is used with the above SAT restriction (where edges replace triangles for clauses with 2 literals) then a 3-colorable graph results. The 3-coloring can be found in a greedy fashion, provided the nodes are colored in the order: clause triangles, clause edges (using just colors 1 and 2), uncomplemented variables, and complemented variables.

We are about to prove that the separation game value is hard to calculate. Our construction makes use of a structure we call a **spider**. Define an $(h,w,t)$-spider to be a graph with vertex set $V =$
\{v_1, \ldots, v_h, u_1, \ldots, u_w, x_1, \ldots, x_t\} and edge set \(E = \{(v_i, v_j) : 1 \leq i < j \leq h\} \cup \{(u_i, u_j) : 1 \leq i < j \leq w\} \cup \{(x_i, x_j) : 1 \leq i < j \leq t\} \cup \{(v_i, u_j) : 1 \leq i \leq h, 1 \leq j \leq w\} \cup \{(u_i, x_j) : 1 \leq i \leq w, 1 \leq j \leq t\}.

So a spider is three complete graphs \(K_h, K_w, K_t\), which we call the head, waist and tail respectively, where each node in the waist is connected to each node in the head and tail.

**Proposition 4.2.** If \(G\) is an \((h,w,t)\)-spider then the value of the separation game on \(G\) is equal to \(w + \max(h, t) - 1\).

**Proof:** The waist is the only minimal separating set so, by Proposition 2.4, Player I chooses it as the first move. This leaves \(K_h\) and \(K_w\).

Since \(v(K_h) = p - 1\), Player II will choose the larger. \(\square\)

**Theorem 4.3.** For a graph \(G\) and integer \(k\), determining if the value of the separation game \(v(G) \leq k\) is NP-complete.

**Proof:** The problem is in NP since we can completely specify Player I’s strategy with one label per node.

To show completeness, we will reduce from the result in Theorem 4.1. We will take three spiders and connect their heads with a “web” of arcs forming a tripartite cover instance. We then show that Player I’s optimal strategy is to separate the spiders and to use a single spider, for which we know the optimal strategy. Separating the spiders optimally involves solving the tripartite vertex cover problem.

Let \(G = (V_1, V_2, V_3, E_{12}, E_{13}, E_{23}; k)\) be an instance of vertex cover restricted to tripartite graphs. Set \(T = |V_1| + |V_2| + |V_3|\).

Create three spiders \(H^i, i=1,2,3\), where \(H^i\) is a \((|V_i| + T, k, 2T)\)-spider. Denote the first \(|V_i|\) vertices in the head of \(H^i, v^i_j\) for \(j=1,2\ldots,|V_i|\) for \(i=1,2,3\). Put edges between these vertices...
corresponding to $G$ and call this graph $H$.

We claim $v(H) \leq 2(T+k) - 1 \iff G$ has a vertex cover of size $\leq k$.

($\Leftarrow$) If $G$ has a vertex cover $C$, with $|C| \leq k$, then Player I can choose $S^0$ as the nodes corresponding to $C$. This disconnects $H$ into 3 spiders, each with a larger tail than head. By Proposition 4.2, each has value $k+2T-1$ so $v(H) \leq |S^0| + k + 2T - 1 = 2(T+k) - 1$ as needed.

($\Rightarrow$) By proposition 2.4, Player I's first move removes either

(i) a waist, leaving a component containing two connected spiders. This component requires at least $k+2T$ deletions because a spider has value $k+2T-1$ and at least one node is needed to disconnect the spiders. This gives $v(H) \leq 2(T+k)$. Contradiction.

(ii) nodes attached to the web, separating one spider from the other two. Two cases:

(a) if this separates all three spiders then, since a spider requires $k+2T-1$ deletions, no more than $k$ nodes separated the spider, which corresponds to a vertex cover of size $\leq k$.

(b) if two spiders are still connected, then Player II can pick that component. Player I's response is either a waist (which leads to the contradiction in (i)) or a separation of the spiders (which leads to the calculation in (iia)) hence proving the claim. $\Box$
4.2 Boundary Theorems and Lower Bounds

Theorem 4.2 implies that we cannot find the exact value of the separation game for an arbitrary graph by any known efficient algorithm. Hence, we turn to developing tools for estimating \( v(G) \); in this section we illustrate how to apply boundary theorems to obtain lower bounds on \( v(G) \).

**Theorem 4.4:** If the separation game is played on graph \( G = (V,E) \), then for all integers \( k \) and \( t \), \( 0 \leq t \leq k \leq |V| \). Player II can force player I to either

1) delete at least \( t \) vertices, or

2) create a set of components with total cardinality \( s \), with \( k-t \leq s \leq k \).

**Proof:** Consider the following strategy for Player II:

1) Let \( B=\emptyset \), \( P=G \).

2) Let Player I separate \( P \).

3) Pick an order of the resulting components, say \( P_1, P_2, \ldots, P_p \) and let \( P_0=\emptyset \) and \( P_{p+1}=G \).

4) Let \( i \geq 1 \) be such that \( |B \cup P_0 \cup P_1 \cup \ldots \cup P_{i-1}| \leq k \) and \( |B \cup P_0 \cup P_1 \cup \ldots \cup P_i| > k \).

5) Let \( B'=B \cup P_0 \cup P_1 \cup \ldots \cup P_{i-1} \). If \( k-t \leq |B'| \leq k \) go to 6; if \( i=p+1 \) go to 7; otherwise let \( P = P_i \), \( B=B' \) and go to 2.

6) Stop, \( B' \) is a subgraph of vertices for \( k-t \leq |B'| \leq k \).

7) Stop, Player I has chosen \( t \) vertices.

The termination in step 7 is correct due to the following:

By construction, at step 2 for each iteration \( |B| \leq k-t \leq |B| \) and, for \( i=k+1 \) in the final step 5, \( B' \leq k-t \). But \( P_0 \cup P_1 \cup \ldots \cup P_p = P \setminus S \), where \( S \)
is the separating set chosen by Player I in step 2. So
\[ |S| = |BUP| - |B'| > k - (k - t) = t. \] This means Player I has removed at least 
t vertices as required.

Let \( B(s) \) be the boundary of \( SCV \). That is,
\[ B(s) = \{ v \in V \setminus S : \exists e = (v, w), w \in S \}. \] By fixing \( t \), we get the following corollary:

**Corollary 4.5:** For any graph \( G \) and integer \( t \),
\[ v(G) \geq \min\{ t, \max_{k} \min_{|B(S)| : k - \leq |S| \leq k} |B(S)| \}. \]

**Proof:** From Theorem 4.4, for any \( k \), either \( t \) nodes are removed or a set of components with total cardinality \( s \), \( k - \leq s \leq k \), is created. To create such a set, the boundary of that set must have been among the nodes chosen by Player I. Hence, in this case, Player I removes at least \( \min\{ |B(S)| : k - \leq |S| \leq k \} \) nodes. Since Theorem 4.4 holds for all \( k \), the maximum over all of these values is a lower bound on the value of the game.

One useful value for \( t \) in the Corollary 4.5 is as follows. Let 
\[ \beta(G) = \max_{i} \min\{ |B(S)| : |S| = i \}. \] Setting \( t = \beta(G) \) gives:

**Corollary 4.6:** For any \( G \),
\[ v(G) \geq \min\{ \beta(G), \max_{k} \min_{|B(S)| : k - \leq |S| \leq k} |B(S)| \}. \]

### 4.3 Separator Theorems and Upper Bounds

The strategy in Theorem 4.4 also gives a feasible strategy for Player II in the divide and conquer game, except that this player must also examine the neighbors of one node in each separating set.

Therefore, for graphs where each node is of small degree, upper bounds on
v(G) provide useful upper bounds on L(G). Let \( \delta_{\text{max}}(G) \) be the maximum degree of any node in G and let \( K \) be the number of separating sets used by player I.

**Theorem 4.7:** \( v(G) \leq L(G) \leq v(G) + \delta_{\text{max}}(G)K \).

**Proof:** The first inequality comes from Lemma 2.3. The second inequality is a result of Player I using optimal separators for the separation game in a divide and conquer algorithm. Step 3 of Algorithm Divide and Conquer requires the additional inquiry of at most \( \delta_{\text{max}}(G) \) nodes for one node in each separating set.

We now show how we can use separator theorems to determine upper bounds on \( v(G) \). Suppose \( G \) belongs to a class of graphs \( G \) closed under the subgraph operation. \( G \) satisfies an \( s(n) \)-separator theorem with constants \( \alpha, \beta, \alpha \in [1/2, 1), \beta > 0 \) if any \( G \in G \) with \( n \) vertices can be separated into two sets \( A \) and \( B \) with \( \|A\|, \|B\| \leq \alpha n \) and the separating set has \( \leq \beta s(n) \) vertices. A feasible, though not necessarily polynomially computable, strategy for Player I is to separate the graph by such a separator, forcing Player II to choose a smaller graph to work with. This algorithm gives the following bound:

**Corollary 4.8:** If \( G \) belongs to a class of graphs \( G \) closed under the subgraph operations such that \( G \) satisfies an \( s(n) \)-separator theorem with constants \( \alpha \) and \( \beta \), then

\[
v(G) \leq \beta \sum \{ S(\alpha^i n) : 0 \leq i \leq -\log n / \log \alpha \}.
\]

**Proof:** This follows from the definition of Player I's algorithm. \( \Box \)
There are several separator theorems in the literature for special types of graphs. Here, we give two examples which illustrate how to use these to derive bounds on the value of the separation game.

**Theorem 4.9:** [D] Any planar graph satisfies a $\sqrt{n}$-separator theorem with $\alpha=2/3$ and $\beta=\sqrt{6}$. $\square$

**Theorem 4.10:** For any planar graph on $n$ vertices, the value of the separation game on $G$ is at most $13.35\sqrt{n}$. $\square$

**Corollary 4.11:** A local optimum on a planar graph with $n$ vertices and maximal degree $\delta$ can be found in $13.35\sqrt{n} + \left\lceil \frac{\log n}{\log 3 - 1} \right\rceil$ function evaluations.

**Proof:** This follows from Theorems 4.7 and 4.10 and Corollary 4.8. $\square$

This theorem for planar graphs can be generalized to graphs of fixed genus.

**Theorem 4.12:** [GHT] Any graph of genus $g$ satisfies a separator theorem with $\alpha=2/3$ and $\beta_s(n) = (6\sqrt{g} + 2\sqrt{2}) \sqrt{n} + 1$. $\square$

**Theorem 4.13:** For any graph $G$ of genus $g$ on $n$ vertices, the value of the separation game on $G$ is at most $(6\sqrt{g} + 2\sqrt{2})(3+\sqrt{6})\sqrt{n} + O(\log n)$. $\square$

**Corollary 4.14:** A local optimum on a graph of genus $g$ with $n$ vertices and maximum degree $\delta$ can be found in $(6\sqrt{g} + 2\sqrt{2})(3+\sqrt{6})\sqrt{n} + O(\log n) + \delta\left\lceil \frac{\log n}{\log 3 - 1} \right\rceil$ function evaluations.

**Proof:** This follows from Theorems 4.7 and 4.13 and Corollary 4.8. $\square$

5. The Hypercube

We now apply the machinery developed in the previous sections to the problem of finding a local optimum on the vertices of the hypercube.
Letting $G_d = (V, E)$ denote the graph of the $d$-dimensional cube, and $f$ a function $V \to Z$, how hard is it to find a local optimum of $f$ on $G_d$?

A straightforward application of our divide and conquer algorithm gives:

**Theorem 5.1:** A local optimum on the $d$-cube can be found with
\[
c \left( \frac{2^d \log d}{\sqrt{d}} \right) \text{ function evaluations where } c = \sqrt{\frac{2}{\pi}} + o(d).
\]

**Proof:** We prove this theorem by illustrating that $dc(G)$ has the given value as an upper bound. Hence by Lemma 2.3, the proof will be complete.

Define the $p$th shell of the hypercube to be the vertices with exactly $p$ "1"s, so that the $p$th shell has cardinality $\binom{d}{p}$. Each shell separates the $d$-cube, so we can narrow down our search to a single shell by binary search on the $d+1$ shells. This binary search takes at most $\lg(d+1) + 1$ "steps," each step involving the inspection of a shell where no shell has cardinality exceeding $\binom{d}{\lfloor d/2 \rfloor}$. Further, by the extended Sterling approximation [F], the size of the center shell satisfies:

\[
C_1 \sqrt{\frac{2}{\pi}} \frac{2^d}{\sqrt{d}} \leq \binom{d}{\lfloor d/2 \rfloor} \leq C_2 \sqrt{\frac{2}{\pi}} \frac{2^d}{\sqrt{d}}
\]

where $C_1 = \frac{1}{2} e^{(12d+1)}$, $C_2 = \frac{1}{2} e^{12d}$.
Therefore the binary search requires at most \( \frac{2^d (1 + \log_2 d + 1)}{\sqrt{d}} \left( \frac{2}{\pi} + o(d) \right) \) lookups. There are at most \( d \log d \) "additional" queries so they don't affect this bound. Once the location of a local minimum has been narrowed down to a single shell, an additional \( \left\lfloor \frac{d}{2} \right\rfloor \) queries will certainly suffice to find a local optimum. This gives a total of
\[
\left( \frac{2}{\pi} + o(d) \right) \left( \frac{2^d}{\sqrt{d}} (2 + \log_2 (d + 1)) \right).
\]
proving the theorem.

We remark that the computation of the bound in this theorem is fairly tight. This might seem surprising since all of the shells are not the same size and hence a more efficient algorithm might, for instance, use a sequence of \( p \)'s (querying the \( p \)th shell) more like \( d/2, d/3, \ldots \) rather than \( d/2, d/4, \ldots \). Further, \( \left\lfloor \frac{d}{4} \right\rfloor \) is much smaller than \( \left\lfloor \frac{d}{2} \right\rfloor \), and in the theorem all shell sizes are bounded only by \( \left\lfloor \frac{d}{2} \right\rfloor \). However, note that when \( 0 \leq i \leq \sqrt{d}/2 \).

\[
\left\lfloor \frac{d}{2} \right\rfloor \cdot \left\lfloor \frac{d}{2} - \frac{\sqrt{d}}{2} \right\rfloor \cdots \left\lfloor \frac{d}{2} - \frac{1}{2} \right\rfloor \geq \frac{d}{2} \left( \frac{d}{2} - 1 \right) \left( \frac{d}{2} - 2 \right) \cdots \left( \frac{d}{2} - \frac{d}{2} \right) + 1
\]

\[
\left\lfloor \frac{d}{2} + \frac{\sqrt{d}}{2} \right\rfloor \left\lfloor \frac{d}{2} + \frac{\sqrt{d}}{2} - 1 \right\rfloor \cdots \left\lfloor \frac{d}{2} + 1 \right\rfloor
\]
This means that there is a stretch of $\sqrt{d}$ shells each at least half the size of the largest shell. Clearly any shell by shell search just in this region would cost at least

$$\frac{1}{2} \left( \frac{d}{d+1} \right) \log \sqrt{d} = \frac{1}{4} \left( \frac{d}{2} \right) \log d,$$

so we aren’t off by more than a factor of 4.

The hypercube is one of the few families of graphs whose genus has been determined: Beineke and Harary [BH] and Ringel [R] show it equals $(d-4)2^{d-2} + 1$. By Corollary 4.14, a local optimum can be found in

$$(6\sqrt{(d-4)2^{d-3} + 1} + 2\sqrt{2})(3 + \sqrt{6})2^{d/2} + 0(d) + \frac{d^2}{\log 3-1}$$

$$> (10)2^{d\sqrt{d-4}} (>> 2^d)$$

function evaluations. This is much weaker than the bound given by Theorem 5.1, because the $\approx 6\sqrt{d} 2^d$ separator from Theorem 4.12 is obviously too large for the $d$-cube.

We can also combine a result about boundary minimization on the $d$-cube with our Corollary 4.6 to get a lower bound on $v(C)$ and hence a lower bound on the number of function evaluations required. A bound on the minimum boundary size $B(G,m)$ is given by [T2]:
Let $P_i$ denote \( \sum_{j=0}^{i} \binom{d}{j} \). If \( m = P_i \), then \( B(C_{d',m}) = \binom{d}{i+1} \). Otherwise let \( i' \) be such that \( P_{i'-1} \leq m \leq P_i \). Then

\[
B(C_{d',m}) \geq \begin{cases} 
\binom{d}{i} & i \leq \frac{d-1}{2} \\
\binom{d}{i+1} & i \geq \frac{d-1}{2}
\end{cases}
\]

Applying the above bounds gives \( \beta(G_d) = \binom{d}{\lfloor \frac{d}{2} \rfloor} \) where \( \beta() \) is as defined in Section 4. Moreover, for \( m \) in the range

\[
P_{\left\lfloor \frac{d-1}{2} \right\rfloor} - 1 \leq 2^{d-1} - \binom{d}{\left\lfloor \frac{d}{2} \right\rfloor} = 2^{d-1} - \beta(G_d) \leq m \leq 2^{d-1}
\]

we have a lower bound of

\[
B(C_{d',m}) \geq \binom{d}{\left\lfloor \frac{d-1}{2} \right\rfloor} \geq 2^d \sqrt{d} \left( \sqrt{\frac{2}{\pi}} - o(d) \right).
\]

Applying Corollary 4.6, we have proved the following:

**Theorem 5.2**: The value of the separation game played on the \( d \)-dimensional hypercube is at least \( \left( \sqrt{\frac{2}{\pi}} - o(d) \right) 2^d \sqrt{d} \).

**Corollary 5.3**: Any algorithm which finds a local minimum on the \( d \)-cube requires at least \( 2^{d-1} \sqrt{d} \) function evaluations.
In this case the gap between the lower and upper bounds is more than a constant factor, but still quite small: $O(\log \log |V|)$.

We make some comments comparing this result to others concerning local optimization on the d-cube. The closest result we know of is due to Aldous [A]. He considers a game where player I, a minimizer, selects an algorithm $A$ to find a local minimum on the d-cube, while player II, a maximizer, selects the function $f$. The outcome of play of the game is the number of function evaluations required by $A$ to find a local minimum of $f$. Aldous shows that the value of the game is roughly $O(2^{d/2})$. In these terms, we have analyzed the value of a modified game when player I has to play before player II (player II sees $A$ before choosing $f$). It is interesting that the modification causes such a large increase in the value of the game.

The problem we have analyzed, when $f$ is computable by a polylog (in $n$, i.e. polynomial in $d$) width and depth circuit, is essentially FLIP, the canonical PLS-Complete problem of [JPY]. So if $\text{PLS} = \text{P}$, the polynomial algorithm for FLIP must make strong use of limitations on $f$, since Theorem 5.2 implies there is no polynomial algorithm that works for arbitrary functions $f$ (or even arbitrary functions $f$ with single local optimum). On the other hand, if PLS is not in P, it might be possible to modify the adversarial argument so as to create a function $f$ which was polylog computable, while retaining a superpolynomial bound. Such a proof would imply $\text{NP} \neq \text{P}$ and would undoubtedly be very difficult! Even a weaker result of the following form would be interesting (this reflects the generally sorry state of lower bound technology): if $f$ is computed by a circuit of fixed width and depth $p(n)$, then the number of function
evaluations required to find a local optimum is at least \( q(p(n)) \), where 
\( q() \) and \( p() \) are polynomial functions (\( q \) of quadratic or higher order).

Theorem 5.2 yields a snake-in-box result. Define a snake in a graph as a simple path such that any two vertices in the path are adjacent in the graph iff they are adjacent in the path. Thus the snake's coils stay a hamming distance of at least 2 from each other. (A snake-in-box in \([DK]\) is defined similarly except that the snake bites its tail, i.e., it is a cycle.) How long can a snake in the \( d \)-cube be? Notice that a local improvement algorithm which iteratively selects the best adjacent vertex must follow a path which is a snake, since a better adjacent vertex further down the path would previously have been selected. Such an algorithm takes at most \( d-1 \) function evaluations per iteration (after the first). But Theorem 5.2 says that in the worst case, any algorithm must use \( 2^d/2\sqrt{d} \) function evaluations. Hence, the local improvement algorithm in the worst case takes at least \( \left\lceil \frac{2^{d-1}/\sqrt{d} - 1}{d-1} \right\rceil \) iterations, so there is a snake that long (of order \( 2^{d-3/2} \)). This is not quite as strong a result as the best known \([DK]\) of order \( 2^{d-1/2} \).

We get a surprising observation if we compare the proof of Theorem 5.2 with the analysis of average performance in \([T2]\). Let \( B(G,m) = \min \{|B(S)| : |S| = m\} \). The driving force behind the exponentially large lower bound is the large size of \( B(G_d,m) \). In particular, the fact that

\[
B(G_d,m) \geq \left\lceil \frac{d}{d-1} \right\rceil, \quad \frac{1}{3} 2^d \leq m \leq \frac{2}{3} 2^d
\]
is sufficient to imply a large exponential lower bound on worst case performance of any algorithm. On the other hand, the $O(d \log d)$ and $O(d^2)$ upper bounds on average performance of local improvement algorithms in [T2] are derived by proving inequalities such as

$$I(d) \leq 2 + \sum_{m=2}^{2^{d-1}} e^k B(G_d,m).$$

where $I(d)$ is the expected number of iterations of a local improvement algorithm for problems with a single local optimum (for any $k$ regular graph). Hence, here the large magnitude of $B(G_d,m)$ leads to a low order polynomial bound on $I(d)$. Thus we have the peculiar situation that the same graphical property of the hypercube, namely the large magnitude of $B(G_d,m)$, appears to be responsible for both the exponentially bad worst case and the polynomially good average case performance of local improvement algorithms.

6. Remarks

Our analysis shows that local search may not be the most efficient method (in the worst case) to find a local optimum of an arbitrary function on a graph. Instead, a divide-and-conquer algorithm tends to give better performance. On a path, this algorithm is essentially Fibonacci search. Fibonacci search can be extended to higher dimensions [ ] but this does not give the right way to find local optima in arbitrary graphs. Binary search has also been extended to higher dimensions by Dobkin and Lipton [DL] and Wood [W]; in [DL] the function values (data) are allowed to be rearranged in any desired way; in [W] the
generalization is highly geometric. The proper generalization to arbitrary graphs, for our problem, is closer to other divide and conquer algorithms on graphs, such as nested dissection for solving a system of linear equations [LRT]. Like generalized nested dissection, our algorithm depends heavily on finding a sequence of graph separators: as in [LRT] Corollary 4.5 shows that no efficient algorithm for finding local optima exists if the graph does not have a good separator.

Theorem 4.2 shows that for general graphs it is NP-hard to find the best set of separators. On the other hand, the analysis of the separation game, though hard, also provides a close lower bound, assuring us of the near-optimality of our algorithm.

It is interesting to see how complicated it can be to design an efficient algorithm that does something very simple, such as finding an entry in a matrix which is smallest in its row and column. The analysis of the d-cube also gave us a new insight into the relationship between the average and worst case behavior of local improvement algorithms.

The exponential lower bound on the work required by any algorithm to find a local optimum on the d-cube suggests that, to find polynomial algorithms for local optimization in combinatorial problems (e.g., integer programming, travelling salesman problem) we need to make heavy use of the restrictions on the objective function implied by the combinatorial problem. Even though it is easy to know which points to check to test for local optimality, and it is easy to know that at least one local optimum must exist, it is hard to find a local optimum for arbitrary \( f \). Our construction employs an \( f \) which (we think) cannot be computed by a polynomial (in \( d \)) width and depth circuit. If it is not
easy to know that at least one local optimum must exist, then finding it can be difficult even with a simple $f$. For example, if we seek a strict local optimum, the trivial adversarial functions, that are constant everywhere, or constant except at one point, ensures that any algorithm is slow. It can also be hard to know how to test for local optimality: Murty [M] shows that it is NP-hard to test for local optimality in continuous quadratic programming, even if function evaluations are assumed to cost unit time.

Acknowledgements

The term "bare tree" was suggested by Kurt Anstreicher.
REFERENCES


FINDING SADDLEPOINTS OF TWO-PERSON ZERO SUM GAMES

Donna Crystal Llewellyn*
Craig Tovey**
Michael Trick

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0205

*Supported by NSF Postdoctoral Fellowship
**Research supported by NSF Grant No. ECS 8451032
ABSTRACT: A strict saddlepoint of a matrix, when it exists, uniquely solves the associated two-person zero sum game. We give an $O(n^{\log_2 3})$ algorithm to find a strict saddlepoint in an $n \times n$ matrix. We also show that $\Omega(n^2)$ work is required to find a non-strict saddlepoint.

1. INTRODUCTION

A two-player zero sum game, as defined by von Neumann and Morgenstern in their classical work, [13], is completely specified by its payoff matrix. Let player I have $m$ strategies and player II have $n$ strategies. Suppose that player I pays player II $a_{ij}$ if player I chooses his $i^{th}$ strategy and player II chooses her $j^{th}$ strategy. Then the payoff matrix of the game is the $m \times n$ matrix $A = [a_{ij}]$. A solution for player I (similarly player II) is a vector of $m$ (respectively, $n$) probabilities, where the $i^{th}$ ($j^{th}$) component is the probability of playing the $i^{th}$ ($j^{th}$) strategy.

In general, finding the optimal solution vectors for the two players is equivalent to solving an $m \times n$ linear programming
problem ([3]). For some games, however, there exist optimal strategies for both players that are pure strategies; that is, each player will use a single strategy with probability one. Define a pair of strategies to be in strong equilibrium if each player can only worsen their payoff by switching to a different strategy. It is clear that a pair of strategies in strong equilibrium is an optimal pair of strategies. This has been well understood since the origins of game theory. (Note that the main contribution of von Neumann and Morgenstern was the development of the theory of mixed strategies to extend the duality to the case where no strong equilibrium exists). However, it appears that the question of how to find strong equilibria efficiently has not been carefully studied. For instance, in [4] it is stated that "a [strong equilibrium] is easy to find, even in a very large matrix." Other standard texts, such as [1], [2], [5]-[11] and [14]-[17] also mention that these strategies are easy to locate but fail to discuss finding an efficient algorithm.

The purpose of this note is to show that a pair of pure strategy solutions that are in strong equilibrium can be found very quickly when such a solution exists. In fact, we give an algorithm that requires knowledge of \( \approx n^{0.58} m \) entries of the payoff matrix (where \( m \geq n \)). Hence, as the number of strategies increases, the proportion of the payoff matrix examined approaches zero. This algorithm also determines when a matrix has no strong equilibrium solution.
A pair of strategies is in weak equilibrium if neither player can improve their payoff by altering strategy. Unfortunately, determining if a weak equilibrium solution exists requires more work than the analogous problem for a strong equilibrium solution. We show that any algorithm to find such a pure strategy solution requires examination of the entire payoff matrix. It is important to note that if all of the entries in the payoff matrix are distinct then any weak equilibrium is also a strong equilibrium and hence the faster algorithm applies.

Pure strategies that are in weak equilibrium are saddlepoints of the payoff matrix. A saddlepoint (SP) of a matrix $A$ is an entry $a_{ij}$ where $a_{ij}$ is the largest element in row $i$ and the smallest in column $j$; that is

$$a_{ij} > a_{ik} \text{ for all } k$$
$$a_{ij} \leq a_{kj} \text{ for all } k.$$ 

If both inequalities are strict, then we call $a_{ij}$ a strict saddlepoint (strict SP).

A SP of a payoff matrix corresponds to a pair of pure strategies that are in weak equilibrium and a strict SP is a strong equilibrium solution to the game. Therefore, finding a pair of pure strategies that are in weak (strong) equilibrium is equivalent to finding a (strict) SP of the payoff matrix. In the remaining sections of this note we will discuss only the mathematical problem of finding (strict) saddlepoints of a matrix, rather than the game theoretic problem of finding weak and strong equilibria.
Given a matrix, we refer to the problem of finding a (strict) saddlepoint or determining that one does not exist as the (strict) SP problem. For ease of notation, we will denote $\log_2 x$ by $\log x$.

II. Finding Strict Saddlepoints

We motivate our algorithm for finding a strict saddlepoint with a couple of observations.

Observation 1: If two entries of $A$, $a_{ij}$ and $a_{kl}$ are known, then at least one of them can be eliminated as a possible strict SP with a single matrix look-up and a constant amount of computation.

Proof: Clearly this follows if either $i = k$ or $j = l$. Hence assume that neither of these holds. Without loss of generality let $a_{ij} \leq a_{kl}$. Look up $a_{il}$. Then, clearly if $a_{ij}$ is a strict SP then $a_{ij} > a_{il}$. Further, if $a_{kl}$ is a strict SP then $a_{il} > a_{kl}$. Since $a_{ij} \leq a_{kl}$ it is clear we can not have $a_{ij} > a_{il} > a_{kl}$. Hence, knowing $a_{il}$ will eliminate either $a_{ij}$ or $a_{kl}$ or both from consideration.

Notice that in the proof of Observation 1, the entry $a_{kj}$ cannot be a strict SP since $a_{ij} \leq a_{kl}$ implies that it is impossible to have $a_{ij} > a_{kj} > a_{kl}$. This leads to our second observation.
Observation 2: If the diagonal terms of $A$ are known and are such that $a_{11} \leq a_{22} \leq \ldots \leq a_{nn}$, then no subdiagonal term of $A$ is a strict SP.

Note that to apply observation 2 we will first have to rearrange the rows and columns of $A$ so that the diagonal is ordered. Our algorithm uses observations 1 and 2 together with proper manipulation of rearrangement pointers. The idea of the algorithm is illustrated in Figures 1 and 2.

The main diagonal is looked up and then sorted to eliminate the subdiagonal. The remaining region is divided into two triangles and one square which are studied recursively. These pieces are then put back together using observation 1. In the diagrams, the shaded regions are the parts of the matrix which have been ruled out by our observations and hence do not have to be searched.

![Figure 1](image1.png)

![Figure 2](image2.png)
We now state the algorithm precisely.

Procedure Square(n, r(1),..., r(n), c(1),..., c(n); candidate)
This procedure returns candidate, a location in A which might be a strict SP. No other element of the nxn submatrix of A, which is indexed by r and c, is a strict SP.

Step 0: If n = 1, return candidate = (r(l),c(l))

Step 1: Look up and sort A(r(i),c(i)), i = 1,...,n in ascending order.
Suppose $\sigma$ is the permutation that gives this sorting;
Set $r(i) = r(\sigma(i))$
$c(i) = c(\sigma(i))$ for $i = 1,...,n$.

Step 2: Call Triangle(n,r(l),...,r(n),c(l),...c(n);candidate)

Step 3: Take $\sigma^{-1}(candidate)$ and return it.

Procedure Triangle(n,r(l),...,r(n),c(l),...c(n);candidate)

Step 0: If n = 1, return candidate = (r(l),c(l))
Step 1: Call Triangle (\(\lfloor \frac{n}{2} \rfloor\), r(l),..., r(\(\lfloor \frac{n}{2} \rfloor\)), c(l),..., c(\(\lfloor \frac{n}{2} \rfloor\)); candidate)
Step 2: Call Square \(\left[\frac{n}{2}\right], r(1), \ldots, r\left(\left[\frac{n}{2}\right]\right), c\left(\left[\frac{n}{2}\right]\right), \ldots, c(n); \text{candidate}2\)

Step 3: Call Triangle \(\left[\frac{n}{2}\right], r(1 + \left[\frac{n}{2}\right]), \ldots, r(n), c(1 + \left[\frac{n}{2}\right]), \ldots, c(n); \text{candidate}3\)

Step 4: Reduce candidatei, i = 1,2,3 to one or no candidate.

Step 5: Return candidate.

Program Strict Saddlepoint

This program solves the strict saddlepoint problem on an nxn matrix A.

Step 1: Call Square(n,1,...,n,1,...,n;candidate)

Step 2: Check to see if candidate is a strict SP; if so, then report the strict SP, if not, then conclude there does not exist a strict SP.

Proposition: Program Strict Saddlepoint solves the strict saddlepoint problem.

Proof: This follows immediately from observations 1 and 2.
Theorem: Program Strict Saddlepoint terminates in time $O(n^{\log 3})$.

Proof: Let $F(n)$ be the time required to run Procedure Square($n,r(1),\ldots,r(n),c(1),\ldots,c(n);\text{candidate}$) and let $G(n)$ be the analogous time for Procedure Triangle($n,r(1),\ldots,r(n),c(1),\ldots,c(n);\text{candidate}$). Hence, the work required to run Program Saddlepoint is $F(n) + O(n)$; (due to the last check required to see if candidate is really a strict SP in step 2 of Program Saddlepoint). Notice that $G(n) = F(n) - c_1n\log n$ for some constant, $c_1$.

By analyzing the steps of Program Saddlepoint, $F(n) = c_1n\log n + F\left(\frac{n}{2}\right) + 2G\left(\frac{n}{2}\right) + c_2 + c_3n$; where the $c_3n$ term is the cost of passing the row and column pointers and the $c_2$ term is the cost of making the final decision between candidates; that is, step 4 of the triangle procedure (using observation 1).

So, $F(n) = c_1n\log n + 3F\left(\frac{n}{2}\right) + c_2 + c_3n - 2c_1\left(\frac{n}{2}\right)\log\left(\frac{n}{2}\right)$

$= c_1n\log n + 3F\left(\frac{n}{2}\right) + c_2 + c_3n - c_1n\log n + c_1n\log\left(\frac{n}{2}\right)$

$= 3F\left(\frac{n}{2}\right) + c_2 + (c_3 - c_1)n$.

Thus $F(n) = 3^{\log n}F(1) + \sum_{i=0}^{\log n} (3^i c_2 + \left(\frac{3}{2}\right)^i (c_3 - c_1)n)$

But, $F(1) = 1$, so after simplification we get

$F(n) = 3^{\log n} \left( 1 + \frac{1}{2}c_2 \right) + 2\left(\frac{3}{2}\right)^{\log n}c_3n - \left(\frac{1}{2}c_2 + 2(c_3 - c_1)n\right)$
So, \[ F(n) = O(3^{\log_3 n}). \] But, \[ 3^{\log_3 n} = n^{\log_3 3}, \] and hence
\[ F(n) = O(n^{\log_3 3}). \]

Hence, the Program Strict Saddlepoint runs in time \( O(n^{\log_3 3}) \) as claimed.

The reader familiar with fractals will note that the unshaded region in Figure 2, corresponding to the portion of the matrix that must be searched, is an illustration of a fractal of dimension \( \log_3 \); and hence has area \( n^{\log_3} \). This provides a geometrically intuitive argument for the running time of Program Strict Saddlepoint. ([12], pages 141-142).

**Corollary:** The strict SP problem on an mxn matrix, with \( m \geq n \), can be solved in \( O((\frac{m}{n})n^{\log_3}) \) time.

**Proof:** Divide the matrix into \( \left\lfloor \frac{m}{n} \right\rfloor \) possibly overlapping nxn square matrices, \( A_1, \ldots, A_{\left\lfloor \frac{m}{n} \right\rfloor} \). Apply the theorem to each \( A_i \) to get at most \( \left\lfloor \frac{m}{n} \right\rfloor \) candidate strict SP's, from which the real strict SP (if it exists) can be found in work \( O(\frac{m}{n} + m + n) \). The total work is thus bounded by \( \left\lfloor \frac{m}{n} \right\rfloor O(n^{\log_3}) + O(m) = O((\frac{m}{n})n^{\log_3}) \) as claimed.
III. Saddlepoints

Note that if we are seeking a SP then observation 1 only holds if \( a_{11} > a_{22} > \ldots > a_{nn} \) and similarly observation 2 will fail unless the inequality is strict. Thus it is not clear whether the search for a SP can be sped up in a similar way. In fact we show next that the SP problem cannot be solved in time better than \( \Omega(n^2) \).

**Theorem:** The most efficient algorithm to solve the SP problem on an \( nxn \) matrix runs in time \( \Theta(n^2) \).

**Proof:** We can put a check mark in each matrix cell which is the maximum in its row in time \( \Theta(n^2) \), and then do the same for the cells which are the minimum in their columns in \( \Theta(n^2) \), while checking for a cell with two check marks which would clearly be an SP.

We show the \( \Omega(n^2) \) lower bound by playing the adversary against an arbitrary algorithm. We argue that the algorithm must check every entry in \( A \).

Our strategy: For the first \( n^2 - 1 \) queries of the values of \( a_{ij} \) we answer 0, unless all the other entries in row \( i \) have been queried, in which case we answer 1. If we answer .5 to the last \( (n^2) \) query, then either that cell is the unique SP or there does not exist an SP. This follows since every 0 entry has either a 1 or a .5 in its row and every 1 has either a 0 or a .5 in its column. If instead, we answer -1, then the
cell is not a SP, and in fact any zero entry in that last row will be a SP. Hence no algorithm can know the answer to the SP problem without \( n^2 \) queries. Note here that this adversarial argument shows that for general \( m \) and \( n \), an entire \( mxn \) matrix must be examined (i.e., to solve the SP problem on an \( mxn \) matrix requires \( \Omega(mn) \) lookups).

In conclusion, we have shown that the strict SP problem can be solved in \( O(n^{1.58}) \) time, but that a similar improvement from \( \Theta(n^2) \) for the SP problem cannot be found. Note that our algorithm for the strict SP problem uses \( O(n^{1.58}) \) matrix lookups and has an \( O(n^{1.58}) \) running time. We conjecture that it may be possible to solve this problem with less than \( O(n^{1.58}) \) matrix lookups but at the cost of a longer running time.

REFERENCES

Routing Printed Circuit Cards
Through an Assembly Cell

Christopher B. Lofgren
and
Leon F. McGinnis

Material Handling Research Center
School of Industrial and Systems Engineering
Georgia Institute of Technology

Craig A. Tovey

School of Industrial and Systems Engineering
Georgia Institute of Technology

Abstract: A printed circuit card must be routed through a set of automated machines in an assembly system so that all of its required components are inserted. Each component required by a card has been assigned to one station in the system. There are precedence constraints on the order that components can be inserted on the card. The problem is to find a minimum length sequence of possibly repeating station visits that inserts all of the components subject to the precedence constraints. This routing problem is NP-Complete. We show that critical path scheduling methods have arbitrarily bad worst case performance. Further, if there exists a polynomial algorithm with a finite worst case performance ratio, then there exists a polynomial approximation scheme for the routing problem. Finally, we discuss the implications of this problem on design and operational issues.

This research was supported by the National Science Foundation, Grant Nos. CDR-8300965, and the Member Companies of the Material Handling Research Center.

1. Introduction

Printed circuit card assembly involves the assembly and joining of various electrical and electronic components to a bare printed circuit card. The components may be discrete devices, such as resistors, capacitors, diodes, transistors, transformers, connectors, etc., or they may be integrated circuits in any of a number of packages types. Assembly may require inserting wires or pins into holes or the placement of a device onto a specified location. Joining can be accomplished manually, via a wave solder bath, using a reflow technique, or by some other means.

In this paper, we focus on the assembly operations. Generally, the assembly operations are sequential, or one at a time. For sparsely populated cards, it may be possible to perform the assembly operations in any desired order. However, as the density of components increases, it may be necessary to perform the assembly operations in a particular order, dictated by such factors as the height of
assembly components, clearance requirements for component grippers, location on the card, etc. When such factors become important, they can be described by a precedence graph, or partial ordering of the assembly operation.

Automated assembly is accomplished using either robots or specialized assembly equipment. Usually the assembly workstation is specialized with regard to the types of components that it can handle, e.g., dual-in-line-packages (DIP), axial leaded, radial leaded, etc., although robotic stations may be capable of handling several component types.

We consider an assembly cell consisting of three or more assembly stations, which may be distinguished by the types of components that they can handle. We assume that a printed circuit card requires a number of components, and that each component type is available at only one (predetermined) station. The component requirements, component-to-station assignments, and assembly precedence constraints may cause a card to visit a workstation more than once.

The problem of interest is to determine a complete ordering of the component assembly operations which minimizes the number of workstation visits.

For example, suppose the twelve components required on a card have been assigned to three machines. The assignment of components is given in Table 1.1.

Table 1.1 Component Assignment

| component: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| machine:   | A | B | C | D | E | F | G | H | I | J  | K  | L  |

Suppose the partial order is \{(1>3), (1>4), (2>4), (2>5), (3>6), (3>10), (4>7), (5>7), (5>8), (5>9), (6>11), (7>12), (9>12)\}, where (a>b) implies component a must be inserted before component b. The optimal sequence of machine visits for this example is given in Table 1.2.

Table 1.2 Optimal Routing

<table>
<thead>
<tr>
<th>Sequence of Machine Visits</th>
<th>Components Inserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2,4,6,10</td>
</tr>
<tr>
<td>C</td>
<td>5,7,11</td>
</tr>
<tr>
<td>A</td>
<td>8,9,12</td>
</tr>
</tbody>
</table>
2. Problem Definition

We define a graph theoretic problem, called the station routing problem, which models the printed circuit card routing problem. An instance of the station routing problem consists of a directed acyclic graph, $G(V,E)$, (representing the partial ordering of component insertions on a card), and a partition of the node set $V$ into $M$ disjoint sets $V = (V_1, \ldots, V_M)$. This partition represents the assignment of components to stations and so we will refer to the $V_i$ as stations.

The key operation we perform on $G$ corresponds to taking the circuit card to a station and performing all possible insertions consistent with the precedence constraints. Define vertex $v$ to be a precursor of vertex $w$ if there exists a directed path in $G$ from $v$ to $w$. For a graph $G = (V_1, \ldots, V_M; E)$ and station $V_i$, we define a visit of $G$ to $V_i$ to be the deletion of all vertices in $V_i$ which have no precursors not in $V_i$. A feasible solution to the station routing problem is a sequence of station numbers $S$ such that the succession of visits of $G$ to the stations given by $S$ leaves $G$ empty. The objective of the station routing problem is to find a feasible solution of minimum length.

In the next section we use this model to establish the NP-completeness of the station routing problem. In section 4 we develop some general negative results regarding worst-case heuristic performance. We discuss a number of heuristics in section 5, and give test results based on a model for generating structured random instances. We conclude in section 6 by discussing the ramifications of this problem on product and process design.

3. NP-Completeness of the Station Routing Problem

Consider the decision version of the printed circuit card routing problem: Given a directed acyclic graph, $G(V,E)$; a partition of the set of vertices into $M$ disjoint subsets; and a positive integer $Q \leq |V|$; does there exist a feasible sequence, $S$, such that the length of the sequence is no more than $Q$? To show the station routing problem is NP-complete, a restricted version of a known NP-complete problem is first shown to be "hard". This restricted problem is then shown to be a special case of the card routing problem.

The shortest common supersequence problem (SCS) can be defined as follows [Garey and Johnson, 1979]: Given a finite alphabet $\Sigma$, a finite set of strings $R$ from $\Sigma^*$, and a positive integer $Q$; is there a string $w \in \Sigma^*$ such that $|w| \leq Q$ and every string $r \in R$ is a subsequence of $w$?
This problem can be illustrated with a two string example. Let string 1 be (1223) and string 2 be (31233). Then a shortest common supersequence, w, is (312323) or (312233), and \(|w| = 6\).

A restricted version of SCS can be defined by requiring that no element in the alphabet may occur consecutively in any string \(r \in R\). We will refer to this as the non-repetitive shortest common supersequence problem (NSCS).

**Theorem 3.1.** NSCS is NP-complete.

**Proof of Theorem 3.1.** Let \(I = (\Sigma, R, Q)\) be an arbitrary instance of SCS \(<\) where \(\Sigma\) is the alphabet, \(R\) is a set of strings, and \(Q\) is a positive integer. Create an instance \(\hat{I} = (\hat{\Sigma}, \hat{R}, \hat{Q})\) of NSCS where for each element \(u \in \Sigma\), \(\hat{\Sigma}\) contains \(u\) and a new element \(u'\); for each \(u \in \Sigma\), every occurrence of \(u\) in each \(r \in R\) is replaced with \(uu'\) to create each \(\hat{r}\) in \(\hat{R}\), and \(\hat{Q} = 2Q\).

If \(S\) solves \(\hat{I}\) then clearly we can use the same \(uu'\) substitution to create \(\hat{S}\) where \(\hat{S} = 2S;\) \(\hat{S} \leq 2Q = Q\), and \(S\) is a superstring of all \(\hat{r}\) in \(\hat{R}\). Conversely, suppose \(\hat{S}\) solves \(\hat{I}\). Extract from \(\hat{S}\) two substrings, \(S\) and \(S'\), containing the unprimed and primed elements of \(\hat{S}\), respectively. It is obvious that since \(\hat{S}\) is a superstring of \(\hat{r}\), \(S\) and \(S'\) must be superstrings of \(r\) and \(r'\), respectively. Thus \(S\) is a common superstring of \(r\) and \(r'\), respectively. Thus \(S\) is a common superstring of \(R = \{r\}\) while \(S'\) is a common superstring of \(R' = \{r'\}\). But \(R'\) differs from \(R\) only in that each element is "primed" instead of "unprimed". The strings \(S\) and \(S'\) therefore each provide a common supersequence of \(R\). Since \(\hat{Q} = 2Q \geq S = S + S'\), the shorter one has length at most \(\frac{\hat{S}}{2} = Q\), which solves \(I\). So \(I\) is solved iff \(\hat{I}\) is, NSCS is obviously in NP, and NSCS is NP-complete.

**Theorem 3.2.** The printed circuit card routing problem routing problem is NP-complete, even when the precedence graph, \(G\), consists of disjoint chains.

**Proof of Theorem 3.2.** The non-repetitive shortest common supersequence problem is a special case of the card routing problem where the precedence graph is a collection of disjoint chains. QED.

A number of comments need to be made with respect to the computational complexity of the station routing problem. First, if the number of stations is no more than two, then computing an optimal sequence is trivial. The algorithm for these instances would start at a selected station, and alternate visits until 6 was empty. After running the algorithm with each station as the starting point, choose the sequence with minimum length.

