SOPHISTICATED VOTING RULES: THE CASE OF TWO TOURNAMENTS

SANJAY SRIVASTAVA AND MICHAEL A. TRICK

June, 1993

ABSTRACT. We characterize when a voting rule defined over two tournaments can be represented by sophisticated voting on a binary tree. This characterization has a particularly simple and intuitive specialization to the case that the two tournaments differ in exactly one place. These characterizations allow us to demonstrate the tree that implements a particular voting rule, and to show the minimality of an example of Moulin that shows that the Copeland rule cannot be implemented on a tree.

1. INTRODUCTION

A longstanding open problem in the theory of social choice is the characterization of *sophisticated* voting rules. To motivate the problem, consider a set C of *candidates*, where members of C can be interpreted as candidates in an election. Let P_i denote the preference ordering of voter i over the candidates. A *sophisticated voting tree* is a binary tree where at each node of the tree, voters choose between the two candidates who have survived the process to that node. For example, in the tree in Figure 1, at node 2 voters choose between candidates a and b, at node 3 they choose between candidates c and d, while at node 1 they choose between the winners at 2 and 3.

Suppose that at each node, the winner is decided by majority vote[1]. Given a preference profile $P = (P_n,...,P_n)$, let $f(P)$ be the candidate who wins at P. Then,

Authors' address: Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA. 15213. The second author was supported by the Office of Naval Research, Grant N00014–92–J–1387.

[1]For simplicity, assume that preferences are strict and that there are an odd number of voters.

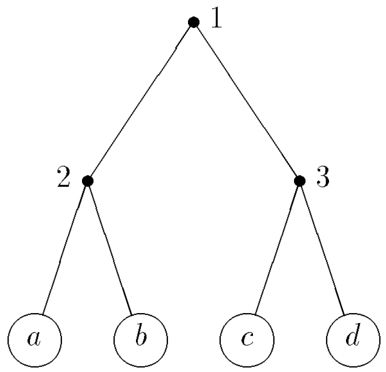

FIGURE 1. Sample Voting Tree

f defines a sophisticated voting rule; for every P, it tells us who will win as long as the election is held according to the voting tree depicted above. Alternatively, we say that f is *implemented* by the tree.

More generally, rule f is a *sophisticated* voting rule if there exists a voting tree whose outcome at P is $f(P)$ (see Farquharson [3]). Of course, different voting trees can, and will, implement different voting rules. What we seek to discover is the set of sophisticated voting rules.

Several papers in the literature have studied properties of sophisticated voting rules. In terms of general characterizations, McKelvey and Niemi [5] show that any onto voting rule must choose from the top cycle[2] and recognize that such a restriction is not sufficient. Moulin [7] gives a further restriction on the voting rule, called adjacency, which holds for pairs of preference profiles, and shows that the Copeland winner is not implementable if there are at least eight candidates. Herrero and Srivastava [4] show that the top cycle condition is sufficient when the number of candidates is three or less. A second branch of the literature has explored the

[2]Given preferences P, the top cycle is the smallest subset of C with the property that every candidate in the subset beats every candidate outside the subset in a majority vote

relationship between the structure of a voting tree and the rule it implements. Miller [6] showed that a specific voting tree, called *amendment voting* leads to outcomes in the *uncovered set*[3] of every tournament. Banks [1] provided a finer characterization of amendment voting showing that the outcomes lie in a subset of the uncovered set now known as the Banks set. A summary of these and related papers can be found in Moulin [7].

Despite these results, a full characterization of such rules has proved elusive. In this paper, we move toward this goal by completely characterizing rules that can be implemented over pairs of preferences. We also point out encouraging results that suggest that pairwise implementation might be sufficient for a broader domain of preferences, at least for voting rules defined over all preferences.

Rather than work with individual preferences, we adopt the approach common in this literature of modeling voting behavior at every node by a *tournament*. A tournament is an irreflexive binary relationship, and in our case, it summarizes the outcome of the voting at every node of a tree. For example, suppose individual preferences over three candidates are given by the following table:

Voter 1	Voter 2	Voter 3
a	b	c
b	c	a
c	a	b

Then, the majority tournament is shown in Figure 2.

Given any pair of candidates, the tournament tells us the outcome of majority voting between the two candidates. Different tournaments would then represent

[3]The uncovered set is a subest U of the top cycle such that for every $x \in U$, $y \in C$, either xTy or there exists z such that $xTzTy$.

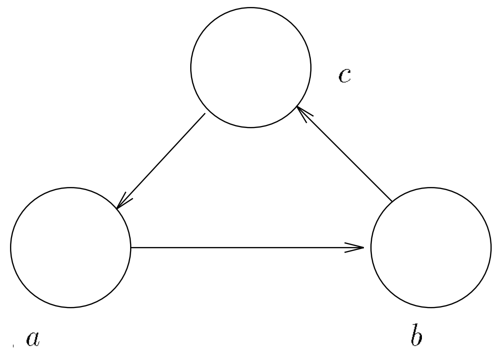

FIGURE 2. Majority Tournament

different preferences. Of course, majority voting is not the only voting procedure that could be employed. Therefore, rather than model the voting procedure, we work directly with tournaments.

Formally, we denote by $\mathcal{T}$ the set of all tournaments over the set C. A voting rule is a function $f : T \to C$. f is a *sophisticated* voting rule if there exists a (finite) voting tree such that when the tournament T is applied at every node, $f(T)$ wins.

In the next section, we present our main result. We illustrate its use by showing that Moulin's counterexample to the implementability of the Copeland rule is minimal. We also present an interesting example of a voting rule that is implementable over an unrestricted domain of tournaments; in doing so, we rely heavily on our characterization across pairs of tournaments to dramatically reduce the size of the problem. In Sections 3 and 4, we show that in some special cases, the characterization is particularly simple and appealing. For instance, Moulin's adjacency condition is necessary and sufficient for any voting rule defined over two tournaments that differ in exactly one place. We show that a generalization of adjacency suffices as long as the number of differing places is not too many. Some extensions to the the case of more than two tournaments are given in Section 5. There, we show how our results can be used to show that implementability across pairs of tournaments is not sufficient for implementation; this disproves a conjecture of Herrero and Srivastava [4], though we are able to refine the conjecture in light of our results.

2. Main Result

Let T and T' be tournaments defined on a ground set C. We are concerned with voting rules defined over (T, T'). If there exists a binary voting tree that implements (i, j), we write $(i, j) \in I$.

Definition 1. *A set $S \subseteq C$ is prime if there does not exist a partition of S into two more more nonempty subsets such that $S = S_1 \cup S_2 \cup \dots S_k$ and*

(1) *$a \in S_i, b \in S_j, i \neq j$ implies either $aTb, aT'b$ or $bTa, bT'a$, and*
(2) *$a \in S_i, b \in S_j, i \neq j, aTb$ implies $a'Tb'$ for all $a' \in S_i$ and $b' \in S_j$.*

Figure 3 illustrates a prime set.

We denote the top cycle of T restricted to a set $S \subseteq C$ as $tc(S)$ and the corresponding top cycle at T' as $tc'(S)$.

We say that $i_1, i_2, \dots, i_k$ is a *T–path* if $i_1 T i_2 T \dots T i_k$.

The following Theorem is our main characterization result.

