Prover / Delayer Game Upper Bounds For Tree Resolution

Alexander Hertel & Alasdair Urquhart *

July 2, 2006

1 Introduction

The Prover / Delayer Game, described in [BSIW04], is a combinatorial game associated with Tree Resolution (T-RES) proofs, and can be used to simplify T-RES lower bounds on formulas. In addition, the game also perfectly captures the amount of Turing Machine space needed to compute a T-RES proof.

The purpose of this paper is to show that the Prover / Delayer Game also gives an upper bound on the size of T-RES proofs, and furthermore can be used to simplify those proofs by allowing one to prove bounds on the points scored in the game rather than having to draw out entire T-RES proofs. In effect, the Prover / Delayer Game can be used to simplify upper and lower bounds for both T-RES size as well as space.

For the purposes of this paper, we assume that the reader is familiar with the basics of proof complexity and general complexity theory. We shall use [CK01] as our reference for proof complexity, and [GJ79] as our reference for $\mathcal{NP}$-Completeness.

2 Prover / Delayer Game Description

The Prover / Delayer Game is an adversarial game between two players, the 'Prover', and the 'Delayer'. The game is played on an unsatisfiable CNF formula F. The point of the game is for the Prover to falsify some initial clause of F, thereby falsifying the formula. Since the formula is unsatisfiable, this is inevitable. Roughly speaking, the Delayer's goal is to delay the falsification of the formula for as long as possible.

The game proceeds in rounds. Each round starts with the Prover choosing a variable, and asking the Delayer what the value of that variable is. The Delayer can give one of three answers:

- True
- False
- You Choose

If the Delayer says 'You Choose', then the Prover gets to decide the value of that variable. In addition, every time 'You Choose' is said, the Delayer wins one point. This is the only way in which points can be scored.

The game finishes when some clause has been falsified. The real goal of the game is not actually to prove or delay; rather, the Delayer's aim is to win as many points as possible, while the Prover's aim is to make sure that the Delayer wins as few as possible.

*This research supported by NSERC and the University of Toronto Department of Computer Science

The T-RES bounds associated with the Prover / Delayer game are all stated in terms of the number of points scored by the Delayer. This leads us to the following definition:

Definition 2.1. *Let F be an unsatisfiable CNF formula. The Prover / Delayer number of F, denoted $PD(F)$ is the greatest number of points the Delayer can score on F with the Prover playing with an optimal strategy.*

3 Prover / Delayer Game & Space

The Prover / Delayer Game can be used to give bounds on the clause space required by a Turing Machine while computing a T-RES proof of an unsatisfiable formula F.

Definition 3.1 ([ET01]). *Let F be an unsatisfiable CNF formula, and let $\pi = C_1, C_2, ..., C_k$ be the clauses of a Resolution refutation of F such that $C_k = \emptyset$, and each C_i appears in F or follows from two previous clauses by Resolution. We say that F has a T-RES refutation bounded by clause space s if there exists a series of CNF formulas (without having repeated clauses) $F_1, F_2, ..., F_x$ such that $F_1 \subseteq F$, $\emptyset \in F_x$, in any F_i there are at most s clauses, and for each $i < s$, F_{i+1} is obtained from F_i by any of the following:*

- *deleting (if desired) some of its clauses,*
- *adding the resolvent of two clauses of F_i* ***and deleting the parent clauses****,*
- *adding (if desired) some of the clauses of F (initial clauses).*

In [ET03], the authors give an identical upper and lower bound for $CS(F)$ based on $PD(F)$:

Theorem 3.2 ([ET03]). *For any unsatisfiable formula F, if $PD(F) = k$, then $CS(F) = k + 1$*

4 Prover / Delayer Game Lower Bounds For Tree Resolution

The Prover / Delayer Game was originally used in [BSIW04] to prove size lower bounds for T-RES proofs. More specifically, lower bounds on $PD(F)$ can be used to prove lower bounds on the size of T-RES proofs.

Definition 4.1. *The size of a T-RES proof is the number of clauses that it contains. This corresponds to the number of nodes in the proof's underlying tree.*

Theorem 4.2 ([BSIW04]). *If an unsatisfiable CNF formula F has a DPLL tree of size $\leq 2^k$, then the Prover has a strategy limiting the Delayer to $\leq k$ points.*

Proof: Let F be any arbitrary unsatisfiable formula with a DPLL tree of size $\leq 2^k$. The Prover uses this tree as a strategy to limit the Delayer to at most k points as follows: the prover starts by querying the variable corresponding to the root of the tree. If the Delayer says "True" or "False", then the Prover proceeds by querying the variable in the corresponding subtree of the root. If the Delayer says "You Choose", then the Prover chooses the smaller subtree, and next queries on that variable. Therefore, even if the Delayer says "You Choose" as much as possible, 2^k can only be split in half at most k times, so the Delayer wins at most k points, as required.

□

The contrapositive of this Theorem gives us the following Corollary, giving a direct connection between the Prover / Delayer Game and T-RES size lower bounds:

Corollary 4.3. *If the Delayer has a strategy guaranteed to win $> k$ points on F, then every DPLL tree for F has size $> 2^k$.*

5 Prover / Delayer Game Upper Bounds For Tree Resolution

In addition to lower bounds, the Prover / Delayer Game gives us upper bounds on the size of T-RES proofs. More specifically, upper bounds on $PD(F)$ can be used to prove upper bounds on the size of T-RES proofs. This result can be proved both constructively and non-constructively. The non-constructive version gives a slightly better bound, but the constructive version yields more insight into the relationship between the game and the T-RES proof that it yields.