Second, if the precedence graph consists of \(K\) disjoint chains, then the problem can be solved in \(O(n^K)\) using dynamic programming. Third, because the proof of NP-completeness for SCS requires \(|\Sigma| \geq 5\), the reduction to the card routing problem only holds for ten or more stations. (This can be reduced to \(\hat{G}\) fairly easily.) A result in [Kiran, 1986] can be used to show the card routing problem is
NP-complete for three or more stations, but this result does not reveal the difficulty associated with generating solutions to simple structured graphs. Finally, we can show through a direct transformation from 3-SAT that, if the number of stations is not fixed, the problem remains hard restricted to graphs of height 12 (no directed path has length >12).
4. Worst Case Performance of Heuristics

If a card must visit only one or two machines it is easy to compute its optimal routing. Alternatively, if the precedence graphs for printed circuit cards are linear orderings (single chains) or two disjoint chains the optimal routings are easily generated. As the precedence graphs become less and less like a linear order, and the number of machines that a part must visit becomes larger, determining optimal solutions may become computationally prohibitive. Consequently, a heuristic is required to compute "good" solutions for the card routing problem. In this section we present negative results about the worst-case performance of some classes of natural heuristics, and then give evidence that no polynomial heuristic has bounded worst case performance.

The first class of heuristics we consider are level strategies. The concept of levels in a precedence graph is often utilized in scheduling theory e.g., [Coffman and Graham, 1972]. Let a vertex in a precedence graph with no successors be defined as a leaf. A vertex $v$, is in the kth level of $G$ if and only if the (edge) length of the longest path from $v$ to a leaf is $k$.

To illustrate this concept, consider the precedence graph in Figure 4.1. Level 0, ($L_0$), is composed of the leaf vertices 10, 11, and 12. Likewise the other levels are: $L_1 = \{6,7,8,9\}$; $L_2 = \{3,4,5\}$; and $L_3 = \{1,2\}$.

A level sequence is any sequence that deletes at least one vertex in the highest nonempty level in the directed graph at each station visit. A level strategy is any algorithm which produces level sequences.

![Figure 4.1 Graphic Representation of a Partial Ordering](image-url)
The levels of an acyclic digraph can be computed in $O(|E|)$ time using a recursive version of depth-first search, [Aho, Hopcroft, Ullman, 1983], which keeps a label for each vertex equal to the longest path to a currently marked leaf. Thus there are level strategies which run quickly. Level strategies have been used successfully to find optimal schedules for job shop problems with unit processing times and tree precedence constraints [Hu, 1961]. A similar idea is used to solve the 2-processor unit processing problem subject to arbitrary precedence constraints [ ]. This makes the level strategies a natural class of heuristics to consider for the station routing problem.

A closely related class of algorithms are the well known critical path strategies [Coffman 1976]. An algorithm is a member of this class iff "when it has finished a task it chooses ... as the next task to start working on ... that task which heads the chain of unexecuted tasks which has the greatest ... [total] of processing times" [Graham 1978]. Applying this definition to the station routing problem, we see that it is not exactly the same as a level strategy, because a path in G might contain consecutive nodes in the same station. For instance, in Figure 4.1, there are two nodes at level 1, but only the node marked B is at the end of a longest critical path. If we define the "critical" length of a sequence to be the length of a longest nonconsecutively repeating subsequence, then we can describe a critical path strategy the same as we define a level strategy where "length" is replaced by "critical length".

How well do level and critical path strategies perform? The example in Figure 2.1 shows that no level or CP strategy can always be optimal: The assignment of nodes to stations is represented by the letter to the left of each vertex in Figure 2.1. A level schedule is given in Table 4.1. The unique optimal sequence, which is not a level or CP sequence is given in Table 4.2.

Table 4.1. Level Sequence

<table>
<thead>
<tr>
<th>Machine</th>
<th>Components Inserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>3, 5</td>
</tr>
<tr>
<td>B</td>
<td>4, 6, 10</td>
</tr>
<tr>
<td>C</td>
<td>7, 11</td>
</tr>
<tr>
<td>A</td>
<td>8, 9, 12</td>
</tr>
</tbody>
</table>

Here is a diagram showing the precedence graph.
Figure 4.1 Critical Path Levels

Table 4.2. Optimal Sequence

<table>
<thead>
<tr>
<th>Machine</th>
<th>Components Inserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2, 4, 6, 10</td>
</tr>
<tr>
<td>C</td>
<td>5, 7, 11</td>
</tr>
<tr>
<td>A</td>
<td>8, 9, 12</td>
</tr>
</tbody>
</table>

The example shows how level and CP sequences can be non-optimal. The next question is: In the worst case, how far from optimal can these strategies be? The following theorem says there is no lower bound on the worst case performance defined as the length of the heuristic sequence divided by the length of an optimal sequence.

**Theorem 4.1.** Any level or CP strategy for the station routing problem has an arbitrarily bad worst case performance.

**Proof of Theorem 4.1.** Let $M = 2N$ be the number of stations in an instance of the card routing problem. We will show that the worst case performance can approach $N$. Let the precedence graph be a collection of disjoint chains constructed as in Figure 4.2. Since all of those chains are nonrepetitive, any CP strategy is also a level strategy so we can analyze the latter. Any level sequence must select all stations found in level $i$ before any in level $i-1$. There are $N$ components in each level assigned to $N$ different machines. In addition, there are $2N$ levels in each of the $k$ blocks. Hence, any level sequence will have a length equal to $2KN^2$.

A shorter feasible solution is as follows: Visit the machines in the order of the first chain in the graph, performing all possible insertions at each machine visit. This sequence is of the form

$$1, N+1, 2, N+2, ..., N, 2N, 1, N+1, ..., 2N,$$

and has length $2KN$. Upon completing this sequence, the second chain has its last two nodes left, and the third chain has its last four nodes remaining. In general, the $p$th chain has its last $(p-1)2$ nodes remaining.

Now, notice that each of the remaining subchains from the second chain through the $N$-lst chain are subchains of the $N$th subchain. Therefore, an additional $2N-2$ visits are sufficient to delete all
nodes making the length of our feasible sequence equal to $2KN + 2N - 2$. (It is not hard to see, by considering the 1st and Nth chains, that this solution is optimal.) Hence, as $K$ increases, the ratio of the lengths of the level sequence and the optimum sequence is

$$\lim_{K \to \infty} \frac{2KN^2}{2KN + 2N - 2} = N.$$ Q.E.D.

To illustrate the construction, consider an example with $M = 2$, and 2 copies of the block. The precedence graph consists of two disjoint chains and four machines. This directed graph is illustrated in Figure 4.2. A level sequence is $(1, 2, 3, 4, 2, 1, 4, 3, 1, 2, 3, 4, 2, 1, 4, 3)$. An optimal sequence is $(1, 2, 4, 1, 3, 4, 2, 1, 4, 3)$. Figure 4.2. Construction for $M = 2, K = 2$

Theorem 4.1 can be extended to handle strategies for the station routing problem which look a limited number of levels down the graph to determine the next machine to visit.

We say that an algorithm is a $Q$-level strategy if it chooses a station that contains at least one vertex from one of the top $Q$ levels of the remaining graphs, at each iteration. Thus a 1-level algorithm is precisely a level algorithm. Similarly, we define a $Q$-critical path strategy as one which always deletes at least one node from one of the top $Q$ critical levels in the remaining graph. In the following, $M$ again denotes the number of stations.

**Theorem 4.2.** Any $Q$-level or $Q$-CP strategy for the station routing problem has worst case performance at least $M/(Q + 1)$. Moreover this bound applies when the graph $G$ is restricted to be a collection of disjoint non-repeating chains.
Proof of Theorem 4.2. Let the precedence graph be $K$ concatenated copies of the collection of disjoint chains constructed as in Figure 4.3, where $N = M/(Q+1)$. Any $Q$-level algorithm will choose every vertex in level $i$ before any vertices in level $i-Q$, and by construction the vertices in any $Q+1$ consecutive levels belong to different stations. Therefore, any $Q$-level algorithm will delete only one vertex per iteration, and will require $KM$ iterations.

As in the proof of Theorem 4.1, an optimal sequence can be constructed by first applying the numbers in the first chain. This has length $KM$. After these numbers are used, the $p$th chain has $(p-1)(Q+1)$ vertices remaining. Now note that each of these subchains remaining, for $p=2$ to $N-1$, is a subsequence of the sequence remaining in the $N$th chain. Therefore, an additional $(N-1)(Q+1)$ iterations are required. So the optimal sequence has total length $KM + (N-1)(Q+1)$. The ratio of the $Q$-level solution to the optimum is

$$\frac{KM}{KM+(N-1)(Q+1)} \quad \text{---} \quad \text{as } K \to \infty$$

Moreover, since all the chains are nonrepetitive, the result applies to $Q$-critical path strategies as well.

Q.E.D.

```
1   2   3   ...   N
N+1 N+2 N+3   ...   2N
...
QN+1 QN+2 QN+3   ...   M-1 (Q+1)N=M
2   3   ...   N   1
N+2 N+3   ...   2N   N+1
...
...
QN+2 QN+3   ...   (Q+1)N QN+1
...
...
N   1   2   ...   N-1
2N   N+1 N+2   ...   2N-1
...
...
(Q+1)N QN+1 QN+2   ...   (Q+1)N-1
```

Kth copy of the block

Figure 4.3. Troublesome Precedence Graph II
Note that the theorem proves unbounded worst-case performance for any strategy that only considers solutions from the top $\sqrt{M}$ levels. But the principal interest is the case where $Q$ is constant:

**Corollary:** For any fixed $Q$, any $Q$-level or $Q$-critical path strategy for the station routing problem has arbitrarily bad worst case performance.

The construction is best visualized with an example. Let $M=12$, $Q=2$, $N=4$, and $K=1$. The precedence graph is illustrated in Figure 4.4.

```
1  2  3  4
5  6  7  8
9 10 11 12
2  3  4  1
6  7  8  5
10 11 12  9
3  4  1  2
7  8  5  6
11 12  9 10
4  1'  2  3
8  5  6  7
12  9 10 11
```

*Figure 4.4. Construction for $M=12$; $Q=2$, $K=1$*

If $K=1$ the 2-level sequence requires 48 station visits while the optimum requires 21. If $K=2$ the station visits counts are 96 and 33, respectively; at $K=10$ they are 480 and 129, respectively.

Theorems 4.1 and 4.2 imply there does not exist a finite bound on the worst case ratio for algorithms that produce level or CPM sequences. In fact, the same construction illustrated in Theorem 4.1 can be used to show that heuristics based on: 1) a simple policy of visiting machines in a cyclic order $A, ..., M$; 2) visiting the machine with the largest number of operations currently able to be performed, (greedy); 3) an affirmative action policy [Fathi and Tovey, 1986]; can have arbitrarily bad worst case performance. While from a practical standpoint, those strategies might produce good solutions to instances encountered in real applications, from a theoretical standpoint it would be attractive to find a heuristic with a finite worst case ratio. The following definitions in conjunction with Theorem 4.3 say, in effect, that if there exists a polynomial algorithm for the station routing problem with a finite worst case ratio, then this algorithm can be used to get arbitrarily close to an optimal solution. The time required to generate such a solution is a polynomial in the accuracy required, and the length of the input for an instance.
An approximation scheme for an optimization problem is an algorithm that takes as input an instance of the problem as well as an accuracy requirement, $\epsilon > 0$. The output from the algorithm is a solution satisfying the requirement that the ratio of the heuristic solution to the optimum is no more than $1 + \epsilon$, [Garey and Johnson, 1979]. An algorithm is a polynomial approximation scheme if for each fixed $\epsilon > 0$ the derived approximation algorithm is a polynomial time algorithm in the length of the input and the accuracy required, [Garey and Johnson, 1979].

Theorem 4.3. If there exists a polynomial time algorithm for the station routing problem with a finite worst case ratio, then there exists a polynomial approximation scheme for the problem.

Proof of Theorem 4.3. See Appendix.

Rather than being an argument for the existence of a polynomial algorithm with a finite worst case ratio, Theorem 4.3 taken with the general properties of the construction techniques of Theorems 4.1 and 4.2 leads to the conjecture of the following negative result.

Conjecture. Unless $P = NP$, there does not exist a polynomial time algorithm for the station routing problem with a finite worst case ratio.

5. Empirical Evaluation of Heuristics for the Card Routing Problem

In this section we present the computational test results for fourteen heuristics. To create a testbed of problems we devised a parameterized printed circuit board generator, which produces fairly realistic random boards in accordance with certain parameter settings (e.g. size, density). The heuristics were tested on the resulting routing problems. Our overall conclusion is that fast heuristics can be moderately effective on problems with this structure, despite the theoretical worst case performance results given in section 4.

5.1 Generating Random Circuit Boards

We want to test our heuristics on a set of generated problems. In section 2 we defined the station routing problem on an arbitrary node-partitioned directed acyclic (DAG) graph, so we could just generate random DAGs. However, the precedence graphs resulting from the application we are concerned with are not purely random. We, therefore, wish therefore to impose some structure on our problem generator. To this end we construct a parameterized model of a random circuit board, and derive a precedence graph from the circuit board.

Our circuit board is rectangular. Components are placed at the points of an $nl$ by $n2$ grid, where $nl$ and $n2$ are parameters. Components come in three sizes: large, medium, and small.
Medium and small components occupy one grid point; large components occupy three consecutive grid points, either horizontally or vertically. Different components can be located anywhere, except that two large components may not be adjacent to each other. The reason for this restriction is that a gripper would not have enough clearance to insert a large component into a space if an adjacent space were taken by another large component, so such a configuration would not be feasible to assemble. (Also of course a large component cannot hang over the edge of the board.) The number of large components is set by a nonnegative integer parameter $L$. The remainder of the components are randomly chosen to be small or medium; the parameter $s$ equals the probability of being small. There is one additional parameter, a binary variable $D$, indicating whether the board is densely packed or not.

Now we describe how the precedence graph is derived from the circuit board. The main idea is that larger components interfere more with each other's insertion, and result in a denser precedence graph. Similarly, dense packings will lead to more precedence relations among the components. If two components could interfere with each other, then in general the one farthest to the right or down must be inserted second. This way, the insertion can proceed from left to right and from the top down. (This avoids directed cycles in the precedence graph.) The only exception to this rule is that a large component will take precedence over a component above it. The specific precedence relations are given in Figure 5.1. Dashed lines indicate relations only present when the parameter $D$ indicates a dense packing. Thus, for example, a pair of adjacent small components in a horizontal line have no effect on each other when the packing is not dense; the one on the left must precede the one on the right when the packing is dense.

![Figure 5.1](image)

Once the precedence graph is created, its nodes are randomly partitioned into $M$ groups, where $M$ is a positive integer parameter.

5.2 The Heuristics

Following are the descriptions of the heuristics. Most come in two versions which differ only in the tie-breaking method.

1) Simple: simply cycle through the sequence $1...M$ repeatedly until the graph disappears. Station visits which do not result in any deletions are not counted.
2) Random: (aka silly) At each iteration randomly choose among all stations for which there is at least one node available. This is equivalent to having no schedule planning at all.

3) MyopicGreedy: At each iteration choose the station containing the largest number of ready nodes. A node is ready if all of its predecessors have already been deleted.

4) Greedy: At each iteration, visit the station which would delete the largest number of nodes from the graph. Note this is not the same as MyopicGreedy, because nodes can be predecessors of other nodes in the same station.

5) Level: This is a level strategy with ties broken lexicographically.

6) CPLever: This is a critical path level strategy as defined in section 4. If there is more than one station at the highest critical level, the tie is broken lexicographically.

7) Altruist: At each iteration, choose the station which would, if visited, make the largest number of nodes become ready.

3AA - 7AA: In heuristics 3-7, ties are broken lexicographically. That is, if two partitions meet the criterion equally, the lower numbered partition is preferred. Heuristics 3AA-7AA are the same, except that the partition which would come next in the cyclic order, starting from the partition chosen the last iteration, is preferred. This is an affirmative action LRC (least recently considered) selection rule [Fathi & Tovey, 1985].

8) CPLeverGreedyAA: Apply the CPLever criterion of (6), break ties by the Greedy criterion of (4), and break any remaining ties by affirmative action LRC selection.

9) CPLeverAltruistAA: Apply the CPLever criterion of (6), break ties by the Altruist criterion of (7), and break any remaining ties by affirmative action LRC selection.

5.3 Lower Bounds on the Optimum Value

It would be nice to know the optimum solution values of the test problems as a standard to measure the heuristics against. (Of course, if we knew how to find these we wouldn't be testing heuristics.) As we remark in section 3, the problem can be solved in \(O(n^k)\) time by dynamic programming when the precedence graph is a collection of \(k\) disjoint chains. We use this fact to compute (loose) lower bounds on our test problems. Our method is to extract three long chains (long in the sense of critical length defined in section 4) from the precedence graph and solve this smaller subproblem exactly by dynamic programming. Clearly this yields a lower bound on the optimum solution value of the original problem. The larger the problem, the weaker the bound obtained by this method will tend to be.

5.4 Computational Results

The heuristics were tested on 62 randomly generated instances. The parameter values used in the generator of the instances were as follows: 1) the number of stations varied from three to five; 2) the probability of small components on a card were taken from the set \(\{0.5, 0.7, 0.9\}\); 3) both dense
and not dense packings of components on the cards were used; 4) the number of large components was in the set \{0,2,4\}; and 5) three replications were made for each of the configurations.

The number of times each heuristic produced the best solution for an instance is given in Figure 5.2. The results of the experiment showed that no heuristic dominated another, even though some heuristics clearly performed better overall. CPLevelGreedyAA and CPLevelAltruistAA computed the best solution for 63% of the instances, and they were within 1 or 2 of the best solution for 81% and 93% of the instances, respectively. Taking the minimum solution for each instance from the top five heuristics overall (SimpleLevelAA, AltruistAA, AltruistAA, and the two mentioned above), we found that in 82% of the instances at least one of the five determined the best solution.

In the absence of any clear winner or small dominating set of heuristics, one would like to be able to provide some insights into how the parameter settings affect comparative heuristic performance. For example, perhaps CP strategies tend to improve in comparison with level strategies as board densities increase, or perhaps affirmative action strategies do relatively better as the number of stations increases. Unfortunately, our testing (including some cases not reported here) did not reveal any such patterns.

Because of the intractability of computing the optimal solutions, the only relative measure for comparing the performance of the heuristics is against the lower bound. It is important to note that as the number of machines and the density of the graph increases, the gap between the lower bound and the optimum tends to increase. The ratio of the best solution over all heuristics over the lower bound was computed for each instance. The number of times this ratio fell within an interval is plotted in Figure 5.3. The plot shows that the ratio of the best solution to the lower bound was 1.5 or less for 95% of the instances. Strangely enough, the one instance where the ratio was equal to 2 was for a sparse graph.

Figure 5.2
The station routing problem for printed circuit board assembly has been defined, and studied both analytically and empirically. We have shown that the problem is computationally intractible, even when the precedence graphs are composed of disjoint chains; that a set of intuitively appealing strategies has unbounded worst case performance; and that if there is a polynomial heuristic with bounded worst case performance then there is a polynomial approximation scheme for the problem. Together, these results support the conjecture that there is no polynomial heuristic with a finite worst case bound, i.e., the station routing problem is theoretically quite difficult.

Our experimental analysis of fourteen heuristics indicates that virtually any "sensible" heuristic will perform best for some problem instances. Because the spread between the various heuristics is fairly large, and no heuristic dominates (or is dominated), the best solution strategy may be to compute a large number of solutions and pick the best. On graphs arising from our model of printed circuit boards, the computational results imply that this strategy is at least moderately effective.

The station routing problem is a particular process planning problem (see, e.g., [1] or [2] on process planning in general and computer aided process planning). It lies at the interface between product design and assembly system operation. The given configuration of the assembly system taken together with printed circuit board design determines whether or not the station routing problem will be easy or hard to solve.

Our results indicate that, insofar as possible, product design should eliminate insertion precedence constraints, either through board layout or alternative component technology. Alternatively, process innovations may eliminate some insertion precedences. Furthermore, the configuration of the assembly system should attempt to simplify the station routing problem. If the nodes could be assigned to stations in any desired way, only subject to cardinality constraints on the $V_p$, obviously one could make assignments so that no station had to be visited twice. Since assembly systems typically produce dozens or hundreds of different circuit boards simultaneously, such an outcome will not be possible for most boards. On the other hand when there are a significant number of precedence constraints (densely populated board) and the assignment of the components required by a card to stations appears random, even the optimal solution will require a large number of station visits. For a fixed amount of work-in-process, longer sequences of station visits imply greater board flowtimes, thus smaller throughputs.

Finally, we note that the problem of specifying the insertion precedences is itself an interesting, and so far as we know, virgin problem. Given a board layout and a set of insertion technologies, it is not at all obvious how the precedence graph should be constructed.
Proof of Theorem 4.3

Theorem 4.3, in effect, states if there exists a polynomial algorithm for the station routing problem that has a finite worst case ratio, then this algorithm can be used to get arbitrarily close to the optimum. For any desired accuracy, the amount of time required by the algorithm is polynomial in the size of the instance.

Theorem. If there exists a polynomial time algorithm for the station routing problem with a finite worst case ratio, then there exists a polynomial approximation scheme.

The idea of the proof is as follows: Assume there exists a polynomial algorithm with a worst case ratio of $Q$. Given an instance $I$ of the problem, we "square" it to produce a larger instance, denoted $I^2$. Our "squaring" process has the property that any feasible solution to $I$ of length $L$ can be converted into a feasible solution to $I^2$ of length $L^2$. Conversely, from any feasible solution to $I^2$ of length $L'$, we can extract a feasible solution to $I$ of length $\leq \sqrt{L'}$. It will follow from these properties of $I^2$ that if we apply the polynomial algorithm with worst case bound of $Q$ to $I^2$, we get a solution to $I$ with worst case bound $\sqrt{Q}$.

Since the construction technique can be recursively applied, given an accuracy requirement, an instance can be constructed by repeating the construction process $p$ times, where $p = \lceil \log \log Q - \log \log(1 + \epsilon) \rceil$. (All logs are base 2.) This guarantees the heuristic solution for the original instance is within $1 + \epsilon$ of the optimal solution.

Proof of Theorem 4.3. Assume there exists a polynomial algorithm for the card routing problem with a finite worst case ratio of $Q$, $Q > 1$.

An instance of the station routing problem consists of an acyclic, directed graph, $G(V,E)$, and a partition of the nodes, $V$, into stations $\{1,\ldots,M\}$. We say an instance is proper if $G$ has no edges that connect a pair of vertices in the same station. Before the squaring operation is performed, a preprocessing of $G(V,E)$ is required. The preprocessing converts any instance into an equivalent proper instance. Let $(i,j) \in E$ and vertices $i$ and $j$ both be in the same station $V_k$. The preprocessing step removes $(i,j)$ from $E$, and adds $(k,j)$ if $(k,i) \in E$. The preprocessing step is illustrated in Figure A.1.

**Proposition:** Let $G' = (V,E')$ be the graph resulting from a preprocessing of $G = (V,E)$. Then the new instance on $G'$ is equivalent to the original instance on $G$.

**Proof:** It is immediate that any feasible solution for $G$ is feasible for $G'$, and vice-versa. Q.E.D.
To illustrate the squaring construction, consider the graph in Figure A.2. The letter labeling each vertex corresponds to the station where the node has been assigned. The "squared" graph, \( G^2 \), is illustrated in Figure B.3. This example should make the general idea clear. Next we specify the "squaring" procedure precisely and define some terminology needed later in the proof.

In the construction of \( G^2 \), each vertex in \( G \) is expanded into a copy of the original graph \( G \).

1. The vertex set \( V^2 \) of \( G^2 \) is therefore the cartesian product \( V \times V \), and members of \( V^2 \) are denoted by an ordered pair \( (i,j) \) written \( ij \) for ease. The stations of \( G^2 \) are the \( M^2 \) subsets of \( V \times V \) induced by the \( M \) subsets of \( V \). Thus if \( i \) is in station \( A \) and \( j \) is in station \( B \), then in \( G^2 \) the node \( ij \) is in station \( AB \). Our convention for the vertex \( ij \) of \( G^2 \) is that the first term, \( i \), denotes the vertex in the original graph \( G \) which has been expanded into a copy of \( G \), called copy \( i \). The second term \( j \) identifies the particular node in copy \( i \). If in the original graph node \( i \) is in station \( X \), then copy \( i \) will be called an \( X \)-copy. For example, Figure A.4 gives a \( C \)-copy of the graph in Figure A.2.
(2) We have now described the vertex set of $G^2$ and its partition into $M^2$ stations. The edge set $E^2$ of $G^2$ is as follows: (i) Within each copy replicate the original graph $G$. That is, for every copy $i$, $(ij,ik) \in E^2$ if $(ij) \in E$. (ii) Between copies of $G$, put in all possible edges if there was an edge between the nodes the copies are expanded from. That is, for every copy $i$ and copy $j$, if $(ij) \in E$ then $(ix,jy) \in E^2$ if $x \neq y$.

In Figure A.3, the edges between copies of $G$ are denoted by the thick arrows between copies. We make the important observation that if $G$ is proper, so is $G^2$.

![Figure A.3. The Square of Figure A.2](image)

![Figure A.4. C-Copy of Figure A.2](image)

A feasible sequence of machine visits that completes the constructed instance is *orderly* if the sequence always completes the deletion of $X$-copy before starting a $Y$-structure, $\Delta X \otimes Y \subseteq Y$.

A sequence that possesses this orderly property will be referred to as an *orderly sequence*. Figure A.5 illustrates an orderly sequence for the constructed instance in Figure A.3.

$$(AA,AB,AC,AB,AA,CA,CB,CC,CB,CA,BA,BC,BB,BA,AA,AC,AB,AA)$$

note to readers: this is Figure A.5.
A feasible sequence of machine visits is *non-orderly* if it is not orderly. Figure A.6 illustrates a non-orderly sequence for the constructed instance in Figure A.3.

\[(\text{AA,AC,AB,AA,BA,BC,BB,CA,BA,CC,CA,BA,BC,BB,BA,AA,AC,AB,AA})\]

**Figure A.6. A Non-Orderly Sequence**

**Claim 1.** Let S be a feasible non-orderly sequence for $G^2$ and $L(S)$ be the length of the sequence. There exists a rearrangement of S, S' such that S' is orderly, $L(S) = L(S')$, and S' is feasible.

**Proof of Claim 1.** As the sequence S is applied to $G^2$, the first copies to be completely deleted must all be X-copies for some X (because the vertices in a station XY are all in X-copies). If S is orderly until this point, the claim follows inductively. If not, truncate S at this point, giving T. T must contain a subsequence of form $(XY_1,XY_2,\ldots,XY_t)$ where $(Y_1,\ldots,Y_t)$ is feasible for G. Rearrange T so that this subsequence is pushed up to the beginning, forming T'. We will show that any vertex deleted by T is also deleted by T', which will prove the claim.

Every station visit of T' to WZ, WTX, must leave the same (or more) vertices deleted since these visits have been postponed, (so more predecessors may have been deleted). The key is to show that the visits which have been moved earlier delete the same set of vertices. Let $ij$ be any vertex in $G^2$, where i is in station X. If $ij$ is deleted by T all its predecessors must have been deleted as well. Since the first copies to be deleted were X-copies, and edges between copies go from every node in one copy to every node in the other, $ij$ cannot have any predecessors in any Y-copies, for any $YTX$. Since the only rearrangements in T' were the postponements of station visits to Y-copies, $YTX$, T' must delete $ij$ as well. Q.E.D.

An example of the reordering operation is illustrated by examining Figure A.6. The underlined machine visits in Figure A.6 are "out of order": interchanging them yields an orderly sequence. Moving the BA ahead of the CA does not affect feasibility, because the CA does not complete the deletion of a C-copy. The ordering of a non-orderly sequence can clearly be performed in polynomial time.

The convolution of a sequence of machine visits is defined analogously to the operation of squaring the graph. It is best illustrated with an example. Let $(ABCA)$ be a sequence of station visits. The convolution of $(A,B,C,A)$ is $(\text{AA,AB,AC,AA,BA,BB,BC,BA,CA,CB,CC,CA,AA,AB,AC,AA})$
Claim 2. Given a feasible sequences of station visits for the original instance a feasible sequence for the squared instance can be created by taking the convolution of S.

Proof of Claim 2. This follows from the construction and the fact that G is proper.

The feasibility of a convolution can be illustrated with an example. The sequence (A,C,B,A) is a feasible sequence for the instance G in Figure A.2. The convolution of this sequence is (AAACABAA; CACCCBCA; BABCBBBA; AAACABAA). The feasibility of this sequence can be verified by applying the sequence to G² in Figure A.3.

As in claim 1, we let L(S) denote the length of a sequence S.

Claim 3. If S is feasible for G² then there exists R feasible for G of length L(R) \leq \sqrt{L(S)}. Moreover, R can be extracted from S in polynomial time.

Proof. By claim 1, S may be taken to be orderly. Thus S begins with a sequence of visits of form XY₁, XY₂, ..., XYₜ. This sequence of visits, which we call an X-block, deletes all X-copies that have no predecessor copies, because Y₁, Y₂, ..., Yₜ is feasible for G. As S is orderly, it consists of a sequence of X-blocks for varying X's. Call this sequence of X's the master sequence of S. From our construction, the master sequence of S must be a feasible solution for G, and can be created easily from S.

Now let L₁(S) = the length of the shortest block in S.
L₂(S) = the length of the master sequence of S.

Obviously then L(S) \geq L₁(S)L₂(S) (since S is a set of blocks arranged according to the master sequence).

Therefore

\[ \text{Min}\{L₁(S),L₂(S)\} \leq \sqrt{L(S)}. \]

The left hand side of this inequality is the length of an easily extracted feasible solution for G.

Q.E.D.

Corollary: If R* is an optimal solution for G then its condition is optimal for G².

Proof: By claim 2, the convolution of R* is feasible for G². If the convolution were not optimal for G² there would exist S feasible for G² with L(S) < (L(R*))². By claim 3 this would imply the existence of an R feasible for G with L(R) \leq \sqrt{L(S)} < L(R*) contradicting the optimality of R*. Q.E.D.

We now finish the proof of Theorem 4.3. Let \( \epsilon > 0 \) and \( Q > 1 \) be fixed arbitrarily. Let \( p = \left[ \log \log Q - \log(1 + \epsilon) \right] \) (logs are to base 2). Then \( 2^p \log(1 + \epsilon) \geq \log Q \) and \( Q^{1/2} 1^p \leq 1 + \epsilon. \)
For any graph G, we preprocess to make it proper, then square it p times, resulting in $G^{2^p}$.

By hypothesis, there is a polynomial algorithm which applied to $G^{2^p}$ produces a solution S with performance bound Q. That is, $L(S)/L(S^*) \leq Q$ where $S^*$ solves $G^{2^p}$ optimally. By our observation that $G^2$ is proper, we may apply claim 3 p times to S to extract R, a feasible solution to G, with $L(R) \leq L(S)^{1/2^p}$. By the corollary to claim 3, where $R^*$ optimally solves G, $L(R^*) \leq L(S^*)^{1/2^p}$. Hence

$$\frac{L(R)}{L(R^*)} \leq \left(\frac{L(S)}{L(S^*)}\right)^{1/2^p} \leq Q^{1/2^p} \leq 1 + \epsilon$$

Moreover, since $p$ is constant, R is computed in polynomial time. Q.E.D.

6. References


VOTING SCHEMES FOR WHICH IT CAN BE DIFFICULT TO TELL WHO WON THE ELECTION

by

John J. Bartholdi, III
Craig A. Tovey
Michael A. Trick

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
404-894-3036

Abstract

We show that a voting scheme suggested by Lewis Carroll can be impractical in that it can be computationally prohibitive (specifically, NP-hard) to determine whether any particular candidate has won an election! We also suggest a class of "impracticality theorems" which say that any fair voting scheme must require excessive computation in the worst-case.

January 1987. Presented at Purdue University, March 1987; at the University of Arizona, April 1987; at Massachusetts Institute of Technology, April 1987.

This research was supported in part by Presidential Young Investigator Awards from the National Science Foundation to the first two authors (ECS-8351313 and ECS-8451032), and by grant N00014-86-K-0173 from the Office of Naval Research.
1. Introduction. We can consider a voting scheme to be an algorithm that takes as input any set C of candidates and any set V of transitive preference orders on C (the preferences of the electorate), and outputs a subset of C, who are the winners (allowing for ties). When a voting scheme is considered to be an algorithm, it is natural to ask about the computational resources required by the scheme. For example, can it be guaranteed to quickly identify a winner of the election?

For both practical and theoretical reasons, an algorithm is considered formally efficient if it requires a number of computational steps that is at most polynomial in the size of the problem. Problems for which there are polynomial-time algorithms are generally considered to be tractable, and those which can require exponential time to solve are considered inherently intractable.

Computational complexity can classify voting schemes based on a well-studied hierarchy of complexity classes that are thought to be distinct. For example, within this hierarchy is the problem class NP, which consists of those questions for which a "yes" answer can be justified in polynomial time. The hardest problems in NP are known as "NP-complete", and all such problems are equivalent in the sense that any problem in NP can be reworded as an instance of an NP-complete problem within polynomial time. Thus if there exists an algorithm to solve any NP-complete problem within guaranteed polynomial time, then all NP-complete problems could be solved by transforming them to instances of the easily solvable problem. That no one has found such an algorithm is taken as strong circumstantial evidence that NP-complete problems are inherently intractable. (For more on complexity and NP-completeness, see [6].)

We will show that, for two voting schemes, several natural questions about the outcome of an election are NP-complete, which suggests that they can be too difficult to answer in practice, and that the voting schemes are therefore impractical.

2. The difficulty of tabulating scores. For any voting scheme the essential question to be answered can be formalized as follows.

ELECTION WINNER

INSTANCE: Set of candidates C, and one distinguished member c of C; set V of transitive, strict preference orders on C.

QUESTION: Is c a winner under the specified voting scheme?
In most historical voting schemes only polynomial time is required to answer this question (a practical necessity when counting paper ballots!). For example, to solve PLURALITY-WINNER requires only $O(|V| + |C|)$ work to count first-place votes and identify the candidate with the most. It is unusual then to discover voting schemes that can apparently require exponential time to tell whether any particular candidate has won the election. We exhibit two such schemes. One was invented by the mathematician Charles Dodgson (better known as Lewis Carroll), and the other was suggested by J. Kemeny. We show that under a either a Dodgson election or a Kemeny election, it is NP-hard to determine whether any particular candidate has won! Thus these schemes are capable of a worst-case behavior that is novel among voting schemes: it can take an impractically long time to determine a winner.

(Others have observed the empirical difficulty of computing winners under these schemes [for example, 5]. However, we establish this in a formal sense, and suggest computational complexity as another criteria by which practical voting schemes should be judged.)

The rationality criterion that troubled Dodgson was the famous one first formalized by the Marquis de Condorcet [3], which requires that a voting scheme elect any candidate (the "Condorcet winner") that would defeat any other candidates in a pairwise election with simple vote counts. Condorcet observed that there can be elections in which no candidate is a Condorcet winner (the "Phenomenon of Cyclic Majorities"). Accordingly, Dodgson sought a voting scheme that would still choose "rationally" in the absence of a Condorcet winner. He suggested the following voting scheme (reprinted at length in [2]; summarized in [11]).

THE DODGSON VOTING SCHEME:

A Dodgson winner is a candidate who is "closest" to being a unique Condorcet winner, where by "closest" we mean the following: Imagine that an election supervisor is empowered to change the ballot of any voter through pairwise interchange of candidates adjacent in the voter's preference order. Then a Dodgson winner is a candidate who requires the fewest interchanges to become a Condorcet winner. (Hereafter we will refer to such pairwise interchanges simply as "switches". The minimum number of switches for a candidate to become a Condorcet winner is the Dodgson score of that candidate. A candidate with the smallest Dodgson score is a winner of the election.)
Dodgson described a winner of the election, but did not completely specify an algorithm to identify a winner. We show that any conceivable algorithm to do this apparently must require excessive time, at least in the worst-case.

In showing that it can be difficult to tell any candidate whether he has won the election, we develop two supporting lemmas. The first and key lemma says that it can be difficult to tell whether a candidate did well. We formalize this question as

DODGSON SCORE
INSTANCE: Set of candidates C, and a distinguished member c of C; set V of transitive, strict preference orders on C; a positive integer K.
QUESTION: Is the Dodgson score of candidate c less than or equal to K?

LEMMA 1. DODGSON SCORE is NP-complete.

Proof. First observe that DODGSON SCORE is in NP, since a "yes" answer can be justified in polynomial time by identifying appropriate switches and tabulating the vote.

Now we contrive an election for which determining the winner entails solving EXACT COVER BY 3-SETS (X3C), which is known to be NP-complete [6].

EXACT COVER BY 3-SETS (X3C)
INSTANCE: Set B with |B| = 3q and a collection S of 3-element subsets of S.
QUESTION: Does S contain an exact cover for S, i.e., a subcollection S' of S such that every element of B occurs in exactly one member of S'?

First we define the set C of candidates: For each element of B, create two corresponding candidates b_i, and f_i. Similarly, we create a candidate s_j for each S_j in S.

Now we devise the set V of voters (which are identified with their preferences). V consists of the following subsets:

1. Swing voters. Create voters corresponding to members of S: For each \{ b_{i1}, b_{i2}, b_{i3} \} create a single voter (b_{i1} > b_{i2} > b_{i3} > s_j > c > ...), where the entries after c are in arbitrary order. We call these voters "swing voters", since their votes will be crucial to the result of the election. Note that among swing voters, to get votes for c over members of B requires at least 4/3 switches per vote on the average, and to
achieve this, any voter who switches at all must switch c upward 4 times, to the very top of his preference order.

The swing voters are the means by which we embed X3C in DODGSON SCORE. However we must pad the electorate with additional voters to make sure that this embedding captures all the difficulty of X3C:

2. Equalizing voters. Let $N_i$ be the number of votes from swing voters that $b_i$ would get in a pairwise election against $c$; let $N_{\text{max}}$ be the largest $N_i$. For each $b_i$ create $N_{\text{max}} - N_i$ additional (identical) voters ($b_i > f_1 > c > ...$) so that each $b_i$ would get exactly $N_{\text{max}}$ votes in a pairwise election against $c$. We call these "equalizing voters" since they make all the $b_i$ score equally well against $c$ among swing voters and equalizing voters. Among equalizing voters, to get votes for $c$ over members of $B$ requires at least 2 switches per vote on the average.

3. Incremental voters. Finally create a class of identical voters ($b_1 > ... > b_{|B|} > f_1 > ... > f_{|B|} > c > ...$) sufficient in number so that any candidate $b_i$ would defeat $c$ by exactly 1 vote. Among incremental voters, to get votes for $c$ over members of $B$ requires at least 2 switches per vote on the average.

Now that we have defined the election, consider whether $c$ can be made a Condorcet winner by no more than $4|B|/3$ switches. If he can, he must convince the electorate to prefer him to each of the $b_i$; but this requires at least $4|B|/3$ switches, and is achievable only if i) all switches are among swing voters, and ii) each swing voter makes 4 switches, to move $c$ to the top of his preferences. But any set of swing voters that can elect $c$ by making no more than $4|B|/3$ switches corresponds to an exact 3-cover of $B$ by members of $S$.

The second supporting lemma says that it can be hard to compare the scores of two candidates, which problem we formalize as

**ELECTION RANKING**

**INSTANCE:** Set of candidates $C$, and distinguished members $c$, $c'$ of $C$; set $V$ of transitive, strict preference orders on $C$.

**QUESTION:** Did $c$ defeat $c'$ in the election?
LEMMA 2. *DODGSON RANKING* is NP-hard.

*Proof.* Assume the construction from the proof of Lemma 1, with the additional stipulations (made without loss of generality) that |B| is even and the total number of voters is odd. Now add a new candidate c' to everyone’s preference orders. To do this, first distinguish a single, arbitrarily chosen equalizing voter v. Divide the remaining voters into two arbitrary groups of equal size. All the voters in one group insert c' at the very top of their preferences; all the voters in the other group insert c' at the very bottom of their preferences. Voter v inserts c' in position $1 + 4|B|/3$ in his preference order.

The Dodgson score of c' is no more than $4|B|/3$: without the vote of v, c' must tie every other candidate in pairwise elections, so that making $4|B|/3$ switches to move c' to the top of v's preferences will enable c' to defeat all the other candidates. On the other hand, the Dodgson score of c' must be at least $4|B|/3$ since at least that many candidates defeat him in pairwise elections (exactly those candidates preferred to him by v). Thus the Dodgson score of c' is exactly $4|B|/3$. By Lemma 1, determining whether the score of c is as small is NP-complete.

Since DODGSON RANKING is as hard as an NP-complete problem, but is not known to be in NP, we conclude that it is NP-hard.

THEOREM 1. *DODGSON WINNER* is NP-hard.

*Proof.* It is straightforward but tedious to pad the contrived election (from Lemmas 1 and 2) with a polynomial number of additional voters so that c and c' (and no others) are tied for first place.

We think Lewis Carroll would have appreciated the idea that a candidate's mandate might have expired before it was ever recognized.

3. Efficiency at the cost of universality. The Dodgson scheme can be made at least formally efficient if we place an *a priori* restriction on the size of the elections in which it will be employed. For example, if we bound in advance the number of voters, then we
can determine the Dodgson winner in polynomial time by enumeration: Any particular
candidate c can be permuted to at most |C| different positions in the preference order of
a voter, so there are at most |C|^|V| possible ways of placing c in the preferences of the
electorate. We can count the number of switches implicit in each of these ways, and the
smallest is the Dodgson score of c. Finally we can compare scores and choose the
smallest. If |V| is bounded by a constant, then this procedure is technically polynomial-
time (even though this might not be reassuring for large |V|).

Similarly, if we bound in advance the number of candidates, the Dodgson scheme can
be made to run in polynomial time by solving an integer linear program with a very large
but fixed number of variables and constraints. The problem of determining the Dodgson
score of candidate c can be formulated as an integer linear program in the following
way. Index by i the types of preference orders found among the voters, and let N_i be
the number of voters of type i. Let x_{ij} be the number of voters with preferences of
type i for which candidate c will be moved upwards by j positions. Let e_{ijk} be 1 if the
result of moving candidate c to position j in a preference order of type i is that c gains
an additional vote against candidate k, and 0 otherwise. Let d_k be the deficit of c with
respect to candidate k, that is, the minimum number of votes that c must gain against k
to defeat him in a pairwise election. If c already defeats k, then d_k = 0. Then the
Dodgson score of c is the value of integer linear program.

\[
\begin{align*}
\min & \sum_{ij} x_{ij} \\
\text{subject to} & \\
\sum_i x_{ij} = N_i & \quad \text{(all types i of preference orders)} \\
\sum_{ij} e_{ijk} x_{ij} \geq d_k & \quad \text{(all candidates k)} \\
x_{ij} \geq 0, \text{ integer}. &
\end{align*}
\]

The first set of constraints restricts the numbers and types of preferences to those
actually present among the voters, and the second set of constraints ensures that c will
become a Condorcet winner. The objective is to minimize the number of switches.

The number of different types of voters (preference orders) is no greater than |C|!,
and the number of different positions in any preference order is |C|. Consequently there
are no more than |C|x(|C|!) variables x_{ij} and no more than |C|! + |C| + |C|^2 constraints.
If we limit the applicability of the Dodgson scheme by restricting |C| to be no larger
than some prespecified number, (3.1) is polynomially solvable, at least by the algorithm of
Lenstra [10], for which the time bound, though potentially enormous, is technically a
polynomial. (Since the number of constraints and the number of variables are fixed, and
only the right-hand side is variable, there might be more efficient ways of solving (3.1).

The effort required to determine a Dodgson winner appears to increase more quickly as a function of $|C|$ than as a function of $|V|$. Even in real elections $|C|$ can be large enough to make the Dodgson scheme potentially impractical: the *New York Times* of 1 April 1986 reported 20 candidates for mayor of Tulsa, Oklahoma!

4. An "impracticality theorem". Kemeny [8, 9] has suggested another voting scheme that extends the Condorcet principle. He defined the outcome of an election to be a consensus ranking of the alternatives, and suggested that the consensus be a preference order that minimizes the sum of "distances" to the preferences of the voters. We show that it can be difficult to determine the outcome of an election under Kemeny scoring. As a corollary we conclude that every voting rule that satisfies certain modest fairness criteria must be inefficient at determining a winner.

Kemeny defined the "distance" between two preferences $P$ and $P'$ as $\text{dist}(P,P') = \sum d(j,k)$, where the sum is taken over all unordered pairs of candidates $j$ and $k$, and where $d(j,k) = 0$ if $P$ and $P'$ agree on candidates $j$ and $k$; $d(j,k) = 2$ if $P$ prefers $j$ to $k$ but $P'$ prefers $k$ to $j$; and $d(j,k) = 1$ if $P$ prefers $j$ to $k$ but $P'$ is indifferent between $j$ and $k$. A Kemeny consensus is a preference that minimizes $\sum N_i \text{dist}(P,P_i)$, where $N_i$ is the number of voters with preference $P_i$.

We will need the following technical result, which enables us to consider only strict preferences.

**Lemma 3.** If all voter preferences are strict, then there exists some Kemeny consensus that is strict.

**Proof.** Let $P$ be a Kemeny consensus that includes ties (indifference), and consider any set $T$ of candidates that are mutually tied under $P$. Let $c$ be any candidate in $T$ and compute $A = \sum \{|\text{voters who prefer } c \text{ to } d|\}$ and $B = \sum \{|\text{voters who prefer } d \text{ to } c|\}$, where the sums are taken over all $d$ in $T$. If $A > B$, then the Kemeny score of $P$ could be improved by breaking ties in favor of $c$; similarly, if $A < B$, the Kemeny score could be improved by preferring the remaining candidates of $T$ to $c$. But since $P$ is a Kemeny consensus, its score must be minimum, so that $A = B$. Finally, since $A = B$, we can break ties arbitrarily in favor of $c$ to produce a new preference order with fewer ties,
but with the same minimum Kemeny score. Repeated application of this produces a Kemeny consensus with no ties.