Theorem 1. *$(a, b) \in I$ if and only if there exists a prime set S with $a \in tc(S), b \in tc'(S)$.*

The proof of this theorem is quite involved, and is presented in the appendix. Intuitively, the theorem states that a pair is implementable if and only if the pair is in their respective top cycles, and for any partition of the candidates into subsets,

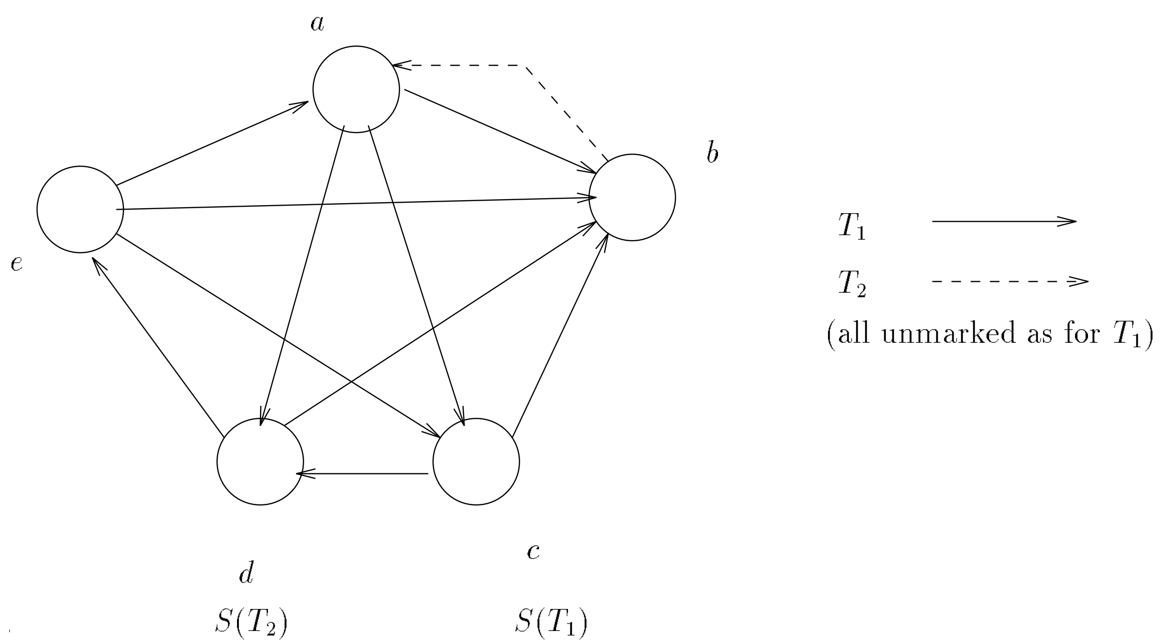

Figure 3. A Prime Set

either the pair occurs in the same subset or the tournaments do not agree on the relationships between subsets.

Prime sets can be quite complicated. The simplest type of prime set is formed by starting with a pair x, y such that xTy and $yT'x$. Such a set is obviously prime. Given a prime set, we can extend it by adding a singleton that differs in it relationship to the prime set. In other words, if P is prime, and $x \notin P$ has the property that either xTi_1 and $i_2T'x$ or i_1Tx and $xT'i_2$ for some $i_1, i_2 \in P$, then $P \cup \{x\}$ is prime. For the prime set in Figure 3, the initial prime set of $\{a, b\}$ is then extended by c, d, and e in turn. The proof works by showing that adding singletons is sufficient to generate all prime sets and then by showing that the top cycles of $P \cup \{x\}$ are implementable.

Some idea of the complexity of the problem can be seen in Figure 4, which give the implementation tree for (d, e) as given by the proof. This implementation tree is also minimum in terms of both depth (the length of the longest path from the root to a leaf) and number of nodes.

This method of adding singletons is also useful in many of the following theorems in order to show the primeness of a set.

In the rest of this section, we consider some implications of the theorem.

2.1. A Class of Pairwise Implementable Rules. Our first corollary to Theorem 1 is a class of rules that are pairwise implementable.

Theorem 2. *If $tc(C) \neq tc'(C)$, then $x \in tc(C)$, $y \in tc'(C)$ implies $(x, y) \in I$.*

Proof: Let $tc(C) \neq tc'(C)$, and choose $x \in tc(C), x \notin tc'(C)$ such that x has a minimal T–path to some $y \in tc'(C)$ (if no such x exists, interchange the roles of T

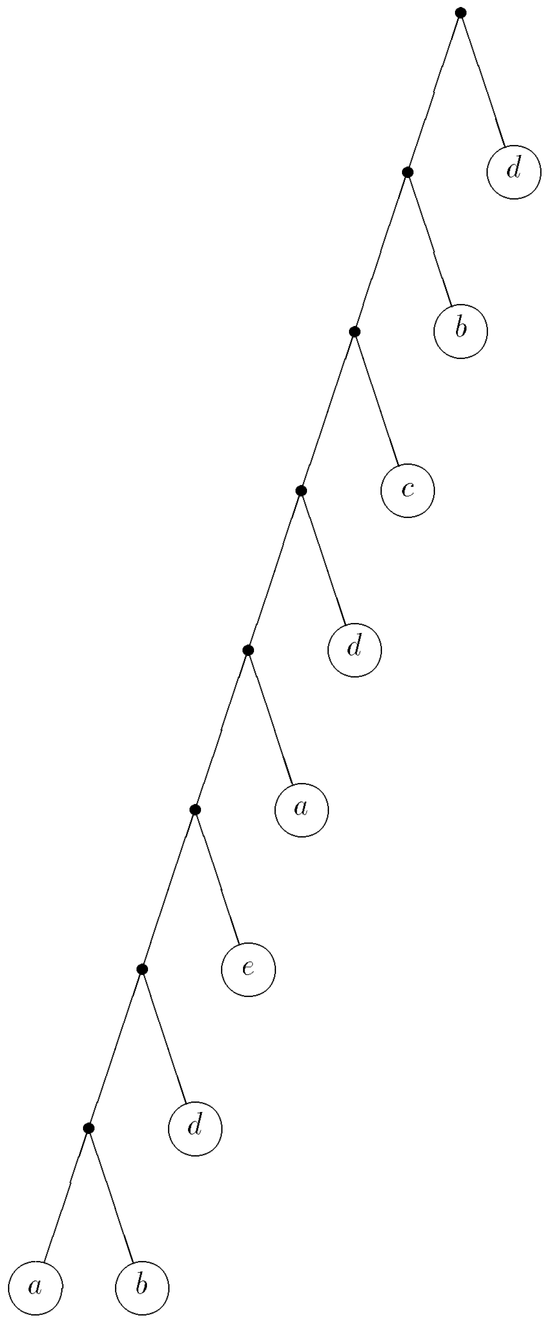

FIGURE 4. Implementation Tree

and T' in the following argument). We will show that $tc(C) \cup tc'(C)$ is a prime set, which will prove the theorem.

If yTx then there is a T–path $x, i_1, \dots, y$ contained in $tcvC$. But this contradicts our minimality assumption in the choice of x and y. Therefore xTy. Since $x \notin tc'(C)$, $yT'x$.

Sort the elements of $tc'(C)$ by the length of the shortest T' path from y, and let the order be $y, y_1, y_2, \dots, y_k$. The claim is that $\{x, y, y_1, y_2, \dots, y_l\}$ is prime for any $l \leq k$. Proof by induction. For $\{x, y\}$, we have xTy and $yT'x$ so that set is prime. For general l, we have y_lTx and $y_pT'y_l$ for some $p < l$ so $\{x, y, y_1, \dots, y_l\}$ is prime if $\{x, y, y_1, \dots, y_{l-1}\}$ is.

Therefore, $\{x\} \cup tc'(C)$ is prime. Now we show that we can add in $tc(C) \backslash tcvpC$. This can be done by induction on the length of the shortest x' to x T–path. For a path $x', x_1, \dots, x$, with $x' \notin tc'(C)$, we have $x'Tx_1$ but $yT'x'$, so we can add x' to the prime set. $\square$

Examples of rules that are therefore pairwise implementable include any rule where the winner is only a function of which candidates are in the top cycle, not the relationships among them.

The above proof does not require that the top cycles be taken with respect to the entire set of candidates. For instance, if the uncovered sets of T and T' are the same, but the top cycles restricted to the uncovered sets are not, then it is possible to implement any choice from the uncovered sets. We can induct on this to implement choices in the top cycle of the uncovered set of the top cycle of the uncovered set and so on. This is related to to the voting rules defined by Dutta [2] and Schwartz [8].

2.2. Implementing a Particular Rule. A second application of Theorem 1 is in the finding of trees that implement particular voting rules. On the surface, our results seem more directed to show what is not implementable than what is. In this section, we show how our results enable us to determine the voting tree for a particular voting rule.

Consider the following voting rule: Assign a total ordering on the candidates. The winner for a tournament T is the element of the top cycle that appears first in the total ordering. This voting rule is practically the simplest onto voting rule possible. Call this class *Lexical Minimal Voting.* By theorem 2, this voting rule is pairwise–adjacent implementable, and we believe that the rule, as defined over all tournaments is implementable. In fact, for three candidates, there is a very simple voting tree, as shown in figure 5.

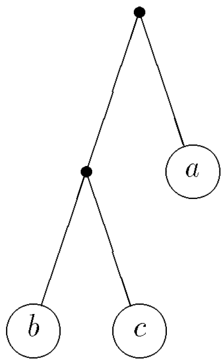

FIGURE 5. Lexical Minimal Voting on Three Candidates

For four candidates, however, the tree is not easy at all to find. A straightforward approach to this problem would be to generate all implementable rules defined on four candidates and determine if Lexical Minimal Voting is among them. Since there are $2^6 = 64$ tournaments on four candidates, there are up to $4^{64} \approx 3.40 \times 10^{38}$ possible voting rules. Fortunately, we are able to show that the vast majority of rules are not implementable. Let the candidates be labeled a, b, c, and d, with alphabetical order providing the ordering.

First, 32 tournaments have a Condorcet winner. If we ensure the voting tree has all the candidates, these tournaments will automatically generate the correct winner.

Of the remaining 32, 24 have all candidates in the top cycle (hence have a as the winner), 6 have three elements in the top cycle including a (and also have a as the winner), and two have three elements in the top cycle but do not include a (so b is the winner).

Now, consider the 24 tournaments with all candidates in the top cycle. These divide into twelve pairs of tournaments, where for each pair the tournaments are adjacent and decompose into three subsets. For each of these pairs, we need only consider one tournament, for the other will have the correct choice by Theorem 1. Similarly, the 6 tournaments with three elements in the top cycle including a will automatically be satisfied as long as the others are.

This leaves us with 14 tournaments: one from each of twelve pairs, and the two tournaments with all elements except a in the top cycle. This leaves us with 268,435,456 possible tournaments. While still beyond the capability of most computers, this is now small enough that a heuristic search would suffice.

Eight tournaments were chosen (including the two that did not have a in the top cycle) and a minimal implementation tree was found for the restriction of the rule onto these eight. The resulting rule was not correct for one of the remaining tournaments, so it was added to the set. When the minimal tree was found for the resulting nine tournaments, it was found to be feasible for all fourteen (hence all 64). The resulting tree is illustrated in figure 6.

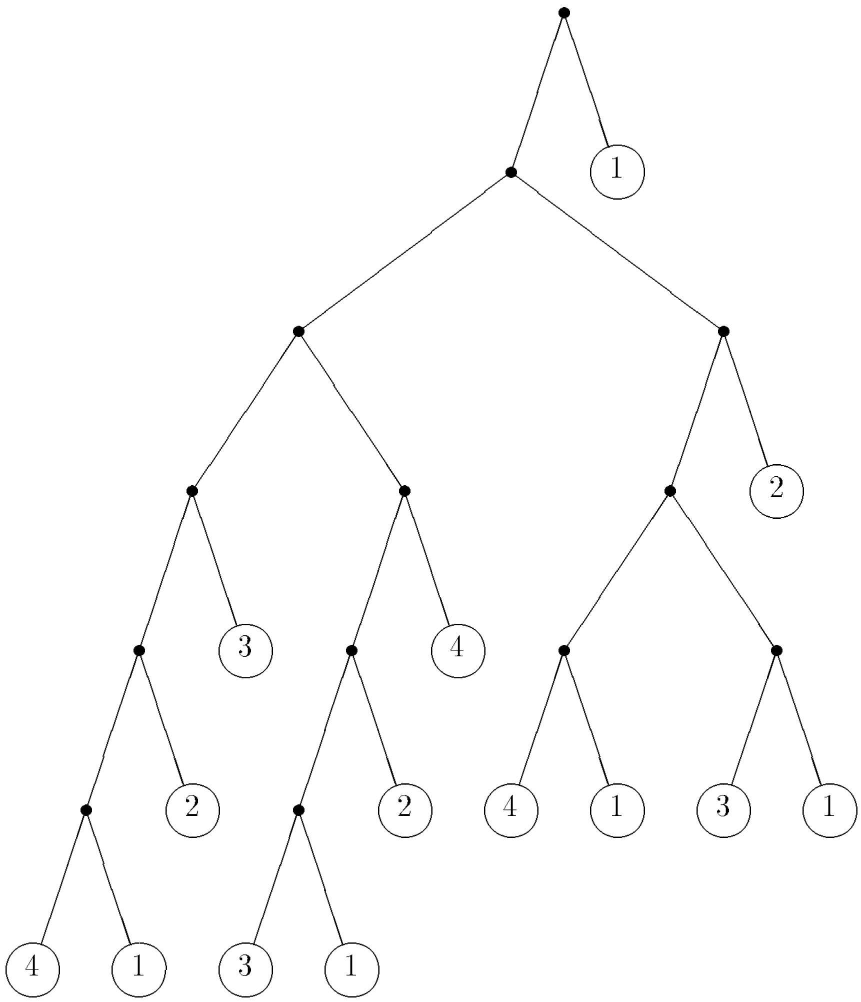

FIGURE 6. Lexical Minimal Voting on Four Candidates

Note that this approach is very computer intensive (the final run required more than 1 week of computer time on an HP–720 workstation), but would be completely infeasible if we did not have our results to limit our search.

2.3. Implementing Copeland Rules. Our final application of Theorem 1 shows that Moulin's [7] example showing that Copeland's rule is not implementable is essentially minimal for pairwise implementability. In other words, for less than eight candidates, there exists a tiebreaking rule that always picks Copeland winners and which is pairwise implementable. It is an open question whether this implies that the entire rule is implementable.

Theorem 3. *For any two tournaments T and T' on seven or fewer candidates, there exist candidates x and y such that x is a Copeland winner for T and y is a Copeland winner for T' and $(x, y) \in I$.*

Proof: Assume there is T and T' on C such that there is no Copeland winners x and y respectively such that $(x, y) \in I$. We will show that $|C| \geq 8$.