**THEOREM 2.** **KEMENY SCORE** is **NP-complete**, and **KEMENY RANKING** and **KEMENY WINNER** are **NP-hard**.

**Proof.** **KEMENY SCORE** is in NP since the score of any candidate can be computed in polynomial time.

We show the problem is hard by embedding within it the following problem, which is known to be NP-complete [6].

**FEEDBACK ARC SET**

**INSTANCE:** Directed graph G, with vertices C; positive integer K.

**QUESTION:** Is there a subset of no more than K arcs which includes at least one arc from every cycle in G?

For any instance of **FEEDBACK ARC SET**, interpret G as representing the outcomes of pairwise contests between candidates C. By [13], G is realizable by a set V of voters such that |V| is even and "small" (polynomial in |C|), whose preferences are all strict, and whose preferences decide each contest by exactly 2 votes. Thus, for any arc (i,j) of G, exactly (|V|+1)/2 voters prefer candidate i to j, and (|V|-1)/2 voters prefer candidate j to i. Therefore any strict preference P must disagree with at least (|V|-1)/2 voters on the relative ranking of candidates i and j, and so must incur a "fixed-cost" of (|V|-1) to its Kemeny score. If P disagrees with the majority of voters and prefers j to i, then there is an additional penalty of (2 voters)(2 points/voter) = 4 points. Since this holds for each of the |V|(|V|-1)/2 pairs of candidates, the Kemeny score of P must be at least |V|(|V|-1)(|V|-1)/2, plus 4 times the number of contests in which P disagreed with the majority.

Now consider the question whether there exists a consensus whose Kemeny score is no larger than |V|(|V|-1)(|V|-1)/2 + 4K. By Lemma 3, the answer is "yes" if-and-only-if there exists a strict preference with the same Kemeny score. By construction, a strict consensus with this score must agree with the majority on all but K of the pairwise contests, and the arcs corresponding to these pairwise contests are a feedback arc set for G. Therefore **KEMENY SCORE** is NP-complete.
As for the Dodgson scheme, the election can be padded to establish that KEMENY RANKING and KEMENY WINNER are both NP-hard.

Now we reword Theorem 2 in a provocative way. Following [14] we define a voting scheme to be neutral if it is symmetric in its treatment of candidates; to be Condorcet if it elects any Condorcet winner; to be consistent if, when two disjoint subsets of the electorate, voting separately, arrive at the same consensus, then their voting together always produces this same consensus. Young and Levenglick [14] proved that Kemeny scoring is the unique voting scheme that is neutral, consistent, and Condorcet. Hence we have the following.

COROLLARY. Under any voting scheme that is neutral, consistent, and Condorcet, the WINNER PROBLEM is NP-hard.

Since only the Kemeny rule satisfies the hypotheses, this corollary is not entirely satisfying. Nevertheless, it can be taken as a model for a new type of impossibility theorem (that might be called an "impracticality theorem"), the general form of which is "Fair elections are impractical". Are there stronger versions of this theorem?

5. Conclusions. Many theorems and some practical experience attest that any conceivable voting scheme is capable of some form of unacceptable behavior, such as violating formalized notions of fairness or rationality [for example: 4, 7, 11; summary in 12]. We suggest computational complexity as an additional worst-case behavior by which to judge voting schemes.

It seems desirable for a voting scheme be dependably quick in its decisions. It would be interesting to explore the extent to which this is reconcilable with notions of fairness. In particular, do there exist more potent impracticality theorems than the one we offer?

In addition, there are other computational aspects of voting to be explored. For example, elsewhere we have exhibited a voting scheme that is easy to operate, but is computationally resistant to manipulation [1].
Finally, we note that computational complexity of voting schemes might become a more immediate issue as society increasingly uses computers to conduct its elections.
REFERENCES


THE COMPUTATIONAL DIFFICULTY OF MANIPULATING AN ELECTION

by

John J. Bartholdi, III
Craig A. Tovey
Michael A. Trick

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
404-894-3036

Abstract

We show how computational complexity might protect the integrity of social choice. We exhibit a voting rule that efficiently computes winners but is computationally resistant to strategic manipulation. It is NP-complete for a manipulative voter to determine how to exploit knowledge of the preferences of others. In contrast, many standard voting schemes can be manipulated with only polynomial computational effort.

January 1987. Presented at Purdue University, March 1987; at the University of Arizona, April 1987; at Massachusetts Institute of Technology, April 1987.

This research was supported in part by Presidential Young Investigator Awards from the National Science Foundation to the first two authors (ECS-8351313 and ECS-8451032), and by grant N00014-86-K-0173 from the Office of Naval Research.
1. Introduction. We consider a voting scheme to be an algorithm that takes as input a set C of candidates and a set V of transitive preference orders on C (the preferences of the electorate), and outputs a subset of C, who are the winners (allowing for ties).

Several celebrated theorems [3, 5, 11] show that any voting scheme that meets certain innocuous-looking rationality criteria must be susceptible to strategic manipulation. That is, a voter with complete knowledge of the preferences of all the other voters might be able to achieve a social choice more to his liking by misrepresenting his preferences.

Like the famous theorem of Arrow, these "impossibility" theorems fascinate because they seem to confound our ideals. They suggest that our methods of social choice must be either unfair or else inherently susceptible to abuse. It is the contribution of this paper to suggest how such a gloomy prospect can be ameliorated on operational grounds: We exhibit a voting scheme that is computationally resistant to manipulation even though it is easy to compute the winner. While manipulable in principle, this scheme might not be manipulable in practice - even assuming free and perfect information - because of excessive computational requirements. This is different from the idea of [10] that degrees of manipulability be distinguished by their information requirements. Instead, we make distinctions based on the time required to process that information.

First we show that many of the well-known and historically used voting schemes can be "efficiently manipulated" in the sense that only polynomial time is required to determine how to vote so as to exploit knowledge of the preferences of other voters. Secondly, we prove that it is NP-complete to determine how to manipulate a scoring method that is in practical use (by, for example, the International Federation of Chess).

2. Computational complexity. For those not familiar with computational complexity, we provide a quick sketch of issues and terms. We urge the reader to consult [4] for more detail.

For both practical and theoretical reasons, an algorithm is considered formally efficient if it requires a number of computational steps that is at most polynomial in the size of the problem. Problems for which there are polynomial-time algorithms are generally considered to be tractable, and those which can require exponential time to solve are considered inherently intractable.

The difficulty of a problem can be measured by the worst-case time to solve an
instance. Complexity theorists have identified a hierarchy of "complexity classes" into which problems might fall. For example, the problem class NP consists of those questions for which a "yes" answer can be justified in polynomial time, and the hardest problems in NP are known as "NP-complete". NP-complete problems are all equivalent in the sense that any problem in NP can be reworded as an instance of an NP-complete problem within polynomial time. Thus if there exists an algorithm to solve any NP-complete problem within guaranteed polynomial time, then all NP-complete problems could be solved by transforming them to instances of the easily solvable problem. Since no algorithm of guaranteed polynomial time has been found for any NP-complete problem, this is widely taken as evidence for their inherent intractability.

3. Many voting schemes are easy to manipulate. Consider a fixed voting scheme, and suppose that a manipulative voter knows in advance the preferences of every other voter; is there a preference ordering (possibly different from his true preferences) that the manipulator can adopt so that a specified candidate c is a winner? We formalize the problem of manipulation as:

SINGLE-VOTER MANIPULATION

INSTANCE: Set of candidates C, and a distinguished member c of C; set V of transitive preference orders on C.

QUESTION: Does there exist a preference order P that will ensure that c will be the winner?

If this can be answered within polynomial time, then the voting scheme is "easily manipulable". We show that many commonly-used voting schemes are easily manipulable by the following simple procedure.
ALGORITHM GREEDY.

INPUT: preferences of all other voters, and a distinguished candidate c.

OUTPUT: either a preference order that, together with those of all the other voters, will ensure that c is a winner, or else a claim that no such preference order exists.

INITIALIZATION: Place c at the top of the preference order.

ITERATIVE STEP: Determine whether any candidate can be placed in the next lower position (independent of other choices) without preventing c from winning. If so, place such a candidate in the next position; otherwise terminate claiming that c cannot win.

The following describes voting schemes that are manipulable by GREEDY. Assume that the preferences of all the voters but the manipulator are fixed and known to the manipulator. For any preference P and candidates i, j, let iPj mean that i is preferred to j under P, so that \{j:iPj\} is the set of candidates to whom i is preferred under P.

THEOREM 1: GREEDY will find a preference order P that will make candidate c a winner (or conclude that it is impossible) for any voting scheme that can be represented as scoring function S(P):C \rightarrow R that is both
- "responsive": a candidate with the highest score, max_i S(P,i), is a winner; and
- "monotone": for any two preference orders P and P' and for any candidate i, \{j:iP'j\} \subseteq \{j:iPj\} implies that S(P,i) \geq S(P',i).

Proof: First note that if GREEDY successfully constructs an order, that order will guarantee victory for candidate c. Now we show that if an order exists that ensures victory for c, then GREEDY will construct it. Suppose GREEDY terminates without having constructed a preference order, and let U be the set of unassigned candidates when GREEDY terminates. Let P' be an order which would enable c to win, and let u be the highest ranked member of U under P'. Consider any completion P of the preference order started by GREEDY that places u in the highest unassigned place. By responsiveness, S(P',c) \geq S(P',u), and because \{j:uPj\} is contained in \{j:uP'j\}, S(P',u) \geq
S(P,u). Furthermore, by the initialization of the algorithm and by monotonicity, c has the highest score that he can get under any preference order, so $S(P,c) \geq S(P',c)$. These inequalities imply $S(P,c) \geq S(P,u)$. But, by the iterative step of GREEDY, $S(P,c) < S(P,u)$ since u cannot go in the assigned slot. This is a contradiction.

**COROLLARY:** Any voting system that satisfies the conditions of Theorem 1, and for which $S$ is evaluable in polynomial time, can be manipulated in polynomial time.

**Proof:** GREEDY executes within polynomial time since no more than $n$ iterations are required, and each iteration requires no more than $n$ evaluations of $S$ (by monotonicity of the scoring function), with each evaluation of $S$ requiring only polynomial time (by assumption).

Many voting schemes in common use satisfy the conditions of Theorem 1 and so are efficiently manipulable by GREEDY:

**Plurality** (Each voter casts 1 vote for their most preferred candidate). Scoring function: Let $b_i$ be the plurality score of candidate $i$ among all voters except the manipulator; then $S(P,i) = b_i + 1$ if $|\{j : iPj\}| = |C| - 1$, else $= b_i$.

**Positional** (Borda count) (Each voter casts $|C|$ votes for their most preferred candidate, $|C|-1$ votes for their next-most-preferred, ..., and 1 vote for their least preferred candidate.) Scoring function: Let $b_i$ be the positional score of candidate $i$ among all voters except the manipulator; then $S(P,i) = b_i + |\{j : iPj\}| + 1$.

**Maximin** (A winner is a candidate who maximizes the minimum number of voters who prefer him to another candidate in pairwise elections.) Scoring function: Let $V_{ij}$ be the voters who prefer $i$ to $j$; then $S(P,i) = \min_j (|V_{ij}| + 1$ if iPj; |V_{ij}| if jPi).

**Copeland’s method** (A winner is a candidate who maximizes the number of victories minus the number of defeats in pairwise elections.) Scoring function: $S(P,i) =$ (number of candidates that $i$ beats in pairwise contests) - (number of candidates to whom $i$ loses in pairwise contests) based on the preferences of all the voters, including the manipulator.

In addition, any monotone increasing function of such scores (which includes lexicographic combinations) still satisfies the hypotheses of Theorem 1. Consequently,
baroque scoring methods such as 1/3 times the positional count plus 1/5 times the Copeland count plus 2/7 times the maximin count are still efficiently manipulable by GREEDY.

4. A voting scheme that is computationally resistant to manipulation. The Copeland voting scheme ranks the candidates according to the number of pairwise contests they win minus the number they lose [9, 8]. When all candidates are compared against each other pairwise (so that they participate in the same number of contests), this is equivalent to scoring simply by the number of contests won. Many organizations use this method of social choice to rank contestants, and extend it by adding a tie-breaking rule that we refer to the "second-order Copeland scheme": In case of a tie, the winner is the candidate whose defeated competitors have the largest sum of Copeland scores.

(The Federation Internationale Des Echecs and the United States Chess Federation implement tie-breaking rules that are either identical to, or are minor variants of, the second-order Copeland scheme [6, 7]. For example, for round-robin tournaments under USCF rules, the primary score of each player is the number of opponents he has defeated plus one-half the number of opponents he has tied. In case of ties with respect to primary score, a secondary score is computed to be the sum of the primary scores of all the opponents he has defeated plus one-half the primary scores of all the opponents with whom he has drawn.)

It requires only polynomial time to compute the winner of an election under either Copeland or second-order Copeland schemes. Furthermore, by the theorem of Gardenfors [3] both schemes are manipulable in principle. However, we make an important distinction here: While a first-order Copeland scheme can be manipulated efficiently (by Theorem 1), second-order Copeland is computationally resistant to manipulation, as we will show. Specifically, it is NP-complete to determine whether one can misrepresent one's preferences to exploit knowledge of the preferences of others. (Note that second-order Copeland violates the hypothesis of monotonicity required by Theorem 1 for manipulability by GREEDY.) Thus second-order Copeland circumvents the Gardenfors impossibility theorem on operational grounds: it satisfies the hypotheses - it is neutral, anonymous, and Condorcet - but it is computationally difficult to manipulate.

Since the proof is lengthy, we highlight here the main points, some of which are of independent interest, and defer supporting technical details to an appendix. To capture
the way in which second-order Copeland is likely to be used in practice, we exhibit an
instance in which a set of candidates is tied under the primary score (Copeland), and the
difficulty is to manipulate the tie-breaking score (second-order Copeland). The outline of
the argument is as follows. First we show that a logic problem that is known to be
hard can be embedded in a problem of scoring tournaments, so the tournament problem is
hard. Then we show that the tournament problem can be embedded in SINGLE-VOTER
MANIPULATION OF SECOND-ORDER COPELAND, which must therefore be hard.

A word of caution to the reader: The argument embeds a logic problem in a graph
problem, and the graph problem in the voting problem. Since we are showing that these
problems are in a sense equivalent, we variously adopt the elementary terminology of
logic, graph theory, and voting. Where one field has a concept we need, we occasionally
switch terminology mid-argument, rather than burden the reader with new but equivalent
definitions.

Now we begin by showing that a problem of tournament scoring is hard. The
tournament problem is illustrated by the final round of a round-robin chess tournament.
The tournament requires each contestant to play every other contestant, but there is one
more round (set of pairwise contests) to be played. The question is whether there exists
a set of outcomes for the final round that will guarantee tournament victory for a
particular competitor. We formalize this as:

TOURNAMENT OUTCOME
INSTANCE: A complete simple graph (with vertices corresponding to candidates), for
which each edge (i,j) can be either directed (corresponding to candidate i having beaten
candidate j) or undirected (corresponding to the contest between candidates i and j not
having been decided); a distinguished vertex (candidate) c.
QUESTION: Is there a way of assigning directions to the currently undirected edges so
that c is the winner?

THEOREM 2: TOURNAMENT OUTCOME UNDER SECOND-ORDER COPELAND is NP-
complete.

Proof: The problem is in NP because we can quickly prove a "yes" answer by showing a
set of outcomes and computing the second-order Copeland scores. We show the problem
is as hard as an NP-complete problem by embedding within it 3,4-SATISFIABILITY, which
is known to be NP-complete [13].

3,4-SAT

INSTANCE: An expression consisting of clauses $C_1, \ldots, C_m$ over variables $X_1, \ldots, X_n$, with each clause containing exactly 3 different variables, and each variable appearing in exactly 4 clauses.

QUESTION: Is there a satisfying truth assignment for the set of clauses?

Given an instance of 3,4-SAT, construct a instance of TOURNAMENT OUTCOME as follows. Create a set of candidates corresponding to clauses $C_1, \ldots, C_m$ and to literals $X_1, \neg X_1, \ldots, X_n, \neg X_n$ (where $\neg X$ is the negation of $X$). In addition, create a distinguished candidate $c$, whose victory is in question. All pairwise contests have been decided except those between the pairs of literal candidates $X_j$ and $\neg X_j$ ($j = 1..n$). In particular, each "clause candidate" $C_i$ ($i = 1..m$) defeated the 3 "literal candidates" corresponding to the literals in the clause, and lost to all other candidates.

Model the candidates and their contests as a graph in which there is a vertex for each candidate. If candidate $i$ has defeated candidate $j$, there is an edge directed from $i$ to $j$; if the contest between candidates $i$ and $j$ has not yet been decided, connect $i$ and $j$ by an undirected edge. Consequently, in $G$ each edge $(X_j, \neg X_j)$ is undirected, and all other edges are directed from winner to loser. (See Figure 1.) Let $R$ be the set of undirected edges corresponding to remaining pairwise contests.

For the candidate corresponding to vertex $v$, his tentative scores $TS(v)$ and $TS^2(v)$ are his Copeland score and second-order Copeland score, respectively, not counting possible points from contests that have not yet been decided. Let $S(v,R)$ and $S^2(v,R)$ denote the eventual Copeland and second-order Copeland scores of $v$, depending on the outcomes of the remaining contests $R$. Note that for all $C_i$, $S(C_i,R) = TS(C_i)$ is independent of $R$.

In Appendix 1, we show how to pad graph $G$ with additional candidates and assign outcomes to their pairwise contests so that the following properties hold:

Property 1: For any specification of outcomes for $R$, the candidates $c$ and all $C_i$ will be tied for first place with respect to (first-order) Copeland score; that is, $S(c,R) = S(C_1,R) = \ldots = S(C_m,R) > S(v,R)$ for all other $v$.

Property 2: The second-order Copeland score of the distinguished candidate $c$ is independent of $R$. (Accordingly, we abbreviate $S^2(c,R)$ as $S^2(c)$.)

Property 3: Each candidate $C_i$ ($i = 1..m$) has $TS^2(C_i) = S^2(c) - 3$.  

7
Property 4: Each $C_i$ ($i = 1..m$) defeated the 3 candidates corresponding to the literals that clause $C_i$ contains in the instance of 3,4-SAT, and lost to all other candidates.

Property 5: For all outcomes $R$, $S^2(c) > S^2(v, R)$ for every $v$ other than a clause candidate.

Now we claim that if $G$ has properties 2-5, then there is a set of outcomes for the remaining contests $R$ which will make $c$ the unique winner if-and-only-if the instance of 3,4-SAT is satisfiable. To see this, imagine candidate $C_i$ ($i = 1..m$) during the last round of contests. His own contests are over, so his Copeland score has been determined, and (by property 3) he is currently 3 points short of a share of first place with $c$ and possibly other $C_j$. By property 4, $C_i$ has lost to all but 3 of the literals, so the outcomes of only 3 contests (those containing the literals defeated by $C_i$) could improve his second-order score. If all 3 contests go as $C_i$ wishes (the candidates that $C_i$ defeated win their contests), then, by properties 2 and 3, $C_i$'s second-order score will equal that of $c$, and by property 5 $C_i$ will own a share of first place; otherwise, $c$ will beat $C_i$. Interpreting a literal losing to its complement as the literal being set to TRUE in the instance of 3,4-SAT, candidate $C_i$ will lose to $c$ if-and-only-if clause $C_i$ is satisfied. Thus satisfiability of the 3,4-SAT expression corresponds precisely to all of the $C_i$'s ($i = 1..m$) being defeated by $c$.

COROLLARY: TOURNAMENT MANIPULATION under Copeland scoring with second-order Copeland tie-breaking is NP-complete.

Proof. This follows from property 1 of the proof of Theorem 2.

REMARK: This means that in a chess tournament it can be difficult for a team to play strategically when the tournament will be decided on tie-breaks. Furthermore, note that in the proof of Theorem 2, the undecided contests were candidate-disjoint; that is, no candidate was involved in more than one undecided contest. This means that it can be hard to manipulate even the final round of a tournament. (We will require this fact in the proof of the next theorem.)
THEOREM 3: SINGLE-VOTER MANIPULATION OF SECOND-ORDER COPELAND is NP-complete.

Proof: The problem is in NP since the outcome of the election can be computed in polynomial time.

We prove difficulty by showing that the problem can contain the tournament manipulation problem just shown to be NP-complete. First recall that any complete graph of outcomes (including undirected edges to represent ties) is realizable by a small number of voters (where "small" means polynomial in the number of candidates) so that every non-tied contest is decided by 2 votes [12]. Apply this to the graph of outcomes of pairwise contests from the construction of Theorem 2, where we interpret the undirected edges as indicating a tie between the corresponding candidates in a simple vote count. Let V be a set of voters that realizes this graph. Assume that the manipulator knows these preferences, and now wonders whether there exists a preference he can claim that will make c a winner.

Since all contests corresponding to directed edges have been decided by 2 votes, the manipulator cannot affect these outcomes; he can affect only the contests that are currently tied. As we have previously observed, those contests are candidate-disjoint, so the manipulator can vote to achieve any of the $2^R$ possible sets of outcomes by simply ranking the candidates of each pair in desired relative order. However, to determine whether any of these sets of outcomes will make the distinguished candidate c a winner, the manipulator must solve the difficult instance of TOURNAMENT OUTCOME contrived in the previous theorem.

COROLLARY: SINGLE-VOTER MANIPULATION OF FIRST-ORDER COPELAND WITH SECOND-ORDER COPELAND TIE BREAKS is NP-complete.

Proof: This follows from Theorem 3 and the corollary to Theorem 2.

Finally we observe that second-order Copeland can be manipulated with (at least formal) efficiency if we restrict its use to elections in which there are not "too many" candidates. In fact this is true of any voting scheme for which the winner can be efficiently computed. The manipulator could in principle compute the outcome of the election for each of the $|C|!$ possible preference orders. If each evaluation requires only
polynomial time, and if $|C|$ is restricted so that $|C|! = O(p(|V|))$ for some polynomial $p$ of fixed degree, then the total effort is, strictly speaking, polynomial.

5. Conclusions. Computational complexity is a new criterion by which to evaluate methods of social choice. Worst-case behavior in this regard might be a practical consideration for some decision methods (see, for example, [1, 2]), just as is worst-case behavior with respect to formalized notions of fairness.

Methods of social choice should be easy to use but hard to abuse; that is, they should identify winners within polynomial time, but they should also be provably difficult to exploit. This is similar to the issues in cryptography, where both encryption and authorized deciphering should be easy, but unauthorized deciphering should be difficult. As for a cryptographic scheme, we would prefer to know that a voting scheme is dependably hard to abuse, rather than merely hard in the worst-case, as we have proved. Unfortunately, this sort of result seems beyond the reach of current complexity theory. However, we can appeal to practical experience, which suggests that NP-hard problems are typically difficult in practice.
REFERENCES

1. Bartholdi JJ III, Tovey CA, and Trick MA (1987) Voting schemes for which it can be difficult to tell who won the election. Submitted for publication.


We show how, given an instance of 3,4-SAT, to construct in polynomial time a graph satisfying properties 1-5. Since the verification of the properties requires only tabulating scores, this is left to the devoted reader (or can be had by writing to the authors).

Before proceeding we give an overview of the construction of the graph: G will contain three main types of vertices: those corresponding to clauses in the instance of 3,4-SAT; those corresponding to literals in the instance of 3,4-SAT; and "fillers", which pad the graph to make the scores suit our purpose. We will construct G so that all scores within each set are equal, or nearly so, but in pairwise contests clause candidates defeat fillers, fillers defeat literals, and literals defeat clauses. Finally, in addition to clause, literal, and filler vertices, there will be two distinguished vertices. One is C₀, corresponding to the contestant whose victory is in question, and the other is b, a "balancing" vertex that will enable us to adjust the second-order Copeland score of C₀.

Let m be the number of clauses and n the number of variables in the instance of 3,4-SAT. Note that 3m = 4n, so m is even and m/2 is an integer.

The vertices of G correspond to
- clauses: Cᵢ (i = 1..m);
- literals: Xⱼ, Xⱼ (j = 1..n);
- fillers: fₖ (k = 1..30m);
- the contestant whose victory is in question: C₀;
- the balancing contestant: b.

Recall that G is a complete graph, so there is an edge between every pair of vertices. Each edge is directed according to the outcome of the pairwise contest between that pair of contestants, from the winner to the loser. Some contests will be as yet undecided, and so will be represented by undirected edges.

 Orient the edges of G according to the following outcomes:
- contests between clauses and literals: Each literal defeats all clauses except the
clauses containing it. Thus each literal defeats between \( m - 4 \) and \( m \) clauses.

- contests between clauses and \( C_0 \): Each of the \( C_i \) (\( i = 0 \ldots m \)) wins \( m/2 \) and loses \( m/2 \)
of these contests. This can be arranged by imagining the \( m + 1 \) contestants \( C_i \) seated at
every other chair around a symmetric \( 2(m + 1) \)-seat round table. Each \( C_i \) sits
diametrically opposite an empty chair. Let each \( C_i \) defeat the \( m/2 \) people on his right,
and lose to the \( m/2 \) people on his left.

- contests between literals: Each literal defeats exactly \( n - 1 \) literals and loses to
exactly \( n - 1 \) other literals. This can be arranged by imagining the \( 2n \) literals seated at
a round table with \( 2n \) seats, with each \( X_j \) seated diametrically opposite \( \sim X_j \). Let each
literal defeat the first \( n - 1 \) literals on his right (up to \( \sim X_j \)), and lose to the first \( n - 1 \)
literals on his left (up to \( \sim X_j \)). The contest between each \( X_j \) and \( \sim X_j \) has not yet been
decided and so the corresponding edge is undirected.

- contests between fillers: Each \( f_k \) (\( k \) odd) defeats \( 15m - 1 \) other \( f_k \) (\( k = 1 \ldots 30m \)).
Each \( f_k \) (\( k \) even) defeats \( 15m \) other \( f_k \) (\( k = 1 \ldots 30m \)). (It is this slight difference that
allows us to achieve property 3.) This can be arranged by imagining all the fillers
seated around a \( 30m \)-seat round table in the sequence \( f_1, f_3, \ldots, f_{30m-1}, f_2, f_4, \ldots, f_{30m} \).
Each \( f_k \) defeats the \( 15m - 1 \) fillers to his right, and loses to the \( 15m - 1 \) fillers to his
left. Contests between fillers seated diametrically opposite each other are always
between a filler of even index and a filler of odd index, and are won by the filler of
even index.

- contests between literals and fillers: Every filler defeats every literal;
- contests between \( C_0 \) and literals: \( C_0 \) defeats \( X_1 \) and \( \sim X_1 \), but loses to \( X_j \) and \( \sim X_j \)
for \( j = 2 \ldots n \). (This gives property 2.)
- contest between \( C_0 \) and \( b \): \( C_0 \) defeats \( b \);
- contests between \( b \) and clauses: \( b \) defeats all clauses;
- contests between \( C_0 \), clauses, and fillers: Define \( f_{26i+1}, f_{26i+2}, \ldots, f_{26i+26} \) to be
associates of \( C_i \) (\( i = 0 \ldots m \)). Thus each \( C_i \) has 26 associates, no filler is an associate of
more than one \( C_i \), and nearly 4m fillers are not associates.

Now let \( L(C_i) \) be the total number (counting repetitions) of clauses of which the
literals in \( C_i \) are members, plus 1 if \( C_i \) contains \( X_1 \) or \( \sim X_1 \). (For example, in the
expression \("(X_1 \text{ or } \sim X_2 \text{ or } \sim X_3)\) and \("X_2 \text{ or } \sim X_3 \text{ or } X_4\)\", \( L(C_1) = 1 + 2 + 2 = 5 \). We
count \( C_0 \) as containing \( X_1 \) and \( \sim X_1 \) so \( L(C_0) = 4 + 2 = 6 \).) Since no literal appears in
more than 4 clauses, and \( X_1 \) and \( \sim X_1 \) do not appear in the same clause, we have that the
number of literals in \( C_i = 3 \leq L(C_i) \leq (3)(4) + 1 = 13 \). \( L(C_i) \) is the number of contests
won by one of $C_0$, $C_1$, ..., $C_m$ against the three literals that comprise $C_i$. Thus $C_i$ defeats all $f_k$ which are either not an associate of $C_i$; or for which $k = 26i + 2, 26i + 4, ..., 26i + 2L(C_i)$; or for which $k = 26i + 1, 26i + 3, ..., 26i + 2(13 - L(C_i)) - 1$. In other words, $C_i$ defeats all of its non-associates, and 13 of its 26 associates ($L(C_i)$ of the 13 defeated associates being fillers of even index).

- contests between $b$ and $f_k$: $b$ loses to all but the last $n + 4$ fillers; that is, $b$ loses to $f_k$ for $k = 1, 2, ..., 27m, ..., 30m - n - 4$, but defeats $f_k$ for $k = 30m - n - 3, ..., 30m$.

- contests between $b$ and literals: $b$ loses to all literals.
FIGURE 1: Essential structure of the graph corresponding to an instance of 3,4-SAT. The undirected edges correspond to contests that have not yet been decided. Each $C_i$ has an edge directed toward each of the 3 literals contained in clause $C_i$. (For clarity most of the directed edges are omitted, as are the candidates with which the graph is padded.)
Progress Report 1988

My research in the past year falls into three areas: (1) complexity; (2) algorithms; (3) probabilistic analysis.

(1) I have continued to study the complexity of various questions associated with the linear complementarity problem (LCP). If recognizing P-matrices is not hard, then it turns out solving LCP's is not hard unless $P = co(NP)$. (See Appendix 1). I have also analyzed the complexity of test problems for the travelling salesman problem, generated by the Pilcher-Rardin random cut generator. When the different aspects (optimization, threshold, etc.) of these problems are considered, they are of provable intermediate complexity unless $NP = co(NP)$ (see Appendix 2).

(2) Working with Richard Borie and R. G. Parker, I studied the solution of combinatorial problems on recursively constructed graphs. The principal questions we sought answers to were, “how far can series-parallel graphs be generalized?”, and “what problems can be solved in linear time on these graphs via dynamic programming?” We found good answers to both questions: necessary and sufficient conditions on composition rule sets to yield easily factorable graph classes (Appendix 3); and a predicate calculus to write problems in, from which linear time algorithms can be generated automatically (Appendix 4). We also found some related results on forbidden minors of partial 3-trees (Appendix 5).

(3) I developed swindling techniques for improving the efficiency of simulated annealing, a popular, general purpose probabilistic algorithm (Appendix 6). I analyzed the probabilistic distributions of outcomes to players under the Stackelberg solution criterion. When preferences are monotonic, the advantage accrues to the leader (see Appendix 7).

Status Update on Papers from 1987 report

The paper on finding local optima in arbitrary discrete structures has been accepted by Discrete Applied Mathematics; the paper on the station routing problem is to appear in Operations Research; the two papers on complexity applied to social choice have been revised and will appear in Social Choice and Welfare (Appendices 8,9).

Research Directions for 1988-1989

I plan to continue applying probabilistic analytic methods, and complexity analysis, to a variety of problems, particularly in game theory and economics. Another major area is in the design and manufacturing scheduling of electronic circuit boards, where I will apply both probabilistic and deterministic optimization tools. In addition, I hope to complete some work on unifying linear programming algorithms, and study the problem of assigning classes to rooms in secondary schools and universities.
Solving LCP on Recognized P-Matrices is Probably Not NP-Hard

Daniel Solow
School of I.E.
Case Western University

Richard Stone
Operations Research Dept.
AT&T Bell Labs

Craig A. Tovey
School of ISyE
Georgia Tech

November 1987

Abstract: We apply a result of Saigal's [1973] to recent work by Johnson, Papadimitriou, and Yannakakis [1985] to show that solving linear complementarity problems on P-matrices cannot be NP-hard unless \( P = \text{co(NP)} \), if recognizing P-matrices is not hard.

The linear complementarity problem, LCP, given square matrix \( M \) and vector \( q \), is to find nonnegative vectors \( x \) and \( w \) such that \( w - Mx = q \), \( x \cdot w = 0 \). It is easy to see that for general \( M \) and \( q \) this problem is NP-hard. For instance, any 0–1 integer programming feasibility problem of form \( Ay \leq b \) can be transformed into an equivalent LCP with

\[
-M = \begin{pmatrix} I & 0 \\ A & 0 \end{pmatrix}; \quad q = (1, 1, \ldots, 1, b).
\]

When \( M \) is a P-matrix (all its principal minors are positive [Berman and Plemmons, 1979, Nonnegative Matrices in the Mathematical Sciences, Academic Press, NY]), a classic result of Samelson, Thrall, and Wesler [1958] guarantees that for all \( q \) a unique solution exists, and algorithms by Lemke [1965] and others are guaranteed to find it, unfortunately with exponential worst-case performance [Murty 1978, Fathi 1979]. Thus the complexity of finding the solution to such an LCP remains a vexing open question. The complexity issue is complicated by the fact that recognition of P-matrices is itself an open problem. While it is obviously in \( \text{co(NP)} \) (just exhibit a principal submatrix with nonpositive determinant), nothing more is known [Murty, personal communication, 1987].

In this note we point out that, if recognizing P-matrices isn't too hard, then the problem of finding the solution to an LCP with \( M \) a P-matrix is not as hard as LCP in general, unless \( NP = \text{co(NP)} \). Recently, in a study of the difficulty of local search, [JPY] defined a class of function computation problems, PLS (Polynomial-time Local Search), with the property that

**Proposition [JPY].** No problem in PLS can be NP-hard unless \( NP = \text{co(NP)} \).

We apply a result of Saigal's [1972] to show that LCP on P-matrices is in PLS, from which our remark follows. First we state the definition of PLS [JPY]:

A function computation problem \( L \) is in the class PLS if for each instance \( x \in \{0,1\}^* \) there is a finite set of (feasible) solutions \( F_L(x) \), all with the same length, which is polynomial in \( |x| \), each solution \( s \) having nonnegative integer cost \( c_L(s, x) \), such that the following three polynomial-time algorithms \( A_L, B_L, \) and \( C_L \) exist: \( A_L \), given instance \( x \), finds a solution \( A(x) \in F_L(x) \). The second algorithm, \( B_L \), takes instance \( x \) and possible solution string \( s \), outputs a special "no" symbol if \( s \notin F_L(x) \), and otherwise outputs the cost \( c_L(s, x) \). The third algorithm gives information about the neighborhood of \( s \), denoted \( N(s, x) \subseteq F_L(x) \). Given \( x \) and \( s \in F_L(x) \), algorithm \( C_L \) produces a solution \( t \in N(s, x) \) with \( c(t, x) < c(s, x) \), or it reports that no such \( t \) exists. In the latter case \( s \) is locally optimal. The problem \( L \) is, given \( x \), to find a locally optimal solution \( s \).

Next, we define the function computation version of LCP on P-matrices:

**Definition.** The problem of solving LCP for P-matrices is, over the set \( S \) of pairs \((M, q)\), where \( M \) is a P-matrix and \( q \) an appropriate size vector, given \( s \in S \), to exhibit a solution \((w, z)\) to the LCP defined by \( s \).

**Theorem.** The problem of solving LCP for P-matrices is in PLS.

Proof: We will define a computational problem \( L \). We then verify two properties of \( L \): first, it is in PLS; second, it is equivalent to LCP on P-matrices, as defined above. Let \( L \) be the set of instances as \( S \) in the definition above. For each instance \( x \), the set of solutions \( F_L(x) \) is the set of all possible choices of lexicographically feasible almost complementary and complementary bases. Formally, if \( M \) is an \( n \) by \( n \) matrix, we let \( E_L(x) = \{ s : s \in \{-1, 0, 1\}^n; s \text{ contains at most one -1}\} \). A member \( s \) of this set corresponds to a submatrix \( B_s \) of \( \{I, -M, -e\} \) (where \( e \) is a vector of 1's), where the \( i \)-th column of \( B_s \) is the \( i \)-th column
of $I$ or $-M$ if $s_i = 0$ or 1, respectively, and equals $-e$ if $s_i = -1$. If all the row vectors of the matrix $W \equiv B_L^{-1} [q, I]$ are lexicographically positive, then $s$ is feasible and $s \in F_L(x)$.

The cost function $c_L(x)$ corresponds to the value of the artificial variable (for the column $-e$) in the Lemke method tableau, with lexicography taken into account. Precisely, for $s \in F_L(x)$, if $s_i \geq 0 \, \forall i$ then $c(s, x) = 0$. Otherwise, let $j$ be the index where $s_j = -1$. Then

$$c(s, x) = \sum_{i=1}^{n+1} W_{ij} \times e^{i-1},$$

where $c$ is a suitably small power of 2 such as $2^{-|s|}$. (For ease we have defined $c$ rational instead of integer, obviously scaling by $2^{|s|}$ would convert to integers.) The $\epsilon$ term serves as the perturbation term to avoid degeneracy.

We can now define algorithm $B_L$ as follows: given $s$, verify that it is a string of 0's and 1's with at most one $-1$, and compute the matrix $W$. If its rows are lexicographically positive, compute $c(s, x)$ as given above, otherwise report “no”. It is clear that $B_L$ runs in polynomial time, the most expensive step being the solution of $n$ simultaneous linear equations to compute $W$.

The other two algorithms correspond directly to steps in Lemke’s algorithm. Algorithm $A_L$, given $M, q$, returns the initial almost complementary basis for Lemke’s algorithm, with the $q$ vector perturbed to $\tilde{q} \equiv q + (\epsilon, \epsilon^2, \ldots, \epsilon^n)$. Precisely, if $q \geq 0$ then $s_i = 0 \, \forall i$; otherwise let $j$ be the index of the most negative component of $\tilde{q}$, and set $s_j = -1; s_i = 0 \, \forall i \neq j$. Clearly algorithm $A_L$ runs in polynomial time.

The neighborhood for this problem is the one induced by Lemke’s method: a string $t \in F_L(x)$ is in $N(s, x)$ if either $s$ and $t$ differ in exactly one component $i$, where $s_i$ or $t_i$ equals $-1$, or if $s$ and $t$ differ in exactly two components $i, j$, where $s_i = t_j = -1$. (Actually this permits a few extra neighbors, but this doesn’t matter.) Algorithm $C_L$, given (lexicographically) feasible (i.e. for the perturbed problem) solution $s$, calculates $B_L^{-1} [I, -M, -e, q]$, which is the corresponding tableau. It then performs one iteration of Lemke’s algorithm, yielding new solution $\tilde{s}$. If $s \geq 0$ it simply reports that $s$ is locally optimal.

Saigal [1973], Eaves (“The Linear Complementarity Problem,” Management Science 17, 612–634, 1971), proved that when $M$ is a P-matrix, and the solutions are nondegenerate, that the sequence of artificial variable values is strictly decreasing in each iteration of Lemke’s algorithm. Hence, $c_L(\tilde{s}, x) < c_L(s, x)$. The algorithm $C_L$ therefore satisfies the requirements of the class PLS, and so we conclude that $L \in PLS$. This is the first property claimed for $L$.

To see that $L$ is equivalent to the function computation version of LCP on P-matrices, we must establish that the only local optima are in fact solutions to the LCP. By [Cottle and Dantzig, 1968], Lemke’s algorithm always terminates in a LCP solution when $M$ is a P-matrix. Suppose $s$ does not correspond to a solution to the LCP: then $c(s) > 0$ and Lemke’s algorithm will find a neighboring solution with smaller cost. On the other hand, if $s$ solves the LCP then $c(s) = 0$ and algorithm $C_L$ will report local optimality. Therefore, when $M$ is a P-matrix, the only local optimum is the solution to the LCP. So solving $L$ is equivalent to solving the LCP, which verifies the second property claimed for $L$. The theorem follows.

Corollary. The problem of solving LCP for P-matrices is not NP-hard unless NP = co(NP).

Having established membership in PLS, the natural open question is whether LCP on P-matrices is PLS-complete. However, because instances of LCP on P-matrices all have only one local optimum, and all known PLS-complete problems have multiple local optima, it seems to us unlikely that a such a reduction exists (unless the whole class PLS collapses to P). There is a neighborhood search algorithm for LCP by Solow [1985] which coincides with Lemke’s algorithm for P-matrices. Solow’s algorithm, when applied to an arbitrary $M$, can get “stuck” at local optima that are not LCP solutions. The problem of finding one of these local optima, which includes solving LCP on P-matrices, is a more plausible candidate for PLS-completeness.
Test Travelling Salesman Problems of Intermediate Complexity

Ronald L. Rardin*  Craig A. Tovey†

May 5, 1988

Abstract

Test Instance Generators (TIGs) are important to evaluate heuristic procedures for NP-hard problems. We establish a framework for evaluating TIGs, and analyze a TIG in use due to Pilcher and Rardin. We show that this TIG creates a class of instances of intermediate complexity:

- not as hard as the hardest TSP instances unless \( \text{NP} = \text{co(NP)} \);
- not as easy as the easiest TSP instances unless \( \text{P} = \text{NP} \).

Our result plays on the distinctions between the threshold, value, and search versions of an optimization problem.

We also define a new class of languages, PC (Polynomially Constructable), for which efficient test instance generators exist. PC is in \( \text{NP} \), contains all known NP-complete languages, but is not known even to contain \( \text{P} \).

1 Introduction

A tremendous amount of effort in the past two decades has been directed towards developing good heuristics for NP-hard problems. A good heuristic, classically, has performance ratio \( \frac{v^h}{v^*} \) close to 1, where \( v^h \) (resp. \( v^* \)) = value of the heuristic (resp. optimal) solution. Since we have no practical way to find \( v^* \) for an arbitrary instance (if we did we wouldn't be resorting to heuristics) we cannot empirically determine \( \frac{v^h}{v^*} \) by testing instances generated purely at random. Hence we need a Test Instance Generator (TIG) that supplies instances with known optimal solutions.

The design and analysis of TIGs is a fairly recent development. In this paper we establish a framework for evaluating TIGs; we apply it to a specific TIG in use for the TSP (due to Pilcher and Rardin (PR) [6]) which turns out to have some interesting and unusual characteristics. In the last section of the paper we study the class of problems for which good TIGs exist: this class, too, turns out to have some unusual characteristics.

Any TIG designer faces a problem analogous to the cryptographer's — how to simultaneously satisfy three conditions:

*School of Industrial Engineering, Purdue University, research supported by ONR Grant N00014-86-K-0689
†School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta GA 30332, research supported by a Presidential Young Investigator NSF Grant ECS-8451032
1. The TIG should generate instances efficiently, i.e. it should run in time polynomial in the length of the output.

2. The TIG should generate instances with known optimal solutions.

3. The TIG should generate instances that are hard to solve.

If we did not enforce condition (2), we could just generate an instance at random. If we did not enforce condition (1), we could just generate an instance at random and solve it by brute force.

For ease, we focus on a specific problem, the TSP. Ideally, to satisfy condition (3), we would like to be able to generate instances as hard as the hardest TSP instances. It is very unlikely that PR accomplishes this; we show that unless \( NP = co(NP) \), its instances are not at the same complexity level as the general TSP. This turns out not to be a specific failing of the PR generator, but rather an inevitable consequence of satisfying conditions (1) and (2). Essentially, this is because the language generated by anything satisfying (1) and (2) must be in NP, because the "solver" could nondeterministically replicate the generative process and thereby have a succinct proof of optimality. That is, there must be a short computation that solves the instance, namely the computation path of the TIG itself. Therefore, the full complexity of an optimization problem cannot be captured by a TIG satisfying (1) and (2).

A simple example for Hamiltonian graphs will illustrate the difficulty. To generate test instances with \( n \) vertices for the Hamiltonian graph problem, form the tour \( 1, 2, \ldots, n, 1 \). For each other possible edge, decide at random whether to include it. Now randomly permute the vertices. Obviously any graph so produced is Hamiltonian, and any Hamiltonian graph can be so produced. The difficulty is in the co-NP side of the question: generating test graphs for which the answer is "No" (Not Hamiltonian). Moreover, if one uses the "promise" that the test instances are generated as above, one can solve the instances with no computational effort by just answering "yes."

Does this mean that a TIG must produce easy problems? Surprisingly, the answer is no. Analyzing Pilcher and Rardin's TIG, we show that unless \( P = NP \), the instances cannot be solved in (deterministic) polynomial time. This is true even if the instance comes with a "promise" that it was created by the generator. At the heart of the proof is a nice illustration that the issue of recognizing valid input is not a mere technicality.

Taken together, these results imply that the class of instances so generated is of intermediate complexity. This depends on distinguishing different questions associated with an instance. The value question,

Given an instance and value \( v \), is \( v \) the optimal solution value?
is \( D^P \)-complete for the TSP (harder than NP-Complete unless \( NP = co(NP) \)), but only in NP for the PR TIG. On the other hand, the search problem,

Given an instance, find an optimal solution

turns out to be NP-hard for the PR TIG instances. This is true even if we are "promised" the instance comes from the TIG.

TIGs with similar properties could be constructed for many other optimization problems for which integer programming formulations are known.

For what problems can good test generators be built? We define a new class, \( PC \), (for Polynomials Constructable), to capture properties shared by languages produced by TIGs. Roughly speaking, \( PC \) is the class of languages produced by TIGs satisfying (1), (2) and which are guaranteed to output a string greater than equal to some (input) desired length. \( PC \) has some interesting properties: \( PC \subseteq NP \); \( PC \) contains all known NP-complete problems; if \( L \in NP \) and \( L \) contains an infinite subset in \( PC \), then \( L \in PC \) also. However, \( PC \) is not known to equal \( NP \). In particular, \( Primes \) are not known to be in \( PC \), even if the Generalized Riemann Hypothesis is assumed (and hence \( Primes \in P \)). If the requirements on the TIG are varied, these properties are fairly robust.

2 Notation

As Papadimitriou [4] and others have observed, an optimization problem is several questions. Since our instances are seen to be "intermediate" by careful classification of different questions, we define several:

\( TSP^\leq \): TSP Threshold. Instance, a complete edge-weighted graph \( G \) and an integer \( k \). Question: is there a tour on \( G \) of weight less than or equal to \( k \)?