First note by Theorem 2 that we need only be concerned with T and T' with the same top cycles, and a minimal counterexample would have that top cycle be C. Let x be any Copeland winner for T and y be any Copeland winner for T'. By Theorem 1, if $(x, y) \notin I$, there must be a decomposition of C into prime sets such that $C = S_1 \cup S_2 \cup \ldots S_k$ and x and y are in different S_i. We will do a case analysis on the number of maximal prime sets. This number cannot be 2 since that would imply that the top cycle of T was not C.

Number of prime sets is 3: Suppose $S_1 T S_2 T S_3 T S_1$, and $x \in S_1, y \in S_2$. Let $|S_i| = n_i$. Note that since T and T' agree except within the S_i, x beats all of S_2 for both T and T', and y beats all of S_3. For x to be a Copeland winner in T, then, $n_1 - 1 + n_2 > \lfloor n_2/2 \rfloor + n_3$ (the inequality is strict because otherwise there would be a candidate in S_2 that would also be a Copeland winner). Similarly, we have $n_2 - 1 + n_3 > \lfloor n_1/2 \rfloor + n_2$ or $n_3 - 1 > \lfloor n_1/2 \rfloor$. This implies $n_3 \geq 2$.

If $n_3 = 2$, then $n_1 = 1$, so $n_2 > \lfloor n_2/2 \rfloor + 2$, so $n_2 \geq 5$.

If $n_3 = 3$, then $n_1 \leq 3$. If $n_1 = 1$, $n_2 \geq 7$; if $n_1 = 2$, $n_2 \geq 5$; if $n_1 = 3$, $n_2 \geq 3$.

If $n_3 = 4$, then the only relevant n_1 values are 1 and 2. If $n_1 = 1$, then $n_2 \geq 9$. If $n_1 = 2$, then $n_2 \geq 7$.

Finally, if $n_3 = 5$ then n_1 must be 1, which requires $n_2 \geq 11$.

In all cases $n_1 + n_2 + n_3 \geq 8$

Number of primes sets is 5 (there are no cases where the number of prime sets is 4 without also having 3 prime sets). For there to be seven or fewer candidates, there are at most two extra candidates beyond the five required (one for each prime set). Suppose they are in separate prime sets, S_i and $S_{i'}$. At least one of T and T' must choose from these two prime sets so suppose T chooses from S_i. Note that changing the preferences within a pair does not increase the maximum Copeland value. Therefore, T' also has a choice from S_i with the same Copeland value.

Therefore, the two extra candidates must be in the same prime set (making three in that set). Note that changing the preferences among these three can only change the maximum Copeland value by 1. This implies that there is a tie for the top Copeland score in either T or T'. If there is a tie in both, then break the tie by choosing outside the prime set with three candidates. If there is a tie in only one, choose inside the prime set with three candidates. This will give choices within prime sets.

Since there cannot be 6 prime sets without there also being 5 or fewer, and there must be at least eight candidates for two nonidentical tournaments to have at least seven prime sets, we see that $|C| \geq 8$. $\square$

It is not difficult to deduce a tie breaking rule that holds for all tournaments and has the property that for any pair of tournaments, the rule chooses Copeland winners from the same prime sets, as long as the number of candidates is seven or fewer. It would be interesting to determine if such a tie–breaking rule is implementable over all tournaments simultaneously.

3. Adjacent Tournaments

In this section, we consider the special case of *adjacent* tournaments, i.e. when T and T' differ in exactly one place. It turns out that for such pairs of tournaments, there is only one prime set that is relevant, and this set has a particularly elegant structure. In the next section, we show that this simple characterization does extend to what we call neighboring and distant tournaments, but does not extend to general pairs of tournaments.

Thus, suppose that $T = T'$ except that aTb and $bT'a$. Our characterization is as follows: Let B be the smallest subset of C such that

(1) $a \in B, b \in B$,

(2) For all $y \notin B$, either xTy for all $x \in B$, or yTx for all $x \in B$.

B is then the largest set with containing a and b with the property that any alternative outside B bears the same relation to all members of B. Our characterization is that the top cycle set of B at the two tournaments completely defines the set of non-constant implementable choice functions on T, T'.

Moulin [7] defined a related property called *adjacency*. For any tournament T, B

is an adjacent set of T if every element not in B had a constant relationship with every element in B. In other words, if $a \notin B$, $b, b' \in B$, then $aTb \iff aTb'$.

An adjacent set could be merged into a single outcome. If the resulting tournament was the same for two differing tournaments then either the choice on those two tournaments would be the same, or they both would choose from the adjacent set.

The B we define is the only relevant adjacent set for this case. Here we are able to show both the necessity and sufficiency of this condition.

Let $tc(B)$ denote the alternatives in the top cycle of T restricted to B and $tc'(B)$ be the alternatives in the top cycle of T' restricted to B.

Theorem 4. *(x, y) is implementable if and only if $x \in tc(B)$ and $y \in tc'(B)$ or $x = y$.*

If $x = y$, the proof is trivial, so suppose $x \neq y$. In proving this theorem, we first provide an algorithm for finding B. Then, using properties of this algorithm, we prove the necessity of the condition. Finally, sufficiency follows from the fact that B is a prime set (if it was not prime, a subset of B would be prime, contradicting the minimality of B).

The following algorithm computes B.

Algorithm COMP

0) Let $B_0 = \{a, b\}$. $i = 0$.

1) If there is no $x \notin B_i$ such that xTa and bTx or aTx and xTb then go to step 3.

2) For an x_i satisfying the conditions of step 1, let $= i + 1$, $B_i = B_{i-1} \cup \{x\}$. Go to step 1.

3) $B = B_i$. Stop.

Theorem 5. *Algorithm COMP correctly finds B.*

Proof: By the initialization condition, $a \in B$ and $b \in B$. By the stopping criteria in step 1, at the end of the algorithm, for $y \notin B$ either xTy for all $x \in B$ or yTx for all $x \in B$.

To show minimality, we simply note that if any subset of B satisfied the conditions, we would have been unable to add any nodes to it, and the algorithm would have stopped at that point. $\square$

We now prove the necessity of the conditions in Theorem 4. Define the *height* of a voting tree to be the length of the longest path from a leaf to the root. A single reversal has height two.

Theorem 6. *If $(x, y) \in I$ then $x \in tc(B)$ and $y \in tc'(B)$ or $x = y$.*

Proof: We will prove this theorem by induction on the height of the tree that implements a pair. If the height of the tree is 1, then $x = y$. If the height of the tree is 2, then $x = a$ and $y = b$, for that is the only place that the tournaments differ. Since $a \in tc(B)$ and $b \in tc'(B)$, the theorem holds for height ≤ 2.

Suppose the theorem is true for heights $\leq k-1$. We will now prove it for height k. Let (x, y) be implemented by a height k voting tree. A height k voting tree is formed by taking two voting trees of height no more than $k-1$ and combining them. Let the two trees implement (x, y') and (x', y) with xTx' and $yT'y'$. By induction, (x, y') and (y, x') meet the conditions of the theorem. If $x \in tc(B)$ and $y \in tc'(B)$ or $x = y$ then the conditions of the theorem are met. Also, if $x = y'$ and $x' = y$, then, since the tournaments differ only in (a, b) the conditions are also met.