\( TSP^\geq \): TSP Exact Value. Instance, same as for \( TSP^\leq \). Question, is the minimum weight tour on \( G \) of weight \( k \)?

\( TSP \): TSP. Instance, a complete edge-weighted graph \( G \). Search Question: find a minimum weight tour on \( G \).

The complexity levels of these questions have been established. \( TSP^\leq \) is NP-complete [1]; \( TSP^\geq \) is \( D^P \)-Complete [5]; the problem \( TSP \) is NP-hard [1].
3 Pilcher-Rardin Generator

For brevity, we will hereinafter refer to the Pilcher-Rardin generator as the TIG. It operates in the following way on a general integer program:

1. Choose an arbitrary integer extreme point \( z \) to be optimal.
2. Select some set of valid cuts (e.g. Chvátal cuts) that are tight at \( z \).
3. Randomly choose nonnegative multipliers on the constraints to compute a cost vector \( c \) which is optimized at \( z \) in the linear relaxation of the model plus cuts added in (2).
4. Discard the cuts, and output \( c \).

It is obvious from standard linear programming theory that this TIG satisfies property (2).

Pilcher and Rardin [6] apply this generator idea to the standard integer programming model of the TSP (2-matching and subtour elimination constraints). They show how to find tight constraints quickly from each of the families of subtour, comb, and clique-tree cuts. Only polynomially many constraints are generated, so the TIG satisfies property (1). One attractive feature here is that the separation problems for combs and clique-trees are open. Thus by the equivalence [2] between separation and optimization, there is no obvious way to solve even the linear programming relaxations in polynomial time. Define:

\[
[TSP] = \text{TSP polytope}
\]

\[
[\text{subtour}] = \text{the LP relaxation polytope including subtours}
\]

\[
[\text{comb}] = \text{the LP relaxation polytope including combs as well}
\]

\[
[\text{clique}] = \text{the LP relaxation polytope including clique-trees as well}
\]

\[
[\text{any}] = \text{any of these three LP relaxations}
\]

These different linear programs can be succinctly encoded. All that is needed is the cost vector. Any cost, applied to \([TSP]\), yields an integer solution since all the extreme points of \([TSP]\) by definition are integer. But \([\text{any}]\) has noninteger extreme points, as well.

**Definition.** A cost vector \( c \) is good with respect to \([\text{any}]\) iff there is an integer extreme point \( z \) of \([\text{any}]\) that minimizes \( c \).

**Theorem 3.1** ("Good TIG"). The set of costs produced by the TIG is precisely the set of good costs (with respect to \([\text{any}]\)).
Proof Sketch: Suppose the TIG picks \( z \). Because the cost \( c \) generated is a nonnegative combination of binding constraints at \( z \), \( z \) must be optimal. So any TIG cost is good.

Conversely, let \( c \) be a good cost with respect to [any], and let \( z \) be an integer extreme point of [any] that minimizes \( c \). Since all integer extreme points of [any] are tours, the TIG could have picked this \( z \). Moreover, \( c \) supports [any] at \( z \). So by standard polyhedral theory it must be in the cone generated by the facets at \( z \). Therefore the TIG could generate \( c \), and any good \( c \) is a TIG cost. It remains to verify that only polynomially many constraints receive nonzero weights to generate \( c \). This follows because the dual of the linear programming relaxation has only polynomially many constraints.

Armed with Theorem 3.1, we can define subclasses of TSP constructible by the TIG. The definitions are as before with prefix \( G \) denoting "good" and with a subscript to indicate what set of facets is involved. For example, \( \text{GTSP}_{\text{comb}} \) is the set of TSP instances restricted to good costs with respect to \( \text{comb} \).

We now state the principal question of this paper as: **what is the complexity of \( \text{GTSP}^S_{\text{any}}, \text{GTSP}^n_{\text{any}}, ... \)** compared with the complexity of the corresponding problem \( \text{TSP}^S, \text{TSP}^n, ... \)?

### 4 Upper Limits on the Complexity of Generated Instances

To begin, we give evidence that the \( \text{GTSP}'s \) aren't as hard as the hardest in TSP. In contrast with \( \text{TSP}^n \), which is \( \text{D}^P \text{-Complete} \) (harder than \( \text{NP-complete} \) unless \( \text{NP} = \text{co(NP)} = \text{D}^P \),

**Theorem 4.1.** \( \text{GTSP}^n_{\text{any}} \in \text{NP} \).

**Proof:** An instance is a cost \( c \) and integer \( k \). The answer is "yes" iff the cost is good and the optimum solution value is \( v^* = k \). If the cost vector \( c \) is good, then by the good TIG theorem, the TIG could construct the pair \( c, k \). Since the TIG satisfies condition (1), an NDTM, given \( c, k \) could emulate this process.

**Complexity under a Promise**

If the answer to one of the TIG threshold instances was "no", so the value \( k \) were unattainable, a short proof of this fact would be to guess the true optimum, verify its optimality, and show it worse than \( k \) in value. Thus it is tempting to say that \( \text{GTSP}^S \in \text{NP} \cap \text{co(NP)} \). Technically, we cannot conclude this because \( \text{GTSP}^S_{\text{any}} \) is the set of all pairs \((c, k)\) such that \( c \) is good with respect to [any] and there is a tour of cost \( \leq k \), using cost vector \( c \). Thus \( \text{co(GTSP}^S_{\text{any}}) \) is the union of sets \{all good \( c \)'s with false \( k \)} \cup \{all bad \( c \)'s with any \( k \}\) and we don't know we can quickly nondeterministically recognize bad \( c \)'s. (note that in Theorem 4.1 the emulation verifies good \( c \)-ness.)
But what if the \( c \) is reported to have come from the TIG? If Pilcher and Rardin are trustworthy, then we "know" in some sense their test instance has a good \( c \). We are again tempted to claim membership in \( \text{NP} \cap \text{co(NP)} \). This kind of situation, where we have a "promise" about an instance (e.g. given a graph, we are promised that it has at most one Hamiltonian cycle), was initially investigated by Valiant and Vazirani [7]. Employing their terminology we can define \( PGTSP \leq_{\text{any}}^\text{P} \), Promised Good TSP, as \( GTSP \leq_{\text{any}}^\text{P} \) with an extra bit of information, that the TIG created it. We can then say that \( PGTSP \leq_{\text{any}}^\text{P} \in \text{co(Promise-NP)} \), because if the bit is off, we reject the string; if it is on, we guess and prove the optimum value, then show \( k \) is less. The interesting feature is that the promise is real. It turns out that even with this promise, the PR TIG instances are not so easy to solve.

5 Lower Bounds on Complexity

As we remarked earlier, the Grötschel-Lovász-Schrijver [2] equivalence between separation and optimization suggests that the TIG instances will not be very easy to solve, because even the linear programming relaxations with combs or clique-trees are open problems. We give much stronger evidence to this effect.

Definition. Let \( G \) be any graph. Then \( c_G \) is the cost vector which is the indicator vector for the edges of \( G \) (1 if the edge is in graph, 0 otherwise).

Lemma 5.1 ("Key Lemma"). If \( G \) is Hamiltonian then \(-c_G\) is good with respect to \([\text{any}]\).

Proof: If \( G \) is a Hamiltonian graph on \( n \) vertices, there exists an integer extreme point with minimum value \(-n\) for cost vector \(-c_G\) (namely any tour in \( G \)). Since for all \( x \in [\text{any}] \), \(-n\) is the best value achievable, this extreme point must be a global minimum for \(-c_G\) over \([\text{any}]\). This is the definition of good.

It now follows from Theorem 3.1 that all negative incidence vectors of Hamiltonian graphs, \(-c_G\), can be generated by the TIG. We can in fact show this generation explicitly just using subtour constraints.

Theorem 5.2. The problem "Good \( c \): given \( c \), is it good with respect to \([\text{any}]\)?" is NP-complete.

Proof Sketch: By reduction from Hamiltonian graph. Let \( G \) be any graph. Solve

Minimize \(-c_G \cdot x\) subject to \( x \in [\text{subtour}]\),

getting value \( v^* \). If \( v^* = -n \) set \( c = -c_G \), otherwise set \( c \) to some vector \( d \) that is not good.

If \( G \) is Hamiltonian \( v^* \) will be zero and \( c = c_G \) is good by the key lemma. Conversely, if \( G \) is not Hamiltonian then either \( v^* > -n \) i.e. the subtour relaxation reveals this, and so \( c = d \) is not good; or \( v^* = -n \) but there is no integer solution achieving that value, whence \( c = c_G \) is not good. Thus \( G \) is Hamiltonian iff \( c \) is not good.
Informally, the TIG can generate all indicator vectors of Hamiltonian graphs. The only other graphs it can generate are provably non-Hamiltonian by solving LP relaxation over \( \text{any} \).

The following theorem, together with theorem 3.1, implies that the PR TIG instances are hard:

Theorem 5.3. \( \text{GTSP}_{2} \) is NP-complete.

Proof: Membership in NP follows by the same argument as in Theorem 4.1. Now, by reduction from "Good c", above, let \( c \) be any instance. Our instance of \( \text{GTSP}_{2} \) is \( (c, T) \), where the threshold number \( T = \sum |c_i| \), or some other suitably large value.

We have formally established that the TIG instances are hard. However, we have not taken into account the promise of "goodness" that in reality accompanies the instance. Since the proof of the preceding theorem depended on the hardness of establishing "goodness," it seems the TIG instances might become easy if the promise is taken into account. What happens is this: \( \text{PGTSP}_{\text{subtour}} \) is in Promise-P. But it is NP-hard, even with the promise, to exhibit a solution.

Theorem 5.4. \( \text{PGTSP}_{\text{subtour}} \) is NP-hard.

Proof Sketch: By the Key Lemma, any Hamiltonian graph \( G \) has good cost \( c_G \). If we could exhibit the solution for any good cost, given merely the promise of goodness, then we could exhibit a Hamiltonian cycle for any Hamiltonian graph, given merely the promise of Hamiltonicity.

Discussion — Recognition of Valid Input

As David Johnson points out [3], phrases such as "remains NP-complete when restricted to cubic planar graphs," and "polynomial on trees" are the clichés of the field. To be absolutely complete, proofs of these assertions are required to address the technicality of an NDTM recognizing cubic planar graphs, or of recognizing trees in deterministic polynomial time, respectively. Here we have an illustration that this requirement to recognize valid input is not an empty technicality. The promise may make it easy to know the optimum value of an instance of GTSP\(^*\), but producing the actual solution can still be hard. And the technical hardness of \( \text{GTSP}_{2} \), the problem of recognizing a good \( c \), is at the heart of the difficulty in producing the actual solution.

6 When Can Test Instances Be Efficiently Generated?

Until this point we have taken a specific TIG and asked, "is it good?" In this section we turn the question around and ask, "for what kinds of problems do good TIGs exist?". We impose weak conditions on a TIG
which lead to an interesting class called PC. We then consider how altering the conditions affects PC and its properties. Among the properties we establish is that PC contains all known NP-complete problems.

A generator $T$ satisfying (1) and (2), but which sometimes didn’t produce any output and just “hung”, could hardly be called efficient. To avoid trivialities we would also like $T$ to produce large instances on command.

**Definition.** Let $T$ be an NDTM which takes as input an integer $n$, and produces a string $s$, with $|s| \geq n$, in time polynomial in $|s|$. (Along every computation path, $T$ halts and outputs string $s$ as above.) Let the language $L \in PC$ be

$$L \equiv \{s | \text{there is a computation of } T \text{ which produces string } s\}.$$  

Then $PC$ is the set of of all such languages $L$.

The following follow from the discussion in the introduction:

**Proposition 6.1.** $PC \subseteq NP$.

**Proposition 6.2.** Hamiltonian Graph $\in PC$.

The class $PC$ is a bit peculiar. If $L \in PC$ and $L' \propto L$, this does not imply that $L' \in PC$. If $L$ and $L'$ were polynomial time isomorphic, this would imply $L' \in PC$.

A function $f$ that takes strings to strings is honest if there is a polynomial $p$ such that for all $y$ in the range of $f$, there is an $x$ such that $f(x) = y$ and $p(|y|) \geq |x|$. We need something a little stronger: a function $f$ is consistently honest iff there is a polynomial $p$ such that for all $x$ in the domain of $f$, $p(|f(x)|) \geq |x|$.

**Lemma 6.3.** If $A \in PC$, and there is a consistently honest polynomially computable function $f$ from $A$ onto $B$, then $B \in PC$.

**Proof Sketch:** Let $T$ be an NDTM that generates $A$ according to the definition of PC, running in time $p()$, and let $q()$ be a monotonically increasing (wlog) polynomial governing the honesty of $f$. Create an NDTM $V$ that takes an integer $n$ as input, selects an integer $Q$ between $q(n)$ and $q(n + 1) - 1$, inclusive, computes the polynomial $q(Q)$, and gives this integer as input to an embedded $T$. The embedded NDTM $T$ produces $s$. Our NDTM $V$ then computes $f(x)$, outputs it, and halts. It is not hard to verify that $V$ is a generator for $B$. 

8
Recall that a p-isomorphism between languages A and B is a bijection \( f \) from A to B such that both \( f \) and \( f^{-1} \) are computable in (deterministic) polynomial time. The next result follows easily from Lemma 6.3.

**Theorem 6.4.** If \( A \) and \( B \) are p-isomorphic, then \( \{ A \in PC \} \iff \{ B \in PC \} \).

Let \( NPC \) denote the class of NP-complete languages. Applying Proposition 6.2, we have:

**Corollary 6.5.** All known NP-complete languages are in PC.

**Corollary 6.6.** If the Berman-Hartmanis conjecture is true, then \( NPC \subseteq PC \).

We next give a characterization of PC. Essentially, if there is an NDTM \( D \) that can efficiently produce arbitrarily long strings from \( L \), and \( L \in NP \), then \( L \in PC \). The idea of the proof is to randomly generate a string \( s \) and a certificate \( c \). If \( s \in L \) and \( c \) happens to be valid, output \( s \). Otherwise resort to \( D \). The reason we need \( D \) and not some specific member of \( L \) is that we need to recoup the time spent checking our random string, (and we need to produce a string of length \( \geq n \).) Notice that \( D \) could be described as \( D \) being able to produce efficiently members of some infinite subset of \( L \), of any minimum length.

**Theorem 6.7.** \( L \in PC \) if and only if \( L \in NP \), and there is an NDTM \( D \) which: (i) takes as input an integer \( n \); (ii) produces as output some \( s \in L \) with \( |s| \geq n \); and (iii) runs in time bounded by \( p(|s|) \) (not necessarily \( p(n) \)).

When we defined PC, we required our NDTM to be able to produce every member of \( L \), of all sizes. Theorem 6.7 shows that to be in PC, all we need is to be able to produce some member of \( L \), of arbitrarily large size, together with membership in NP. An example of the distinction here comes from prime numbers.

We don't know how to produce all the prime numbers according to the definition of PC. The theorem proves that technically, primes would be in \( PC \) if we could produce some infinite subset of the primes. For instance, if Fermat's conjecture (that \( 2^{2n} + 1 \) is prime for all \( n \)) were true, primes would be in \( PC \) even though we would have a handle on only this tiny subset of primes.

Primes are an example of a language in NP not known to be in PC. In fact, even assuming the GRH (Generalized Riemann Hypothesis), from which primes \( \in P \), does not give a method for producing a large prime on demand. If in addition, Kramer's conjecture were true (there exists at least one prime between \( x \) and \( x + \log^2 x \)) then we could test only quadratically many integers and produce at least one prime of desired size.

The preceding suggests that "hardness" of recognition is not equivalent to "hardness" of constructability. An easy result corroborates this:
Theorem 6.8. If for any $L \in NP$ we have $L \notin PC$ then $P \subsetneq PC$.

Finally, if recognition of a language is easy enough, then it is in PC.

Theorem 6.9. If $L$ can be recognized by a logspace-bounded Turing machine with one-way input tape, then $L \in PC$.

The major open question here of course is, does $PC = NP$?

We have considered two other definitions. The first one is weaker, requiring $T$ to halt with probability 1 along any computation path. The second is stronger: $T$ takes integer $n$ as input, and must either output a string $s$ with $n \leq |s| \leq q(n)$ (here $q$ is a polynomial), or conclude that no such $s$ exists. All the results in this section carry over in a suitable way (the stronger definition sometimes requires an additional non-sparse assumption). Thus there is a robustness of the properties of efficiently generatable test languages with respect to the definition.

Theorem 6.7 exploits a weakness in the requirement on the class PC. The generator it constructs is heavily biased towards a subset of $L$, though every string in $L$ has a nonzero chance of being generated. Another interesting way to restrict PC would be to require that for each $k$, every string of length $k$ be generated with equal probability.

7 Acknowledgements

The authors thank Eric Allender, Vijay Chandru, Andrew Odlyzko, Martha Pilcher, and Don Wagner for helpful discussions.

References

Unambiguous Factorization of Recursive Graph Classes

Richard B. Borie* R. Gary Parker† Craig A. Tovey‡

June 6, 1988

Abstract

The popular class of series-parallel graphs can be built recursively from single edges by combining smaller components via connections only at a fixed pair of vertices called terminals. This recursive construction property with a limited number of terminals is essential to the linear time solution of problems on these graphs. A second useful property of these graphs is that decomposition is unambiguous with respect to the series-parallel rules. This implies that the parse-tree of decomposition (which is required by the algorithms) can be determined in a straightforward manner by repeatedly applying the decomposition rules. Subject to retaining these properties, and requiring a natural consistency of the rules, we determine exactly how far the series-parallel graphs may be generalized. We show specifically that any consistent set of rules satisfying these two properties can be described by at most four triples of integers. Two such sets yield the series-parallel and Halin graphs, respectively.

Key Words. series-parallel graph, decomposition, recursive graph class, dynamic programming, linear-time algorithm.

AMS(MOS) subject classifications. 05C70, 68R10, 90C39.

Running Head: Factorization of Recursive Graph Classes.

*Supported by a National Science Foundation Graduate Fellowship, School of Information and Computer Science, Georgia Institute of Technology, Atlanta GA 30332
†School of Industrial and Systems Engineering, Georgia Institute of Technology
‡Supported by a Presidential Young Investigator Award from the National Science Foundation (ECS-8451032), and a matching grant from Digital Equipment Corporation, Schools of Information and Computer Science and Industrial and Systems Engineering, Georgia Institute of Technology
1 Introduction

It is by now well known that a large number of hard problems on graphs can be solved in linear time over series-parallel graphs. Most of the work in the field was initiated by Takamizawa et al. [9]. Since then, Bern, Lawler, and Wong [3], Wimer, Hedetniemi, and Laskar [11],[10], and others [4],[5],[6] have shown the existence of thousands of such algorithms.

Bern et al. observed that the linearity of these algorithms rested on the fact that the family of series-parallel graphs can be recursively defined in terms of decomposition rules: any graph from this class can be separated into smaller component graphs in the same class; the components are interconnected only at a limited number of distinguished vertices called terminals. Thus, linear time algorithms for other recursively-definable graph classes exist as well. A few such classes are known, such as the Halin graphs.

Given that so many ordinarily difficult problems can be solved efficiently on recursively-definable graphs, it is desirable to know how far can these classes extend? The goal of this paper is to answer that question. In section 2, we develop natural properties that a set of decomposition rules should satisfy in order to allow efficient dynamic programming algorithms. In the remainder of the paper we derive necessary and sufficient conditions for a rule set to satisfy these properties. The conditions turn out to be quite compact.

2 Recursively decomposable graphs

2.1 Decomposition via \((j,k,r)\)-operations

In this section we formally define operations that connect component graphs together at a limited number of distinguished vertices. For \(k \geq 2\), a \(k\)-terminal graph \(G = (V, U, E)\) has vertex set \(V\), a subset \(U \subseteq V\) of terminal vertices where \(|U| \leq k\), and edges specified by \(E \subseteq 2^V\). Now, given \(k\)-terminal graphs \(G_i = (V_i, U_i, E_i)\) for \(1 \leq i \leq m\), define a function \(f(G_1, \ldots, G_m) = (V, U, E)\) with \(V = \bigcup_{1 \leq i \leq m} V_i\), \(U \subseteq \bigcup_{1 \leq i \leq m} U_i\), and \(E = \bigcup_{1 \leq i \leq m} E_i\). Here \(V_i, U_i,\) and \(E_i\) are identified in the obvious way with the respective \(G_i\).

We shall call \(f\) a \((j, k, r)\)-operation if \(|U| = j \leq k\) and \(|\bigcup_{1 \leq i \leq m} U_i| = r \geq k\). Conceptually, a \((j, k, r)\)-operation joins \(k\)-terminal graphs \(G_i\) at their common terminals, producing a graph \(G\) with at most \(j\) terminals, and where up to \(r - j\) vertices that were terminals in the constituent \(G_i\) relinquish their distinguished status in the resultant \(G\). To illustrate this, three graphs are given in Figure 1, where the doubly circled vertices denote the terminals. If \(U\) is chosen to be \(\{1, 3, 9\}\), and \(f\) denotes a \((3, 4, 5)\)-operation, then \(G = f(G_1, G_2, G_3)\) results as shown in Figure 2.
Figure 1: Three 4-terminal graphs

Figure 2: The resulting 3-terminal graph
2.2 Efficient algorithms for recursively definable classes

We now illustrate how problems over graphs built by \((j, k, r)\)-operations can be solved efficiently using the standard recurrence relation method employed for series-parallel graphs \([3],[11]\). We take as a sample problem, that of 3-coloring the vertices of a graph. First, we state the well-known solution method for series-parallel graphs. (Actually, any series-parallel graph can be 3-colored, but the method will serve well for our expository purpose.) Suppose we have two series-parallel graphs \(G\) and \(H\) with terminals \(g_1, g_2\) and \(h_1, h_2\), respectively, which are to be connected in parallel. Can the resulting graph \(J\) be 3-colored? Call the colors red, white, and blue. Recursively, assume we know for each component \(G\) and \(H\) whether it can be 3-colored. Moreover, assume we know, for each possible coloring of \(g_1\) and \(g_2\), whether \(G\) can be 3-colored. Thus we might know, for example, that \(G\) cannot be 3-colored so that \(g_1\) and \(g_2\) are both red, but \(G\) can be 3-colored so that \(g_1\) is red and \(g_2\) is white. We know the corresponding information about \(H\) as well. When \(G\) and \(H\) are connected in parallel, \(g_i\) is identified with \(h_i\), resulting in the terminal \(j_i\) of \(J\). Thus, if there is any "match" between colorings of the \(g_i\) and \(h_i\) that is admissible for 3-coloring both \(G\) and \(H\), then \(J\) can be 3-colored. Moreover, the more detailed information, of which colorings of the \(j_i\) are consistent with \(J\) being 3-colored, can be obtained in the same way.

This is the standard dynamic programming method which underlies the linear time algorithms on series-parallel graphs. The time required is linear because there is only a fixed amount of information to be computed for each component \(J\): for each of the \(3^2 = 9\) colorings of the terminals \(j_i\), can \(J\) be 3-colored? It should now be obvious that if \(G, H,\) and \(J\) each had 3 terminals instead of 2, with \(G\) and \(H\) connected in parallel as described above, then the same method would work. We would now have to recursively determine, for each component \(J\), and each of the \(3^3 = 27\) colorings of the terminals \(j_i\), whether \(J\) can be 3-colored. On the other hand, if the number of terminals were unbounded, the amount of information required would grow exponentially.

As another more formal example, consider the generally \(NP\)-hard problem of finding the minimum vertex cover of a graph. Let \(n_X(G)\) denote the cardinality of a smallest vertex cover \(A\) of \(G = (V, U, E)\) subject to the conditions \(X = A \cap U\). Suppose that \(G = f(G_1, \ldots, G_m)\) for some \((j, k, r)\)-operation \(f\) and that \(n_{X_i}(G_i)\) is known for each \(G_i = (V_i, U_i, E_i)\) and for each \(X_i \subseteq U_i\). Then we can obtain the value of \(n_X(G)\) as

\[
\min\left\{| \bigcup_{i=1}^{m}(X_i) | + \sum_{i=1}^{m}(n_{X_i}(G_i) - | X_i |) : \exists Y s.t. X \subseteq Y \subseteq (\bigcup_{i=1}^{m} U_i) - (U - X); X_i = Y \cap U_i \forall i\right\},
\]

which is efficiently computable for bounded \(j, k, r\). Notice that the vertex cover problem is trivial on single edges. Thus it is solvable in polynomial time by a
dynamic programming algorithm on any recursive class of \( k \)-terminal graphs such that the rules are \((j, k, r)\)-operations, given the decomposition tree.

This discussion motivates our first requirement of the rule set:

**Finite Terminal Set:** the number of terminals must be limited by a finite number.

It should be clear that this condition is essential to the polynomial time performance of the algorithms [3],[11].

### 2.3 Unambiguous factorization

We must require something more than a finite terminal set: to apply an algorithm to a graph \( G \), the decomposition tree for a graph **must be known**. Given a graph \( G \), it must be easy to determine whether \( G \) is in the class or not, and if so to exhibit its decomposition tree. Therefore, a second useful property of a set of decomposition rules is:

**Unambiguous Factorization:** membership in the class can be recognized by simply applying the rules until no remaining subgraph can be decomposed further.

For example, series-parallel graphs may be decomposed simply by repeatedly applying the 2-terminal series and parallel decomposition rules.

Given a set of rules \( R \), we let \( \{R\} \) denote the recursive class of graphs that can be built from edges with \( R \). We say that an element \( G \in \{R\} \) is prime if there do not exist \( f_i \in R \) and \( G_1, \ldots, G_k \in \{R\} \) such that \( G = f_i(G_1, \ldots, G_k) \). A prime that is not a mere edge is a nontrivial prime.

A recursive class \( \{R\} \) has the unambiguous factorization property with respect to a partial order \( P \) on \( R \) if, for any decomposition of \( G \in W \) which uses minimally (with respect to \( P \)) applicable rules of \( R \) until each remaining element is prime, it is guaranteed that each such prime is an edge iff \( G \in \{R\} \). Thus many \( NP \)-hard problems are solvable in polynomial time with dynamic programming algorithms when instances are restricted to such graph classes.

### 2.4 Consistent rule sets

We will also require the rule set to satisfy a modest consistency property. Informally, a set \( R \) of rules is consistent if, when some decomposition rule \( r \) is in \( R \), then any rule \( r' \) weaker than \( r \) is in \( R \) also. For example, if the rules allow a parallel-type operation at 3 vertices, they must allow the same operation at just 2 vertices.

The decomposition of a graph \( G = (V, U, E) \) with \( |U| = j \) using a \((j, k, r)\)-operation requires the selection of a set \( Z \) such that \( U \subseteq Z \subseteq V \) and \(|Z| = r\). Denote
the components of $G - Z$ by $H_1, \ldots, H_m$. If each $H_i$ satisfies $|\Gamma_G(H_i) \cap Z| \leq k$ then the $(j,k,r)$-operation succeeds. (Here $\Gamma$ represents the neighborhood set.) This operation decomposes $G$ into $G_1, \ldots, G_m$, and possibly some components which are merely edges between terminals, where each $G_i$ is obtainable from $H_i$ by adding the terminals $\Gamma_G(H_i) \cap Z$ and the edges from $G$ which connect them to $H_i$.

A key outcome of unambiguous factorization is that rules from $R$ can be applied in any order allowed by a partial order $P$ until all remaining graphs are prime, at which point recognition reduces to checking that no primes are non-trivial. The unambiguous factorization property would thus guarantee a polynomial time recognition algorithm for the recursive class of graphs.

Note also that a graph's membership in $\{R\}$ is not independent of its specification of terminals. That is, there might be some $(V', U', E') \in \{R\}$ and vertices $x \in U', y \notin U'$ such that $(V', U' - \{x\} \cup \{y\}, E') \notin \{R\}$. For example, the graph below would not be series-parallel if $w$ and $y$ were the terminals.

Now, it might be best to refrain from making decisions during decomposition, such as whether to include $x$ or $y$ in the set of terminals, if such a decision could be postponed until later when additional information might be available. This leads to a critical observation: operations which allow vertices to be added into the terminal set arbitrarily during decomposition lead to ambiguous factorization. To see this, suppose $f \in R$ is an operation such that some $x \in U_i$ but $x \notin U$ and $x$ is not in any of the identifications. Let $G_i = (V', U', E')$ where $U' = U_i$. Select $G_1, \ldots, G_i-1, G_{i+1}, \ldots, G_m \in \{R\}$, so that $G = f(G_1, \ldots, G_m) \in \{R\}$. But then $G = f(G_1, \ldots, G_{i-1}, H, G_{i+1}, \ldots, G_m)$, where $H = (V', U' - \{x\} \cup \{y\}, E') \notin \{R\}$, so arbitrarily choosing $H$ rather than $G_i$ eventually leads to a nontrivial prime.

Consider an example from the (non-standard) class of non-biconnected series-parallel graphs, where the jackknife (e.g. see [6]) operation arbitrarily selects a vertex to become a terminal. If $x$ or $z$ is chosen, the graph will factor completely; but if $y$ is chosen, a nontrivial prime is reached.
These outcomes indicate that there should be some minimality requirement on the size of the set $Z$ employed to separate the graph, so that decisions regarding which vertices are to become terminals can be delayed until later during decomposition. These arguments motivate the following.

We shall call a rule set $R$ **consistent** if for each $(j, k, r)$-operation in $R$ (i.e., $(j, k, r) \in R$), the following properties hold.

1. If $k < r$ then $(j, k, r - 1) \in R$.
2. If $j < k$ then $(j, k - 1, r) \in R$.
3. If $j < k$ then $(j + 1, k, r) \in R$.
4. If $j > 1$ and $k > 2$ then $(j - 1, k - 1, r - 1) \in R$.

The first of these properties provides that it is never necessary to coerce vertices into terminals merely to raise the number of terminals to $r$. The second property simply states that it is not required that any of the values $|\Gamma_G(H_t) \cap Z|$ be exactly $k$. The third property satisfies the intuitive notion that classifying a vertex as a terminal only restricts the possible ways in which the graph can be separated at the next step, and hence reduces the freedom possible during selection of the separator. Thus the $(j + 1, k, r)$-operation is in a sense weaker than the $(j, k, r)$-operation.

The final property is the least obvious, so its inclusion is justified more formally. (A similar argument could be given for each of the previous three as well.) Suppose $G = (V, U, E)$ where $|U| = j - 1$ and $|V| \geq r$, and that $G$ can be decomposed with a $(j - 1, k - 1, r - 1)$-operation by choosing a separator $Z \supseteq U$ with $|Z| = r - 1$. Now, select any vertex $t \in V - Z$, and let $H = (V, U \cup \{t\}, E)$. Then $H$ can be decomposed by a $(j, k, r)$-operation using the separator $Z \cup \{t\}$. So the $(j, k, r)$-operation is at least as powerful as the $(j - 1, k - 1, r - 1)$-operation. Hence if $(j, k, r) \in R$ then we may assume that the weaker operation $(j - 1, k - 1, r - 1)$ is in $R$ as well.

Finally, let $(R, \triangleleft)$ denote the partial order on a consistent rule set as follows.

- $(j', k', r') \triangleleft (j, k, r)$ if $(j', k', r')$ is any of the operations given by the consistency properties.
• The $<$ relation is transitive.

If $R$ is consistent and finite, then it is completely specified by a list of its maximal elements with respect to $<$. In the following we take unambiguous factorization to be with respect to the natural partial order $(R, <)$.

In the following sections we establish succinct necessary and sufficient conditions on a consistent set of decomposition rules to satisfy the finite terminal and unambiguous factorization properties. This answers the question of how far series-parallel graphs can be extended, while preserving their desirable properties. It turns out that every consistent set satisfying the two properties can be characterized by specifying at most four triples of integers. Our results naturally include the series-parallel and Halin classes as particular cases.

3 Rule sets with ambiguous factorization

We now establish, in a series of lemmas, the classes of rule sets not satisfying unambiguous factorization. In the remainder of the paper, we assume the rule set $R$ to be consistent.

Lemma 1 If $(j, k, r) \in R$ is maximal, where $j < k$, $3 \leq k \leq r - 1$, $r \leq 2k - 2$, then $C$ does not satisfy unambiguous factorization.

Proof: Consider the graph $G = (V, U, E)$ with $|U| = j$, $t \in U$, $|A| = k - j + 1$, $|B| = \min(r - j, k - 1)$, $V = U \cup A \cup B$, $E_1 = \{t, a\} : a \in A$, $E_2 = \{\{a, b\} : a \in A, b \in B\}$, $E_3 = \{\{b, c\} : b \in B, c \in B - \{b\}\}$, $E_4 = \{\{b, u\} : b \in B, u \in U - \{t\}\}$, and $E = \bigcup_{1 \leq i \leq 4} E_i$. The following figure illustrates $G$ for the case $j = 3$, $k = 5$, $r = 7$. (For ease the edge set $E_3$, which denotes a clique on $B$, is depicted by encircling the vertices in $B$.)

![Graph illustration](image)

3 Rule sets with ambiguous factorization

We now establish, in a series of lemmas, the classes of rule sets not satisfying unambiguous factorization. In the remainder of the paper, we assume the rule set $R$ to be consistent.

Lemma 1 If $(j, k, r) \in R$ is maximal, where $j < k$, $3 \leq k \leq r - 1$, $r \leq 2k - 2$, then $C$ does not satisfy unambiguous factorization.

Proof: Consider the graph $G = (V, U, E)$ with $|U| = j$, $t \in U$, $|A| = k - j + 1$, $|B| = \min(r - j, k - 1)$, $V = U \cup A \cup B$, $E_1 = \{t, a\} : a \in A$, $E_2 = \{\{a, b\} : a \in A, b \in B\}$, $E_3 = \{\{b, c\} : b \in B, c \in B - \{b\}\}$, $E_4 = \{\{b, u\} : b \in B, u \in U - \{t\}\}$, and $E = \bigcup_{1 \leq i \leq 4} E_i$. The following figure illustrates $G$ for the case $j = 3$, $k = 5$, $r = 7$. (For ease the edge set $E_3$, which denotes a clique on $B$, is depicted by encircling the vertices in $B$.)

![Graph illustration](image)
Now, one minimal separator of $G$ is $U \cup B$, which has cardinality $j + \min(r - j, k - 1) \leq r$. It separates $G$ into some edges and $k - j + 1$ stars, $G_i = (B \cup \{t, a_i\}, B \cup \{t\}, \{\{a_i, x\} : x \in B \cup \{t\})$. Each star $G_i$ easily decomposes into edges at the next step.

Another minimal separator is $U \cup A$, which has cardinality $k + 1 \leq r$. It leaves a factor $H = (V - \{t\}, A \cup U - \{t\}, E - E_1)$, which has $k$ terminals. Observe that $|V - \{t\}| = k + \min(r - j, k - 1) = \min(r + k - j, 2k - 1) \geq r + 1$, so $H$ cannot be completely factored into edges at the next step. Suppose without loss of generality that $b_0 \in S$ and $b_1 \notin S$, where $S$ is the separator at the next step. So there will exist a factor containing $b_1$, the $k$ terminals of $H$, and another terminal $b_0$. Thus no such separator $S$ can exist, $H$ is prime, and there is not unambiguous factorization.

Lemma 2 If $(j, k, r) \in R$ is maximal, where $j < k$, $3 \leq k \leq r - 1$, $r \geq 2k - 1$, then $C$ does not satisfy unambiguous factorization.

Proof: Consider the graph $G = (V, U, E)$ with $|U| = j$, $t \in U$, $A = \{a_0, \ldots, a_{k-j}\}$, $B = \{b_0, \ldots, b_{r-j-1}\}$, $V = U \cup A \cup B$; $E_1 = \{\{t, a\} : a \in A\}$, $E_2 = \{\{a, b\} : (y - x) \mod (r - j) \leq k - 2\}$, $E_3 = \{\{b, c\} : b \in B, c \in B - \{b\}\}$, $E_4 = \{\{b, u\} : b \in B, u \in U - \{t\}\}$, and $E = \bigcup_{i=1}^{4} E_i$. The following figure illustrates $G$ for the case $j = 3$, $k = 5$; $r = 9$.

One minimal separator of $G$ is $U \cup B$. It has cardinality $r$, and separates $G$ into some edges and $k - j + 1$ stars, $G_i$, with a central nonterminal $a_i$ adjacent to $k$
terminals. Each star \( G_i \) easily decomposes into edges at the next step.

Another minimal separator is \( U \cup A \), which has cardinality \( k + 1 \leq r \). It leaves a factor \( H = (V - \{ t \}, A \cup U - \{ t \}, E - E_1) \), having \( k \) terminals. Observe that \( |V - \{ t \}| = r + k - j \geq r + 1 \), so \( H \) cannot be completely factored into edges at the next step. Now consider the bipartite subgraph \( K = (A \cup B, E_2) \), and notice \( G \) has been constructed such that each \( A' \subset A \) satisfies \( |\Gamma_K(A')| \geq |A'| + 1 \). Let \( S \) be the separator at the next step, suppose without loss of generality that \( b_i \not\in S \), and let \( A' \) be the subset of \( A \) which is not in the component containing \( b_i \). So there will exist a factor containing \( b_i \) and at least \( k + 1 \) terminals \( A \cup U \cup \Gamma_K(A') - \{ t \} - A' \). Thus no such separator \( S \) can exist, \( H \) is prime, and there is not unambiguous factorization.

\[ \square \]

**Lemma 3** If \((1, 2, r) \in R \) is maximal, where \( r \geq 3 \), then \( C \) does not satisfy unambiguous factorization.

Proof: Consider the graph \( G = (V, U, E) \) with \( U = \{ t \}, A = \{ a_0, \ldots, a_{r-3} \}, V = A \cup \{ t, b, c \}, E_1 = \{ \{ t, a \} : a \in A \}, E_2 = \{ \{ a, d \} : a \in A, d \in A - \{ a \} \}, \) and \( E = E_1 \cup E_2 \cup \{ \{ a_0, b \}, \{ a_1, b \}, \{ b, c \} \} \). The following figure illustrates \( G \) for the case \( r = 5 \).

![Graph Illustration](image)

One minimal separator of \( G \) is \( U \cup A \), which obviously leads to complete decomposition.

Another minimal separator of \( G \) is \( \{ t, b \} \), which leaves a factor \( H = (V - \{ c \}, \{ t, b \}, E - \{ \{ b, c \} \}) \). Observe that \( |V - \{ c \}| = r + 1 \), so \( H \) cannot be completely factored into edges by separator \( S \) at the next step. If \( \{ a_0, a_1 \} \not\subset S \), then some component will contain at least 3 terminals \( t, b, a_i \), where \( a_i \in S \). But if \( \{ a_0, a_1 \} \subset S \), then some component will contain at least 3 terminals \( t, a_0, a_1 \). Thus no such separator \( S \) can exist, \( H \) is prime, and there is not unambiguous factorization.

\[ \square \]
Lemma 4 If \((j, k, k) \in R\) is maximal, where \(j < k\), then \(C\) does not satisfy unambiguous factorization.

Proof: Consider the graph \(G = (V, U, E)\) with \(|U| = j\), \(|A| = k - j\), \(V = U \cup A \cup \{b, c\}\), \(E_1 = \{(u, a) : u \in U, a \in A\}\), \(E_2 = \{(a, d) : a \in A, d \in A - \{a\}\}\), \(E_3 = \{(a, b) : a \in A\}\), and \(E = E_1 \cup E_2 \cup E_3 \cup \{\{b, c\}\}\). The following figure illustrates \(G\) for the case \(j = 4\), \(k = 9\).

One minimal separator of \(G\) is \(U \cup A\), which has cardinality \(k\). It separates \(G\) into some edges and a graph \((A \cup \{b, c\}, A, E_3 \cup \{\{b, c\}\})\), which decomposes into edges by choosing the separator \(A \cup \{b\}\) at the next step.

Another minimal separator is \(U \cup \{b\}\), which has cardinality \(j + 1 \leq k\) and which leaves a factor \(H = (V - \{c\}, U \cup \{b\}, E - \{\{b, c\}\})\). Observe that \(|V - \{c\}| = k + 1\), so \(H\) cannot be completely factored into edges at the next step. Suppose without loss of generality that there exists a sequence of minimal separators \(S_0 = U \subset S_1 = U \cup \{b\} \subset S_2 \subset \ldots \subset S_n = V - \{c\}\) which decomposes \(H\) into edges, where each \(|S_i| = m_i\). This amounts to applying a sequence of operations with the forms \((m_0, m_1, m_1), (m_1, m_2, m_2), \ldots, (m_{n-1}, m_n, m_n)\). But this is equivalent to applying a single \((m_0, m_n, m_n)\)-operation, which is actually a \((j, k+1, k+1)\)-operation. But this operation contradicts the minimality of \((j, k, k)\), so \(H\) does not factor completely and there is not unambiguous factorization.

\(\square\)

Lemma 5 If \((k, k, r) \in R\) is maximal, where \(4 \leq k \leq r - 2\), then \(C\) does not satisfy unambiguous factorization.

Proof: Consider the graph \(G = (V, U, E)\) with \(|U| = k\), \(\{t_0, t_1\} \subseteq U\), \(A = \{a_0, \ldots, a_{r-1}\}\), \(V = U \cup A \cup \{b\}\), \(E_1 = \{(u, a) : u \in U - \{t_0, t_1\}, a \in A\}\), \(E_2 = \)
\[ \{a_x, a_y\} : x < y \}, \text{ and } E = E_1 \cup E_2 \cup \{\{t_0, b\}, \{t_1, b\}, \{b, a_0\}, \{b, a_1\}\}. \] The following figure illustrates \( G \) for the case \( k = 5, r = 9 \).

One minimal separator of \( G \) is \( U \cup \{a_0, a_1\} \), which has cardinality \( k + 2 \leq r \). It separates \( G \) into some edges and two graphs, both of which easily decompose into edges at the next step.

An alternative minimal separator is \( U \cup \{b\} \), which leaves a factor \( H = (V - \{t_0, t_1\}, U - \{t_0, t_1\} \cup \{b\}, E - \{\{t_0, b\}, \{t_1, b\}\}) \) with \( k - 1 \) terminals. Observe that \( |V - \{t_0, t_1\}| = (k - 1) + (r - k + 2) = r + 1 \), so \( H \) cannot be completely factored into edges by the separator \( S \) used at the next step. If \( \{a_0, a_1\} \not\subseteq S \), then some component will contain \( m = |S| \geq k \) terminals, so \( (k - 1, m, m) \in R \), but this is impossible due to the previous lemmas. But if \( \{a_0, a_1\} \subseteq S \), then some component will contain \( |S| - 1 \geq k \) terminals, which yields a similar contradiction. Thus no such separator \( S \) can exist, \( H \) is prime, and there is not unambiguous factorization.

4 Characterization of unambiguously factorable graph classes

The preceding lemmas exclude certain classes of rules if decomposition is to be unambiguous. The lemmas can thus be taken as providing necessary conditions for
unambiguous decomposition. The main result of this paper is that these conditions are sufficient as well. To prove this we will need a definition and a technical lemma.

Definition 6 The function $P(G, X)$, where $X$ is a subset of the nonterminals of $G$, denotes the maximum number of vertex-disjoint paths from not necessarily distinct vertices in $X$ to distinct terminals of $G$.

Now the lemma:

Lemma 7 If $f$ is a $(k, k, r)$-operation with either $k \leq 3$ or $r \leq k + 1$, $G = f(\ldots, H, \ldots)$, and $X$ is a subset of the nonterminals of $H$, then $P(G, X) = P(H, X)$.

Proof: Suppose $P(G, X) \neq P(H, X)$. Obviously $P(G, X) \leq P(H, X)$, because the same disjoint paths as in $G$ can be followed outward from $X$ until reaching distinct terminals of $H$. Hence it must be that $P(G, X) < P(H, X)$. Suppose the operation $f$ extends the terminal set $U$ of $G$ to $Z$ before performing the separation. Then in the decomposition tree, $H$ has a sibling $H'$ with vertex set $V'$, such that there exists a cutset $C \subseteq V'$ between $V' \cap U$ and $V' \cap (Z - U)$ which satisfies the inequality

$$1 \leq |C| < |V' \cap (Z - U)| \leq |V' \cap U|.$$  

(The cutset $C$ restricts the number of paths in $G$ between vertices in $X$ and terminals in $V' \cap U$, but then the number of paths in $H$ between $X$ and $V' \cap (Z - U)$ is not restricted by $C$ because they do not pass through $C$.) Therefore it must be that

$$k \geq |V' \cap Z| \geq |V' \cap (Z - U)| + |V' \cap U| \geq 2 + 2 = 4$$

and

$$r - k \geq |V' \cap (Z - U)| \geq 2.$$

Thus if $k \leq 3$ or $r \leq k + 1$, then $P(G, X) = P(H, X)$. $\square$

We come now to the main theorem which establishes exactly when factorization is unambiguous.

Theorem 8 If $R$ is consistent, then the recursive class $\{R\}$ satisfies unambiguous factorization with respect to $(R, \triangleleft)$ iff each maximal operation in $R$ has one of the following forms.

- $(2, 2, r)$
- $(3, 3, r)$
- $(k, k, k)$
(k, k, k + 1)

Proof: If \{R\} is unambiguously factorable, then Lemmas 1, 2, 3, 4, and 5 guarantee that only the given forms for maximal \((j, k, r)\)-operations are allowed in \(R\).

Conversely, suppose each maximal operation in \(R\) has one of the given allowable forms, and let \(G_0\) be a graph in \{R\} with the smallest number of vertices which is not unambiguously factorable. Thus there must exist an operation \(h \in R\) which extends the terminals of \(G_0\) by adding new terminals \(T\) before performing the separation, such that \(G_0 = h(\ldots, G_0', \ldots)\), the graph \(G_0'\) is smaller than \(G_0\) and hence is unambiguously factorable, and \(G_0'\) will eventually lead to a non-trivial prime containing a nonterminal \(w\).

Since \(G_0 \in \{R\}\), it has a decomposition tree \(D\). Consider the path in \(D\) from the root to the node at which \(w\) becomes a terminal, denoted by \(G_i = f_i(\ldots, G_{i+1}, \ldots)\) for \(0 \leq i \leq m - 1\), where \(w\) is a nonterminal of \(G_m\) but a terminal of \(G_i\). We will exhibit a path \(G_i' = f_i'(\ldots, G_{i+1}', \ldots)\) for \(0 \leq i \leq m - 1\), such that \(w\) is a terminal in \(G_i'\), which will contradict the unambiguous factorability of \(G_0'\) and thus also the selection of \(G_0\). Denote each \(G_i = (V_i, U_i, E_i)\) and \(G_i' = (V_i', U_i', E_i')\). Let \(f_i = (k_i, k_i, r_i)\) use the cutset \(C_i \supseteq U_i\), and observe that each \(k_i \geq k_{i+1}\). Thus \(f_i' = (k_i', k_i', r_i')\) has cutset \(C_i' = (C_i \cup T) \cap V_i'\), where \(C_i' \supseteq U_i'\). Hence \(U_i' = (U_i \cup T) \cap V_i'\).

It is necessary to show that each \(f_i'\) is in \(R\) and that each \(f_i'\) can be applied to decompose the corresponding \(G_i'\).

To verify that \(f_i'\) is in \(R\), it suffices to show that \(f_i' < f_i\), or equivalently that \(k_i' \leq k_i\) and \(r_i' - k_i' \leq r_i - k_i\). But \(k_i' = |U_i'|\), \(k_i = |U_i|\), \(r_i' = |C_i'|\), and \(r_i = |C_i|\).

First notice that

\[
r_i' - k_i' = |C_i' - U_i'| = |(C_i - U_i) \cap V_i'| \leq |C_i - U_i| = r_i - k_i.
\]

Next suppose that \(k_i' > k_i\). If \(X\) denotes the set of nonterminals of \(G_i'\), then \(P(G_i', X) > P(G_i, X)\). Letting \(Y = U_i' - U_i\), then \(G_i\) must have a cutset \(C\) between \(Y\) and some \(Z \subseteq U_i - U_i'\), such that \(1 \leq |C| \leq \min(|Y| - 1, |Z|)\). Thus \(|T| \geq |Y| \geq 2\), which implies that \(k_0 \leq 3\). But \(k_0 \geq |Y| + 1 \geq 3\), so \(|Y| \leq k_0 = 3\) and there are two cases to consider.

1. If \(|Y| = 2\) then consider the operation \(f_j\) at which the number of terminals in \(G_j\) reachable from \(X\) through \(Y\) first exceeds the number of such terminals in \(G_{j+1}\), which must exist since \(k_0 > 2\). But then \(k_j \geq 4\), which is not true.
2. If \(|Y| = 3\) then at most 2 terminals are reachable by paths from \(X\) through \(Y\), and each of these is reachable through at most 2 of the 3 vertices in \(Y\), which contradicts the minimality of operation \(h\).

This contradiction guarantees that

\[
P(G_i', V_i' - U_i') = P(G_i, V_i' - U_i'), \quad (1)
\]
and therefore that \( k_i' \leq k_i \). Hence \( f_i' \prec f_i \) and \( f_i' \) is in \( R \).

To verify that each \( f_i' \) can be applied to decompose \( G_i' \), first observe that any operation in \( R \) (which is the closure of the maximal operations under the consistency properties) satisfies the hypotheses of Lemma 7. Choose the smallest \( i \) such that \( f_i' \) is not applicable because \( G_{i+1}' \) has more terminals than its parent \( G_i' \). If \( X \) denotes the set of nonterminals of \( G_{i+1}' \), then \( P(G_{i+1}', X) > P(G_i', X) \). But \( P(G_i', X) = P(G_i, X) \) by choice of \( i \), and \( P(G_i, X) = P(G_{i+1}, X) \) by operation \( f_i \), so \( P(G_{i+1}', X) > P(G_{i+1}, X) \). But this contradicts Equation 1, so each \( f_i' \) can be applied.

\( \square \)

**Corollary 9** Any consistent \( R \) satisfying the unambiguous factorization property can be specified by a set of at most four incomparable maximal operations.

An example of such operations is \{\((2, 2, 9), (3, 3, 8), (5, 5, 6), (7, 7, 7)\)\}.

**Corollary 10** The subclass of biconnected series-parallel graphs can be specified by the maximal \((2, 2, 8)\) operation, and thus satisfies unambiguous factorization. The Halin graphs are contained in \{\((3, 3, 4)\)\}, and thus satisfy unambiguous factorization.

## 5. Summary

In a very real sense, and subject to rather mild assumptions, we have shown how far the popular class of series-parallel graphs can be extended. In this regard, it would be interesting to discover what kinds of problems yield to polynomial time algorithms on these classes. It would also be worthwhile to establish NP-completeness results for problems remaining intransigent on the more extended classes. For example, some problems that are hard on arbitrary graphs, but open on series-parallel graphs, may be NP-hard on a somewhat larger class of recursive graphs.

It would also be interesting to determine other existing or newly formed graph classes which are generalized by the \((j, k, r)\) operations. Those that are recursively defined but do not satisfy the unambiguous factorization property would also be interesting to identify.

One class of recursively constructed graphs that has received recent attention, the partial \( k \)-trees [1], do not satisfy unambiguous factorization (for \( k \geq 3 \)). Hence, their factorization is somewhat more problematic. Arnborg, Corneil, and Proskurowski [2] have shown that a “bottom-up” method (as opposed to the usual top-down procedure) can be used to achieve recognition in \( O(n^{k+2}) \) time. Recently, Robertson and Seymour have announced [7],[8] that an asymptotically more efficient factorization of partial \( k \)-trees exists: for an actual implementation for any \( k \), the set of minimal forbidden minors must be explicitly known. In contrast, factorization of the classes given in Theorem 8 is straightforward, but not implied by the Robertson-Seymour work because the classes are not closed under the subgraph operation.
References


Automatic Generation of Linear Algorithms from Predicate Calculus Descriptions of Problems on Recursively Constructed Graph Families

Richard B. Borie*    R. Gary Parker†    Craig A. Tovey‡

July 21, 1988

Abstract

We develop a calculus in which problems on graphs can be expressed. Any problem so expressible can be solved in linear time on any recursively constructed graph family. Moreover, the linear time algorithm can be generated automatically from the expression. Our approach has several advantages over prior efforts to construct linear algorithms on series-parallel and other recursive graphs. The calculus frees us from the ad hoc nature of the previous constructions; it is built from only canonical graph properties (such as vertex-edge incidence) and the most fundamental logical operations (and, or, not, quantification); and it accounts for nearly all the known problems so solvable, as well as clearing up many open or unverified problems.

*School of Information and Computer Science, Georgia Institute of Technology, Atlanta GA 30332
†School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta GA 30332
‡Supported by a Presidential Young Investigator Award from the National Science Foundation (ECS-8451032) and a matching grant from Digital Equipment Corporation, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta GA 30332
Introduction

Numerous research efforts have focused on the development of efficient algorithms for $NP$-hard problems when instances are restricted to graphs belonging to certain recursively constructed families. Among these are trees, series-parallel graphs, Halin graphs, partial $k$-trees, and bandwidth-$k$ graphs. The common feature of these families is that any sufficiently large member can be composed from smaller members of the same family, joined at special vertices called terminals. Fast algorithms on these graphs are typically based on dynamic programming, so that a solution to a large member can be determined directly from solutions to the smaller members which constitute it, by a recurrence relation specific to the problem. The number of terminals is restricted to a fixed $k$, so this recurrence relation can be evaluated efficiently, which in turn leads to an efficient algorithm (assuming a decomposition tree for any graph in the recursive family can be found quickly).

Usually, the recurrence relation is found by intuitive methods, with complicated problems requiring much attention to details. In this paper we show how to automatically generate the recurrence from a problem description in a particular predicate calculus. To obtain a linear time algorithm for a problem on recursively constructed graphs, one simply states the problem in our calculus, and the rest follows automatically. (This idea was partly inspired by Lauriere in [12], where problems are solved by branch-and-bound techniques starting from a formal problem description.) Our method employs the notions of homomorphic classes and regular predicates (suitably extended) introduced by Bern, Lawler, and Wong in [6].

Takamizawa, Nishizeki, and Saito in [15] appear to have initiated the research activity on efficient algorithms for recursively constructed graphs, working with series-parallel graphs. Richey in [14] developed other algorithms on the same family. Arnborg, Corneil, and Proskurowski in [1], [2], and [4] have generalized to partial $k$-trees, while Wimer, Hedetniemi, and Laskar in [16] and [17] list a large number of problems and graph families for which such problems are linear time solvable.

Alternately, Bern, Lawler, and Wong in [6] have examined why certain $NP$-hard problems on restricted families of graphs have linear time algorithms. We summarize their results as follows: Let $F$ be a recursive graph family, and let $P$ be a predicate on $F_S = \{(G, S) : G = (V, E) \in F, S \subseteq V\}$. A problem is to find the optimum $S \subseteq V$ that satisfies $P$. If $o$ is a binary
recursive operator of $F_S$ and there exists a set $C$ with multiplicative operator $\odot$, then $h : F_S \rightarrow C$ is a homomorphism if both

1. $h(G_1, S_1) = h(G_2, S_2) \rightarrow P(G_1, S_1) = P(G_2, S_2)$ and

2. $h((G_1, S_1) \circ (G_2, S_2)) = h(G_1, S_1) \odot h(G_2, S_2)$.

If $C$ is finite, then explicit multiplication tables can be constructed, and $P$ is said to be regular. Further, if a decomposition tree for $G \in F$ is given and $P$ is a regular property, then there exists a linear algorithm to find an optimal subset $S$ satisfying $P$, using a straightforward dynamic programming approach. The set of regular properties is closed under logical negation, conjunction, and disjunction, as well as restricted forms of maximality and minimality that would be more precisely called vertex criticality.

More recently, Mahajan and Peters in [13] show that computation of the finite multiplication tables is undecidable, but that for certain properties which are local (independence, domination, irredundance, etc.), these tables are effectively computable. Bauderon and Courcelle in [5] and [8] define context-free graph grammars, where a graph refers to an oriented hypergraph, and thereby obtain an alternative notion of recursive graph families, with a rewriting rule corresponding to a recursive operation; they also develop a systematic method for expressing some local properties. Arnborg, Lagergren, and Seese in [3] extend these results to allow directed or undirected graphs with edge or vertex labels, and consider problems involving counting and summing which can be solved in linear, polynomial, and pseudopolynomial time. Finally, Bodlaender in [7] solves problems which satisfy a composition of certain locality conditions, where instances are restricted to graphs with bounded tree-width. Our results build substantially on the efforts of Bern et al [6] summarized above, and free us from the ad hoc nature of previous constructions.

In this paper we develop a primitive calculus for stating graph problems, such that any expressible problem is solvable in linear time on recursively constructed graph families. The proofs embody a method for automatically generating homomorphic classes and recurrence relations to encode the relevant information about a graph problem, thus mechanically creating a dynamic programming algorithm. The calculus comprises a small number of easily checkable primitive predicates, combined by logical operators and
quantifiers. Furthermore, the calculus is extensible, merely by the addition of other primitive predicates, and one useful extension is given.

Many common graph properties and concepts such as adjacency, vertex degree, connectivity, forests, cycles, matchings, minor subgraphs, and isomorphism are expressible in the calculus. Most of the known linear algorithms on $k$-terminal graph families, including those for vertex cover, Hamiltonian cycle, Steiner subgraph, and chromatic number, are expressible as well. Maximality and minimality are expressed in complete generality, so problems such as min-max matching are included too. Not only the existence of a solution to an expressible predicate can be determined: optimum cardinality sets, optimum weighted sets, and even the number of distinct solutions can also be found.

We also give a small number of problems that are solvable in linear time on recursive families, but that do not appear to be expressible in the calculus without further extension. Other problems that are solvable in polynomial, but not linear, time on recursive graph families are discussed, as are problems which remain $NP$-hard on recursive families. The latter are especially important from the perspective of the implied restrictions on further extensions to the calculus.

Before proceeding further, we wish to emphasize several important characteristics of our calculus for expressing graph problems. First, the calculus is elegantly defined in terms of a small canonical set of primitive predicates, which are combined by logical operators and quantifiers to form legal expressions.

Second, many graph problems stated in Garey and Johnson [10] can be translated into our calculus in a straightforward manner. For example, consider the vertex cover problem:

**INSTANCE:** Graph $G = (V, E)$, positive integer $k \leq |V|$.

**QUESTION:** Is there a subset $V_1 \subseteq V$ with $|V_1| \leq k$ such that for each edge $\{u, v\} \in E$ at least one of $u$ and $v$ belongs to $V_1$?

This problem can be stated succinctly in our calculus as

$$\min |V_1| : (\forall e_1)(\exists v_1 \in V_1)(\text{Inc}(v_1, e_1)).$$

We incorporate the use of macros to make expressions concise, so the Hamil-
tonian cycle problem can be stated as

\[(\exists E_1)(\text{Conn}(V, E_1) \land \text{Reg}_2(V, E_1)),\]

where \(\text{Conn}\) and \(\text{Reg}_m\) are predicates (expressed in the calculus) that test for connectedness and regularity of degree \(m\).

Third, our calculus has considerable scope. For example, it easily solves nine of the ten problems listed by Johnson in [11] for recursively constructed graphs; this includes many which were listed as open or unverified (see Corollary 8). Indeed, there are only a few problems known to be solvable in linear time by dynamic programming on recursive graph families that are not known to be expressible in our calculus, so the coverage of our calculus is almost complete. In later sections of this paper we examine several such problems and conjecture that there does not exist any calculus which completely covers all such problems while still yielding an automatic algorithm generator, and hence that our results are in a sense the best possible.
Definitions

A \textit{k-terminal graph} $G = (V, T, E)$ is a graph with an ordered set $T \subseteq V$ of distinguished vertices, known as \textit{terminals}, such that $| T | \leq k$. Here $T = \{ t_1, \ldots, t_{| T |} \}$, where $t(G) = | T |$.

A \textit{c-ary k-terminal recursive operation} $f$ is written $G = f(G_1, \ldots, G_c)$, where $G$ and the $G_j$ are k-terminal graphs. $f$ can be represented by a matrix $M(f)$ with $r$ rows and $c$ columns, such that $0 \leq M_{i,j}(f) \leq t(G_j)$ for $1 \leq i \leq r$, $1 \leq j \leq c$. The nonzero elements $M_{i,j}(f)$ of the $i$th row indicate which terminals of each $G_j$ are merged together, or \textit{identified}, to create the $i$th new vertex of $G$. (If an element has value 0, then no terminal from $G_j$ is used in the creation of the $i$th vertex.) The first $t(G)$ rows of $M(f)$ indicate the ordered set of terminals of $G$, and the remaining rows indicate new vertices which result by identifying terminals from the $G_j$ but which become nonterminals of $G$. Therefore we let $t(f) = t(G)$, and $f$ is completely specified as $f = (M, t)$. As an example, consider the following 4-ary 3-terminal operation $G = f(G_1, \ldots, G_4)$, with $M(f)$ as shown and $t(f) = 3$. 

\begin{figure}
\centering
\includegraphics[width=\textwidth]{diagram}
\caption{Example of a 4-ary 3-terminal operation}
\end{figure}
A base graph is a k-terminal graph with no nonterminal vertices. A decomposition tree of a k-terminal graph $G$ is a rooted tree with vertex labels $g$ and $f$ such that

1. $g_v$ is a base graph if $v$ is a leaf,
2. $f_v$ is a recursive operation if $v$ is interior,
3. $g_v = f_v(g_{v_1}, \ldots, g_{v_c})$ if $v$ is interior with children $v_1, \ldots, v_c$, and
4. $g_v = G$ if $v$ is the root.

Given a set of $k$-terminal base graphs $B$ and a finite set of $k$-terminal recursive operations $R$, the $k$-terminal recursive family $F = (B, R)$ is the closure of $B$ by the operations in $R$. Well known recursive families include trees, outerplanar graphs, series-parallel graphs, Halin graphs, partial $k$-trees, and bandwidth-$k$ graphs. Notice that each $G \in F$ has a decomposition tree such that $g_v \in B$ if $v$ is a leaf and each $f_v \in R$. Our aim is to determine which $NP$-hard problems can be solved efficiently when the instance is restricted
to a graph $G \in F$, assuming a decomposition tree of $G$ is given along with
the instance. Note that the instance is not allowed to include any numbers,
so recognition versions of problems often must either include constants in the
problem description or be expressed as optimization versions instead. Also,
the instance cannot contain more than one graph, so problems like subgraph
isomorphism cannot be expressed in full generality.

We now develop a predicate calculus in which to express problems that
can be solved efficiently on a recursive family $F = (B, R)$. A problem in-
stance includes a graph $(V, E) \in F$ and possibly specified subsets $V' \subseteq V$
or $E' \subseteq E$. Let variables range over obvious domains; for example, $v_i \in V$,
e_i \in E, $V_i \subseteq V$, and $E_i \subseteq E$. Also, we shall let a $t$-ary predicate take the
form $P(x_1, \ldots, x_t)$, where the $x_i$ are variables.

Following [6], the predicate $P$ is regular if there is a finite set $C$ and a
homomorphism $h : F_{x_1, \ldots, x_t} \rightarrow C$ such that, for each $f \in R$ and corresponding
$\circ$ (which guarantees that the free variables of $P$ are selected in a compatible
way when $f$ is applied), there exists a multiplicative operator $\odot$ of $C$ such that

1. $h(G_1, x_{1,1}, \ldots, x_{1,t}) = h(G_2, x_{2,1}, \ldots, x_{2,t})$
   \[ \rightarrow P(G_1, x_{1,1}, \ldots, x_{1,t}) = P(G_2, x_{2,1}, \ldots, x_{2,t}) \text{ and} \]

2. $h(\circ((G_1, x_{1,1}, \ldots, x_{1,t}), \ldots, (G_n, x_{n,1}, \ldots, x_{n,t})))$
   \[ = \odot(h(G_1, x_{1,1}, \ldots, x_{1,t}), \ldots, h(G_n, x_{n,1}, \ldots, x_{n,t})). \]

An element $c \in C$ is said to be accepting if $h(G, x_1, \ldots, x_t) = c$ implies that
$P(G, x_1, \ldots, x_t)$ is true.

If a decomposition tree for $G \in F$ is given and $P$ is regular, then there
is a linear algorithm to find optimal values $x_1, \ldots, x_t$ satisfying $P$, using a
straightforward dynamic programming approach. In the next section, we
give a detailed example of this after first finding some regular predicates.
Regular Predicates

In this section we establish the regularity property with respect to a set of primitive predicates. These primitives are then used to generate other predicates, or macros, which are also shown to be regular. (In the next section we will demonstrate the use of these results by formalizing the existence of linear time algorithms on recursively constructed graph families.) We begin by showing that the primitive predicates vertex equality, vertex-edge incidence, membership, and a congruence property for cardinality are regular.

Theorem 1 Each of the following predicates is regular.

1. \( v_1 = v_2 \).
2. \( \text{Inc}(v_1, e_1) \).
3. \( v_1 \in V_1 \).
4. \( e_1 \in E_1 \).
5. \( |E_1| \equiv a \pmod{b} \).
6. \( |V_1| \equiv a \pmod{b} \).

Proof:

1. \( v_1 = v_2 \). Let \( C = \{0,1\} \), with 1 the accepting class (such that the predicate is true). If \( G \) is a base graph, let \( h(G,v_1,v_2) \) be 1 if both \( v_1 \) and \( v_2 \) are vertices of \( G \) such that \( v_1 = v_2 \), and 0 otherwise. If \( G = o(G_1, \ldots, G_m) \) (where we slightly abuse the notation and assume that \( o \) guarantees the selections of \( v_1 \) and \( v_2 \) in each \( G_i \) are mutually compatible), then let \( \circ \) be \( \lor_m \) (m-ary logical disjunction).

2. \( \text{Inc}(v_1, e_1) \). Again consider \( C = \{0,1\} \). If \( G \) is a base graph, let \( h(G,v_1,e_1) \) be 1 if both \( v_1 \) and \( e_1 \) are in \( G \) and \( v_1 \) is incident to \( e_1 \), and 0 otherwise. If \( G = o(G_1, \ldots, G_m) \), then let \( \circ \) be \( \lor_m \) as before.

3. \( v_1 \in V_1 \). \( C = \{0,1\} \), and for \( G \) a base graph with vertices \( V \), let \( h(G,v_1,V_1) \) be 1 if \( v_1 \in V_1 \cap V \) and 0 otherwise. If \( G = o(G_1, \ldots, G_m) \), then again let \( \circ \) be \( \lor_m \).
4. $e_1 \in E_1$. $C = \{0,1\}$, and for $G$ a base graph with edge set $E$, let $h(G, e_1, E_1)$ be 1 if $e_1 \in E_1 \cap E$ and 0 otherwise. If $G = \circ(G_1, \ldots, G_m)$, let $\varnothing$ be $\vee_m$.

5. $|E_1| \equiv a \pmod{b}$. Let $C = \{0, \ldots, b - 1\}$, with accepting state 0. If $G$ is a base graph with edge set $E$, let $h(G, e_1, E_1)$ be $|E_1 \cap E| \pmod{b}$. If $G = \circ(G_1, \ldots, G_m)$, let $\varnothing$ be the sum $\pmod{b}$ of the $h(G_j)$.

6. $|V_1| \equiv a \pmod{b}$. This case is essentially the same as the previous one, except that $\varnothing$ compensates for multiple counts of $v_1$ when $v_1$ is a terminal which appears in more than one $G_j$. (This predicate is not needed in what follows, and is in fact later rendered redundant by a extension to the calculus based on a stronger primitive, but it is included here for completeness.)

\[ \square \]

The next theorem shows that the primitive predicates can be employed to construct others which also satisfy the regularity property. Specifically, any predicate constructed from the primitive predicates using logical negation, conjunction, disjunction, universal and existential quantification is regular.

**Theorem 2** The set of regular predicates is closed under $\neg$, $\land$, $\lor$, $\forall$, and $\exists$, where quantification is over variables which range over vertices, edges, vertex sets, and edge sets.

**Proof:** To simplify matters, transform any expression $Q$ into prenex normal form, such that all the quantifiers are brought to the outermost nesting level. That is, let $Q(x_1, \ldots, x_t) = (q_1 y_1) \cdots (q_s y_s) P(y_1, \ldots, y_s, x_1, \ldots, x_t)$, where each $q_i$ is either $\forall$ or $\exists$. Let the primitive properties which are embedded in $P$ (those that contain no $\neg$, $\land$, or $\lor$) be $P_1, \ldots, P_r$. Each $P_i$ is regular, so it is recognizable by a homomorphism with some constant number $c_i$ of classes. Thus the number of different values of the vector $[P_1, \ldots, P_r]$, which represents the classes corresponding to $P$, is a constant $c = \prod_{i=1}^{r} c_i$. But whether a class is accepting for particular values of the $y_i$ can be determined from whether each of the components $P_i$ of the vector is accepting for those values of the $y_i$, by applying the operators $\neg$, $\land$, and $\lor$. Hence for fixed $y_i$, the expression $P$ is regular.

10
Let $Q_i = (q_1y_i) \cdots (q_my_y)P(y_1, \ldots, y_m, x_1, \ldots, x_t)$. Since it has just been shown that $Q_{i+1}$ is regular, it suffices to show for $s \geq i \geq 1$, that if $Q_{i+1}$ is regular then $Q_i$ is regular. There are four cases.

1. $y_i$ is a vertex.
   For $1 \leq j \leq k$ let $C_j$ be the set of homomorphic classes of $Q_{i+1}$ when $y_i$ is the $j$th terminal. Let $I$ be the set of possible sets of homomorphic classes of $Q_{i+1}$, one set for $y_i$ being each interior (nonterminal) vertex. Let $E$ be the set of homomorphic classes of $Q_{i+1}$ when $y_i$ is an exterior vertex (i.e., one which is not in this component of the decomposed graph). Then the cross product $C'' = C_1 \times \cdots \times C_k \times I \times E$ is the set of homomorphic classes of $Q_i$, where $c' \in C''$ is accepting iff either
   - $q_i = \exists$, each $c'[j]$ contains an accepting class for $1 \leq j \leq k$, and each set in $c'[k + 1]$ contains an accepting class, or
   - $q_i = \forall$, each $c'[j]$ contains only accepting classes for $1 \leq j \leq k$, and each set in $c'[k + 1]$ contains only accepting classes.

   Notice that $|C''| = (|C|)^k(|E|)^2$, so $Q_i$ is regular.

2. $y_i$ is an edge.
   Let $I$ be the set of possible sets of homomorphic classes of $Q_{i+1}$, one set for $y_i$ being each interior edge. Let $E$ be the set of homomorphic classes of $Q_{i+1}$ when $y_i$ is an exterior edge. The cross product $C'' = I \times E$ is the set of homomorphic classes of $Q_i$, where $|C''| = 2^{|C|} |E|$, so $Q_i$ is regular.

3. $y_i$ is a vertex set.
   Let $C'' = \Pi_{A \subseteq T} C_A$, where $C_A$ denotes the set of homomorphic classes of $Q_{i+1}$ such that $A = y_i \cap T$. Thus $|C''| = (2^{|C|})^2$, and $Q_i$ is regular.

4. $y_i$ is an edge set.
   $C'' = \{C_A : A \subseteq E\}$, where $C_A$ denotes the homomorphic class of $Q_{i+1}$ such that $A = y_i \cap E$. So $|C''| = 2^{|C|}$ and $Q_i$ is regular.

\hfill \Box

The proofs of the preceding theorems indicate a method for automatically generating homomorphic classes and recurrence relations to encode the
relevant details about a graph problem expressed in terms of the primitive predicates with logical operators and quantifiers. Although the number of classes generated by this method could conceivably be enormous, in practice a reduction technique can be employed to minimize the number of such classes, using notions of class equivalence. The theorems merely guarantee that the number of classes is constant, with respect to the size of a problem instance.

We illustrate the aforementioned automatic algorithm generator with an example involving vertex covering. The recognition version of this problem can be expressed in the calculus as

\[ (\exists e_1)(\forall v_1 \in V_1)(\text{Inc}(v_1, e_1)). \]

We solve this problem on the following series-parallel graph, by explicitly generating the classes implied by the above proofs.

Each base graph

\[
\begin{array}{c}
\text{x} \\
\text{z} \\
\text{y}
\end{array}
\]

belongs to the homomorphic class

\[
( \{ \{(01, 01, \emptyset, 00)\}, (00, 00, \emptyset, 00)\} ,
\{ \{(11, 01, \emptyset, 00)\}, (10, 00, \emptyset, 00)\} ,
\{ \{(01, 11, \emptyset, 00)\}, (00, 10, \emptyset, 00)\} ,
\{ \{(11, 11, \emptyset, 00)\}, (10, 10, \emptyset, 00)\} )
\]

which can be seen as follows. The outermost 4-tuple contains sets corresponding to the possible values of \( V_1 \cap \{ x, y \} \): \( \emptyset \), \( \{ x \} \), \( \{ y \} \), and \( \{ x, y \} \). Each of these sets has only one element, because there is no interior vertex, and
hence no alternative ways to select non terminals in $V_1$. This single element is a pair, whose first entry is a set corresponding to interior edges (of which there is only one), and whose second entry corresponds to an exterior edge. At the innermost layer are 4-tuples, this time corresponding to possible values of $v_1$: $x$, $y$, interior, and exterior. Each element contains a value of the vector $[v_1 \in V_1, \text{Inc}(v_1, e_1)]$. For instance, the first such value corresponds to $V_1 \cap \{x, y\} = \emptyset$, $e_1$ is interior, and $v_1 = x$; here $v_1 \in V_1$ is false and $\text{Inc}(v_1, e_1)$ is true, so the vector is 01. Next a series operation produces the graph

![Diagram](image)

which belongs to the homomorphic class

( \{(01,01,00,00), (01,01,00,00), (00,01,00,00)\}, (00,00,00,00)\),
(\{(01,01,11,00), (00,01,11,00)\}, (00,00,10,00)\),
(\{(11,00,01,00)}, (10,01,00,00)\), (10,00,00,00)\),
(\{(11,10,01,00)}, (10,11,00,00)\), (10,10,00,00)\),
(\{(11,10,11,00)}, (10,11,11,00)\), (10,10,10,00)\)).

Following, a parallel operation yields

![Diagram](image)

which belongs to the homomorphic class

( \{(01,01,00,00), (01,00,01,00)}, (00,01,01,00)\), (00,00,00,00)\),
(\{(01,01,00,00), (01,00,01,00)}, (00,01,01,00)\), (00,00,00,00)\),
(\{(11,10,01,00)}, (10,11,01,00)\), (10,10,00,00)\),
(\{(11,10,11,00)}, (10,11,11,00)\), (10,10,10,00)\)).
To determine whether this is an accepting class, we first remove the cases involving exterior vertices and edges. This yields

\[
\{(01, 01, \{00\}, 00), (01, 00, \{01\}, 00), (00, 01, \{11\}, 00), (00, 00, \{10\}, 00)\}, \{(11, 01, \{00\}, 00), (11, 00, \{01\}, 00), (10, 01, \{01\}, 00), (10, 00, \{00\}, 00)\}, \{(11, 01, \{10\}, 00), (11, 00, \{11\}, 00), (10, 01, \{11\}, 00), (10, 00, \{10\}, 00)\}, \{(01, 11, \{00\}, 00), (01, 10, \{01\}, 00), (00, 11, \{01\}, 00), (00, 10, \{00\}, 00)\}, \{(11, 11, \{00\}, 00), (11, 10, \{01\}, 00), (10, 11, \{01\}, 00), (10, 10, \{00\}, 00)\}, \{(11, 11, \{10\}, 00), (11, 10, \{11\}, 00), (10, 11, \{11\}, 00), (10, 10, \{10\}, 00)\}.
\]

To determine whether this is an accepting class, we first remove the cases involving exterior vertices and edges. This yields

\[
\{(01, 01, \{00\}, 00), (01, 00, \{01\}, 00), (00, 01, \{01\}, 00), (00, 00, \{10\}, 00)\}, \{(11, 01, \{00\}, 00), (11, 00, \{01\}, 00), (10, 01, \{01\}, 00), (10, 00, \{00\}, 00)\}, \{(11, 01, \{10\}, 00), (11, 00, \{11\}, 00), (10, 01, \{11\}, 00), (10, 00, \{10\}, 00)\}, \{(01, 11, \{00\}, 00), (01, 10, \{01\}, 00), (00, 11, \{01\}, 00), (00, 10, \{00\}, 00)\}, \{(11, 11, \{00\}, 00), (11, 10, \{01\}, 00), (10, 11, \{01\}, 00), (10, 10, \{00\}, 00)\}, \{(11, 11, \{10\}, 00), (11, 10, \{11\}, 00), (10, 11, \{11\}, 00), (10, 10, \{10\}, 00)\}.
\]

Next each accepting class is replaced by true (1) and each nonaccepting class is replaced by false (0); for the given problem this step has no effect. Then the logical operators \(\neg\), \(\land\), and \(\lor\) are applied to obtain the results of the expression \(v_1 \in V_1 \land \text{Inc}(v_1, e_1)\).

\[
\{(0, 0, \{0\})\}, \{(0, 0, \{0\}), (0, 0, \{1\})\}, \{(1, 0, \{0\}), (0, 0, \{0\})\}, \{(1, 0, \{0\}), (1, 0, \{1\}), (0, 0, \{1\})\}, \{(0, 1, \{0\}), (0, 0, \{0\})\}, \{(0, 1, \{0\}), (0, 0, \{1\}), (0, 1, \{1\})\}, \{(1, 1, \{0\}), (1, 0, \{0\}), (0, 1, \{0\})\}, \{(1, 1, \{0\}), (1, 0, \{1\}), (0, 1, \{1\})\}.
\]

The innermost quantifier, \(\exists v_1\), is now applied to obtain
(\{\{0\}, \{0,1\}\}, \\
(\{0,1\}, \{1\}\}, \\
(\{0,1\}, \{1\}\}, \\
(\{1\}\}).

Finally \((\forall e_1)\) is applied to obtain \((\{0\}, \{0,1\}, \{0,1\}, \{1\}\), and \((\exists V_t)\) is applied to obtain \((0,1,1,1)\), which means that a vertex cover \(C\) exists for each \(V_1 \cap C \subseteq T\) other than \(\emptyset\). As explained in the next section, the cardinality of a minimum vertex cover can be computed, or (by maintaining back pointers) an optimum cover can be produced.

The last two theorems of this section are useful in making expressions more concise.

**Theorem 3** If \(P\) and \(Q\) are regular properties, then each of the following predicates is a regular property.

\[
egin{align*}
P &\rightarrow Q &\iff &\neg P \lor Q \\

P &\leftrightarrow Q &\iff & (P \rightarrow Q) \land (Q \rightarrow P) \\

(\exists x \in X)(P(x)) &\iff (\exists x)(x \in X \land P(x)) \\

(\forall x \in X)(P(x)) &\iff (\forall x)(x \in X \rightarrow P(x)) \\

Q(P(x)) &\iff (\exists y)((y = P(x)) \land Q(y)) \\

P(x_i)(1 \leq i \leq m) &\iff P(x_1) \land \cdots \land P(x_m)
\end{align*}
\]

\(\square\)

**Theorem 4** Each of the following predicates is a regular property.

\[
\begin{align*}
e_1 &= e_2 &\iff & (\forall v_1)(\text{Inc}(v_1, e_1) \leftrightarrow \text{Inc}(v_1, e_2)) \\

\text{Adj}(v_1, v_2, E_1) &\iff (\exists e_1 \in E_1)(\text{Inc}(v_1, e_1) \lor \text{Inc}(v_2, e_1)) \land \neg v_1 = v_2 \\

V_1 \cup V_2 &= V_3 &\iff & (\forall v_1)((v_1 \in V_1 \lor v_1 \in V_2) \leftrightarrow v_1 \in V_3) \\

V_1 \cap V_2 &= V_3 &\iff & (\forall v_1)((v_1 \in V_1 \land v_1 \in V_2) \leftrightarrow v_1 \in V_3) \\

V_1 \setminus V_2 &= V_3 &\iff & (\forall v_1)((v_1 \in V_1 \land \neg v_1 \in V_2) \leftrightarrow v_1 \in V_3)
\end{align*}
\]

\[
\begin{align*}
\text{Part}(V_0, V_1, \ldots, V_m) &\iff (V_1 \cup \ldots \cup V_m = V_0) \land (V_i \cap V_j = \emptyset)(1 \leq i < j \leq m) \\

\text{Part}(E_0, E_1, \ldots, E_m)
\end{align*}
\]
\[ V_1 \subseteq V_2 \quad \iff \quad (\forall v_1)(v_1 \in V_1 \rightarrow v_1 \in V_2) \]
\[ E_1 \subseteq E_2 \quad \iff \quad V_1 \subseteq V_2 \land V_2 \subseteq V_1 \]
\[ V_1 = V_2 \quad \iff \quad V_1 \subseteq V_2 \land -V_2 \subseteq V_1 \]
\[ E_1 = E_2 \quad \iff \quad V_1 \subseteq V_2 \land \neg V_2 \subseteq V_1 \]
\[ |V_1| \geq m \quad \iff \quad (\exists v_1 \in V_1, \ldots, v_m \in V_1)(\neg v_i = v_j(1 \leq i < j \leq m)) \]
\[ |E_1| \geq m \quad \iff \quad \neg(|V_1| \geq m + 1) \]
\[ |V_1| \leq m \quad \iff \quad |V_1| \leq m \land |V_1| \geq m \]
\[ |E_1| = m \quad \iff \quad |E_1| = m \]
\[ \max P(V_1) \quad \iff \quad P(V_1) \land (\forall V_2)(V_1 \subset V_2 \rightarrow \neg P(V_2)) \]
\[ \max P(E_1) \quad \iff \quad \neg P(V_2) \]
\[ \min P(V_1) \quad \iff \quad P(V_1) \land (\forall V_2)(V_2 \subset V_1 \rightarrow \neg P(V_2)) \]
\[ \min P(E_1) \quad \iff \quad \neg P(V_2) \]
\[ E_1 = \text{Ind}(V_1) \quad \iff \quad (\forall e_1)(e_1 \in E_1 \iff (\forall v_1)(\text{Inc}(v_1, e_1) \rightarrow v_1 \in V_1)) \]
\[ E_1 = \text{IncE}(V_1) \quad \iff \quad (\forall e_1)(e_1 \in E_1 \iff (\exists v_1 \in V_1)(\text{Inc}(v_1, e_1))) \]
\[ V_1 = \text{IncV}(E_1) \quad \iff \quad (\forall v_1)(v_1 \in V_1 \iff (\exists e_1 \in E_1)(\text{Inc}(v_1, e_1))) \]
\[ \deg(v_1, E_1) = m \quad \iff \quad |E_1 \cap \text{IncE}(|v_1|)| = m \]
\[ \text{Reg}_m(V_1, E_1) \quad \iff \quad (\forall v_1 \in V_1)(\deg(v_1, E_1) = m) \]
\[ \text{Isom}_H(V_1, E_1) \quad \iff \quad (\exists v_1 \in V_1, \ldots, v_{|H|} \in V_1)(\neg v_i = v_j \land \neg (\text{Adj}(v_i, v_j, E_1) \iff \text{Adj}(i, j, H))(1 \leq i < j \leq |H|)) \]
\[ \text{Conn}(V_1, E_1) \quad \iff \quad (\forall V_2, V_3)(\text{Part}(V_1, V_2, V_3) \rightarrow (V_2 = \emptyset \lor V_3 = \emptyset \lor (\exists v_2 \in V_2, v_3 \in V_3)(\text{Adj}(v_2, v_3, E_1)))) \]
\[ \text{Conn}_m(V_1, E_1) \quad \iff \quad \text{Isom}_K(V_1, E_1) \lor (\forall v_1 \geq m + 1 \land (\forall V_2)(|V_2| \leq m \rightarrow \text{Conn}(V_1 \setminus V_2, E_1 \setminus \text{IncE}(V_2)))) \]
\[ \text{Forest}(V_1, E_1) \quad \iff \quad (\forall V_2)(\text{Conn}_2(V_2, \text{Ind}(V_2) \setminus E_1) \rightarrow \neg (V_2 \subseteq V_1)) \]
Tree($V_1, E_1$) $\iff$ Forest($V_1, E_1$) $\land$ Conn($V_1, E_1$)
Path($V_1, E_1$) $\iff$ Tree($V_1, E_1$) $\land$ ($\forall v_1 \in V_1$)(deg($v_1, E_1$) $\leq$ 2)
Cycle($V_1, E_1$) $\iff$ Conn($V_1, E_1$) $\land$ Reg$_2$($V_1, E_1$)
Match($E_1$) $\iff$ Reg$_1$(IncV($E_1$), $E_1$)

Minor$_H$($V_0, E_0$) $\iff$ ($\exists V_1, \ldots, V_{|H|})$ (Part($V_0, V_1, \ldots, V_{|H|}$)$\land$
Conn($V_i$, Ind($V_i$))(1 $\leq$ i $\leq$ |$H$|)$\land$
(Adj($i, j, H$) $\rightarrow$ ($\exists v_1 \in V_i, v_2 \in V_j$)(Adj($v_1, v_2, E_0$)))
(1 $\leq$ i $<$ j $\leq$ |$H$|))
Applications

We now demonstrate how the predicate calculus developed in the previous section provides the basis for linear time solvability on recursively constructed graph families.

Theorem 5 If $P(x_1, \ldots, x_t)$ is regular (for fixed $t$), then each of the following problems can be solved in linear time on recursively constructed graph families.

1. Recognition.
   \[(\exists x_1) \ldots (\exists x_t) P(x_1, \ldots, x_t).\]

2. Optimization.
   If $t = 1$ and $x_1$ denotes a set, then
   \[\min |x_1| : P(x_1) \text{ or } \max |x_1| : P(x_1).\]

3. Enumeration.
   Compute \[\{ (x_1, \ldots, x_t) : P(x_1, \ldots, x_t) \} |.\]

Proof: First observe that a decomposition tree $D$ for a graph $G$ in a recursive family $F$ has leaves which are mutually edge-disjoint base graphs, so the size of $D$ is linear in the size of $G$. Let $r$ denote the root node of $D$.

1. Recognition.
   By Theorem 2, if $P(x_1, \ldots, x_t)$ is regular, then $Q = (\exists x_1) \ldots (\exists x_t) P(x_1, \ldots, x_t)$ is also regular. Thus there are a finite set $C$ and homomorphism $h : F \rightarrow C$ corresponding to $Q$. The set of base graphs is finite, so $h$ can be evaluated at each leaf of $D$ in constant time. The set of recursive operations is also finite, and for each one there is a corresponding finite multiplication table, so $h$ can be evaluated at each interior node of $D$ in constant time. Therefore $h$ is found for $r$ by a linear time dynamic programming algorithm, and $Q$ is true for $G$ iff $h(r)$ is an accepting class of $C$.

2. Optimization.
   Since $P(x_1)$ is regular, it has a corresponding finite set $C$ and homomorphism $h : F_{x_1} \rightarrow C$. For each class $c \in C$ and node $v$ of $D$, let $z_c(v)$ denote the optimum of $|x_1|$ over all values of $x_1$ for which
\( h(g(v), x_1) = c \). Computation of each \( z_c \) is trivial at leaves of \( D \); at interior nodes, it involves summation — modified so that duplicate elements are counted only once — and either minimization or maximization over the possible sums. The solution to the optimization problem is found by taking the minimum or maximum over those \( z_c(r) \) such that \( c \) is an accepting class.

3. Enumeration.

\( P(x_1, \ldots, x_t) \) is regular, and the corresponding finite set and homomorphism are \( C \) and \( h : F_{x_1, \ldots, x_t} \rightarrow C \). For each class \( c \in C \) and node \( v \) of \( D \), let \( z_c(v) \) denote \( \# \{ (x_1, \ldots, x_t) : P(x_1, \ldots, x_t) \land h(g(v), x_1, \ldots, x_t) = c \} \). Computation of each \( z_c \) is trivial at leaves of \( D \); at interior nodes, it involves computing sums of products. The solution to the optimization problem is found by taking the sum of all \( z_c(r) \) such that \( c \) is an accepting class.

To illustrate, recall the vertex cover problem described earlier:

\[
VC(V_1) = (\forall e_1)(\exists v_1 \in V_1)(Inc(v_1, e_1)).
\]

Theorem 5 shows that linear algorithms can be found for each of

1. \( (\exists V_1)(VC(V_1)), \)
2. \( \min | V_1 | : VC(V_1) \), and
3. \( \text{Compute } | \{ V_1 : VC(V_1) \} |. \)

In the optimization case, a minimum cardinality solution can be reconstructed by maintaining back pointers, in the standard manner.

The next theorem summarizes the status of numerous popular problems.

**Theorem 6** An optimal solution to each of the following problems is solvable in linear time on recursively constructed graph families.

Vertex cover
\[
VC(V_1) \iff E = IncE(V_1)
\]
Independent set
\[ IS(V_1) \iff VC(V \setminus V_1) \]

Dominating set
\[ DS(V_1) \iff V = IncV(IncE(V_1)) \]

Clique
\[ Clique(V_1) \iff (\forall v_1, v_2 \in V_1)(Adj(v_1, v_2, E)) \]

Maximal matching
\[ MaxMatch(E_1) \iff \text{max Match}(E_1) \]

Hamiltonian cycle
\[ HC(E_1) \iff \text{Cycle}(V, E_1) \]

t-Partite subgraph
\[ Partite_t(E_1) \iff (\exists V_1, \ldots, V_t)(Part(V, V_1, \ldots, V_t) \wedge \ldots) \]

Cubic subgraph
\[ Cubic(E_1) \iff \text{Reg}_3(V, E_1) \]

Planar subgraph
\[ Planar(E_1) \iff \neg \text{Minor}_{K_5}(V, E_1) \wedge \neg \text{Minor}_{K_{3,3}}(V, E_1) \]

Eulerian subgraph
\[ Euler(E_1) \iff \text{Conn}(IncV(E_1), E_1) \wedge (\forall v_1 \in IncV(E_1))(\text{deg}(v_1, E_1) \equiv 0 \pmod{2}) \]

\( d \)-Degree-bounded spanning tree
\[ DST(E_1) \iff \text{Tree}(V, E_1) \wedge (\forall v_1)(\text{deg}(v_1, E_1) \leq d) \]

Leafset of spanning tree
\[ LST(V_1) \iff (\exists E_1)(\text{Tree}(V, E_1) \wedge (\forall v_1 \in V_1)(\text{deg}(v_1, E_1) = 1)) \]
Steiner tree \((V')\)
\[\text{Steiner}(E_1) \iff V' \subseteq \text{IncV}(E_1) \land \text{Tree}(\text{IncV}(E_1), E_1)\]

Rural postman \((E')\)
\[\text{RP}(E_1) \iff E' \subseteq E_1 \land \text{Cycle}(	ext{IncV}(E_1), E_1)\]

t-Colorability
\[\text{Color}(V_1, \ldots, V_t) \iff \text{Part}(V, V_1, \ldots, V_t) \land \text{IS}(V_i)(1 \leq i \leq t)\]

t-Edge colorability
\[\text{EColor}(E_1, \ldots, E_t) \iff \text{Part}(E, E_1, \ldots, E_t) \land \text{Match}(E_i)(1 \leq i \leq t)\]

Partition into \(t\) forests
\[\text{PartF}(V_1, \ldots, V_t) \iff \text{Part}(V, V_1, \ldots, V_t) \land \text{Forest}(V_i, \text{Ind}(V_i))(1 \leq i \leq t)\]

Partition into \(t\) perfect matchings
\[\text{PartPM}(V_1, \ldots, V_t) \iff \text{Part}(V, V_1, \ldots, V_t) \land \text{Reg}_1(V_i, \text{Ind}(V_i))(1 \leq i \leq t)\]

Partition into \(t\) cliques
\[\text{PartC}(V_1, \ldots, V_t) \iff \text{Part}(V, V_1, \ldots, V_t) \land \text{Clique}(V_i)(1 \leq i \leq t)\]

Covering by \(t\) cliques
\[\text{CC}(V_1, \ldots, V_t) \iff (E = \text{Ind}(V_1) \cup \ldots \cup \text{Ind}(V_i)) \land \text{Clique}(V_i)(1 \leq i \leq t)\]

\[
\]

Of course, many well known \(NP\)-hard problems are described on graphs with weighted vertices and/or edges. For example, vertex cover, independent set, and dominating set are no less difficult in the general case when vertices are weighted. Steiner tree and rural postman normally have weighted edges, while graph \(b\)-partitioning often has both weighted vertices and edges. Our next result demonstrates that cardinality cases can be replaced by their weighted analogues.