That leaves only the case where $x = y'$, $y \in tc'(B)$ and $x' \in tc(B)$, with $x \notin tc(B)$ (there is a symmetric case with $y = x'$). Since xTx', we must have that $x \notin B$ (if it was in B, then it would be in $tc(B)$). But $x \notin B$ and xTx' implies xTy, so $y'Ty$, which is a contradiction. $\square$

4. Neighboring and Distant Tournaments

While prime sets for general tournaments can be quite complex, the characterization we have obtained for adjacent tournaments is quite simple. The question we ask in this question is the extent to which the simple characterization extends. We prove the rather curious result that if T and T' either differ in many places or differ in only a few, then the structure of the prime sets is quite simple.

Suppose T and T' disagree on at most ten rankings (i.e. if we consider the graphs of the tournaments, then the maximum number of arrows that are reversed is ten). Then we say that T and T' are *neighboring*. Alternatively, suppose that T and T' agree on at most three rankings. Then we say that T and T' are *distant*. We characterize implementable choice functions defined on a pair of tournaments that are either neighboring or distant.

Our characterization is as follows: Let (a, b) be a pair on which T and T' differ (so aTb and $bT'a$). Let Δ_{ab} be the minimum set of alternatives that satisfies the following conditions:

(1) $a \in tc(\Delta_{ab}), b \in tc'(\Delta_{ab})$, and
(2) For all $x \in \Delta_{ab}$, either $x \in tc(\Delta_{ab})$ or $x \in tc'(\Delta_{ab})$.
(3) For all $x \notin \Delta_{ab}$, either
    (a) xTy for all $y \in \Delta_{ab}$, or

(b) $xT'y$ for all $y \in \Delta_{ab}$, or

(c) yTx and $yT'x$ for all $y \in \Delta_{ab}$.

Theorem 7. *If T and T' are either neighboring or distant then $(x, y) \in I$ if and only if $x \in tc(\Delta_{ab})$ and $y \in tc'(\Delta_{ab})$ for some pair (a, b) on which T or T' differ, or $x = y$.*

As in the previous section, we prove this theorem in three steps. First, we provide an algorithm for finding Δ_{ab}. Sufficiency again follows from Δ_{ab} being a prime set. Finally, we prove necessity by a rather lengthy case analysis.

4.1. Algorithm for Δ_{ab}. In this section, we give an algorithm for finding Δ_{ab} and prove its correctness.

Algorithm FIND_DELTA

0) Let $B_0 = \{a, b\}$. $i = 0$.

1) If there is no $x \notin B_i$ such that

(1) $a \in tc(B_i \cup \{x\})$

(2) $b \in tc'(B_i \cup \{x\})$

(3) $x \in tc(B_i \cup \{x\})$ or $x \in tc'(B_i \cup \{x\})$

go to step 3.

2) For an x_i satisfying the conditions of step 1, let $= i + 1$, $B_i = B_{i-1} \cup \{x\}$. Go to step 1.

3) $\Delta_{ab} = B_i$. Stop.

Theorem 8. *Algorithm FIND_DELTA correctly finds Δ_{ab}.*

Proof: By the initialization condition and by the conditions of step 1, $a \in tc(\Delta_{ab})$ and $b \in tc'(\Delta_{ab})$. By the stopping criteria in step 1, at the end of the algorithm, for $y \notin \Delta_{ab}$, y does not enter either top cycle, or y dominates Δ_{ab} in either T or T'.

The only remaining step is to show that this algorithm finds the minimum set, which we do by a similar argument to that for COMP. $\square$

4.2. Sufficiency. Sufficiency again follows from the fact that Δ_{ab} is a prime set.

4.3. Necessity. In this section, we prove the following theorem:

Theorem 9. *If T and T' are neighboring and $(x, y) \in R$ then $x \in tc(\Delta_{ab})$ and $y \in tc'(\Delta_{ab})$ for some (a, b) where T and T' differ or $x = y$.*

Proof: Slightly abusing notation, we will write $(x, y) \in \Delta_{ab}$ to mean $x \in tc(\Delta_{ab})$ and $y \in tc'(\Delta_{ab})$. We will prove this theorem by induction on the height of the tree that implements a pair. If the height of the tree is 1, then $x = y$. If the height of the tree is 2, then $x = a$ and $y = b$, for some place that the tournaments differ. Since $a \in tc(\Delta_{ab})$ and $b \in tc'(\Delta_{ab})$, the theorem holds for height ≤ 2.

Suppose the theorem is true for heights $\leq k-1$. We will now prove it for height k. Let (x, y) be implemented by a height k voting tree. A height k voting tree is formed by taking two voting trees of height no more than $k-1$ and combining them. Let the two trees implement (x, x') and (y', y) with xTy' and $yT'x'$. By induction, (x, y') and (y, x') meet the conditions of the theorem. Let (x, x') come from Δ_{ab} and let (y', y) come from Δ_{cd}. If $(a, b) = (c, d)$ then the theorem holds. There are a number of cases:

Case 1. ($x = x'$ or $y = y'$). If $x = x'$ then xTy' and $yT'x$. Now, if yTx then x must be in Δ_{cd} and, since it beats y', must be in $tc(\Delta_{cd})$, so the theorem follows. If

xTy then, since $yT'x$, we get (x, y) by combining (x, x) with (y, y), which is proved by the induction hypothesis. The $y = y'$ case is identical.

Case 2. $(x \neq x'$ and $y \neq y')$.

Case 2.1. $(xTy$ or $yT'x)$. If xTy then combining (x, x) with (y', y) gives (x, y), so this reduced to case 1. Similarly, if $yT'x$ then (x, x') with (y, y) gives (x, y).

Case 2.2. $(yTx$ and $xT'y)$.

Case 2.2.1 $(x'Ty$ or $y'T'x)$. If $x'Ty$ then, since yTx, $xT'y$, and $yT'x'$, we must have $y \in tc'(\Delta_{ab})$, so the theorem is proved. The case of $y'Tx$ is symmetric.

Case 2.2.2 $(yTx'$ and $xT'y')$. Note that if we show $y \in \Delta_{ab}$ then $y \in tc'(\Delta_{ab})$ and the theorem is proved. Similarly, the theorem is proved if $x \in \Delta_{cd}$.

Case 2.2.2.1 $(y'Ty$ or $x'T'x)$. If $y'Ty$ then $xTy'TyTx$ and $xT'y$ and $xT'y'$. Therefore, $(x, y) \in \Delta_{yx}$. Again, $x'Tx$ is symmetric.

Case 2.2.2.2 $(yTy'$ and $xT'x)$. Since $y' \in tc(\Delta_{cd})$ there must be a minimal path $y'Tw_1Tw_2T \dots Tw_mTy$ with all $w_i \in \Delta_{cd}$. Furthermore, by our sufficiency proof, we can choose y' such that w_iTx for all i (otherwise we can choose w_i to be y'). Similarly, there is a path in Δ_{ab} such that $x'T'z_1T'z_2T' \dots T'_nT'x$ with $z_jT'y$. Note that if $w_iT'x$ for any i then $x \in \Delta_{cd}$ (hence in $tc(\Delta_{cd})$) so $xT'w_i$ (and similarly yTz_j).

Case 2.2.2.2.1 $(y'T'x'$ or $x'Ty')$. If $y'T'x'$, then $y' \in \Delta_{ab}$ and in $tc'(\Delta_{ab})$. But this implies that $w_1 \in \Delta_{ab}$, $w_2 \in \Delta_{ab}$ and so on up to $y \in \Delta_{ab}$, proving the theorem. If $x'Ty'$ then we can conclude similarly that $x \in \Delta_{cd}$.