**Theorem 7**  The cardinality of a vertex or edge set, \(|x_i|\), can be replaced by the sum of the weights of its elements, \(\sum_{y \in x_i} w(y)\), in either a regular predicate or an optimization function, while maintaining a linear time algorithm.
Proof: It is obvious that the proof of the optimization portion of Theorem 5 can be modified to handle sums of vertex or edge weights, by altering the values of \( z_c(v) \) appropriately for each leaf \( v \).

The case in which a subexpression \( |x_i| \geq m \) is replaced by \( \sum_{y \in x_i} w(y) \geq m \) is more difficult. Let \( x'_j = \{ z : w(z) = j \} \) (for \( j < m \)) and \( x'_m = \{ z : w(z) \geq m \} \) be specified with the instance. The subexpression can be replaced by

\[
\bigvee_{S} \bigwedge_{1 \leq j \leq m} (\exists y_1 \in x_i \cap x'_j, \ldots, y_{a_j} \in x_i \cap x'_j)(\neg y_a = y_b)(1 \leq a < b \leq a_j),
\]

where \( S = \{(a_1, \ldots, a_m) : m \leq \sum_{1 \leq j \leq m}(ja_j) < m + \max_{v \in V} w(v)\} \) is a finite set corresponding to possible partitions of the weight \( w(x_i) \) among members of the various \( x_i \cap x'_j \).

We conclude this section with an explicit demonstration of the strength of our results. Specifically, we shall formalize the status of a host of problems which were listed as open or unverified by Johnson in [11]. Note that we are not suggesting that none of these problems have since been resolved by other means. Rather, we simply show that their resolution follows from the results established herein.

**Corollary 8** The following problems are linear time solvable on the recursive graph classes shown, once a decomposition tree is found in polynomial (but not necessarily linear) time.

1. Partition into \( t \) cliques on series-parallel graphs, partial \( k \)-trees, and bandwidth-\( k \) graphs.
2. \( t \)-Edge colorability (\( t \)-chromatic index) on partial \( k \)-trees and bandwidth-\( k \) graphs.
3. Hamiltonian cycle on bandwidth-\( k \) graphs.
4. Bipartite subgraph (max cut) on partial \( k \)-trees.
5. Steiner tree on Halin graphs, partial \( k \)-trees, and bandwidth-\( k \) graphs.
Extensions

The calculus can be extended with additional variable domains and primitive predicates, while still maintaining regularity. One useful way in which this can be done is by defining a new domain consisting of certain restricted sets of vertex sets. Let the domain PART consist of partitions of the vertices such that either

1. each piece of the partition induces a connected subgraph (as when partitioning into cliques), or

2. each piece of the partition is hereditary in the following sense: each of its connected components meets the same criteria that the entire piece must meet (as when partitioning into forests).

Theorem 9 If $A_1$ has domain PART, then the predicate $V_1 \in A_1$ is regular.

Proof: Consider $C = \{0, 1\}$, with 1 the accepting class. If $G$ is a base graph with vertices $V$, let $h(G, V_1, A_1)$ be 1 if $(V_1 \cap V) \in \{V_2 \cap V : V_2 \in A_1\}$, and 0 otherwise. If $G = \circ(G_1, \ldots, G_m)$, let $\circ$ be $\land_m$ ($m$-ary logical conjunction).

The proof of Theorem 9 does not generalize beyond the domain PART, even to arbitrary partitions of the vertices. The reason is that, when $\circ$ is applied, $\circ$ must receive enough information from the different $h(G_j)$ to determine which pieces of the partition from the different $G_j$ are combined into the same piece for $G$. While this is obvious whenever a piece has nonempty intersection with the terminals of $G_j$ (because each terminal is in exactly one piece), it is more difficult when this intersection is empty. But an empty intersection with the terminals of $G_j$, together with the conditions on the domain PART, can be handled: If the pieces must induce connected subgraphs, then no piece which is entirely interior to $G_j$ can be combined with any piece which is entirely exterior to $G_j$. And if instead the heredity condition holds, then each piece which is entirely interior to $G_j$ can be combined arbitrarily with any piece which is entirely exterior.

Theorem 10 The set of regular predicates is closed under $\neg$, $\land$, $\lor$, $\forall$, and $\exists$, where quantification is over variables which range over vertices, edges, vertex sets, edge sets, and PART.
Proof: Theorem 2 is extended to handle quantified variables whose domain is \( \text{PART} \), by adding the case where \( y_i \) is in \( \text{PART} \): Let \( C' = \prod_{p \in \pi(T)} C_p \), where \( \pi(T) \) denotes the set of partitions of the terminals (note \( | \pi(T) | \leq k! \)), and \( C_p \) denotes the set of homomorphic classes of \( Q_{i+1} \) such that \( p = \{ V_i \cap T : V_i \in y_i \} \). Thus \( | C' | \leq (2^{|C|})^k \), and \( Q_i \) is regular. 

**Theorem 11** An optimal solution to each of the following problems is solvable in linear time on recursively constructed graph families, where \( A_i \) is in \( \text{PART} \).

Chromatic number

\[
\text{Color}(A_1) \iff (\forall V_1 \in A_1)(\text{IS}(V_1))
\]

Partition into forests

\[
\text{PartF}(A_1) \iff (\forall V_1 \in A_1)(\text{Forest}(V_1, \text{Ind}(V_1)))
\]

Partition into perfect matchings

\[
\text{PartPM}(A_1) \iff (\forall V_1 \in A_1)(\text{Reg}(V_1, \text{Ind}(V_1)))
\]

Partition into cliques

\[
\text{PartC}(A_1) \iff (\forall V_1 \in A_1)(\text{Clique}(V_1))
\]

Generalized (connected) \( H \)-matching

\[
\text{GMatch}_H(A_1) \iff (\forall V_1 \in A_1)(\text{Isom}_H(V_1, \text{Ind}(V_1)))
\]

Graph \( G \)-partitioning

\[
\text{GPart}_G(E_1) \iff (\exists A_1)((\forall V_1 \in A_1)(| V_1 | \leq k \land \text{Conn}(V_1, \text{Ind}(V_1))) \land (\forall e_1 \in E_1)(e_1 \in \text{Ind}(V_1)))
\]

\( \square \)

This theorem shows that the optimization version of partition into cliques is solvable in linear time on each of series-parallel graphs, partial \( k \)-trees, and bandwidth-\( k \) graphs. We show in the next section that the optimization version of edge chromatic number (chromatic index) is solvable in polynomial time on partial \( k \)-trees and bandwidth-\( k \) graphs. This completes the demonstration of polynomiality for nine out of the ten graph problems listed by
Johnson in [11] for recursively constructed families such as series-parallel graphs, Halin graphs, partial $k$-trees, and bandwidth-$k$ trees. (The remaining problem, graph isomorphism, takes an instance containing two graphs, and hence does not fit our model.)
Limits

Even though the calculus is extensible, it is not simple to discover new primitives that increase the number of significant expressible problems, so that every such problem remains regular. There exist a few problems solvable in linear time on recursively constructed families that are not thought to be expressible in the extended calculus of the previous section. There also exist some closely related problems that are not thought to be solvable in linear time on these families, some of which are known to be solvable in polynomial time by dynamic programming and some of which remain \( NP \)-hard.

One additional problem that could be included by a trivial modification of the extension in the previous section is \( m \)-Multichromatic number, coloring each vertex with \( m \) colors so that no adjacent vertices have any color in common. Merely define a domain \( P_m \) of sets of vertex sets such that no vertex is in more than \( m \) sets, and that maintains the connectedness and heredity conditions as defined on \( \text{PART} \), which is now simply \( P_1 \).

Covering by cliques is another problem which is solvable in linear time on recursive families, and for which the connectedness condition holds. A dynamic programming algorithm merely needs a homomorphic class for each possible combination of

1. which subsets of the \( k \) terminals are contained in any single clique,

2. which of these cliques also contains an interior vertex (and hence cannot contain any exterior vertices), and

3. which edges between terminals already exist.

There are less than \( 2^2 \times 2 \times 2^2 \) combinations. Observe that there is no bound on the number of cliques in which any particular vertex can appear, so this problem is not included by the extension to the \( P_m \) domains. Rather, it is the fact that no interior vertex can be in the same clique as any exterior vertex which allows a bounded number of classes; hence the only graph covering problems for which regularity is maintained in this way are covering by cliques and special cases, such as covering by triangles.

Many similar covering problems such as covering by matchings (edge chromatic number), by perfect matchings, by a fixed connected \( H \), by cycles, by Hamiltonian cycles, by \( f \)-factors, by \( f \)-regular subgraphs, and even by
hypocliques (cliques with one missing edge) can be solved in polynomial time, but do not appear to be solvable in linear time. As an example, consider edge chromatic number. A homomorphic class must maintain information about which edge colors are already incident to each terminal, and with \( t \) colors this implies \( 2^{kt} \) classes. By realizing that the minimum information a class needs to contain is not which colors, but rather how many colors are incident to exactly each subset of the terminals, this number can be reduced only to \( \Theta(t^{2k}) \). But Vizing’s theorem states that the minimum cardinality edge coloring is equal either to the maximum vertex degree \( \Delta \) or to \( \Delta + 1 \), and there is no degree bound even for simple recursive families like trees. Hence the best dynamic programming algorithm for edge chromatic number appears to take \( \Omega(n^{2k}) \) time.

Covering by forests and by trees (which are actually two equivalent problems) are solvable in linear time, however, and are included in the extension to the \( P_m \) domains through the following observation: each recursively constructed graph family uses a finite cardinality rule set and therefore has some associated bound \( r \) on its connectivity, so at most \( r \) forests or trees are sufficient.

Another related problem that is solvable in linear time is \( f \)-factorization (note this is distinct from covering by \( f \)-factors, in which a given edge could occur in more than one \( f \)-factor). Any \( f \)-factorable graph must be \( cf \)-regular, where \( c \) is the number of factors. But each graph in a \( k \)-terminal recursive family has a vertex with degree not more than \( k \) (consider a vertex that is a nonterminal at any node which has only leaves as children in the decomposition tree). Hence the number of factors \( c \leq k + f \).

A similar argument to those above can be used to show that chromatic number can be solved in linear time. By an easy induction on the size of a decomposition tree, for each recursive family there exists a constant \( r \) such that any \( k \)-terminal graph in the family can be \( r \)-colored with each terminal assigned a distinct color. Thus at most \( r \) vertex sets suffice. But chromatic number fits into the calculus extended with the domain PART, so this argument is unnecessary (but still interesting).

One more problem which appears to be polynomial but not linear is the optimization version of degree-bounded spanning tree. For any fixed \( d \), it suffices to have one homomorphic class for each possible combination of

1. a partition of the \( k \) terminals into already-connected components and
2. assignments of degrees $0 \leq d_i \leq d$ for $1 \leq i \leq k$.

This is $\pi(T) \times (d + 1)^k$ classes. But in the optimization version it might be that $d = \Theta(n)$, so the best dynamic programming algorithm appears to take $\Omega(n^k)$ time.

Some of the problems which do have linear algorithms are quite similar to problems which remain $NP$-hard on recursive graph families. Generalized $H$-matching appears to be nonlinear if $H$ is not connected, and it appears to remain $NP$-hard if $H$ is connected and given as part of the instance. To see why no dynamic programming algorithm seems to be applicable, even on trees, consider a graph $H$ with a central vertex $v$ and paths of length $1, 2, \ldots, m$ emanating from it. The number of homomorphic classes needed just for the cases when vertex $v$ coincides with a terminal vertex during application of an algorithm is at least $\Theta(2^m) = \Theta(2^{n^2})$, where each class corresponds to a distinct subset of the $m$ paths and indicates which paths begin with an interior edge.

A closely related problem, subgraph isomorphism, is solvable in linear time on recursive families and in $O(n^{[H]})$ time on all graphs for a fixed subgraph $H$. But this problem is $NP$-hard if $H$ is given as part of the instance, even for the special case of subtree isomorphism on series-parallel graphs.

The graph partitioning problem with $b$ given as part of the instance cannot be handled efficiently by dynamic programming, because the subexpression $|V_1| \leq b$ translates into $\neg(\exists v_1, \ldots, v_{b+1})$, which is not a fixed size expression. Notice that the expression given for graph $b$-partitioning uses the unboundedness of $|A_1|$ to ensure the permissibility of the assumption that $V_1$ induces a connected component. However, if $|A_1|$ and $b$ are both fixed, the problem becomes trivial on all graphs, because $|V| \leq b |A_1|$. Furthermore if edges are ignored and the problem is to minimize $|A_1|$, then the problem becomes equivalent to bin packing, with a piece of the partition corresponding to a bin. And if edges are ignored, $|A_1|$ is fixed, and $b$ is either part of the instance or to be minimized, then the problem becomes equivalent to processor scheduling, with each piece of the partition corresponding to the jobs executed by a certain processor.

Certain problems are known to be difficult on recursively constructed graphs. For example, bandwidth is shown to be $NP$-hard on trees in [9], while optimal linear arrangement and minimum cut linear arrangement are polynomial on trees but still open on series-parallel graphs. Interestingly,
bandwidth can be expressed in a calculus with just a slight bit more freedom than our extended calculus, merely by generalizing to allow a domain of arbitrary sets of vertex sets:

\[ \text{t-Bandwidth} \]
\[ BW_t(A_1) \iff V \in A_1 \land (\forall V_1 \in A_1)((V_1 = \emptyset \lor (\exists v_1 \in V_1)(V_1 \setminus \{v_1\} \in A_1)) \land \\
(\forall v_1 \in V_1, v_2 \in V_1)((V_1 \setminus \{v_1\} \in A_1 \land V_1 \setminus \{v_2\} \in A_1) \rightarrow v_1 = v_2)) \land \\
(\forall v_1, v_2)(\text{Adj}(v_1, v_2, E) \rightarrow \neg(\exists V_1 \in A_1, \ldots, V_{t+1} \in A_1) \\
(v_1 \in V_1 \land \neg v_2 \in V_{t+1} \land V_i \subset V_{i+1}(1 \leq i \leq t))) \]

(Informally, this expression maintains a set \( S_v = \{u : l(u) \leq l(v)\} \) in place of the label \( l(v) \) of each vertex \( v \), and \( A_1 = \{S_v : v \in V\} \cup \emptyset \).)

Finally, problems which are PSPACE-hard are not expressible in the calculus because expressions are restricted to a finite number of quantifiers. As an example consider Generalized Kayles, a game in which two players alternate moves, with a move being the removal of any vertex and all its neighbors from a graph, until the graph becomes empty. Given a recursive graph, does the first player have a winning strategy to defeat his adversary, i.e., one which guarantees he can remove the final vertex? Observe that the first player's moves are existential, while his opponent's moves are universal. It appears that for a dynamic programming strategy to prove useful, there would have to be a homomorphic class for each possible sequence of the form \( (q_1v_1) \cdots (q_mv_m) \), where each \( q_i \) is either \( \forall \) or \( \exists \), and \( m = O(n) \) is the maximum size of an independent set of vertices. This is because there is no locality to the game — players can remove any vertex remaining on each move — so the sequence of \( \forall s \) and \( \exists s \) which occur within a particular subgraph will not necessarily alternate. That the calculus cannot come near to expressing PSPACE-hard problems is surely an indication that the calculus captures much of the essence of the dynamic programming paradigm on recursive graph families.
Conclusions

In this paper, we have developed a calculus for expressing graph problems which are regular, and which can thus be solved in linear time using dynamic programming when instances are confined to recursive graphs families. Indeed, we have provided an automatic algorithm generation mechanism. Our demonstration of the strength of these results has included the resolution of various problems of apparently unknown or unverified complexity status. We have also shown how to extend the calculus, including examples which illustrate the difficulties inherent with doing so. Unfortunately, our attempts to discover a precise characterization of the set of all regular problems have been unsuccessful. We conclude by conjecturing that there are theoretical reasons for this outcome.

Conjecture 12. For any problem which can be solved in linear time when the instance is restricted to be a recursively constructed graph, there is a set of primitives such that this problem is expressible in terms of these primitives and the connectives ∧, ∨, ¬, and ∃.

Conjecture 13. There is no finite set of primitives such that every problem solvable in linear time on recursively constructed graphs can be expressed in terms of those primitives and the logical connectives.
References


Simulated Simulated Annealing

Craig A. Tovey *
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

July 15, 1988

Abstract

The principal shortcoming of simulated annealing (SA) is that it takes too long. We present a few swindling ideas for speeding up SA by simulating its action on a problem. The increase in speed is attained at the cost of decreasing generality — the methods all require the use of problem-specific information. We illustrate the ideas with examples, and discuss our computational experience. We conclude with some remarks on the usefulness of simulated annealing.

1 Introduction

Simulated annealing (SA), introduced by Kirkpatrick et al. [5] and Cerny [2] is a powerful general purpose algorithm, based on a simulation method of Metropolis et al. [7], for solving optimization problems. It is useful in finding globally good solutions for a large variety of problems, and has several appealing features: it is easy to code, adaptable, and finds very high quality solutions if given enough time. Its principal drawback is its gargantuan appetite for cpu time. For instance, David Johnson et al. [4] report that for a partitioning problem, SA outperformed all other heuristics, but only when the cpu time allowance was raised to 24 hours on a Cray.

*Supported by a Presidential Young Investigator Award from the National Science Foundation (ECS-8451032) and a matching grant from Digital Equipment Corporation
In this paper we present a few ideas which we think can be widely used to speed up SA. The ideas are easy to understand, and should be accessible to most practitioners. The common theme is to approximate or simulate the actions of the classic simulated annealing algorithm, rather than following it precisely. The simulated SA algorithm should run faster than the original while retaining the ability to find good quality solutions.

There is a trade-off between the general and the specific here. One expects that the more generally applicable an algorithm, the less efficient it will be. SA is very general, and very slow. To gain the increase in efficiency, we specialize the algorithm using information about the particular problem we are trying to solve. The methods we present here, however, are “generic” ideas which in our experience can be applied usefully to many different problems.

The outline of the paper is as follows: In section 2 we give a precise statement of the SA algorithm, and establish our notation. In sections 3, 4, and 5 we develop the poor man’s, neighborhood prejudice, and target prejudice swindling ideas, respectively. In each section we develop the ideas by example, and discuss our computational experience with them. In section 6 we discuss how to put these ideas together, and make some observations about the usefulness of SA and simulated SA in general.

2 Background

Because of the other articles in this issue, a complete statement of the simulated annealing algorithm should not be necessary. However, it will be convenient to state precisely our terminology and the steps in a single iteration of the SA procedure. We adopt the following conventions:
• The objective is to find a global minimum of the function $f$.

• The temperature parameter is denoted $T$.

• The neighborhood set of a point $x$ is denoted $N(x)$.

The steps in a single iteration, with current solution $x$, are:

1. Choose $y$, a random member of $N(x)$.

2. Calculate $\Delta f \equiv f(y) - f(x)$.

3. If $\Delta f \leq 0$ Then set $x \leftarrow y$ (Accept)
   Else
   with probability $e^{-\Delta f / T}$ set $x \leftarrow y$ (Accept anyway); otherwise (with complementary probability) leave $x$ unchanged (Reject.)

4. Return

We will refer to the above procedure as a "pure" SA iteration.

For some very simple problems, the largest portion of cpu time goes into computing the exponential in step 3. If this is the case, SA can be sped up by building a table of $e^{-z}$ for a range of positive values of $z$.

Ordinarily, the bulk of the computer time is spent evaluating $\Delta f \equiv f(y) - f(x)$ in Step 2 of the algorithm. It is well known from experience with local optimization algorithms that it may be cheaper to compute $\Delta f$ directly, instead of computing $f(y)$. For example, a 2-swap neighborhood for the travelling salesman problem on $n$ locations requires $O(1)$ time to compute $\Delta f$ (four edge costs are involved), while $f(y)$ for arbitrary $y$ takes $O(n)$ time.

Before trying any more sophisticated speedup techniques, the SA user should first check for these possibilities (as well as data access problems such as too many page
faults). Hereafter we assume that the crucial use of cpu time is in the calculation of $\Delta f$.

## 3 Poor Man's Swindle

Since this evaluation is expensive, how can we avoid it? Our first swindle is based on the use of some other function, $P$, which has some kind of approximate information about $\Delta f$. Our $P$ is cheaper to evaluate than $\Delta f$, but of course not as accurate: it is a "poor man's" $\Delta f$. The problem is, how can inexact information about $f$ be useful? If we just implemented SA using $P$ instead of $f$, we would have a faster program which optimized the wrong function. The idea is, with some probability $p$, calculate $P$ and use it. With the complementary probability, calculate both $P$ and $f$, and use Bayesian reasoning to balance out the inaccuracy due to $P$.

As an example, suppose at some iteration of SA the temperature $T = 10$, the true function change $\Delta f = 5$, and the poor man's estimate of this value is $P = 8$. Suppose further that $p$, the probability of using just the poor man's estimate, has been set to $2/3$. What procedure do we follow? Since $e^{-\Delta f/T} = e^{-1/2} \approx 0.61$, we'd like the procedure to accept 61% of the time. Two thirds of the time, the only information we have to go on is $P$: in this case we accept with probability $e^{-P/T} = e^{-8/10} \approx .45$, and reject with the complementary probability. The other third of the time, therefore, we should accept with probability 0.93, (otherwise reject with probability 0.07). Thus the overall probability of accepting is $(2/3)(.45) + (1/3)(.93) = .61$ as desired. Moreover, if $P$ is much cheaper to calculate than $\Delta f$, we have achieved a computational savings: $\Delta f$ only gets calculated one third of the time.

The general form of the poor man's swindle is:
**Poor Man's Swindle**

1. Calculate the poor man's acceptance rate, \( q = e^{-P/T} \) (if \( P > 0 \)), \( q = 1 \) (if \( P \leq 0 \)).

2. With probability \( p \) go to 3, with probability \( 1 - p \) go to 4.

3. Accept with probability \( q \); otherwise reject. Return.

4. Calculate \( \Delta f \) and the true acceptance probability, \( r = e^{-\Delta f / T} \) (if \( \Delta f > 0 \)), \( r = 1 \) (if \( \Delta f \leq 0 \)).

5. Calculate \( s = (r - pq)/(1 - p) \).

6. If \( s > 1 \) set \( s = 1 \); if \( s < 0 \) set \( s = 0 \).

7. Accept with probability \( s \); otherwise reject. Return.

We say that this procedure simulates SA because the calculation of \( s \) in step 5 tries to yield the same overall acceptance probability as the "pure" SA procedure. That is, \( pq + (1 - p)s = r \), the true acceptance probability. Clearly, if in Step 5, we get \( 0 \leq s \leq 1 \), the simulation is completely successful. If \( s \) has to be adjusted in Step 6, the simulation has some inaccuracy.

If \( \Delta f \) is \( \alpha \) times as expensive to calculate as \( P \), then the expected computational cost per iteration of the poor man's swindle is \((1 - p) + 1/\alpha\) of the cost of a pure iteration.

**Computational Factors**

Using the poor man's swindle is a tradeoff between the computational savings \((1 - p) + 1/\alpha\), and the errors that force \( s \) to be adjusted in step 6. Naturally, we would like \( p \) to be as large as possible, but not incur too many errors.

One of the nice things about this swindle is that it is self-checking: it is easy to collect data in step 6 on what kinds of errors occur. If \( P \) seems consistently too high
or too low, adjust its definition to get a better estimate of $\Delta f$.

Call an error “optimistic” if $s < 0$ in step 5 (because the poor man’s estimate gave too high an acceptance probability.) We could also call this a Type 2 error, since we accepted something we shouldn’t have. If $s > 1$ the error is pessimistic (i.e. Type 1). If there are many errors of each type, $P$ is just too poor an estimate, and $p$ will have to be reduced.

In many applications, one of these types will dominate. For example, in a electronics design problem we have worked with, the objective is to minimize materials cost, plus a number of penalty functions to avoid undesirable design characteristics such as excessive density and awkward placements. The penalty functions are all nonnegative, and comprise the major part of the computational effort in calculating $\Delta f$. A natural poor man’s function would be just the materials cost. The only possible errors in this case are optimistic: a design could be cheap but undesirable for other reasons. We can thus modify the poor man’s procedure:

**Optimistic Poor Man’s Swindle**

1. Calculate the poor man’s acceptance rate, $q = e^{-P/T}$ (if $P > 0$), $q = 1$ (if $P \leq 0$).
2. Reject with probability $1 - q$; continue to 3 otherwise.
3. Calculate $r = e^{-\Delta f/T}$ (if $\Delta f > 0$), $r = 1$ (if $\Delta f \leq 0$), the true acceptance probability.
4. Accept with probability $s = r/q$; otherwise reject. Return.

The chance of a move being accepted then equals $qs = r$ as desired. If $q \geq r$ then there is no error.

There is an obvious corresponding Pessimistic Poor Man’s swindle. In practice,
the Optimist is the more useful because SA generally spends the great bulk of its time 
rejecting bad moves.

4 Neighborhood Prejudice Swindle

In most applications of SA, fewer than 1% of moves are accepted overall. Thus the 
algorithm often spends many consecutive iterations at the same point. The goal of the 
neighborhood prejudice swindle is to reduce the time spent in this way. If most of the 
neighbors, (the points in $N(x)$), have a substantially greater $f$ value than does $x$, SA 
will spend a lot of time just rejecting them anyway. With neighborhood prejudice we 
use problem-specific information to identify a promising subset $S(x) \subset N(x)$. As with 
the previous swindle, we also employ a parameter $0 < p < 1$ to control the frequency 
of use. We replace the steps of the SA iteration by 

\textit{Neighborhood Prejudice Swindle}

1. With probability $p$, choose $y$ randomly from $S(x)$; with probability $1 - p$ choose $y$ randomly from $N(x)$.

2. Calculate $\Delta f$ and the nominal acceptance probability, $r = e^{-\Delta f/T}$, (if $\Delta f > 0$), $r = 1$ (if $\Delta f \leq 0$).

3. If $y \notin S(x)$, adjust $r \leftarrow r/(1 - p)$.

4. Accept with probability $r$; otherwise reject. Return.

This modified procedure shows favoritism to the members of $S(x)$. If $S(x)$ is 
well-chosen, it contains the neighbors of $x$ that have decent acceptance probabilities: 
pure SA would probably eventually choose one of these anyway, and the neighborhood 
prejudice just helps SA come to this decision a bit sooner.
In many applications, it is fairly easy to find a set of promising neighbors. For instance, variables that contribute directly to a constraint violation or penalty function are good candidates for change. Here we illustrate the swindle on the problem of finding Ramsey graphs.

A **clique** of size $k$ in a graph is a set of $k$ vertices, each pair of which is connected by an edge. A triangle is a clique of size 3. An **independent set** of size $k$ is a set of $k$ vertices, no pair of which is connected by an edge. We call a graph a $(k, l)$-Ramsey graph if it has no cliques of size $k$ and no independent sets of size $l$. For example, figure 1 shows a 3,3-Ramsey graph of size 5; figure 2 shows a 3,4 Ramsey graph of size 8. A famous theorem due to Ramsey states that, for any fixed $k, l$, there is a finite upper bound on the size (number of vertices) of the $(k, l)$-Ramsey graphs. Thus, there is no 3,3 Ramsey graph of size 6. Finding large Ramsey graphs is widely known as a very difficult combinatorial problem, hence it is a good sample problem for SA.

![Figure 1](image1)

![Figure 2](image2)

We implement SA as follows: the set of possible solutions is the set of all graphs of some desired size $n$; the neighborhood $N(z)$, where $z$ is a graph, consists of all graphs that differ from $z$ in one or two edges. The objective is to minimize the number
of cliques (and independent sets) of size \( k \) (or) and greater. The main cost of an SA iteration is in counting the number of these cliques and independent sets. (Finding the largest clique in a graph is NP-hard, so this counting has to be done enumeratively.)

Once we are finding and counting the large cliques (and independent sets) in the graph \( x \), it is easy to find promising neighbors. We say a vertex pair is "troublesome" if it occurs in one or more large cliques (or large independent sets.) If we change \( x \) by removing an edge between a troublesome pair, (or adding an edge if there is none), then we will have removed one or more large cliques (or independent sets). Hence we have a neighboring graph with a good chance of being better than \( x \). This leads to our definition of \( S(x) \): \( y \in S(x) \) iff \( y \in N(x) \) and \( y \) differs from \( x \) at one or two troublesome pairs of \( x \).

Table 1 gives some computational test results for differing values of \( p \). All tests were performed with a PASCAL implementation on a DEC MicroVAX II. The important thing to notice is that some use of the swindle is good, but too much is bad. As \( p \) increases from 0, the required cpu time drops until \( p \approx .8 \). As \( p \) increases beyond this point, the required cpu time increases. The worst algorithm performance occurs at \( p = 1 \). Obviously, one could construct cases where SA would never find the optimum when \( p = 1 \).

This experimental evidence supports the theoretical convergence properties of SA. The first convergence proofs of SA (e.g. [6],[8]) assumed symmetry of the neighborhood-generating transition probability matrix. That is, it was assumed that \( R_{ij} = R_{ji} \) where \( R_{ij} \) \( \equiv \) the probability \( j \) is the neighbor generated from \( i \). Any use of neighborhood prejudice would violate this condition. More recent proofs, however, (e.g. [1], [3], [10]) require only irreducibility and a kind of path cost symmetry condition (if you can get
Table 1: Performance of Neighborhood Prejudice Swindle: $k = l = 4$

<table>
<thead>
<tr>
<th>$n = 14$</th>
<th>$n = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$ (in %)</td>
<td>$p$ (in %)</td>
</tr>
<tr>
<td>cpu (minutes)</td>
<td>cpu (minutes)</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>12.3</td>
<td>120</td>
</tr>
<tr>
<td>75</td>
<td>77</td>
</tr>
<tr>
<td>2.4</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>57</td>
</tr>
<tr>
<td>20.8</td>
<td>230</td>
</tr>
<tr>
<td>25</td>
<td>47</td>
</tr>
<tr>
<td>26.6</td>
<td>270</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>335</td>
<td>280</td>
</tr>
<tr>
<td>1</td>
<td>370</td>
</tr>
</tbody>
</table>

from $i$ to $j$ without the $f$ value ever exceeding $z$, then you can get from $j$ back to $i$ without exceeding $z$). Partial use of neighborhood prejudice satisfies these weaker conditions, whereas total use does not. Hence the empirical results corroborate the theory.

One way to use this neighborhood prejudice is to invoke it only when SA has spent more than some fixed number of consecutive iterations at the same point. Intuitively, the swindle should be more useful when the neighbors of $x$ have relatively poor $f$ values.

5 Target Prejudice Swindle

When solving an optimization problem, if we knew where we were trying to go, it might help get us there faster. This plausible statement motivates the third swindle, Target Prejudice. Our target is an $x$ that minimizes $f$. Suppose we know (or guess) some characteristic of our target solution $x$. For example, all the classic maximum size Ramsey graphs are highly symmetric: all nodes in Figure 1 are degree 2 (touch 2 edges); all nodes in Figure 2 are degree 4. We might guess that for other values of $k, l$ there is a Ramsey graph whose nodes all have the same degree. This is a pre-judgement
about what our target looks like. The idea of Target Prejudice is to use this guess to guide SA. We can add a penalty to the function $f$ for violations of this property. If the penalty is very large, this is equivalent to restricting the search space to points with this property.

Will this method work? If the penalty function $h$ is nonnegative, and the target (global optimum) solution $x^*$ does not incur a penalty, then clearly $x^*$ globally minimizes $f + h$, also. On the other hand, if our guess about the characteristics of the target is wrong, then $(f + h)$ may not be minimized at $x^*$: SA will be misled.

The interesting (and disappointing) thing about Target Prejudice is, even if the prejudice is correct, it may hurt the SA algorithm efficiency. As an example, consider the problem in Figure 3. Each point is a node; neighbors are connected by edges; function values are given next to the nodes. How efficient will SA be on this problem? As Hajek and others [H, Ts, AF] have shown, the controlling factor for the convergence is the maximum depth of any point, denoted $d$. The depth of a point is the smallest necessary amount of increase in $f$ along any path to a point with better $f$ value. For instance, the depth of $H$ in Figure 3 is 0, because one could go directly to a point with better $f$ value ($G, A, I,$ or $L$). The depth of $C$ is 1, because the path $C, J, M$ brings $f$ up 1 from 3 to 4, and no other path increases $f$ less. The maximum depth $d$ for the problem in Figure 3 is 2 because of node $A$.

The smaller $d$, the faster SA will work. As an extreme example, if $d = 0$ then the temperature $T$ can be set to 0 right away; there are no poor local optima to get stuck at. Target Prejudice changes the effective $f$ values, and hence can change $d$. This change could be good, but it could be bad. For example, in Figure 3, if we add a large penalty to the corner nodes $A, C, E, G$, the maximum depth $d$ drops to 0. But if we
penalize or eliminate the square nodes $I, J, K, L$, then $d$ increases (up to 7 at node $C$).

Note that the global optimum $F$ is unchanged in both cases.

We can see from this example that Target Prejudice alters the topology of the problem. If the points removed (or penalized) are locally but not globally optimal solutions, then SA may work faster. But if the points removed are on paths towards the global optimum, the topology can become hillier and SA will be slowed. In our experience, the latter case is more common. This is probably because one rarely knows in advance how to recognize that a local optimum is not a global optimum — this is what makes global optimization so hard in the first place.

Reiner [9] reports some experimental results on an IBM mainframe, using Target Prejudice on the Ramsey graph problem introduced in section 4. He restricts the
search space to a highly symmetric class of graphs known as “cyclic 2-colorings”. The results are very poor: SA was unable to find even a 4,5-Ramsey graph of size 14. Several other restrictions were tried, with the same negative computational result. In contrast, we consistently find 4,5-Ramsey graphs of size 15 in negligible time (less than 5 seconds on a MicroVAX, or less than 30 seconds on an IBM PC); and have matched the size of the largest known 4,5-Ramsey graph, 24 vertices (at a cost of 30 cpu hours).

Target prejudice is an attractive idea. The prospect of reducing the huge size of a search space is particularly appealing. However, simulated annealing is not like other optimization methods (e.g. branch and bound): it is more sensitive to the topology than the cardinality of the search space.

6 Conclusions

The three techniques we have presented can all help reduce the cpu requirements of SA. Of the three, the poor man’s swindle appears the safest to use, since it checks itself. The optimistic poor man can be particularly safe and useful in applications where the objective function includes penalties. In our experience, neighborhood prejudice is often quite useful in increasing the efficiency of SA. Some amount of self-validation is possible with this swindle, as well. We can keep a profile of how good the choices from the promising set \( S(x) \) have been, and compare it with a profile of the entire neighborhood set \( N(x) \). This will give some information about optimistic (Type 2) errors. If in addition it is possible to check for membership in \( S(x) \), we can detect “good” neighbors of \( x \) that are not in \( S(x) \) (Type 1 errors). However, the technique should be used with some restraint. As we have seen, the third technique, target prejudice, appears to be the most risky. Changing the topology of the problem can
have a dramatic negative effect on performance.

**Computational Factors**

Based on our computational experience, the following are a few points to consider:

- **Inaccuracies in simulating SA** (e.g. type 1 and 2 errors) can be especially harmful at the beginning of the SA procedure, when the temperature $T$ is high. It is at this stage that SA seems to settle on a rough global form of the solution; the bias introduced by errors during this process can change the outcome tremendously.

- **Be wary of interaction effects between swindles.** Two different swindles may each, when tested, be virtually error-free, but have many errors when used simultaneously. In one application, we had an error rate of 0.5% for a poor man’s swindle, and an error rate of about 2% for a neighborhood prejudice swindle. But when we used the two together, we found that more than 60% of the promising neighbors (the members of $S(x)$) led to errors with the poor man’s swindle.

- **Small test problems can give misleading results.** If the test problem is small enough to be solved by local improvement (equivalent to running SA at $T = 0$), the estimates of error rates, etc. are apt to be inaccurate.

**How good is simulated annealing?**

The following comments about SA have been made at conferences or in technical articles: “powerful”, “expensive”, “snake oil,” “effective”, “based on a faulty analogy”, “very good results...”, “disappointing”, “ineffective”. We believe that the great variation in evaluations is a natural consequence of SA’s newness. As people gain more experience with SA, its appropriate place in the optimization toolbox will become clearer. Here we make a rough guess at the eventual verdict:

14
• SA is a very general tool for solving hard problems.

• SA is not usually as good for a particular problem as a specialized optimization method would be. Thus SA is especially appropriate when there is no alternative method available, or you have no idea how to go about finding a solution.

• SA is relatively easy to implement.

• SA is robust: if the problem changes, it is not hard to adjust SA accordingly.

• SA is very slow.

• SA is responsive to the allocation of computer resources: the more time you give it, the better quality solution you get. As computers continue to get faster and cheaper, but programmers do not, SA may become more attractive.

One reason for the widespread disappointment at the slowness of SA is that some of the initial test results were considerably better than those attainable by the "pure" simulated annealing algorithm. In particular, the travelling salesman test results reported by Vecchi and Kirkaptrick ([5],[11]) are one or two orders of magnitude better than "pure" SA yields. This gave some people the mistaken impression that SA was a competitive algorithm for (heuristically) solving travelling salesman problems. In their paper, however, Vecchi et al. state that that they monitor the progress of the algorithm, and "reheat" (i.e. increase $T$) when appropriate. (This corresponds to the way annealing is performed by metallurgists and crystallographers). The algorithm's progress is monitored by displaying the current solution on a computer screen. Humans have excellent visual abilities, and can easily recognize when the solution takes on a globally "stupid" characteristic. When this happens, the temperature is raised a bit until the algorithm settles on a more intelligent global configuration. We have
run SA interactively, and verified that it dramatically increases the rate at which $T$ can be dropped. But we know how to detect globally good or bad structures in only a few problems, such as two-dimensional routing. And this detection is necessarily interactive: it is not known how to program a computer to perform this kind of visual reasoning. We conclude that interactive SA has promise for a limited variety of problems, where some kind of feedback on global solution quality is available.

*How good is simulated simulated annealing?*

Simulated simulated annealing is not a panacea for the often devastating cpu requirements of simulated annealing. In our experience, its impact is usually not enough to make SA the method of choice when it otherwise would not be. But when SA is the method of choice, perhaps because of other considerations such as ease of use or robustness, simulated SA can reduce cpu time usage by an order of magnitude or two, without detracting much from its good characteristics.
References


Asymmetric Probabilistic Prospects of Stackelberg Players

Craig A. Tovey*
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

July 12, 1988

Abstract

Alkan et al. [1] consider the family of all bimatrix games with ordinal payoffs, and conclude that the average leader and follower enjoy symmetric prospects under the Stackelberg solution concept. In contrast, economics lore stresses the asymmetry between leader and follower, the leader generally enjoying the more favored position. We replace the computational analysis of [1] by a simple probabilistic combinatorial argument. We then impose monotonicity conditions on the player preferences. With this regularity condition, the symmetry between leader and follower breaks down, and most of the resultant advantage accrues to the leader. Thus, the monotonicity largely restores the advantage ascribed by economics folklore to the leader. Our analysis extends to non-ordinal payoff matrices.

ISyE Technical Report J-88-22

*Supported by a Presidential Young Investigator Award from the National Science Foundation (ECS-8451032)
1 Introduction

The Stackelberg solution concept has found many applications in control theory, games, and economics. Part of its strong appeal is its distinguishing between leader and follower, which is consistent with many commonly observed situations. Many games, such as chess, checkers, or go, are inherently sequential. In the business world, many market competitors have a long history of leader-follower behavior, e.g. one company sets the pace, and the others respond.

Thus, one expects an asymmetry: if at the outset of a game it is undecided who is the leader and who the follower, the players will not be indifferent between the two alternatives.

Traditionally, the leader is ascribed the advantage in this situation. In chess, for example, it is a statistically well documented advantage to move first. Often the leader in a competitive market is the company with largest market share. In the economics folklore it is considered advantageous to be the leader.

Alkan, Brown, and Sertel [1] question the asymmetry of the Stackelberg solution concept. They consider a class of bimatrix games in which the two players are each drawn at random from a population of leaders and followers, respectively. They find the following counterintuitive result: on average, the Stackelberg solution favors neither the leader nor the follower, when the populations are the sets of all ordinal payoff matrices. Instead, the distributions of payoffs to leader and follower are symmetric.

In section 2 we replace the combinatorial calculations of [1] by a simple probabilistic combinatorial argument. This line of reasoning exposes more clearly the nature of the payoff distribution. We also show a minor asymmetry between leader and follower.

A question posed by [1] is, what would be the effect of imposing a regularity condition such as monotonicity on the player preferences? We motivate these more complex cases in section 3. It turns out (section 4) that the symmetry between leader and follower breaks down. Interestingly, the leader emerges with the advantage. Thus the outcomes are more in accord with the economics folklore when preferences are monotonic instead of arbitrary. Since monotonicity is generally regarded as a natural, plausible condition on a family of preferences, we take this accord as a confirmation of economics lore. All of the analysis (section 5) holds for a general class of payoff distributions of which the ranked outcomes are a particular case. We conclude with this and a few other remarks in section 7.

2 Symmetric Probabilistic Outcomes

2.1 Definitions and Notation

First we settle some notation and define the game. The two players, leader and follower, will be referred to as L and F, respectively. For clarity we also assign nominal genders of male and female respectively to L and F. Each player has a payoff matrix, denoted L and F, respectively. Each matrix has m rows and n columns; the i,j cell of L is denoted L_{ij}.

Following [1], the values in the payoff matrix are the integers 1, 2, through mn, arranged in some random order. These integer can be taken to represent the most favored outcome, second most favored outcome, through least favored outcome of the player. We will call such a matrix an ordinal matrix. The players are minimizers under this interpretation of the matrix entries.

As we remark in section 7, it is not necessary to our analysis to work with ordinal matrices. For example, the cells in the matrix could contain independently identically distributed values from some distribution (binomial, normal, exponential, etc.), and the results hold for the actual numerical distributions, not just the ordinal characteristics. The players are typically maximizers under this interpretation
of the matrix entries. The analysis is the same whether the players are minimizing ordinal ranks or maximizing monetary payoffs. All we use about the distribution is its exchangeability among cells. That is, the joint distribution of values is unchanged if we exchange the locations of any two cells.

The game is played as follows: first $L$ picks a row index $i : 1 \leq i \leq m$; then $F$ picks a column index $j : 1 \leq j \leq n$. The players receive payoffs $L_{ij}$ and $F_{ij}$ respectively. The Stackelberg solution is the outcome of this game if each player has full information about $F$ and $L$ and plays optimally.

The follower's optimal play is easy to determine. A row index $i$ is given; $F$ chooses $j$ to get her best value in that row. Independent of $L$, $F$'s strategy can be summarized as: for each row, pick the column with best value in that row. Given $F$, we can build a template which selects one cell per row. For example, if $F$ is

\[
\begin{pmatrix}
1 & 2 & 3 \\
6 & 5 & 4 \\
7 & 8 & 9
\end{pmatrix}
\]

the template is

\[
\begin{pmatrix}
\square & \text{\large \blacksquare} & \text{\large \blacksquare} \\
\text{\large \blacksquare} & \text{\large \blacksquare} & \square \\
\square & \text{\large \blacksquare} & \text{\large \blacksquare}
\end{pmatrix}
\]

A cell is marked if it is selected by the template. We can now describe $L$'s strategy in terms of the template. He looks at $F$, derives the template, and lays it on top of his matrix $L$. Imagine the template as opaque, with $m$ square holes cut out. When the template is laid down over $L$, $L$ can see $m$ marked values, one in each row. He picks the row with the best value. This "template" construct will be useful in the following analysis.

2.2 Symmetry of Random Leader vs. Random Follower

We begin by proving that if $F$ and $L$ are drawn randomly from the population of all possible ordinal matrices, then $F$ and $L$ enjoy symmetric prospects. Let $F$ be generated randomly, yielding a template. Put the template over a blank leader matrix $L$, and now let the $nm$ terms be distributed randomly. What will $L$ see through the template? Because all cells are interchangeable, regardless of the specific template $L$ will see $m$ terms randomly selected from among the $mn$ terms. He then picks his best, which is simply the best of $m$ values selected randomly from the $mn$. This payoff distribution will recur; we will call it "best of $m$".

Now notice that, regardless of the template, the best of the $m$ terms seen by $L$ is equally apt to be in any row. So from $F$'s point of view, regardless of her matrix $F$, a random leader picks the row index $i$ at random. $F$ then picks her best of the $n$ cells in that row. So by exchangeability, $F$'s payoff is simply the best of $n$ values extracted randomly from the $mn$. As before, we will call it the "best of $n$". We have proved the following theorem:

**Theorem 2.1** If $L$ and $F$ are both random, then the distribution of the Stackelberg payoff to each player is "best of $k$", where $k$ is the number of strategies available to the player.

Obviously when $n = m$ the payoff distributions are identical.
2.3 Symmetry of Fixed Player vs. Random Opponent

Actually, we have proved something stronger with respect to the leader: his payoff, for any fixed template, has distribution best of m. Since the template contains all the pertinent information from L's point of view, we have:

Lemma 2.1 If L is random then for any fixed F the distribution of payoff to L is best of m.

The proof of Theorem 1 does not imply the symmetric statement about payoffs to random followers against fixed leaders. We prove it separately:

Lemma 2.2 If F is random then for any fixed L the distribution of payoff to F is best of n.

Proof: Fix the leader's matrix L. His strategy can be summed up by giving, for each possible template, which row he would choose.

Consider any template T, and without loss of generality suppose the row L would pick is row 1. If we choose randomly from among all F that yield T, what value will be in row 1, under the template? We can create a "random" F that yields T in two easy steps:

1. Generate a completely random F.
2. In each row of F, swap the least (best) element with the element under the template T.

For example, if the template is diagonal then

\[
\begin{pmatrix}
3 & 2 \\
4 & 1 
\end{pmatrix}
\]

would become

\[
\begin{pmatrix}
2 & 3 \\
4 & 1 
\end{pmatrix}
\]

The transformation in step 2 maps the set of all order matrices into the set of all order matrices with template T. There are n ways each possible row can be created, hence for each T exactly n^n order matrices get transformed to a matrix that yields T. Therefore the two steps produce all such matrices with equal probability.

Now that we know how to produce the random F that yields T, what value is under T in row 1? This value is simply the best of the entries in row 1 of the matrix F as it was in step 1, i.e. our usual best of n distribution.

Alkan et al. prove the same leader-follower symmetry implied by Lemmas 2 and 3. They derive explicit numerical formulas for the distributions. Our constructive "best of k" description is simpler to derive, shows the true nature of the distribution, and leads more easily to further insights.
2.4 Minor Asymmetries between Leader and Follower

So far, only one asymmetry between leader and follower is apparent: the lengths of the author's proofs of Lemmas 2 and 3 are not equal. If we consider the payoffs to the fixed players in these Lemmas, we can find an objective (though minor) asymmetry.

Lemma 3 says that if the population of leaders pick a representative with care, and the followers pick one at random, the followers do exactly as well as they would against a random leader. Lemma 2 makes the equivalent statement about random leaders and chosen followers. But not every leader does equally well against a random follower. By the analysis in the lemmas, a random leader (follower) is equally likely to select each row (column) against any fixed opponent. Therefore, no cell in a leader's (follower's) matrix can have a chance more than $1/n$ ($1/m$) of being the payoff against a random opponent. This observation implies the following

**Lemma 2.3** For any fixed $L$ (resp. $F$), the payoff distribution against a random opponent cannot be better than a random selection from the best $n$ (resp. $m$) outcomes.

*Proof:* We have observed that no single outcome can have probability more than $1/n$ (resp. $1/m$). Therefore the best distribution of outcomes gives probability $1/n$ (resp. $1/m$) to the best $n$ ($m$) outcomes.

When the payoff matrix is ordinal, a payoff distribution can be described as a vector $P$, where $P_j$ equals the probability that the outcome is $j$.

**Definition 1** A payoff distribution $P$ is better than a payoff distribution $Q$ iff

$$\sum_{j=1}^{r} P_j \geq \sum_{j=1}^{r} Q_j$$

for all $r = 1, 2, ...$

This definition is simply the notion of stochastic dominance applied to the discrete probability density functions associated with $Q$ and $P$ based on the preference ordering of the outcomes.

**Remark:** When we assert that one payoff distribution, $P$, is better than another one, $Q$, we mean that $P$ dominates $Q$ in such a way that any rational individual will prefer $P$ to $Q$. A payoff distribution is no more than a lottery, with prespecified probabilities for each outcome. If we assume that

1. The individual's preferences are transitive on lotteries (transitivity),

2. If the individual prefers $i$ to $j$, and $\Delta \geq 0$, then the individual prefers the lottery $P = (p_1, p_2, ...)$ to the lottery $P' = (p_1, p_2 - \Delta, ..., p_i + \Delta, ...)$ (monotonicity).

then if $P$ is better than $Q$ according to definition 2.1, the individual must prefer $P$ to $Q$.

These assumptions are very weak. (In particular, they do not constrain the individual's attitude towards risk.) In fact they are a subset of the axioms satisfied by any individual whose preferences can be modelled by a linear utility function according to the Von Neumann-Morgenstern utility theory (so-called rational).

Now we come to an asymmetry. Let's count the number of leaders who, favored by the gods of probability, actually achieve the distribution given in Lemma 2.3. A "favored" leader has his best $n$ outcomes all in the same row. There are $m$ rows, $n!$ ways to arrange the top $n$ outcomes in that row, and $(mn - n)!$ ways to arrange the remaining terms. Therefore there are

$$m(n!)(mn - n)!$$
favored leaders. If the symmetry between leader and follower were complete, then this would equal the number of favored followers when \( m = n \). But, a follower is favored if her top \( m \) choices are spread over all \( m \) rows, i.e. each row contains exactly one of the top \( m \) choices. When \( n = m \) this means there are \( n^2 \) places to put the top choice, \( n^2 - n \) places to put the second choice, \ldots, \( n \) places to put the \( n \)th choice, and \( (n^2 - n)! \) ways to arrange the remaining terms. Therefore there are

\[
n^nn!(n^2 - n)!
\]

favored followers. This exceeds the number of favored leaders by a factor of \( n^{n-1} \).

As an example, suppose \( n = m = 2 \). The favored leaders are

\[
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}, \begin{pmatrix}
1 & 2 \\
4 & 3
\end{pmatrix},
\]

and the six matrices obtained by rearranging rows and columns to put the 1 in the other three cells, making 8 altogether. The favored followers with a 1 in the upper left corner are

\[
\begin{pmatrix}
1 & 3 \\
2 & 4
\end{pmatrix}, \begin{pmatrix}
1 & 3 \\
4 & 2
\end{pmatrix}, \begin{pmatrix}
1 & 4 \\
2 & 3
\end{pmatrix}, \begin{pmatrix}
1 & 4 \\
3 & 2
\end{pmatrix},
\]

making a total of 16 favored followers altogether.

2.4.1 Implications

Thus the symmetry between leader and follower is not complete. The greater number of followers who do very well, together with the fact that the average follower payoff equals the average leader payoff, suggests a wider dispersion of outcomes among follower than leaders. This in turn suggests that if one is risk averse, one would prefer to be a leader than a follower.

We caution the reader that this observation makes sense under one of two possible interpretations of the random bimatrix game. Mathematically, the leader's matrix is taken from a probability distribution on matrices. The source of the randomness admits of two interpretations. First, there could be a single leader, whose payoff matrix is generated randomly whenever he starts to play the game. Second, there could be a large population of leaders, each with his own fixed payoff matrix, one of whom is selected randomly to play the game. Our observation is with respect to the first interpretation. We favor the first interpretation because we feel it more accurately depicts a dynamic or uncertain situation, and because the second interpretation cannot apply to cases where the matrix payoffs come from a continuous distribution.

3 Stackelberg Outcomes with Monotonic Player Preferences

3.1 Monotone Payoffs: Definition and Motivation

In the game defined in section 2.1, \( L \) and \( F \) are randomly taken from the population of all possible payoff matrices. Lemmas 2.2 and 2.3 have either \( L \) or \( F \) fixed though unknown, while the opponent is random as before. We call these two populations of payoff matrices random and fixed respectively. In this section we introduce two other populations, the row monotone and column monotone (row and column for short), and analyze the payoff distributions among these four populations.

First, let us motivate the introduction of these two new populations. If arbitrary payoff matrices are allowed, a player would have the possibility of facing, among others, each of two opponents, with completely opposite preferences. For example,
would both be possible opponents. But, in the competitive market equilibrium situations which originally motivated the Stackelberg criterion, it could be unlikely for possible opponents to differ so widely in their preferences. This is because all the players want roughly the same thing, though they have different production functions. Thus, it might be more realistic to impose some regularity condition on the player preferences. Monotonicity is one of the simplest and most common such conditions.

There are two ways to impose monotonicity on the player preferences, within row and within column. If the payoff values in a matrix are decreasing in each row (respectively column) we say the matrix is row (respectively column) monotone.

3.2 Random monotone matrices

A monotone matrix is assumed to come randomly from the set of all possible matrices satisfying that monotonicity. For brevity, we will use terms such as "row leader" to mean "a leader whose payoff matrix is randomly selected from the population of row monotone matrices."

It will help to have a conceptual model of how to randomly generate a monotone matrix. To create a random row monotone matrix, we can simply generate a random matrix, and arrange the numbers in each row from largest to smallest. It is obvious that any matrix so generated is row monotone. It is also clear that every row monotone matrix has the same probability of being generated, since exactly \(\binom{n}{m}\) random matrices get "mapped" into each row monotone matrix. Therefore our generating model is correct.

4 Summary of Results

The leader and follower can each be of four possible types: random, row, column, or fixed. As we have seen in section 2, the leader's and follower's payoffs must be considered separately. Hence there are 16 possible cases, and 32 analyses to perform.

We leave it to the reader to verify that it is better to defer the analyses until the next section, and present the results now.

Table 1 below summarizes the results. In each cell, the outcome to the leader is described in the lower left corner, and the outcome to the follower is described in the upper right corner. A short glossary of terms used in the table follows:

usual: best of \(m\) for the leader; best of \(n\) for the follower.

better: better than usual. (better means the dominance defined in section 2.3).

worse: worse than usual (i.e., dominated by the usual.)
depends: The outcome can be better or worse, depending on the fixed matrix (used only when one of the players is fixed). In other words, for some fixed matrices the outcome is better; for some fixed matrices the outcome is worse.

**perfect:** The player always gets his most favored outcome.

Of the 32 outcomes to analyze, we have rigorous justification for 29, and heuristic arguments for the other 3. The results for the latter three are set in italics. (The heuristic arguments are also supported numerically for small cases).

### 4.1 Discussion — Symmetry and Asymmetry

Apparently, in the majority of cases, the leader/follower symmetry breaks down. There are only four cases where the distribution to $L$ and $F$ are sure to be the same: if one player is random and the other is random or row monotone, or if we have a row-leader and column-follower.

In our view, the layout of Table 1 overstates the asymmetry present. The symmetry between leader and follower, if it exists, is based on the equivalence between the leader's rows and the follower's columns. That is, the leader's rows are the followers columns. For a moment, let $n = m$, and consider some fixed $L$. If there were a fixed $F$ that did just as well against random leaders as $L$ did against random followers, the natural guess would be that $F = L^T$, the transpose of $L$. Similarly, it is really not surprising that if a row follower plays a row leader, the two players do not enjoy the same prospects. This is not a deep asymmetry. But it would be natural to expect that a row leader gets the same prospects against a row follower, that a column follower gets against a column leader. Table 4.2 shows the payoffs that match up against each other when the leader-row / follower-column duality is taken into account. Two payoff locations have the same number if they match with each other, so the integers 1 to 16 each appear twice in the table.

Table 3 compares the results in Table 1 through the pairings given in table 2. There is still a fair amount of asymmetry present, particularly when one or both of the players is monotone.

### 4.2 Discussion — Leader versus Follower

One thing that emerges very clearly from the results is that the leader has regained his traditional advantage. The leader comes out worse than usual only once, while the follower comes out better than...
usual only once. Suppose one were to play the following 2-stage game: in stage 1 a cell in table 4.1 is selected at random; in stage 2 the corresponding game is played. Clearly in this game, it is advantageous to be the leader.

Moreover, we argue that the leader's advantage is even stronger. When we compare the payoff distributions via the row/column duality the leader dominates the follower. The leader does better than the follower in six cases and equally well in the other ten, as seen in Table 3.

5 The Analyses

Theorem 5.1 The payoff to the random player against any of the four types of opponents is the usual.

Proof: This follows from Lemmas 2.2 and 2.3. If the payoff has the same distribution against any fixed opponent, it has the same distribution against any class of opponents, as well.·

By considering a fixed opponent, we can dispense with 8 more analyses with just two more theorems.

It will be convenient to name some special templates.

Definition 2 A template is a straight column template if its distinguished cells form a single column. If that column is the nth, the template is called the right column template.

Theorem 5.2 The payoff to a row follower against any fixed leader is the usual.

Proof: The templates of all row-monotone follower are the same: the right column template. Therefore, any fixed L will react in the same way to all row-followers. Fix L and suppose without loss of generality that L will choose row index i = 1. What payoff does a randomly generated row follower get against i = 1? She gets the best of the n values that were randomly put in that row (and then sorted). This is precisely her usual.·

Theorem 5.3 Against any straight column template the column leader's payoff distribution is the usual. Against any other template, the column leader's payoff is better than the usual.

Proof: We claim that if L were to always choose the first row, i = 1, his payoff distribution would be the usual, against any fixed template. Let the template's special cell in row 1 be in column 1, without loss of generality. Then L's payoff would always be the first number in column 1. Following our model on
Table 3: Leader Domination Based on Table 2 Case Matching

<table>
<thead>
<tr>
<th>Leader</th>
<th>Follower</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>usual</td>
</tr>
<tr>
<td>2</td>
<td>better</td>
</tr>
<tr>
<td>3</td>
<td>usual</td>
</tr>
<tr>
<td>4</td>
<td>depends</td>
</tr>
<tr>
<td>5</td>
<td>usual</td>
</tr>
<tr>
<td>6</td>
<td>better</td>
</tr>
<tr>
<td>7</td>
<td>worse</td>
</tr>
<tr>
<td>8</td>
<td>depends</td>
</tr>
<tr>
<td>9</td>
<td>usual</td>
</tr>
<tr>
<td>10</td>
<td>usual</td>
</tr>
<tr>
<td>11</td>
<td>perfect</td>
</tr>
<tr>
<td>12</td>
<td>depends</td>
</tr>
<tr>
<td>13</td>
<td>usual</td>
</tr>
<tr>
<td>14</td>
<td>better</td>
</tr>
<tr>
<td>15</td>
<td>depends</td>
</tr>
<tr>
<td>16</td>
<td>depends</td>
</tr>
</tbody>
</table>

how a random column monotonic matrix is created, this payoff would be the best number in column 1 of a purely random matrix, which is the usual best of \( m \) distribution. Therefore, \( L \)'s payoff distribution is sure to be at least as good as the usual.

Now, if the template is a straight column template, then it will always be optimal for \( L \) to choose \( i = 1 \). This is so because column monotonicity ensures \( L_{i1} \) is worse than \( L_{11} \) for all \( i > 1 \). Thus \( L \)'s payoff distribution will be exactly the usual.

On the other hand, if the template is not a straight column template, we claim it will sometimes be better for \( L \) to choose a row index \( i \neq 1 \). Assume as before the template's marked cell in row 1 is in column 1. Suppose further the template has some marked cell in a column other than 1, in some row \( k \). Then if the worst \( m \) values are in \( L \)'s first column, that is \( L_{i1} = nm - m + i \), \( L \) will get a better payoff choosing row \( k \) than row 1. Thus there exists a strategy for \( L \) which gets a better than usual payoff distribution. Therefore, if \( L \) follows his best strategy, his payoff distribution will also be better than usual.

The payoffs to column leaders in table 1 now follow. As remarked in the proof of theorem 4.2, a row follower always has a straight column template, so \( L \) only gets the usual against her. It is easy to see that templates of column followers are not so restricted. Also, out of the \( n^2 \) fixed templates, only \( n \) are straight column.

We can dispense with the fixed leader versus fixed follower payoffs as the only trivial case in the table.

We evaluate two straightforward row leader payoffs next.

Lemma 5.1 The payoff to a row leader against a row follower is always his best outcome.

Proof: The leader's best outcome must be rightmost in some row of \( L \). Since \( F \)'s template must be the right column template, the leader's best outcome must be in a marked cell, and he can select it.
Lemma 5.2 The payoff to a row leader against a random follower is best of $m$.

Proof: Consider what happens to the numbers in $L$. First they are split randomly into $m$ groups (the rows). Then each group is arranged in order. Finally, the follower picks one from each group to be in a marked cell. Since $F$ is totally random, which number it picks from a group is totally random. Therefore, it doesn’t matter that the group was arranged in order — the selection is random anyway. The leader now selects the best of these $m$ values, which lead to the usual distribution.

The next case is a little more difficult:

Theorem 5.4 A column follower playing a random leader receives a payoff worse than usual.

Proof:

From the proof of theorem 2.1, even for any fixed template a random leader will randomly select among the row indices with equal probability. Combining this fact with our method for generating random column monotone matrices, we can give a narrative for generating $F$’s payoff: start with a random matrix, sort the values in each column from best to worst, select a row at random, and extract the best value in that row.

Intuitively, the outcome from this narrative should not be very good. Think of the values as eggs, some good, some bad. Think of the rows as baskets, each containing $n$ eggs. The leader is going to randomly select one basket, and $F$ gets to take the best egg in that basket. Therefore, $F$ would like to have the good eggs scattered among the baskets so each basket contains a pretty good egg. When the values are sorted within columns, too many of the good eggs will end up in the one basket, and the average outcome should suffer.

We make this intuition precise. For any $k$, consider the probability that a follower with matrix $F$ gets one of her $k$ best outcomes (versus a random leader). This probability is exactly equal to $\beta(F,k)/m$, where $\beta(F,k) \equiv$ the number of rows of $F$ “hit” by (containing at least one of) the $k$ best outcomes. For example, if $k = 2$, then the probability is $1/m$ if 1 and 2 hit the same row ($\beta = 1$); and $2/m$ if they are in different rows ($\beta = 2$). Now let $M$ be any matrix, and let $\tilde{M}$ be the column monotone matrix that results when $M$’s columns are sorted. We will show that

\[ \beta(M,k) \geq \beta(\tilde{M},k) \]

for all $k$ and $M$. Moreover, the inequality can be strict. This implies that the follower’s prospects with $M$ are better (according to definition 2.1) than with $\tilde{M}$.

If the $k$ best values hit $\beta$ rows of a column monotone matrix $\tilde{M}$, then $\tilde{M}$ must have a column containing $\beta$ of the $k$ best values. This is because the $k$ best values must occupy the top portions of whatever columns they are in. But, if a column of $\tilde{M}$ has $\beta$ of the $k$ best values, then no matter how values are disarranged within columns to get $M$, that same column of $M$ must contain $\beta$ of the $k$ best values. Therefore, at least $\beta$ of $M$’s rows are hit by the top $k$ values. This verifies the above inequality and concludes the proof.

Here is an example of the inequality being strict. If

\[ M = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix} \]

then $\beta(M,2) = 2$ but $\beta(\tilde{M},2) = 1$.

We are now in a position to deal with several more cases as corollaries of the results obtained so far. We employ the following lemma:
Lemma 5.3 If a row or column or specific player's outcome against a particular kind of opponent is better or worse, then a fixed player's outcome against the same kind of opponent depends.

Proof: By Theorem 5.1, a random player's outcome against that particular kind of opponent is the usual. If some player (or class of players) does better (resp. worse) than usual, then some other player must do worse (resp. better) so that payoffs average out to the usual.

(Note: There is a slight weakness here. To illustrate, take the case where some player does worse than usual. we can not actually deduce that there is some player whose distribution dominates the usual as in definition 2.1. The disadvantage of the worse player might be balanced out among a group of other players. We can deduce that, for every possible preference among payoff distributions, there exists a player whose payoff is strictly preferred to the usual. It is possible that this preferred player varies depending on the preferences. In all the cases where we employ this lemma, we can also show the stronger result, but the proofs take more space.)

Corollary 5.1 The following payoffs depend:

1. Fixed leader's payoff versus random follower.
2. Fixed leader's payoff versus row follower.
3. Fixed leader's payoff versus column follower.
4. Fixed follower's payoff versus random leader.

Proof: Follows from Lemma 5.3 and

1. Theorem 5.3.
2. Lemma 5.1.
3. Theorem 5.3.
4. Theorem 5.4.

Corollary 5.2 A row leader's payoff against a fixed follower depends.

Proof: Similar to Lemma 5.3. Since a row leader's payoff against a random follower is the usual (Lemma 5.2), while against a row follower it is perfect (Lemma 5.1), we can deduce the claimed result.

Theorem 5.5 The following payoffs depend:

1. Column follower's payoff versus fixed leader.
2. Fixed follower's payoff versus column leader.
3. Fixed follower's payoff versus row leader.

Proof:

1. If L's top row contains his best n outcomes, L will always choose i = 1. Then the column follower's outcome is perfect against this kind of leader. By Theorem 5.4, the column follower's outcome can also be worse.
2. If the follower’s matrix $F$ is:

$$
\begin{pmatrix}
1 & 2 & 3 & \ldots & n \\
2 & 2 & 2 & \ldots & 2n \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
(2n) & (2n-1) & (2n-2) & \ldots & 1
\end{pmatrix}
$$

then any column leader will choose $i = 1$ and the follower’s payoff is perfect. The result follows as in Lemma 5.3.

3. If the rightmost column of the follower’s matrix contains her best $m$ outcomes, then her template is a right column template. By the proof of Theorem 2.1, $L$ is equally likely to choose among all rows. Therefore $F$’s payoff distribution is a random selection from among her best $m$ outcomes. By lemma 2.3, this is better than the usual. The result follows now from Lemma 5.3.

### 5.1 Heuristic Arguments — Behavior of Column Followers

This completes the rigorous analysis of 29 cases. We now come to the heuristic arguments for the remaining 3. They are: the two payoffs when a column follower plays a row leader, and the payoff to a column follower against a column leader. These cases are complicated because a column follower’s templates are irregular.

#### 5.1.1 Irregular Distribution of Column Follower Templates

**Proposition 1** All templates can occur with a column follower, but the templates do not all occur with the same probability.

Proof: Let $T$ be an arbitrary template. We construct a column monotone matrix that produces $T$. Fill in a 1 (most favored outcome) in the marked cell in row 1. Find the marked cell in row 2. If it is in the same column as the 1, fill in a 2; if it is in a different column, fill in a 3 and fill in a 2 above it. In general, for row $i$, find the marked cell in that row and let $k \geq 0$ be the number of unfilled cells above it. Put the $k+1$ best remaining values in the marked cell and the unfilled cells above it. After $i = m$, fill in the remaining values arbitrarily (but maintain column monotonicity.) This procedure constructs the desired matrix.

The second statement can be proved by example. For $n = m = 2$, a simple calculation verifies that the straight column templates occur with twice the probability of the “diagonal” templates. In contrast, the templates of a random follower occur with equal probability.

The nature of the irregular probabilities is a correlation within columns. For example, if the marked cell in row 3 is in column 5, then column 5 is most likely to contain the marked cell in row 4. This is clear intuitively because the better the value of a cell, the better the possible range of value of the cell below it (in a column monotone matrix). That is why the straight column templates are more likely to occur than the “diagonal” ones in the case $n = m = 2$. 

13
5.1.2 Payoff to Column Leader Against Column Follower

We first refine our analysis of the payoff distribution for the column leader against a column follower. By theorem 5.3, \( L \) gets only the usual payoff against a straight column template, but gets a better payoff against other templates. Proposition 5.1 ensures \( L \) really does do better than usual overall, because some of \( L \)'s opponents will not have straight column templates. However, the correlation within columns makes straight column (and similar) templates more likely. Hence the outcome is not as good as against equally probable templates. We conclude that the column leader does better than usual against a column follower, but less well than he does against a random follower.

5.1.3 Payoff to Column Follower Against Column Leader

Next we consider \( F \)'s payoff in the contest of the previous section. Theorem 5.3 tells us that \( L \) could achieve the usual payoff by always selecting \( i = 1 \). Whenever \( L \) selects the first row, \( F \) is sure to get her best outcome, because her best outcome is sure to be in her first row. However, \( L \) can sometimes improve his payoff by selecting \( i > 1 \). When this happens, \( F \)'s payoff will suffer. Intuitively, we expect that \( L \) selects \( i > 1 \) rather infrequently, because his best values strongly tend to be in his first row, and because \( L \) doesn't do a lot better than usual. We conclude that the column follower's outcome, while not perfect, is extremely good.

5.1.4 Payoff to Column Follower Against Row Leader

Now we consider a column follower's payoff when she plays a row leader. We know from Theorem 5.4 that if the leader were random, \( F \)'s payoff would be worse than usual. That is, if the leader were equally likely to select any row, \( F \)'s payoff would be as described in Theorem 5.4. When \( F \) has a straight column template, it is easy to see that this is the case. When \( F \) is not a straight column template, the column exchangeability of \( F \) implies that marked cells with few other marked cells in their column are more likely to be the targets for \( L \)'s row selection. But these cells tend to contain poor values for \( F \). We conclude (heuristically) that the payoff distribution to \( F \) is even poorer than if she were playing a random leader.

5.1.5 Payoff to Row Leader Against Column Follower

Finally, if \( L \) faces a random follower, he will get the usual by Lemma 5.2. But the column correlation of the column follower means \( L \) sees a higher proportion of straight column templates, (and other similar templates). What does a straight column template do to \( L \)'s outcome? By the column exchangeability, all the straight column templates must be equally likely. The payoff to \( L \) against a set of equally likely straight column templates is: select \( i \) at random, take \( n \) th order statistics, and choose the best of them. This is precisely the payoff to a column follower against a random leader, proved worse than usual in Theorem 5.4. So intuitively, we would expect that \( L \)'s payoff is a mix between the usual of Lemma 5.2 and the worse of Theorem 5.4. We conclude that \( L \)'s payoff is worse, but not as bad as a column follower's outcome against a random leader.

6 Optional section: small numerical examples

In this section we give the exact distributions for the case \( n = m = 2 \). Our convention for representing a payoff distribution is as a vector \((p_1, p_2, p_3)\) where \( p_i \) is the probability of getting payoff \( i \). Thus the perfect outcome is represented as \((1,0,0)\); it is easy to compute the usual outcome is \((3/6,2/6,1/6)\). In
all cases there is no chance of getting 4, the least preferred outcome, so $p_4$ is omitted. (We include $p_3$ for clarity, although it could be deduced from $p_1$ and $p_2$.) All calculations are omitted. The results are given in table 4. Note that they are consistent with table 1, except that a column follower does no worse against a row leader than against a random leader. This is nonetheless consistent with the heuristic analysis in section 5.1.4, because $n < 3$.

### Table 4: Payoff Distributions When Each Player Has Two Strategies

<table>
<thead>
<tr>
<th>Follower</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>(.50, .33, .17)</td>
<td>(.50, .33, .17)</td>
<td>(.50, .33, .17)</td>
</tr>
<tr>
<td>(.50, .33, .17)</td>
<td>(.50, .33, .17)</td>
<td>(.50, .33, .17)</td>
</tr>
<tr>
<td>(.50, .33, .17)</td>
<td>(1.0, 0)</td>
<td>(.50, .28, .22)</td>
</tr>
<tr>
<td>(.50, .42, .08)</td>
<td>(.50, .33, .17)</td>
<td>(.50, .39, .11)</td>
</tr>
</tbody>
</table>

7. **Remarks**

The key property of the payoff distribution used in the proofs is the exchangeability. Even when the numbers in the rows are arranged in decreasing order, the distribution of the entire row is identical and in fact exchangeable with any other row. The “best of $k$” distribution is simply the first order statistic (or last if maximizing) of a sample of size $k$ from the distribution. The ordinal matrices are just one particular case of an exchangeable distribution; the “best of $k$” will be well-defined for other distributions as well. Thus the payoffs in the matrix could be independent identically distributed values from any distribution: normal, gamma, Bernoulli (1 with probability $p$, 0 with probability $1 - p$), geometric, etc. and the results still hold for that distribution, not just for the ordinals of that distribution outcomes. Let $S$ be any finite set of numbers; the payoffs in the matrix could be drawn at random without replacement from $S$ and the results would still hold (the ordinal payoffs are a special case of this).

Two other cases involving monotonicity are possible: the players could be both column monotonic, or both row monotonic, but with opposing directions of preference. In the former case, the leader’s payoff is somewhat better than usual, from Theorem 5.3 and the discussion in section 5.1.2; the follower’s payoff is much worse than usual, because the leader will tend to pick F’s worst row. In the latter case, the follower will always pick L’s worst column: the leader’s payoff will be much worse than usual, while the follower’s payoff will be the usual. Thus if we pit the player outcomes strongly against each other, the game is quite unbalanced, the leader overall does a bit better, but does not appear to retain the dominance of section 4.2.

8. **Acknowledgments**

I thank Ahmed Alkan, for showing me the interesting paradox; Bob Foley, Steve Hackman, and Kris Calvin Haney for helpful comments.
References

VOTING SCHEMES FOR WHICH IT CAN BE DIFFICULT TO TELL WHO WON THE ELECTION

by

John J. Bartholdi, III
Craig A. Tovey
Michael A. Trick

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
404-894-3036

Abstract

We show that a voting scheme suggested by Lewis Carroll can be impractical in that it can be computationally prohibitive (specifically, NP-hard) to determine whether any particular candidate has won an election. We also suggest a class of "impracticality theorems" which say that any fair voting scheme must, in the worst-case, require excessive computation to determine a winner.


This research was supported in part by Presidential Young Investigator Awards from the National Science Foundation to the first two authors (ECS-8351313 and ECS-8451032), and by grant N00014-86-K-0173 from the Office of Naval Research.
1. Introduction. We can consider a voting scheme to be a well-defined rule by which any input consisting of a set C of candidates and a set V of transitive preference orders on C (the preferences of the electorate), one can determine a subset of C whose elements are the winners (allowing for ties). When a voting scheme is considered to be a rule to determine the winner(s) of an election, it is natural to ask about the computational resources required by the scheme. For example, can the scheme be guaranteed to quickly identify a winner? In other words, is there an efficient algorithm to find a winner under the given voting scheme?

For both practical and theoretical reasons, an algorithm is considered formally efficient if it requires a number of computational steps that is at most polynomial in the size of the problem. Problems for which there are polynomial-time algorithms are generally considered to be tractable, and those which can require exponential time to solve are considered inherently intractable.

Computational complexity can classify voting schemes based on a well-studied hierarchy of complexity classes that are thought to be distinct. For example, within this hierarchy is the problem class NP, which consists of those questions for which a "yes" answer can be justified in polynomial time. The hardest problems in NP are known as "NP-complete", and all such problems are equivalent in the sense that any problem in NP can be reworded as an instance of an NP-complete problem within polynomial time. Thus the existence of a polynomial-time algorithm to solve a single NP-complete problem implies that all problems in NP can be solved in polynomial time. That no one has found such an algorithm is taken as strong circumstantial evidence that NP-complete problems are inherently intractable. (For more on complexity and NP-completeness, see [7].)

We will show that, for two voting schemes, several natural questions about the outcome of an election are NP-complete. This suggests that it is very unlikely that one can find an efficient algorithm to answer them. Thus these questions can be too difficult to answer (at least for "sufficiently large" elections without special structure), and so the voting schemes might be impractical.

2. The difficulty of tabulating scores. We assume throughout that the preferences of all voters are strict (irreflexive and antisymmetric), transitive, and complete. We formalize as follows the the essential question to be answered by any voting scheme.
ELECTION WINNER

INSTANCE: Set of candidates C, and one distinguished member c of C; set V of preference orders on C.

QUESTION: Is c a winner under the specified voting scheme?

In most historical voting schemes only polynomial time is required to answer this question (a practical necessity when counting paper ballots!). For example, to solve PLURALITY-WINNER requires only $O(|V| + |C|)$ work to count first-place votes and identify the candidate with the most. It is unusual then to discover voting schemes that can apparently require exponential time to tell whether any particular candidate has won the election. We exhibit two such schemes. One was invented by the mathematician Charles Dodgson (better known as Lewis Carroll), and the other was suggested by J. Kemeny. We show that under either a Dodgson election or a Kemeny election, it is NP-hard (that is, at least as hard as an NP-complete problem) to determine whether any particular candidate has won! Thus these schemes are capable of taking an impractically long time to determine a winner. Others have observed the empirical difficulty of computing winners under these schemes [for example, 6]. However, we establish this in a formal sense, and suggest computational complexity as another aspect to be taken into consideration when practical voting schemes are to be judged.

The rationality criterion that troubled Dodgson was the famous one first formalized by the Marquis de Condorcet [4], which requires that a voting scheme elect any candidate (the "Condorcet winner") that would defeat any other candidates in a pairwise election with simple vote counts. Condorcet observed that there can be elections in which no candidate is a Condorcet winner (the "Phenomenon of Cyclic Majorities"). Accordingly, Dodgson sought a voting scheme that would still choose "rationally" in the absence of a Condorcet winner. He suggested the following voting scheme (reprinted at length in [3]; summarized in [13]).

THE DODGSON VOTING SCHEME:

A Dodgson winner is a candidate who is "closest" to being a unique Condorcet winner, where by "closest" we mean the following: Imagine that an election supervisor is empowered to change the ballot of any voter through pairwise interchange of candidates adjacent in the voter's preference order. Then a Dodgson winner is a candidate who requires the fewest interchanges to become a Condorcet winner. (Hereafter we will refer to such pairwise interchanges simply as "switches". The minimum number of switches for
a candidate to become a Condorcet winner is the Dodgson score of that candidate. A candidate with the smallest Dodgson score is a winner of the election.)

As an example of the Dodgson scheme, consider the candidates A, B, C, D, and three voters, one with each of the preferences A>D>B>C, B>C>A>D, and C>A>D>B. Then the Dodgson score of A is 1, since switching C and A in the preference C>A>D>B is sufficient to make A the Condorcet winner, and no fewer switches will do so. Similarly, the Dodgson score of B is 2, since at least 2 switches are necessary to beat both A and D, and this can be accomplished by switching B to the top of the preference A>D>B>C. The Dodgson score of C is 1, since C can be made to defeat B by a single switch within the preference B>C>A>D to become a Condorcet winner. Finally, the Dodgson score of D is 4 since at least 3 switches are required to defeat A (one in each preference); and an additional switch is required for D to defeat C. Thus A and C tie for Dodgson winner of this election.

Dodgson described a winner of the election, but did not specify an algorithm to identify a winner. We show that any conceivable algorithm to do this apparently must require excessive time, at least in the worst-case.

In showing that it can be difficult to tell any candidate whether he has won the election, we develop two supporting lemmas. The first and key lemma says that it can be difficult to tell whether a candidate did well. We formalize this question as

**DODGSON SCORE**

**INSTANCE:** Set of candidates C, and a distinguished member c of C; set V of preference orders on C; a positive integer K.

**QUESTION:** Is the Dodgson score of candidate c less than or equal to K?

**LEMMA 1.** DODGSON SCORE is NP-complete.

**Proof.** First observe that DODGSON SCORE is in NP, since a "yes" answer can be justified in polynomial time by identifying appropriate switches and tabulating the vote.

Now we contrive an election for which determining the winner entails solving EXACT COVER BY 3-SETS (X3C), which is known to be NP-complete [7].
EXACT COVER BY 3-SETS (X3C)

INSTANCE: Set B with \(|B| = 3q\) and a collection S of 3-element subsets of B.

QUESTION: Does S contain an exact cover for B, i.e., a subcollection S' of S such that every element of B occurs in exactly one member of S'? 

First we define the set C of candidates: For each element b_i of B, create two corresponding candidates b_i and f_i. Similarly, we create a candidate s_j for each S_j in S.

Now we devise the set V of voters (which are identified with their preferences). V consists of the following subsets:

1. Swing voters. Create voters corresponding to members of S: For each subset S_j = \(\{b_{j1}, b_{j2}, b_{j3}\}\) create a single voter \((b_{j1} > b_{j2} > b_{j3} > s_j > c > ...\)) , where the entries after c are in arbitrary order. We call these voters "swing voters", since their votes will be crucial to the result of the election. Note that switching c up 1 position in such a preference order gains 0 votes against members of B; switching 2 times gains 1 vote; switching 3 times gains 2 votes; switching 4 times, so that c is at the very top of the preference order, gains 3 votes against members of B in pairwise elections. Thus, among swing voters, to get additional votes for c over members of B requires at least 4/3 switches per vote on the average, and to achieve this, any voter who switches at all must switch c upward 4 times, to the very top of his preference order.

The swing voters are the means by which we embed X3C in DODGSON SCORE. However we must pad the electorate with additional voters to make sure that this embedding captures all the difficulty of X3C:

2. Equalizing voters. Let N_i be the number of votes from swing voters that b_i would get in a pairwise election against c; let N_max be the largest N_i. For each b_i create \(N_{\text{max}} - N_i\) additional (identical) voters \((b_i > f_i > c > ...\)) so that each b_i would get exactly \(N_{\text{max}}\) votes in a pairwise election against c. We call these "equalizing voters" since they make all the b_i score equally well against c among swing voters and equalizing voters. Among equalizing voters, to get votes for c over members of B requires at least 2 switches per vote on the average.

3. Incremental voters. Finally create a class of identical voters \((b_1 > ... > b_{|B|} > f_1 > ... > f_{|B|} > c > ...\)) sufficient in number so that any candidate b_i would defeat c by exactly 1 vote. Among incremental voters, to get votes for c over members of B requires at least 2 switches per vote on the average.
Now that we have defined the election, consider whether \( c \) can be made a Condorcet winner by no more than \( 4|B|/3 \) switches. If he can, he must convince the electorate to prefer him to each of the \( b_i \); but this requires at least \( 4|B|/3 \) switches, and is achievable only if i) all switches are among swing voters, and ii) each swing voter makes 4 switches, to move \( c \) to the top of his preferences. But any set of swing voters that can elect \( c \) by making no more than \( 4|B|/3 \) switches corresponds to an exact 3-cover of \( B \) by members of \( S \).

The second supporting lemma says that it can be hard to compare the scores of two candidates, which problem we formalize as

**ELECTION RANKING**

**INSTANCE:** Set of candidates \( C \), and distinguished members \( c, c' \) of \( C \); set \( V \) of preference orders on \( C \).

**QUESTION:** Did \( c \) defeat \( c' \) in the election?

**LEMMA 2.** **DODGSON RANKING** is \( NP \)-hard.

**Proof.** We use the construction from the proof of Lemma 1 with the additional properties that the total number of voters is odd and there exists at least 1 equalizing voter.

To get the total number of voters odd, it suffices to get \( |B| \) odd and \( |S| \) even. If \( |B| \) is not odd, add 3 new artificial base elements, \( a_1, a_2, a_3 \), and add the set \( \{a_1, a_2, a_3\} \) to \( S \). If \( S \) is not now even, choose any member of \( S \) and add another copy of it. If there are no equalizing voters in the election corresponding to this enlarged instance of \( X3C \), then choose any element of \( S \) and add 2 copies of it; this preserves the parity of \( S \), and ensures the existence of at least 1 equalizing voter. These changes to the instance of \( X3C \) are merely cosmetic, since whether there exists a 3-cover remains invariant under these modifications.

Now enlarge the election by adding a new candidate \( c' \) to everyone's preference orders. To do this, first arbitrarily chose some equalizing voter \( v \) to be special. Divide the remaining voters into two arbitrary groups of equal size. All the voters in one group insert \( c' \) at the very top of their preferences; all the voters in the other group...
insert \( c' \) at the very bottom of their preferences. The special voter \( v \) inserts \( c' \) in position \( 1 + 4|B|/3 \) in his preference order.

The Dodgson score of \( c' \) is no more than \( 4|B|/3 \): without the vote of \( v \), \( c' \) must tie every other candidate in pairwise elections, so that making \( 4|B|/3 \) switches to move \( c' \) to the top of \( v \)'s preferences will enable \( c' \) to defeat all the other candidates. On the other hand, the Dodgson score of \( c' \) must be at least \( 4|B|/3 \) since at least that many candidates defeat him in pairwise elections (exactly those candidates preferred to him by \( v \)). Thus the Dodgson score of \( c' \) is exactly \( 4|B|/3 \). The proof now follows analogously to the proof of Lemma 1. Thus DODGSON RANKING is as hard as an NP-complete problem; but since we do not know whether DODGSON RANKING is in NP, we can say only that it is NP-hard.

**THEOREM 1.** DODGSON WINNER is NP-hard.

**Proof.** It is straightforward but tedious to pad the contrived election (from Lemmas 1 and 2) with a polynomial number of additional voters so that \( c \) and \( c' \) (and no others) are tied for first place.

We think Lewis Carroll would have appreciated the idea that a candidate's mandate might have expired before it was ever recognized.

3. Efficiency at the cost of universality. The Dodgson scheme can be made at least formally efficient if we place an *a priori* restriction on the size of the elections in which it will be employed. For example, if we bound in advance the number of voters, then we can determine the Dodgson winner in polynomial time by enumeration: Any particular candidate \( c \) can be permuted to at most \(|C|\) different positions in the preference order of a voter, so there are at most \(|C|^{|V|}\) possible ways of placing \( c \) in the preferences of the electorate. We can count the number of switches implicit in each of these ways that lead \( c \) to be a Condorcet winner, and the fewest number of switches is the Dodgson score of \( c \). Finally we can compare scores and choose the smallest. If \(|V|\) is bounded by a constant, then this procedure is technically polynomial-time (even though this might not be reassuring for large \(|V|\)).
Similarly, if we bound in advance the number of candidates, the Dodgson scheme can be made to run in polynomial time by solving an integer linear program with a very large but fixed number of variables and constraints. The problem of determining the Dodgson score of candidate \( c \) can be formulated as an integer linear program in the following way. Index by \( i \) the types of preference orders found among the voters, and let \( N_i \) be the number of voters of type \( i \). Let \( x_{ij} \) be the number of voters with preferences of type \( i \) for which candidate \( c \) will be moved upwards by \( j \) positions. Let \( e_{ijk} \) be 1 if the result of moving candidate \( c \) by \( j \) positions upward in a preference order of type \( i \) is that \( c \) gains an additional vote against candidate \( k \), and 0 otherwise. Let \( d_k \) be the deficit of \( c \) with respect to candidate \( k \), that is, the minimum number of votes that \( c \) must gain against \( k \) to defeat him in a pairwise election. If \( c \) already defeats \( k \), then \( d_k = 0 \). Then the Dodgson score of \( c \) is the value of integer linear program.

\[
\begin{align*}
\min & \quad \sum_{ij} x_{ij} \\
\text{subject to} & \quad \sum_j x_{ij} = N_i \quad \text{(all types } i \text{ of preference orders)} \\
& \quad \sum_{ij} e_{ijk} x_{ij} \geq d_k \quad \text{(all candidates } k) \\
& \quad x_{ij} \geq 0, \text{ integer.}
\end{align*}
\] (3.1)

The first set of constraints restricts the numbers and types of preferences to those actually present among the voters, and the second set of constraints ensures that \( c \) will become a Condorcet winner. The objective is to minimize the number of switches.

The number of different types of voters (preference orders) is no greater than \(|C|!\), and the number of different positions in any preference order is \(|C|\). Consequently there are no more than \(|C| \times (|C|!)\) variables \( x_{ij} \) and no more than \(|C|! + |C|\) non-trivial constraints. If we limit the applicability of the Dodgson scheme by restricting \(|C|\) to be no larger than some prespecified number, (3.1) is polynomially solvable, at least by the algorithm of Lenstra [12], for which the time bound, though potentially enormous, is technically a polynomial.

The effort required to determine a Dodgson winner appears to increase more quickly as a function of \(|C|\) than as a function of \(|V|\). Even in real elections \(|C|\) can be large enough to make the Dodgson scheme potentially impractical: the *New York Times* of 1 April 1986 reported 20 candidates for mayor of Tulsa, Oklahoma!

4. An "impracticality theorem". Kemeny [10, 11] has suggested another voting
scheme that extends the Condorcet principle. He defined the outcome of an election to be a consensus ranking of the alternatives, and suggested that the consensus be a preference order that minimizes the sum of "distances" to the preferences of the voters. We show that it can be difficult to determine the outcome of an election under Kemeny scoring. As a corollary we conclude that every voting rule that satisfies certain modest fairness criteria must be inefficient at determining a winner.

(We thank the editor and referees for pointing out that the difficulty of scoring a Kemeny election has been independently established by others, including J. Orlin (private correspondance), and, most notably, Wakabayashi [16], who comprehensively analyzed the complexity of median and mean procedures. In addition, the complexity of related problems has been discussed elsewhere [for example, 1, 8].

Kemeny defined the "distance" between two preferences P and P' as dist(P,P') = \sum d(j,k), where the sum is taken over all unordered pairs of candidates j and k, and where d(j,k) = 0 if P and P' agree on candidates j and k; d(j,k) = 2 if P prefers j to k but P' prefers k to j; and d(j,k) = 1 if P prefers j to k but P' is indifferent between j and k. A Kemeny consensus is a preference that minimizes \sum N_j dist(P,P_j), where N_j is the number of voters with preference P_j.

We will need the following technical result, which enables us to consider only strict preferences.

**Lemma 3.** If all voter preferences are strict, then there exists some Kemeny consensus that is strict.

**Proof.** Let P be a Kemeny consensus that includes ties (indifference), and consider any set T of candidates that are mutually tied under P. Let c be any candidate in T and compute A = \sum \{voters who prefer c to d\} and B = \sum \{voters who prefer d to c\}, where the sums are taken over all d in T. If A > B, then the Kemeny score of P could be improved by breaking ties in favor of c; similarly, if A < B, the Kemeny score could be improved by preferring the remaining candidates of T to c. But since P is a Kemeny consensus, its score must be minimum, so that A = B. Finally, since A = B, we can break ties arbitrarily in favor of c to produce a new preference order with fewer ties, but with the same minimum Kemeny score. Repeated application of this produces a Kemeny consensus with no ties.
THEOREM 2. KEMENY SCORE is NP-complete, and KEMENY RANKING and KEMENY WINNER are NP-hard.

Proof. KEMENY SCORE is in NP since the score of any candidate can be computed in polynomial time.

We show the problem is hard by showing that the following problem, which is known to be NP-complete [7], can be polynomially transformed to it.

FEEDBACK ARC SET
INSTANCE: Directed graph G, with vertices C; positive integer K.
QUESTION: Is there a subset of no more than K arcs which includes at least one arc from every cycle in G?

For any instance of FEEDBACK ARC SET, interpret G as representing the outcomes of pairwise contests between candidates C. By [15], G is realizable by a set V of voters such that |V| is even and "small" (polynomial in |C|), whose preferences are all strict, and whose preferences decide each contest by exactly 2 votes. Thus, for any arc (i,j) of G, exactly (|V|+2)/2 voters prefer candidate i to j, and (|V|-2)/2 voters prefer candidate j to i. Therefore any strict preference P must disagree with at least (|V|-2)/2 voters on the relative ranking of candidates i and j, and so must incur a "fixed-cost" of (|V|-2) to its Kemeny score. If P disagrees with the majority of voters and prefers j to i, then there is an additional penalty of (2 voters)(2 points/voter) = 4 points. Since this holds for each of the |V|(|V|-1)/2 pairs of candidates, the Kemeny score of P must be at least |V|(|V|-1)(|V|-2)/2, plus 4 times the number of contests in which P disagreed with the majority.