Case 2.2.2.2.2 $(x'T'y'$ and $y'Tx')$. Note that if $y' \in tc'(\Delta_{cd})$ then $x' \in \Delta_{cd}$ and the method in Case 2.2.2.2.1 holds. Therefore $y' \notin tc'(\Delta_{cd})$ and $x \notin tc(\Delta_{ab})$. At this point, let us determine the number of places in which T and T' must differ:

(1) yTx and $xT'y$;

(2) $y'Tx'$ and $x'T'y'$;

(3) Since xTx' and $xT'x'$, there must be some reversal to implement (x, x');

(4) Similarly for (y', y);

(5) At least one node a must be added to implement (x, x'), and one node b to implement (y', y). Each of a must form a reversal with each of y', y, b (for a total of 3) and b forms a reversal with x and x' (for 2 more) to give five differences between T and T'.

(6) A more involved argument shows that there must be two more reversals.

This gives a total of 11 places where T and T' differ. Similarly we have shown that there are 4 places where T and T' must be the same. Therefore, if the pair of tournaments differ in ten places or fewer or are the same in three places or fewer, then one of the preceding cases must occur. □

4.4. Intermediate Cases. Our Δ characterization does not hold for cases where there are 11 or more differences and four or more concurrences between the two tournaments. Consider tournaments in figure 7.

In the figure, the T tournament is represented by solid lines while the T' tournament is represented by dashed lines. Any relationship not shown goes down for T and up for T'. Table 1 lists the Δ sets for every reversal.

Note that this table gives no way to implement (x, y). But x, y is implementable through the tree in figure 8.

5. Extensions

An intriguing possibility is suggested by the following theorem.

Theorem 10. *Let T_1, T_2, T_3 be tournaments such that*

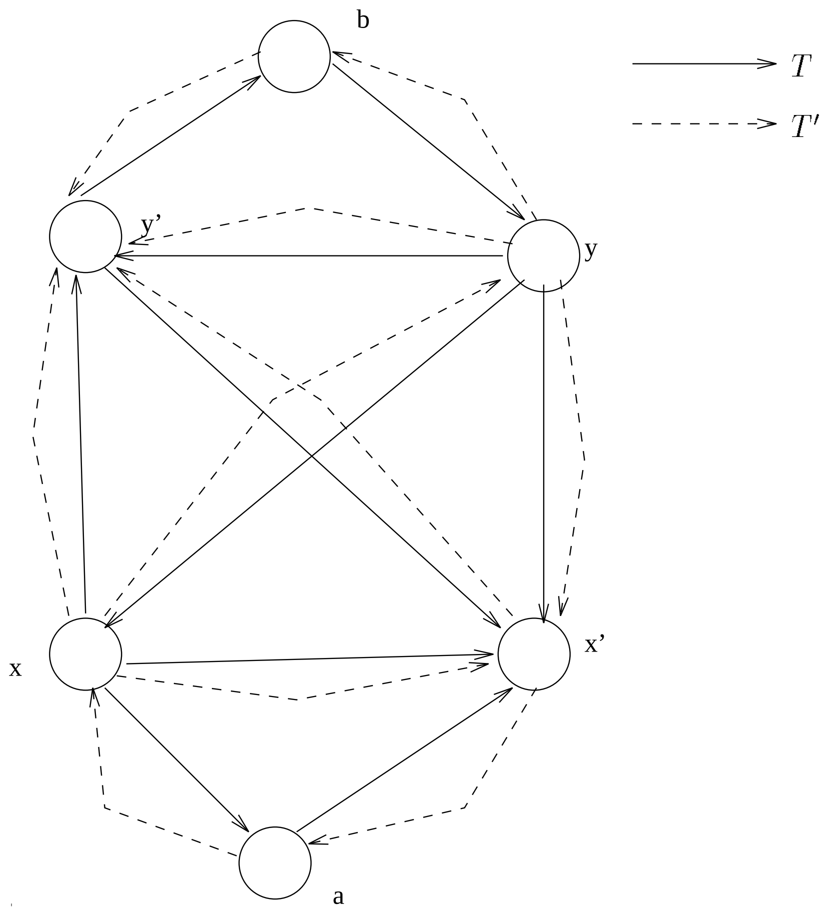

FIGURE 7. Example for intermediate case

Reversal	$tc(\Delta)$	$tc'(\Delta)$
(x,a)	x	a,x,x'
(a,x')	a	x'
(y',a)	y'	a
(y,a)	y	a,x,x',y
(b,a)	b	a
(y,x)	y	x
(b,x)	x,y',y,b	x
(y',x')	y'	x'
(b,x')	b	x'
(y',b)	y'	b
(b,y)	y',y,b	y

TABLE 1. Deltas for Counterexample

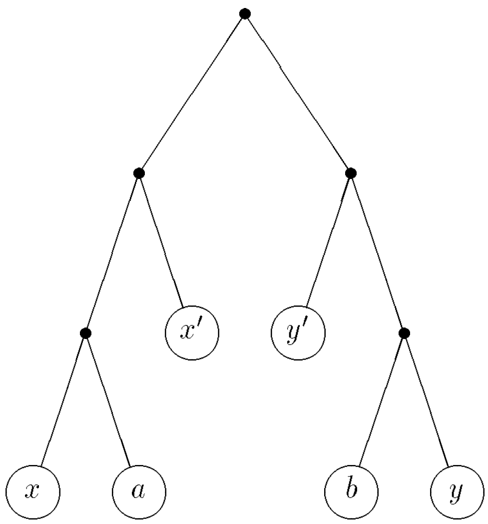

FIGURE 8. Implementation tree for (x,y)

(1) *T_1 and T_2 are adjacent,*

(2) *T_2 and T_3 are adjacent, and*

(3) *$T_1 \neq T_3$ (and hence must differ in two places).*

Let x, y, z be such that $(x, y) \in I(T_1, T_2)$ and $(y, z) \in I(T_2, T_3)$, and $y \neq z$. Then $(x, z) \in I(T_1, T_3)$.

Proof: Let T_1 and T_2 differ on (a, b) and T_2 and T_3 differ on (c, d).

Suppose $x \neq y$ and $y \neq z$. By Theorem 4, we have $x \in tc\Delta_{ab}(T_1, T_2)$, $y \in tc'\Delta_{ab}(T_1, T_2) \cap tc\Delta_{cd}(T_2, T_3)$ and $z \in tc'\Delta_{cd}(T_2, T_3)$. In this case, let $\Delta = \Delta_{ab}(T_1, T_2) \cup \Delta_{cd}(T_2, T_3)$.

If $x = y$ and $y \neq z$, let $\Delta = \Delta_{cd}(T_2, T_3)$. Otherwise, let $\Delta = \Delta_{ab}(T_1, T_2)$.

We know that Δ is prime in (T_1, T_3) for otherwise that decomposition would work equally well for either $\Delta_{ab}(T_1, T_2)$ or $\Delta_{cd}(T_2, T_3)$.

We now show that $x \in \mathrm{tc}(\Delta)$. Let z' be some member of Δ and suppose there is no T_1–path from x to z'. This implies that $z' \in \Delta_{cd}T_2, T_3$. Therefore, there is a T_1 path in Δ from x to y and a T_2–path Q from y to z'. If the union does not contain a T_1–path from x to z, then T_1 and T_2 must differ on Q. But T_1 and T_2 differ only on (a, b), so Q must contain the segment (b, a) in T_2. There is a T_1–path from x to a in Δ. Adding the (a, z') path in T_2 where T_1 agrees gives a T_1 path from x to z'. Therefore $x \in tc(\Delta)$.