Now consider the question whether there exists a consensus whose Kemeny score is no larger than |V|(|V|-1)(|V|-2)/2 + 4K. By Lemma 3, the answer is "yes" if-and-only-if there exists a strict preference with the same Kemeny score. By construction, a strict consensus with this score must agree with the majority on all but K of the pairwise contests, and the arcs corresponding to these pairwise contests are a feedback arc set for G. Therefore KEMENY SCORE is NP-complete.

As for the Dodgson scheme, the election can be padded to establish that KEMENY RANKING and KEMENY WINNER are both NP-hard. ■
Now we reword Theorem 2 in a provocative way. Following [17] we define a voting scheme to be neutral if it is symmetric in its treatment of candidates; to be Condorcet if it elects any Condorcet winner; to be consistent if, when two disjoint subsets of the electorate, voting separately, arrive at the same consensus, then their voting together always produces this same consensus. Young and Levenglick [17] proved that Kemeny scoring is the unique voting scheme that is neutral, consistent, and Condorcet. Hence we have the following.

COROLLARY. Under any voting scheme that is neutral, consistent, and Condorcet, the WINNER PROBLEM is NP-hard.

Since only the Kemeny rule satisfies the hypotheses, this corollary is not entirely satisfying. Nevertheless, it can be taken as a model for a new type of impossibility theorem (that might be called an "impracticality theorem"), the general form of which is "Fair elections are impractical". Are there stronger versions of this theorem?

5. Conclusions. Many theorems and some practical experience attest that any conceivable voting scheme is capable of some form of unacceptable behavior, such as violating formalized notions of fairness or rationality [for example: 5, 9, 13; summary in 14]. We remark that the computational complexity of the corresponding election winner problem is also an important aspect one should consider when judging voting schemes.

It seems desirable for a voting scheme to be dependably quick in its decisions. It would be interesting to explore the extent to which this is reconcilable with notions of fairness. In particular, do there exist more potent impracticality theorems than the one we offer?

In addition, there are other computational aspects of voting to be explored. For example, elsewhere we have exhibited a voting scheme that is easy to operate, but is computationally resistant to manipulation [2].

Acknowledgements. The authors thank the editor and three anonymous referees for many helpful suggestions.
THE COMPUTATIONAL DIFFICULTY OF MANIPULATING AN ELECTION

by

John J. Bartholdi, III
Craig A. Tovey
Michael A. Trick

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
404-894-3036

Abstract

We show how computational complexity might protect the integrity of social choice. We exhibit a voting rule that efficiently computes winners but is computationally resistant to strategic manipulation. It is NP-complete for a manipulative voter to determine how to exploit knowledge of the preferences of others. In contrast, many standard voting schemes can be manipulated with only polynomial computational effort.


This research was supported in part by Presidential Young Investigator Awards from the National Science Foundation to the first two authors (ECS-8351313 and ECS-8451032), and by grant N00014-86-K-0173 from the Office of Naval Research.
1. Introduction. We consider a voting scheme to be an algorithm that takes as input a set $C$ of candidates and a set $V$ of preference orders that are strict (irreflexive and antisymmetric), transitive, and complete on $C$. The algorithm outputs a subset of $C$, who are the winners (allowing for ties).

Several celebrated theorems [4, 6, 14] show that any voting scheme that meets certain innocuous-looking rationality criteria must be susceptible to manipulation by strategic voting; that is, a voter with complete knowledge of the preferences of all the other voters can achieve a social choice more to his liking by misrepresenting his preferences.

Like the famous theorem of Arrow, these "impossibility" theorems fascinate because they seem to confound our ideals. They suggest that our methods of social choice must be either unfair or else inherently susceptible to abuse. The contribution of this paper to suggest how such a gloomy prospect can be ameliorated on operational grounds: We exhibit a voting scheme that is computationally resistant to manipulation even though it is easy to compute the winner. While manipulable in principle, this scheme might not be manipulable in practice - even assuming free and perfect information - because of excessive computational requirements. This is different from the idea of [13] that degrees of manipulability be distinguished by their information requirements. Instead, we make distinctions based on the time required to process that information.

First we show that many of the well-known and historically used voting schemes can be "efficiently manipulated" in the sense that only polynomial time is required to determine how to vote so as to exploit knowledge of the preferences of other voters. Secondly, we prove that it is NP-complete to determine how to manipulate a scoring method that is in practical use (by, for example, the International Federation of Chess).

2. Computational complexity. For those not familiar with computational complexity, we provide a quick sketch of issues and terms. We urge the reader to consult [5] for more detail.

An algorithm is considered formally efficient if it requires a number of computational steps that is at most polynomial in the size of the problem. Problems for which there are polynomial-time algorithms are generally considered to be tractable, and those which require exponential time to solve are considered inherently intractable. Practical experience confirms the validity of this distinction.

One way to measure the difficulty of a problem is by the worst-case time to solve
an instance of given size. Based on this measure, complexity theorists have identified a hierarchy of "complexity classes" into which problems might fall. For example, the problem class NP consists of those questions for which a "yes" answer can be justified in polynomial time, and the hardest problems in NP are known as "NP-complete". NP-complete problems are all equivalent in the sense that any problem in NP can be reworded as an instance of an NP-complete problem within polynomial time. Thus if there exists an algorithm to solve any NP-complete problem within guaranteed polynomial time, then all NP-complete problems could be solved by transforming them to instances of the easily solvable problem. Since no algorithm of guaranteed polynomial time has been found for any NP-complete problem, this is widely taken as evidence for their inherent intractability. To show that a problem is NP-complete it is enough to show that it is within the class NP, and that it is "at least as difficult as" some other problem known to be NP-complete. The first point is generally trivial to establish; the latter point can be established by showing how the known hard problem can be quickly converted to a special case of the problem in question. In showing that the two problems are "equivalently difficult", the argument is frequently rather formal, and with few concessions to intuition.

Classifying the complexity of voting problems is a natural refinement of previous work in social choice which has dealt with the distinction between the impossible and the possible [4, 6, 14], and, more recently, between the noncomputable and the computable [7, 9]. Here we distinguish between the intractable and the tractable, and conclude that, even if some electoral problems admit of a computable solution, that solution might be impractical. The solution might be impractical because the time required to compute a solution can increase exponentially in the size of the problem.

3. Many voting schemes are easy to manipulate. Consider a fixed voting scheme, and suppose that a manipulative voter knows in advance the preferences of every other voter; is there a preference ordering (possibly different from his true preferences) that the manipulator can adopt so that a specified candidate c is a winner? If a voter can answer this question, he can determine whether he can manipulate the election. We formalize this question as follows.
EXISTENCE OF A WINNING PREFERENCE

INSTANCE: Set of candidates C, and a distinguished member c of C; set V of transitive preference orders on C.

QUESTION: Does there exist a preference order P that will ensure that c will be the winner?

If this can be answered within polynomial time, then the voting scheme is "easily manipulable". We show that many commonly-used voting schemes are easily manipulable by the following simple procedure.

ALGORITHM GREEDY-MANIPULATION

INPUT: preferences of all other voters, and a distinguished candidate c.

OUTPUT: either a preference order that, together with those of all the other voters, will ensure that c is a winner, or else a claim that no such preference order exists.

INITIALIZATION: Place c at the top of the preference order.

ITERATIVE STEP: Determine whether any candidate can be placed in the next lower position (independent of other choices) without preventing c from winning. If so, place such a candidate in the next position; otherwise terminate claiming that c cannot win.

The following describes voting schemes that are manipulable by GREEDY-MANIPULATION. Assume that the preferences of all the voters but the manipulator are fixed and known to the manipulator. For any preference P and candidates i, j, let iPj mean that i is preferred to j under P, so that (j:iPj) is the set of candidates to whom i is preferred under P.
THEOREM 1: GREEDY-MANIPULATION will find a preference order $P$ that will make candidate $c$ a winner (or conclude that it is impossible) for any voting scheme that can be represented as function $S(P):C \rightarrow \mathbb{R}$ that is both
- "responsive": a candidate with the largest $S(P,i)$ is a winner; and
- "monotone": for any two preference orders $P$ and $P'$ and for any candidate $i$, $(j:iP'j) \subseteq (j:iPj)$ implies that $S(P,i) \geq S(P',i)$.

Proof: First note that if GREEDY-MANIPULATION successfully constructs an order, that order will guarantee victory for candidate $c$. Now we show that if an order exists that ensures victory for $c$, then GREEDY-MANIPULATION will construct it. Suppose GREEDY-MANIPULATION terminates without having constructed a preference order, and let $U$ be the set of unassigned candidates when GREEDY-MANIPULATION terminates. Let $P'$ be an order which would enable $c$ to win, and let $u$ be the highest ranked member of $U$ under $P'$. Consider any completion $P$ of the preference order started by GREEDY-MANIPULATION that places $u$ in the highest unassigned place. By responsiveness, $S(P',c) \geq S(P',u)$, and because $(j:uPj)$ is contained in $(j:uP'j)$, $S(P',u) \geq S(P,u)$. Furthermore, by the initialization of the algorithm and by monotonicity, $S(P,c) \geq S(P',c)$. These inequalities imply $S(P,c) \geq S(P,u)$. But, by the iterative step of GREEDY-MANIPULATION, $S(P,c) < S(P,u)$ since $u$ cannot go in the assigned slot. This is a contradiction.

COROLLARY: Any voting system that satisfies the conditions of Theorem 1, and for which $S$ is evaluable in polynomial time, can be manipulated in polynomial time.

Proof: GREEDY-MANIPULATION executes within polynomial time since no more than $n$ iterations are required, and each iteration requires no more than $n$ evaluations of $S$ (by monotonicity of $S$) with each evaluation of $S$ requiring only polynomial time (by assumption).

Many voting schemes in common use satisfy the conditions of Theorem 1 and so are efficiently manipulable by GREEDY-MANIPULATION. These include:

- **Plurality** (Each voter casts 1 vote for their most preferred candidate). Function: Let $b_i$ be the plurality score of candidate $i$ among all voters except the manipulator; then
S(P,i) = b_i + 1 if |\{j : iPj\}| = |C| - 1, else = b_i.

**Positional (Borda count)** (Each voter casts |C| votes for their most preferred candidate, |C| - 1 votes for their next-most-preferred, ..., and 1 vote for their least preferred candidate.) Function: Let b_i be the positional score of candidate i among all voters except the manipulator; then S(P,i) = b_i + |\{j : iPj\}| + 1.

**Maximin** (A winner is a candidate who maximizes the minimum number of voters who prefer him to another candidate in pairwise elections.) Function: Let \(V_{ij}\) be the voters who prefer i to j; then S(P,i) = \(\min_j (|V_{ij}| + 1 \text{ if } iPj; |V_{ij}| \text{ if } jPj)\).

**Copeland's method** (A winner is a candidate who maximizes the number of victories minus the number of defeats in pairwise elections.) Function: S(P,i) = (number of candidates that i beats in pairwise contests) - (number of candidates to whom i loses in pairwise contests) based on the preferences of all the voters, including the manipulator.

In addition, any monotone increasing function of such functions (which includes lexicographic combinations) still satisfies the hypotheses of Theorem 1. Consequently, baroque scoring methods such as 1/3 times the positional count plus 1/5 times the Copeland count plus 2/7 times the maximin count are still efficiently manipulable by GREEDY-MANIPULATION.

4. A voting scheme that is computationally resistant to manipulation. The Copeland voting scheme ranks the candidates according to the number of pairwise contests they win minus the number they lose [11, 12]. When all candidates are compared against each other pairwise (so that they participate in the same number of contests), this is equivalent to scoring simply by the number of contests won. Many organizations use this method of social choice to rank contestants, and extend it by adding a tie-breaking rule that we refer to the "second-order Copeland scheme": In case of a tie, the winner is the candidate whose defeated competitors have the largest sum of Copeland scores.

(The Federation Internationale Des Echecs and the United States Chess Federation implement tie-breaking rules that are either identical to, or are minor variants of, the second-order Copeland scheme [8, 10]. For example, for round-robin tournaments under USCF rules, the primary score of each player is the number of opponents he has defeated plus one-half the number of opponents he has tied. In case of ties with respect to primary score, a secondary score is computed to be the sum of the primary scores of
all the opponents he has defeated plus one-half the primary scores of all the opponents
with whom he has drawn.)

It requires only polynomial time to compute the winner of an election under either
Copeland or second-order Copeland schemes. Furthermore, by the theorem of Gardenfors
[4] both schemes are manipulable in principle. However, we make an important
distinction here: While a first-order Copeland scheme can be manipulated efficiently (by
Theorem 1), second-order Copeland is computationally resistant to manipulation, as we
will show. Specifically, it is NP-complete to determine whether one can misrepresent
one's preferences to exploit knowledge of the preferences of others. (Note that second-
order Copeland violates the hypothesis of monotonicity required by Theorem 1 for
manipulability by GREEDY-MANIPULATION.) Thus second-order Copeland circumvents
the Gardenfors impossibility theorem on operational grounds: it satisfies the hypotheses—it
is neutral, anonymous, and Condorcet—but it is computationally difficult to
manipulate.

We can easily modify Second Order Copeland to similarly circumvent the Gibbard-
Satterthwaite theorem [6, 14]: simply impose an arbitrary order on the candidates to
break ties in their Second Order Copeland score; then the voting scheme is single-valued
(but no longer neutral). By the Gibbard-Satterthwaite theorem, this modified scheme is
in principle subject to manipulation but by our results it is computationally resistant to
manipulation. More formally, the Gibbard-Satterthwaite theorem states

THEOREM: No social choice function is simultaneously
(1) single-valued;
(2) non-dictatorial;
(3) non-manipulable.

In contrast, we prove the following.

THEOREM: There exists a social choice function (Second Order Copeland) that is
simultaneously
(1) single-valued;
(2) non-dictatorial;
(3) easy to compute, but computationally difficult to manipulate.
Moreover, this social choice function is anonymous, Pareto optimal, and Condorcet.
Intuitively, it is difficult to construct a manipulative preference under Second Order Copeland because it is difficult to know where to place candidates in the preference. For example, placing a favored candidate at the top can unintentionally improve the scores of rivals because of second order effects in the scoring. This forces the manipulator to consider all the exponentially-many possible preference orders.

To capture the way in which second-order Copeland is likely to be used in practice, we exhibit an instance in which a set of candidates is tied under the primary score (Copeland), and the difficulty is to manipulate the tie-breaking score (second-order Copeland). The outline of the argument is as follows. First we show that a logic problem that is known to be hard can be embedded in a problem of scoring tournaments, so the tournament problem is hard. Then we show that the tournament problem can be embedded in EXISTENCE OF A MANIPULATIVE PREFERENCE FOR SECOND-ORDER COPELAND, which must therefore be hard.

A word of caution to the reader: The argument embeds a logic problem in a graph problem, and the graph problem in the voting problem. Since we are showing that these problems are in a sense equivalent, we variously adopt the elementary terminology of logic, graph theory, and voting. Where one field has a concept we need, we occasionally switch terminology mid-argument, rather than burden the reader with new but equivalent definitions.

Now we begin by showing that a problem of tournament scoring is hard. The tournament problem is illustrated by the final round of a round-robin chess tournament. The tournament requires each contestant to play every other contestant, but there is one more round (set of pairwise contests) to be played. The question is whether there exists a set of outcomes for the final round that will guarantee tournament victory for a particular competitor. We formalize this as:

TOURNAMENT OUTCOME

INSTANCE: A complete simple graph (with vertices corresponding to candidates), for which each edge (i,j) can be either directed (corresponding to candidate i having beaten candidate j) or undirected (corresponding to the contest between candidates i and j not having been decided); a distinguished vertex (candidate) c.

QUESTION: Is there a way of assigning directions to the currently undirected edges so that c is the winner?
THEOREM 2: TOURNAMENT OUTCOME UNDER SECOND-ORDER COPELAND is NP-complete.

Proof: The problem is in NP because we can quickly prove a "yes" answer by showing a set of outcomes and computing the second-order Copeland scores. We show the problem is as hard as an NP-complete problem by embedding within it 3,4-SATISFIABILITY, which is known to be NP-complete [16].

3,4-SAT

INSTANCE: An expression consisting of clauses $C_1, ..., C_m$ over variables $X_1, ..., X_n$, with each clause containing exactly 3 different variables, and each variable appearing in exactly 4 clauses.

QUESTION: Is there a satisfying truth assignment for the set of clauses?

Given an instance of 3,4-SAT, construct a instance of TOURNAMENT OUTCOME as follows. Create a set of candidates corresponding to clauses $C_1, ..., C_m$ and to literals $X_1, \overline{X}_1, ..., X_n, \overline{X}_n$ (where $\overline{X}$ is the negation of $X$). In addition, create a distinguished candidate $c$, whose victory is in question. All pairwise contests have been decided except those between the pairs of literal candidates $X_j$ and $\overline{X}_j$ ($j = 1..n$). In particular, each "clause candidate" $C_i$ ($i = 1..m$) defeated the 3 "literal candidates" corresponding to the literals in the clause, and lost to all other candidates.

Model the candidates and their contests as a graph in which there is a vertex for each candidate. If candidate $i$ has defeated candidate $j$, there is an edge directed from $i$ to $j$; if the contest between candidates $i$ and $j$ has not yet been decided, connect $i$ and $j$ by an undirected edge. Consequently, in $G$ each edge $(X_j, \overline{X}_j)$ is undirected, and all other edges are directed from winner to loser. (See Figure 1.) Let $R$ be the set of undirected edges corresponding to remaining pairwise contests.

For the candidate corresponding to vertex $v$, his tentative scores $TS(v)$ and $TS^2(v)$ are his Copeland score and second-order Copeland score, respectively, not counting possible points from contests that have not yet been decided. Let $S(v,R)$ and $S^2(v,R)$ denote the eventual Copeland and second-order Copeland scores of $v$, depending on the outcomes of the remaining contests of $R$. Note that, for all $C_i$, $S(C_i,R) = TS(C_i)$ is independent of $R$.

In Appendix 1, we show how to pad graph $G$ with additional candidates and assign outcomes to their pairwise contests so that the following properties hold:
Property 1: For any specification of outcomes for \( R \), the candidates \( c \) and all \( C_i \) will be tied for first place with respect to (first-order) Copeland score; that is, \( S(c,R) = S(C_1,R) = \ldots = S(C_m,R) > S(v,R) \) for all other \( v \).

Property 2: The second-order Copeland score of the distinguished candidate \( c \) is independent of \( R \). (Accordingly, we abbreviate \( S^2(c,R) \) as \( S^2(c) \).)

Property 3: Each candidate \( C_i \) (\( i = 1..m \)) has \( TS^2(C_i) = S^2(c) - 3 \).

Property 4: Each \( C_i \) (\( i = 1..m \)) defeated the 3 candidates corresponding to the literals that clause \( C_i \) contains in the instance of 3,4-SAT, and lost to all other candidates.

Property 5: For all outcomes \( R \), \( S^2(c) > S^2(v,R) \) for every \( v \) other than a clause candidate.

Now we claim that if \( G \) has properties 2-5, then there is a set of outcomes for the remaining contests \( R \) which will make \( c \) the unique winner if-and-only-if the instance of 3,4-SAT is satisfiable. To see this, imagine candidate \( C_i \) (\( i = 1..m \)) during the last round of contests. His own contests are over, so his Copeland score has been determined, and (by property 3) he is currently 3 points short of a share of first place with \( c \) and possibly other \( C_j \). By property 4, \( C_i \) has lost to all but 3 of the literals, so the outcomes of only 3 contests (those containing the literals defeated by \( C_i \)) could improve his second-order score. If all 3 contests go as \( C_i \) wishes (the candidates that \( C_i \) defeated win their contests), then, by properties 2 and 3, \( C_i \)'s second-order score will equal that of \( c \), and by property 5 \( C_i \) will own a share of first place; otherwise, \( c \) will beat \( C_i \). Interpreting a literal losing to its complement as the literal being set to TRUE in the instance of 3,4-SAT, candidate \( C_i \) will lose to \( c \) if-and-only-if clause \( C_i \) is satisfied. Thus satisfiability of the 3,4-SAT expression corresponds precisely to all of the \( C_i \)'s (\( i = 1..m \)) being defeated by \( c \).

COROLLARY: **TOURNAMENT MANIPULATION under Copeland scoring with second-order Copeland tie-breaking is NP-complete.**

Proof. This follows from property 1 of the proof of Theorem 2.

Remark: This means that in a chess tournament it can be difficult for a team to play
strategically when the tournament will be decided on tie-breaks. Furthermore, note that in the proof of Theorem 2, the undecided contests were candidate-disjoint; that is, no candidate was involved in more than one undecided contest. This means that it can be hard to manipulate even the final round of a tournament. We will require this fact in the proof of the next theorem, but first we need a technical lemma.

Let \( G \) be a complete graph on \( n \) vertices, where each edge may be directed or undirected. We show that there exists a set of \( n(n - 1) \) voters whose preferences produce the graph \( G \) of outcomes of pairwise contests between \( n \) candidates (where undirected edges signify ties). This set of voters also decide every non-tied contest by exactly 2 votes.

It is known that any complete directed graph of outcomes is realizable by a small number of voters [15]); our result differs only in allowing \( G \) to contain undirected edges representing ties, and in stipulating that the margin of victory be at least 2. All that is needed for our purposes is that \( G \) can be realized with a number of voters that is a polynomial in \( n \).

Our argument depends on a theorem of elementary graph theory that the edges of \( K_n \), the complete undirected graph on \( n \) vertices, can be partitioned into \((n - 1)/2\) Hamiltonian cycles if \( n \) is odd [see, for example, p. 13 of 3].

**Lemma:** Any set of outcomes of (simple majority) pairwise contests between \( n \) candidates (including ties) is realizable with \( n(n - 1) \) voters in such a way that every non-tied contest is decided by 2 votes.

**Proof:** First assume \( n \) is odd and let \( \{H_i\} \) be a set of \((n - 1)/2\) edge-disjoint Hamiltonian paths on \( K_n \). For each \( H_i \) create 2 sets of voters, \( V_i \) and \( W_i \), as follows. Arbitrarily fix the vertex sequence of \( H_i \) to be \( c_1, c_2, ..., c_n \), and define voter \( V_{ij} \) to have preferences \( (c_j > c_{j+1} > ... > c_{j-1}) \) (where the subscripts are understood to "wrap around" so that the successor of \( n \) is 1). The preferences of each \( W_{ij} \) are exactly the opposite of \( V_{ij} \). Thus we have created \( 2n \) voters corresponding to each of \((n - 1)/2\) Hamiltonian paths, for a total of \( n(n - 1) \) voters. Furthermore, by symmetry, for every voter that prefers \( c_j \) to \( c_k \), there exists an opposite voter who prefers \( c_j \) to \( c_k \), so that every contest is tied.

Now consider any complete graph \( G \) on these \( n \) vertices, with each edge either directed or undirected. We will adjust the preferences of the voters to produce \( G \). For each pair of candidates \( c_j \) and \( c_k \), examine the edge joining \( c_j \) and \( c_k \) in \( G \): if it is
undirected, do nothing. Otherwise, assume the edge is directed from $c_j$ to $c_k$. Since $c_j$ and $c_k$ must be adjacent in some Hamiltonian path $H_i$, there must be a voter whose preference order is ($\ldots > c_k > c_j$); that is, $c_j$ follows $c_k$ at the very bottom of the preference order. Interchange $c_k$ and $c_j$ in this voter's preferences, so that now $c_j$ defeats $c_k$ in a pairwise contest by 2 votes. Since each interchange is between adjacent positions in a preference order, there can be no disruption of the results of other contests.

If $n$ is even, add a dummy vertex (candidate) to $G$, with edges directed arbitrarily. Since the number of candidates is now odd, by the argument immediately preceding, there exists a set of voters for which the graph is realizable. Delete the dummy candidate from the preferences of these voters.

**THEOREM 3: EXISTENCE OF A WINNING PREFERENCE FOR SECOND-ORDER COPELAND IS NP-COMPLETE.**

*Proof:* The problem is in NP since the outcome of the election can be computed in polynomial time.

We prove difficulty by showing that the problem can contain the tournament manipulation problem just shown to be NP-complete. First recall that any complete graph of outcomes (including undirected edges to represent ties) is realizable by a small number of voters (where "small" means polynomial in the number of candidates) so that every non-tied contest is decided by 2 votes [15]. Apply this to the graph of outcomes of pairwise contests from the construction of Theorem 2, where we interpret the undirected edges as indicating a tie between the corresponding candidates in a simple vote count. Let $V$ be a set of voters that realizes this graph. Assume that the manipulator knows these preferences, and now wonders whether there exists a preference he can claim that will make $c$ a winner.

Since all contests corresponding to directed edges have been decided by 2 votes, the manipulator cannot affect these outcomes; he can affect only the contests that are currently tied. As we have previously observed, those contests are candidate-disjoint, so the manipulator can vote to achieve any of the $2^R$ possible sets of outcomes by simply ranking the candidates of each pair in desired relative order. However, to determine whether any of these sets of outcomes will make the distinguished candidate $c$ a winner, the manipulator must solve the difficult instance of TOURNAMENT OUTCOME contrived
in the previous theorem.

COROLLARY: \textit{EXISTENCE OF A WINNING PREFERENCE FOR FIRST-ORDER COPELAND WITH SECOND-ORDER COPELAND TIE BREAKS} is NP-complete. \hfill $\blacksquare$

\textit{Proof:} This follows from Theorem 3 and the corollary to Theorem 2. \hfill $\blacksquare$

We can now establish the main result of this section, that it is computationally difficult to manipulate Second Order Copeland. Intuitively, this follows from Theorem 3: since it is difficult to determine a preference that ensures the election of a particular candidate, it is difficult to manipulate. More formally:

\textbf{THEOREM 4: MANIPULATION OF SECOND-ORDER COPELAND} is NP-complete.

\textit{Proof:} In the construction of Theorem 3, let the social choice function break ties in the order of preference $C_1$, $C_2$, $\ldots$, $C_m$, $c$, $\ldots$. Let the three "literal" candidates defeated by $C_1$ (whose existence is guaranteed by Property 4) be denoted $L_1$, $L_2$, and $L_3$. Suppose now that the sincere preferences of the manipulator are $c$, $C_1$, $L_1$, $L_2$, $L_3$, $\ldots$. Then if the manipulator votes sincerely, Properties 2-5 and the tie-breaking rule result in $C_1$ the social choice. Since the manipulator prefers only $c$ to $C_1$, there is only one way to manipulate: to vote insincerely to make $c$ the winner. But to recognize when this is possible is NP-complete by Theorem 3. \hfill $\blacksquare$

COROLLARY: \textit{MANIPULATION OF FIRST-ORDER COPELAND WITH SECOND-ORDER COPELAND TIE BREAKS} is NP-complete.

Finally we observe that second-order Copeland can be manipulated with (at least formal) efficiency if we restrict its use to elections in which there are not "too many" candidates. In fact this is true of any voting scheme for which the winner can be efficiently computed. The manipulator could in principle compute the outcome of the election for each of the $|C|!$ possible preference orders. If each evaluation requires only polynomial time, and if $|C|$ is restricted so that $|C|! = O(p(|V|))$ for some polynomial $p$ of fixed degree, then the total effort is, strictly speaking, polynomial.
5. Conclusions. Computational complexity is a new criterion by which to evaluate methods of social choice. Worst-case behavior in this regard might be a practical consideration for some decision methods (see, for example, [1, 2]), just as is worst-case behavior with respect to formalized notions of fairness.

Methods of social choice should be easy to use but hard to abuse; that is, they should identify winners within polynomial time, but they should also be provably difficult to exploit. We have shown that Second Order Copeland satisfies these properties. Moreover, Second Order Copeland is a reasonable voting scheme in that it satisfies many appealing rationality criteria, including unanimity (the Pareto principle), neutrality, anonymity, and Condorcet-winner.

These issues are similar to those in cryptography, where both encryption and authorized deciphering should be easy, but unauthorized deciphering should be difficult. As for a cryptographic scheme, we would prefer to know that a voting scheme is dependably hard to abuse, rather than merely hard in the worst-case, as we have proved. Unfortunately, this sort of result seems beyond the reach of current complexity theory. However, we can appeal to practical experience, which confirms that NP-hard problems are difficult to solve.

There are several concerns that might be raised regarding our interpretation of these results. We state and discuss them below.

Concern: Complexity is not an issue because manipulation is not an issue: complete information about preferences is never available.

Discussion: Since we are trying to protect the mechanism of social choice, it is best to make the most conservative assumptions.

Concern: It might be that very few among all possible instances of an election are actually manipulable.

Discussion: True. Unfortunately, both the Gibbard-Satterthwaite and the Gardenfors theorems are worst-case theorems; that is, they claim that there exist instances of manipulable elections, but do not say how many. In the same way, our theorem is a worst-case theorem; it claims that some instances among the manipulable elections are hard to manipulate, but does not say how many. Thus our theorem is weak exactly where the Gibbard-Satterthwaite and Gardenfors theorems are weak.

Concern: It might be that very few among all real instances of an election are
actually manipulable.

Discussion: True. Perhaps real elections have sufficiently special structure that makes them dependably easy to manipulate. For example it might be that real elections are small enough that exponentially-increasing work is not a deterrent to manipulation. It also might be that people's preferences are specially structured so that it is not hard to recognize when manipulation is possible. These are empirical questions that must be tested by experiment.

Concern: It might be that there are effective heuristics to manipulate an election even though manipulation is NP-complete.

Discussion: True. The existence of effective heuristics would weaken any practical import of our idea. It would be very interesting to find such heuristics.

Concern: These results do not "ameliorate" the Gibbard-Satterthwaite or Gardenfors theorems because the real meaning of these theorems is something else: that since there is an incentive to manipulate, researchers must consider game-theoretic issues.

Discussion: Game-theoretic issues are certainly important, but our result weakens this justification for studying them.

ACKNOWLEDGEMENTS: The authors appreciate many helpful comments and suggestions by the editor and three anonymous referees. We also thank Michel Balinski, Salvador Barbera, Jean-Pierre Barthelemy, and Peyton Young for stimulating discussions.
REFERENCES


APPENDIX 1

We show how, given an instance of 3,4-SAT, to construct in polynomial time a graph satisfying properties 1-5. Here is an overview of the construction of the graph: G will contain three main types of vertices: those corresponding to clauses in the instance of 3,4-SAT; those corresponding to literals in the instance of 3,4-SAT; and "fillers", which pad the graph to make the scores suit our purpose. We will construct G so that all scores within each set are equal, or nearly so, but in pairwise contests clause candidates defeat fillers, fillers defeat literals, and literals defeat clauses. Finally, in addition to clause, literal, and filler vertices, there will be two distinguished vertices. One is $C_0$, corresponding to the contestant whose victory is in question, and the other is $b$, a "balancing" vertex that will enable us to adjust the second-order Copeland score of $C_0$.

Let $m$ be the number of clauses and $n$ the number of variables in the instance of 3,4-SAT. Note that $3m = 4n$, so $m$ is even and $m/2$ is an integer. Now define the vertices of G correspond to

- clauses: $C_i$ ($i = 1..m$);
- literals: $X_j, \bar{X_j}$ ($j = 1..n$);
- fillers: $f_k$ ($k = 1..30m$);
- the contestant whose victory is in question: $C_0$;
- the balancing contestant: $b$.

Recall that G is a complete graph, so there is an edge between every pair of vertices. Each edge is directed according to the outcome of the pairwise contest between that pair of contestants, from the winner to the loser. Some contests will be as yet undecided, and so will be represented by undirected edges.

Orient the edges of G according to the following outcomes:

- contests between clauses and literals: Each literal defeats all clauses except the clauses containing it. Thus each literal defeats between $m - 4$ and $m$ clauses.

- contests between clauses and $C_0$: Each of the $C_i$ ($i = 0..m$) wins $m/2$ and loses $m/2$ of these contests. This can be arranged by imagining the $m + 1$ contestants $C_i$ seated at every other chair around a symmetric $2(m + 1)$-seat round table. Each $C_i$ sits diametrically opposite an empty chair. Let each $C_i$ defeat the $m/2$ people on his right, and lose to the $m/2$ people on his left.
- contests between literals: Each literal defeats exactly $n - 1$ literals and loses to exactly $n - 1$ other literals. This can be arranged by imagining the $2n$ literals seated at a round table with $2n$ seats, with each $X_j$ seated diametrically opposite $\neg X_j$. Let each literal defeat the first $n - 1$ literals on his right (up to $\neg X_j$), and lose to the first $n - 1$ literals on his left (up to $\neg X_j$). The contest between each $X_j$ and $\neg X_j$ has not yet been decided and so the corresponding edge is undirected.

- contests between fillers: Each $f_k$ ($k$ odd) defeats $15m - 1$ other $f_k$ ($k = 1..30m$). Each $f_k$ ($k$ even) defeats $15m$ other $f_k$ ($k = 1..30m$). (It is this slight difference that allows us to achieve property 3.) This can be arranged by imagining all the fillers seated around a $30m$-seat round table in the sequence $f_1, f_3, ..., f_{30m-1}, f_2, f_4, ..., f_{30m}$. Each $f_k$ defeats the $15m - 1$ fillers to his right, and loses to the $15m - 1$ fillers to his left. Contests between fillers seated diametrically opposite each other are always between a filler of even index and a filler of odd index, and are won by the filler of even index.

- contests between literals and fillers: Every filler defeats every literal;
- contests between $C_0$ and literals: $C_0$ defeats $X_1$ and $\neg X_1$, but loses to $X_j$ and $\neg X_j$ for $j = 2..n$. (This gives property 2.)
- contest between $C_0$ and $b$: $C_0$ defeats $b$;
- contests between $b$ and clauses: $b$ defeats all clauses;
- contests between $C_0$, clauses, and fillers: Define $f_{26i+1}, f_{26i+2}, ..., f_{26i+26}$ to be associates of $C_i$ ($i = 0..m$). Thus each $C_i$ has 26 associates, no filler is an associate of more than one $C_i$, and nearly $4m$ fillers are not associates.

Now let $L(C_i)$ be the total number (counting repetitions) of clauses of which the literals in $C_i$ are members, plus 1 if $C_i$ contains $X_1$ or $\neg X_1$. (For example, in the expression "($X_1$ or $X_2$ or $X_3$) and ($X_2$ or $X_3$ or $X_4$)", $L(C_1) = 1 + 2 + 2 = 5$. We count $C_0$ as containing $X_1$ and $\neg X_1$ so $L(C_0) = 4 + 2 = 6$.) Since no literal appears in more than 4 clauses, and $X_1$ and $\neg X_1$ do not appear in the same clause, we have that the number of literals in $C_i = 3 \leq L(C_i) \leq (3)(4) + 1 = 13$. $L(C_i)$ is the number of contests won by one of $C_0, C_1, ..., C_m$ against the three literals that comprise $C_i$. Thus $C_i$ defeats all $f_k$ which are either not an associate of $C_i$; or for which $k = 26i + 2, 26i + 4, ..., 26i + 2L(C_i)$; or for which $k = 26i + 1, 26i + 3, ..., 26i + 2(13 - L(C_i)) - 1$. In other words, $C_i$ defeats all of its non-associates, and 13 of its 26 associates ($L(C_i)$ of the 13 defeated associates being fillers of even index).

- contests between $b$ and $f_k$: $b$ loses to all but the last $n + 4$ fillers; that is, $b$ loses to $f_k$ for $k = 1, 2, ..., 27m, ..., 30m - n - 4$, but defeats $f_k$ for $k = 30m - n - 3$.
Verifying that properties 1-5 are satisfied consists entirely of arithmetic computation. First, we compute the first-order Copeland scores of some of the candidates:

(1) \( S(C_i) \) (i=1..m) = \( \frac{m}{2} + 3 + 0 + (30m - 13) \) = \( S(C_0) = \frac{m}{2} + 2 + 1 + (30m-13) \), where the terms are due to victories over contestants from \( C_0..C_m \), literals, \( B \), and fillers, respectively.

(2) \( S(B) = m + 0 + 0 + (n + 4) = m + N + 4 \), where the terms are due to victories over contestants from \( C_1..C_m \), \( C_0 \), literals, and fillers, respectively.

We do not know the Copeland scores of the literals, but we can compute their tentative scores. For any literal \( X_i \),

\[
TS(X_i) = \begin{cases} 
1 & \text{if } i > 1; \\
0 & \text{if } i = 1; \\
m - \text{(# of clauses in which } X_i \text{ appears)} + (n-1) + 0 + 1, & \text{where the terms are due to victories over } C_0, \text{ clauses, literals, fillers, and } B, \text{ respectively.}
\end{cases}
\]

\( S(X_i,R) = TS(X_i) \) or \( TS(X_i) + 1 \). Thus.

(3) \( TS(X_i) = m + n + 1 - \{\text{the contribution of } X_i \text{ to } L(\)} \)

(4) \( ..If \ F_i \text{ is an associate, } S(F_i) = (15m-1) + \{(1, \text{if } i \text{ is even, or } 0) + 1 \cdot 2n + (1 \text{ if it beats its associate)} \), \text{ where the terms are due to victories over fillers, } B, \text{ and literals, respectively.}

Now we verify properties 1-5.

Property 1: The total number of nodes in \( G \) is \( 1 + m + 2n + 30m + 1 = 65m/2 + 2; \) \( B \) loses to at least \( 26m \) fillers; each filler loses to at least \( 14m \) fillers; each literal loses to at least \( 30m \) fillers; hence for all \( v \) not a \( C_i \), \( S(v) \leq 32(m/2) + 2 - 14m < S(C_0) \), which verifies property 1.

Property 2: Since for all \( i, C_0 \) either beats both \( X_i \) and \( \overline{X_i} \) or loses to both these
literals, it follows that $S^2(C_0,R)$ is independent of $R$. This is property 2.

**Property 3:** We compare the second order Copeland scores of the $C_i$ and verify that for any $i$, $1 \leq i < m$, $TS^2(C_i) = TS^2(C_{i+1})$. Define $TS^2(C_i) - TS^2(C_{i+1}) = Diff_C + Diff_B + Diff_L + Diff_F$, where the "Diff" terms are the differences in the contributions from the different classes of nodes, C, B, Literal, Filler. (Note: C includes $C_0$ as well as all the clause nodes.)

$Diff_C = 0$ since $C_i$ and $C_{i+1}$ both beat $m/2$ in class $C$ and by (1) all class $C$ members have identical value of $S()$.

$Diff_B = 0$ since they both beat B.

$Diff_L = 3(m+n+1) - L(C_i) - [3(m+n+1) - L(C_{i+1})] = L(C_{i+1}) - L(C_i)$

$Diff_F = 0$ from non-associate fillers, which they both beat

$+ \{|\text{evens } C_i \text{ beats} \} - \{|\text{evens } C_{i+1} \text{ beats} \}$ by (4)

$+ 13$ for $C_i$ beating 13 of $C_{i+1}$'s associates that beat it

$- 13$ for $C_{i+1}$ beating 13 of $C_i$'s associates that beat it by (4)

$= L(C_i) - L(C_{i+1})$

Thus all the Diff terms cancel out and $TS^2(C_i)$ equals $TS^2(C_{i+1})$. This proves half of property 3, that all the clause nodes have the same $TS^2$ score.

Now compare $TS^2(C_1)$ with $S^2(C_0)$. As before let $S^2(C_0) - TS^2(C_1) = Diff_C + Diff_B + Diff_L + Diff_F$.

$Diff_C = 0$ as before;

$Diff_B = m + (n+4)$ by (2);

$Diff_L = 2(n+m+1) - (\text{contribution of } X_1 \text{ to } L() + \text{contribution of } \neg X_1 \text{ to } L()) - 3(n+m+1) + L(C_1)$ by (3).

$= (n+m+1) + L(C_1) - L(C_0)$;

$Diff_F = L(C_0) - L(C_1)$ as before.

Thus $Diff_F$ is cancelled out by part of $Diff_L$, and the sum of the differences equals $m + (n+4) - (n+m+1) = 3$. (Now it is clear why $b$ beats $(n+4)$ fillers.) This verifies property 3.

**Property 4:** The construction of the edges between $C_i$ and the literals gives property 4.
Property 5: To verify property 5 we need only a rough estimate of the second order Copeland scores. By (4), for any $C_j$, the portion of the second-order score coming from the fillers is about $30m(15m+2n) > 480m^2$. Since the total number of nodes is only $65m/2 + 2$, this is not a bad estimate. Moreover, as shown in the verification of property 1, every other node loses to at least $14m$ fillers. Hence the filler portion of other second order scores will be smaller by at least $14m(15m+2n) > 225m^2$. This is too large a deficit to overcome by contributions from the remaining $2(m/2)$ nodes, which cannot possibly contribute more than $2(m/2)(32m/2) < 81m^2$. Hence, every other node has smaller second order Copeland score than the $C_j$ (smaller by more than $100m^2$). This verifies property 5.

Thus the election we have constructed satisfies the properties claimed.
FIGURE 1: Essential structure of the graph corresponding to an instance of 3,4-SAT. The undirected edges correspond to contests that have not yet been decided. Each $C_i$ has an edge directed toward each of the 3 literals contained in clause $C_i$. (For clarity most of the directed edges are omitted, as are the candidates with which the graph is padded.)
FIGURE 1: Essential structure of the graph corresponding to an instance of 3,4-SAT. The undirected edges correspond to contests that have not yet been decided. Each $C_i$ has an edge directed toward each of the 3 literals contained in clause $C_i$. (For clarity most of the directed edges are omitted, as are the candidates with which the graph is padded.)
PART I - PROJECT IDENTIFICATION INFORMATION

1. Program Official/Org. - Thom J. Hodgson - DDM

2. Program Name - OPERATIONS RESEARCH & PRODUCTION SYSTEMS

3. Award Dates (MM/YY) - From: 07/85 To: 12/90

4. Institution and Address -
   Georgia Tech Research Corp
   Administration Building
   Atlanta - GA - 30332

5. Award Number - 8451032

6. Project Title - Presidential Young Investigator: Computational Complexity and Rescheduling Algorithms

This Packet Contains
NSF Form 98A
And 1 Return Envelope
The project centered on two topics: the development and analysis of heuristics and algorithms for scheduling and related optimization problems; and the interdisciplinary application of computational complexity and related techniques. For the first topic, findings include effective heuristic methods and improved complexity classification for scheduling and routing problems in satellite data processing, circuit card assembly, classroom assignment, and multiprocessing, as well as contributions to the development and analysis of generic techniques: simulated annealing, local improvement, linear programming, augmenting paths, and dynamic programming on recursive graphs. For the second topic, two areas were studied: social choice, and honey bee foraging. In social choice, principal results include construction of a voting procedure that employs computational complexity to thwart manipulation; convergence results and polynomial algorithms for the yolk, minimax set, and other solution concepts. The asymptotic consistency of sample estimators and associated computational methods help form the basic tools needed for empirical study of social choice. The study of honey bees produced a model predicting the pattern and quality of forager allocation among flower patches. Preliminary experimental data support the model.
Publications Resulting from this Award
References


3


The data requested below are important for the development of a statistical profile on the personnel supported by Federal grants. The information on this part is solicited in response to Public Law 99-383 and 42 USC 1885C. All information provided will be treated as confidential and will be safeguarded in accordance with the provisions of the Privacy Act of 1974. You should submit a single copy of this part with each final project report. However, submission of the requested information is not mandatory and is not a precondition of future award(s). Check the “Decline to Provide Information” box below if you do not wish to provide the information.

Please enter the numbers of individuals supported under this grant.
Do not enter information for individuals working less than 40 hours in any calendar year.

<table>
<thead>
<tr>
<th></th>
<th>Senior Staff</th>
<th>Post-Doctorals</th>
<th>Graduate Students</th>
<th>Under-Graduates</th>
<th>Other Participants¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Total, U.S. Citizens</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B. Total, Permanent Residents</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U.S. Citizens or Permanent Residents²:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>American Indian or Alaskan Native</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asian</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black, Not of Hispanic Origin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hispanic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pacific Islander</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>White, Not of Hispanic Origin</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>C. Total, Other Non-U.S. Citizens</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify Country</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. PR China</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. ISRAEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. Total, All participants (A + B + C)</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Disabled³

³Category includes, for example, college and precollege teachers, conference and workshop participants.
²Use the category that best describes the ethnic/racial status for all U.S. Citizens and Non-citizens with Permanent Residency. (If more than one category applies, use the one category that most closely reflects the person's recognition in the community.)
³A person having a physical or mental impairment that substantially limits one or more major life activities; who has a record of such impairment; or who is regarded as having such impairment. (Disabled individuals also should be counted under the appropriate ethnic/racial group unless they are classified as "Other Non-U.S. Citizens.")

AMERICAN INDIAN OR ALASKAN NATIVE: A person having origins in any of the original peoples of North America, and who maintain cultural identification through tribal affiliation or community recognition.

ASIAN: A person having origins in any of the original peoples of East Asia, Southeast Asia and the Indian subcontinent. This area includes, for example, China, India, Indonesia, Japan, Korea and Vietnam.

BLACK, NOT OF HISPANIC ORIGIN: A person having origins in any of the black racial groups of Africa.

HISPANIC: A person of Mexican, Puerto Rican, Cuban, Central or South American or other Spanish culture or origin, regardless of race.

PACIFIC ISLANDER: A person having origins in any of the original peoples of Hawaii; the U.S. Pacific Territories of Guam, American Samoa, or the Northern Marianas; the U.S. Trust Territory of Palau; the islands of Micronesia or Melanesia; or the Philippines.

WHITE, NOT OF HISPANIC ORIGIN: A person having origins in any of the original peoples of Europe, North Africa, or the Middle East.