Similarly, $z \in \mathrm{tc'}(\Delta)$. By Theorem 1, this gives us that $(x, z) \in I(T_1, T_3)$. $\square$

This theorem suggests that forcing the implementation of pairwise adjacent tournaments may force additional consistency. This is not true for tournaments that are not adjacent, as the following shows.

Herrero and Srivastava [4] conjectured that pairwise compatibility is sufficient for implementing voting rules defined on three or more tournaments provided choices were made from the top cycle. By better understanding the role of decomposability, we can show this conjecture to be false.

They already have a counterexample when three tournaments are pairwise implementable but choices are not taken from the top cycle. Let $S = \{a, b, c, d\}$, and let the three tournaments be:

T_1: $aT_1b, aT_1c, dT_1a, bT_1c, dT_1b, dT_1c$

T_2: $bT_2a, cT_2a, dT_2a, cT_2b, bT_2d, dT_2c$

T_3: $aT_3b, cT_3a, dT_3a, cT_3b, bT_3d, cT_3d$

Let the choice for T_1 be a, T_2 be b, and T_3 be c. These three tournaments are pairwise compatible, but $(a, b, c) \notin R(T_1, T_2, T_3)$. Note that $a \notin tc(T_1)$. We can fix this flaw by adding two candidates e and f such that $\{a, b, c, d\}T_ie$, eT_if, and $fT_i\{a, b, c, d\}$ for $i = 1, 2, 3$. Now $a \in tc(T_1)$ and $(a, b, c) \notin I(T_1, T_2, T_3)$.

We can, however, refine the conjecture.

Conjecture 1. *For any voting rule $S(T)$ defined over all tournaments, $S(T)$ is implementable on trees if and only if the rule restricted to any pair of adjacent tournaments is implementable.*

References

1. J.S. Banks, "Sophisticated Voting Outcomes and Agenda Control," *Social Choice and Welfare*, **1:** 295–306 (1985).
2. B. Dutta, "Covering Sets and a New Condorcet Choice Correspondence," *Journal of Economic Theory*, **44:** 63–80 (1988).
3. R. Farquharson, *Theory of Voting*, Yale University Press, New Haven (1969).

4. M. Herrero and S. Srivastava, "Decentralization by Multistage Voting Procedures," *Journal of Economic Theory* (1993).
5. R.D. McKelvey and R.G. Niemi, "A multistage game representation of sophisticated voting for binary procedures," *Journal of Economic Theory,* **18:** 1–22 (1978).
6. N. Miller, "A New Solution Set for Tournaments and Majority Voting: Further Graph–Theoretical Approaches to the Theory of Voting," *American Journal of Political Science,* **24:** 68–96 (1980).
7. H. Moulin, "Choosing from a tournament," *Social Choice and Welfare,* **3:** 271–291 (1986).
8. T. Schwartz, "Cyclic Tournaments and Cooperative Majority Voting: A Solution," *Social Choice and Welfare,* **7:** 19–29 (1990).

Appendix A. Proof of Theorem 1

In proving the theorem, one direction is easy: If (a, b) is implementable, then take a minimal implementation tree. The nodes of the tree form a prime set, and a and b are in their respective top cycles.

Sufficiency is somewhat more involved, and the rest of this section contains the sufficiency proof.

Let S be a prime set. We wish to show that any choice in the top cycle is implementable. We will prove this by induction:

Lemma 1. *If S is a prime set with $|S| > 1$, then $S = S' \cup k$ for some $k \notin S'$ where*

(1) *S' is prime, and*
(2) *For some $i, j \in S'$ either $kTi, jT'k$ or $iTk, kT'j$.*

Proof: Let S be prime. Note that if S' is prime and, for $k \notin S'$, if $kTi, jT'k$ (or the reverse), then $S' \cup k$ is also prime. Starting from each singleton, grow the maximum such prime set. If one of these sets is S, then the lemma is proved. Otherwise, the resulting sets have to be disjoint (if there is an intersection, then it is possible to show a singleton that can be added to one of the sets). These sets, then, define a partition of S', contrary to the hypothesis. $\square$

Therefore, to prove the theorem, we need only prove the following:

Theorem 11. *Let S be prime and let $k \notin S$ be such that kTi and $jT'k$ for some $i, j \in S$. Also, assume $(a, b) \in I$ for all $a \in tc(S), b \in tc'(S)$. Then $(a, b) \in I$ for all $a \in tc(S \cup k), b \in tc'(S \cup k)$.*

Before we begin, we need one lemma about subsets of a prime set:

Lemma 2. *If S is prime and $\emptyset \neq A \subset S$ such that iTj for all $i \in A, j \notin A$, $i, j \in S$, then there exists $j \notin A, i \in A$ such that $jT'i$.*

Proof: Otherwise $A, S/A$ is a partition. $\square$

To prove theorem 11, we will divide the problem into cases. Let $S' = S \cup k$. Note that since $jT'k$, $tc'(S) \subseteq tc'(S')$.

CASE 1. $(tc(S) = tc(S'), tc'(S) = tc'(S'))$. In this case, the theorem is true by induction.

CASE 2. $(tc(S) = tc(S'), tc'(S) \supseteq tc'(S) \cup k)$. Take $a \in tc(S), b \in tc'(S)$. Now, aTk (since $k \notin tc(S)$) but $kT'j'$ for some $j' \in tc'(S')$. So (a, j') combined with (k, k) gives (a, k) for all $a \in tc(S)$.

Now, to get (a, j) for all $j \in tc'(S')$, we induct on the length of the minimum length path from j to k in T'.

If that length is one (so $jT'k$), then to get (a, j) for all $a \in tc(S)$, induct on the shortest a to j path in T. If that length is one, then (a, k) with (j, j) gives (a, j).

If that length is more, so $aTi_1Ti_2 \dots j$, then (i_1, j) (by induction) with (a, k) gives (a, j).

If the minimum length path from j to k in T' is more than one, so $jT'j_1T'j_2T' \dots T'k$, then we get (a, j) by inducting on the a to j path in T. If this length is one (so aTj) then (a, j_1) (by the outer induction) with (j, j) gives (a, j). If this length is more than one (so $aTi_1Ti_2T \dots j$), the (i_1, j) (by the inner induction) with (a, j_1) (by the outer induction) gives (a, j).

CASE 3. $(tc(S') = k,\ tc'(S') = tc'(S))$. Take $b \in tc'(S)$. We need show only (k, b). $bT'k$, but kTa for all $a \in tc(S)$. Taking any such a, we get (a, b) with (k, k) gives (k, b).

CASE 4. $(tc(S') = k,\ tc'(S') \supseteq tc'(S) \cup k)$. For $b \in tc'(S')$ we need (k, b). Induct on the length of the minimum path from b to k in T'. If the length is one, then $bT'k$. But kTb (since $k = tc(S')$) so (k, k) with (b, b) gives (k, b). If the length is more than one, so $bT'j_1T'j_2 \dots T'k$, take (k, j_1) (by induction) with (b, b) to get (k, b).

CASE 5. $(tc(S') \supseteq tc(S) \cup k,\ tc'(S') = tc'(S))$. This is CASE 2, with the roles of T and T' reversed.

CASE 6. $(tc(S') \supseteq tc(S) \cup k,\ tc'(S') \supseteq tc'(S) \cup k)$. This is the hardest case where k enters both top cycles, possibly bringing other nodes into it as well.

In this case, we know that there exits i', i'', j', j'' with the following properties:

(1) $i''Tk, kTi', j''T'k, kT'j'$.

(2) $i' \in tc(S), j' \in tc'(S)$

We prove this case in five steps:

(1) $(a, b) \in I$ for $a \in tc(S), b \in tc'(S)$.

(2) $(a, k) \in I$ and $(k, b) \in I$ for $a \in tc(S), b \in tc'(S)$.

(3) $(a, j) \in I$ and $(i, b) \in I$ for $a \in tc(S), j \in tc'(S'), b \in tc'(S), i \in tc'(S')$.

(4) $(k, j) \in I$ and $(i, k) \in I$ for $j \in tc'(S'), i \in tc(S')$.

(5) $(i, j) \in I$ for $i \in tc(S'), j \in tc'(S')$.

STEP 1. $(a, b) \in I$ for $a \in tc(S), b \in tc'(S)$ follows by induction.

STEP 2.$(a, k) \in I$ and $(k, b) \in I$ for $a \in tc(S), b \in tc'(S)$. This is the most difficult case. We break this case up depending on where i'' and j'' occur relative to the top cycles of S.

SUBCASE 1. $(i'' \in tc(S), j'' \in tc'(S))$.

(k, b) for $b \in tc'(S)$. This follows by induction on the minimal b to j'' path in T'. If this path has length 0 ($b = j''$), then (i', j'') with (k, k) gives (k, j''). In general, if $bT'j_1T'j_2T' \ldots T'j''$, then (k, j_1) (by induction) with (i', b) (by STEP 1), gives (k, b).

(a, k) for $a \in tc(S)$ is similar.

SUBCASE 2. $(i'' \in tc(S), j'' \notin tc'(S))$ (Note that the case where $i'' \notin tc(S), j'' \in tc'(S)$ is identical with the role of T and T' reversed.)

(a, k) for $a \in tc(S)$ is the same as STEP 2, SUBCASE 1.

(k, b) for $b \in tc'(S)$. First we show (a, j'') by induction on the length of the minimum a to j'' path in T. If the length is one (so iTj''), then (a, k) (from the first part of STEP 2) with (j'', j'') gives (a, j''). Otherwise, let path be $aTi_1Ti_2T \ldots Tj''$). Then (a, k) (from the first part of STEP 2) with (i_1, j'') (by induction) gives (a, j'').

Next, we get (k, j'') by combining (i', j'') (from above) with (k, k) to get (k, j'').

Finally, to get (k, b) we take (i', b) (from STEP 1) with (k, j'') (from above).

SUBCASE 3.$(i'' \notin tc(S), j'' \notin tc'(S))$ Note that in this case, kTa for all $a \in tc(S)$

and $kT'b$ for all $b \in tc'(S)$. We choose i'' to be such that the distance from $tc(S)$ to i (taken to be the minimum over all $a \in tc(S)$ of the length of the shortest path from a to i'') is minimized.

Surprisingly, this case is the hardest case, and the only one that uses the fact that S is prime.

First, note that if we had (k, j) for some $j \in S$, then, since there exists $b' \in tc'(S)$, $b'T'j$, combining (k, j) with (i', b') gives (k, b'). Now, we proceed by induction on the length of the minimum b to b' path in T'. Let $bT'b_1T'b_2T' \ldots T'b'$ be the path. Combining (i', b) (by STEP 1) with (k, b_1) by induction gives (k, b) for all $b \in tc'(S)$.

Also, if we have (k, b) for all $b \in tc'(S)$, then let $b'T'i''$ with $b' \in tc'(S)$. Then (k, b') (by assumption) with (i'', i'') gives (i'', b'). This with (k, k) gives (i'', k), and now with (a, j') (from STEP 1) gives (a, k) for all $a \in tc(S)$.

A similar argument holds if we had (i, k) for some $i \in S$. So we need only show (i, k) for some $i \in S$ or (k, j) for some $j \in S$.

Let $A \subset S$ be the set of nodes i such that there exists an i'' to i path in T. By Lemma 2, there exits a node $\hat{\imath} \in A$ such that $\hat{\imath}T'i^*$ for some $i^* \notin A$ (and has $i^*T\hat{\imath}$ by definition of A). Among all choices for $\hat{\imath}$, choose the one that has the shortest path from i'' to $\hat{\imath}$ in T. Let this path be $i''Ti_1Ti_2T \ldots Ti_mT\hat{\imath}$. Rename i'' if necessary in order that kTi_l for all i_l on the path.

Note by our choice of i'' and $\hat{\imath}$, for any i_l on the path, we have i^*Ti_l, $i^*T'i_l$ and kTi_l.

If $i_lT'i_{l+1}$ for all i_l on the path, then $(i^*, \hat{\imath})$ (by definition of i^* and $\hat{\imath}$) with (i_m, i_m) gives (i^*, i_m), which combines with (i_{m-1}, i_{m-1}) to give (i^*, i_{m-1}), and eventually to (i^*, i''). This now combines with (k, k) to give (k, i'') (note kTi^* by our choice of i'').

Otherwise, let p be the minimum index such that for some q, either i_pTi_q and $i_qT'i_p$ or i_qTi_p and $i_pT'i_q$. This gives us either (i_p, i_q) or (i_q, i_p). If we now combine this with (i_{p-1}, i_{p-1}), and so on back, we get either (i'', i_p) or (i'', i_q). Combining this with (k, k) we get (i'', k), as needed.

We now use the previous argument to give us (a, k) and (k, b) for all $a \in tc(S)$ and $b \in tc'(S)$.

STEP 3. (a, j) for $a \in tc(S), j \in tc'(S')$. First we do an outer induction on the minimum j to k path in T'. If $jT'k$, then do an inner induction on the minimum a to j path in T. If aTj, then (a, k) (from STEP 2) with (j, j) gives (a, j). Otherwise $aTi_1Ti_2T \ldots Tj$, so (i_1, j) (by inner induction) with (a, k) (from STEP 2) gives (a, j).

The general case of the outer induction is where $jT'j_1T'j_2T' \ldots T'k$. Again, do an inner induction on the minimum a to j path in T. If aTj, then (a, j_1) (by the outer induction) with (j, j) gives (a, j). In general for the inner induction, we have $aTi_1Ti_2T \ldots Tj$. Take (a, j_1) (by outer induction) with (i_1, j) (by inner induction) to get (a, j).

(i, b) for $i \in tc(S'), b \in tc'(S)$ is similar.

STEP 4. (k, j) for $j \in tc'(S')$. Here we induct on the minimum j to k path in T'. If the length is one (so $jT'k$), take (i', j) (from STEP 3) with (k, k) to get (k, j). In general, $jT'j_1T'j_2T' \ldots T'k$, so take (k, j_1) (by induction) with (i', j) (by STEP 3) to get (k, j).

(i, k) for $i \in tc(S')$ is similar.

STEP 5. (i, j) for $i \in tc(S'), j \in tc'(S'), i \neq k, j \neq k$. Here we induct on the sum of the length of the shortest i to k path in T and of the length of the shortest j to k path in T'. If the total is 2 (so iTk and $jT'k$), take (i, k) (from STEP 4) and (k, j)

(from STEP 4) to get (i,j).

In general, we have $iTi_1Ti_2T\dots Tk$ and $jT'j_1T'j_2T'\dots T'k$. Take (i,j_1) (by induction) with (i_1,j) (by induction) to get (i,j). $\square$