5.1 Constructive Proof

The following Theorem constructively allows us to take the history of a Prover / Delayer Game, and build a DPLL tree:

Theorem 5.1. *If the Prover has a strategy limiting the Delayer to at most k points playing on formula F which contains $\leq n$ distinct variables, then F has a* DPLL *tree of size $O(n^k)$.*

Proof: Proof by (strong) induction on k, the number of points scored by the Delayer.

Basis: $k = 1$. Let F be any arbitrary formula on which the Prover has a strategy limiting the Delayer to at most 1 point. We will use the series of queries and answers from the game played between the Prover and Delayer on F to construct a DPLL tree of size $O(n)$.

Since the Delayer scores at most 1 point, the very last query made by the Prover must result in the Delayer responding with "You Choose" (YC), and the Prover's response must falsify the formula, or else the Delayer would be able to score ≥ 2 points. Therefore all $\leq n-1$ previous variable queries must have been answered with a "True" or "False".

Rename the variables so that they are queried in the order $x_1, x_2, ..., x_j$, where x_j is the variable for which the Delayer responds, "YC". Without loss of generality, assume that the Prover sets x_j to true (the false case is analogous). Use this series of questions and answers to build a path from the root of the tree to a leaf, as shown below in Figure 1.1.

We now fill in the rest of the tree. For every "True" or "False" reply, the Delayer could have said "YC", and we know that this change in strategy could not have increased the number of points scored. This means that every node in the DPLL tree corresponding to one of these $\leq n-1$ variables must be adjacent to a leaf, each marked with an 'x' in Figure 1.2.

Now we need only construct the subtree labelled T_1. If this subtree is a leaf, then we are done. However, it may be the case that the formula F_j at the root of T_1 is one in which the Delayer can win 1 point (the Delayer cannot win ≥ 2 points on F_j, or else the reply to x_j would have been "False" instead of "YC", allowing for ≥ 2 points to be scored on F). In order to build T_1, we therefore play a new game on F_j, and use exactly the same technique as above to turn this game into a tree.

Note that because "YC" was said, preceeded by zero or more non-"YC" responses, F_j contains strictly fewer distinct variables than F, and the height of our entire DPLL tree can be at most n, so this process is guaranteed to terminate, leaving us with a DPLL tree in which each node either is a leaf or is adjacent to at least one leaf. Our tree therefore has height $\leq n$, and contains $\leq 2n+1$ nodes, which is $O(n)$.

Induction Hypothesis: Let F be any arbitrary unsatisfiable formula on $\leq n$ distinct variables. Assume that if the Prover has a strategy limiting the Delayer to at most $k-1$ points, then F has a DPLL tree of size $O(n^{k-1})$.

Induction Step: Let F be any arbitrary formula on which the Prover has a strategy limiting the Delayer to at most k points. As in the basis, we will again use the series of queries and answers from the game played between the Prover and Delayer on F to construct the 'backbone' of a DPLL tree, which we will then fill in.

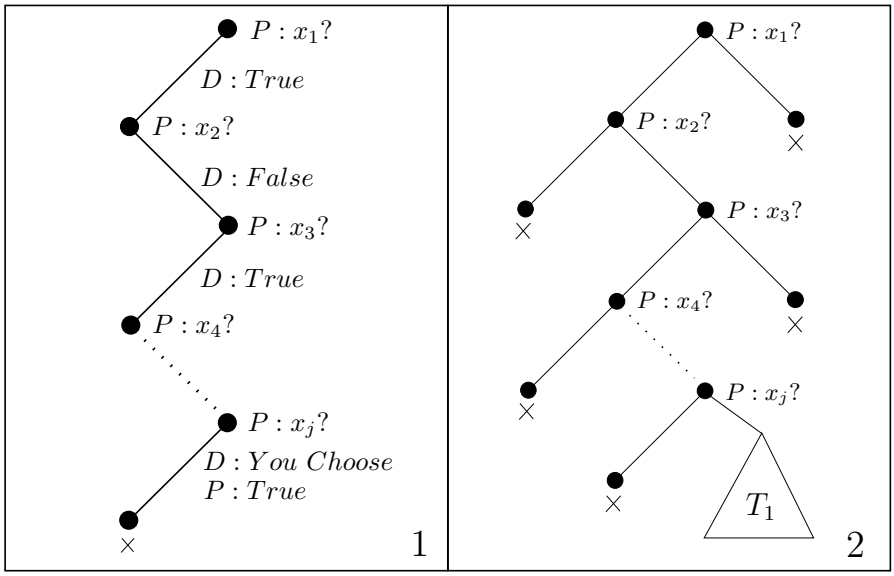

Figure 1: Example of a game in which 1 point is scored together with its corresponding DPLL tree.

Consider the game played on F up to and including the first "YC" reply to the query on variable x_j, as shown below in Figure 2.1. The path corresponding to the remainder of the game in which $k-1$ points are scored is labelled P_{k-1}.

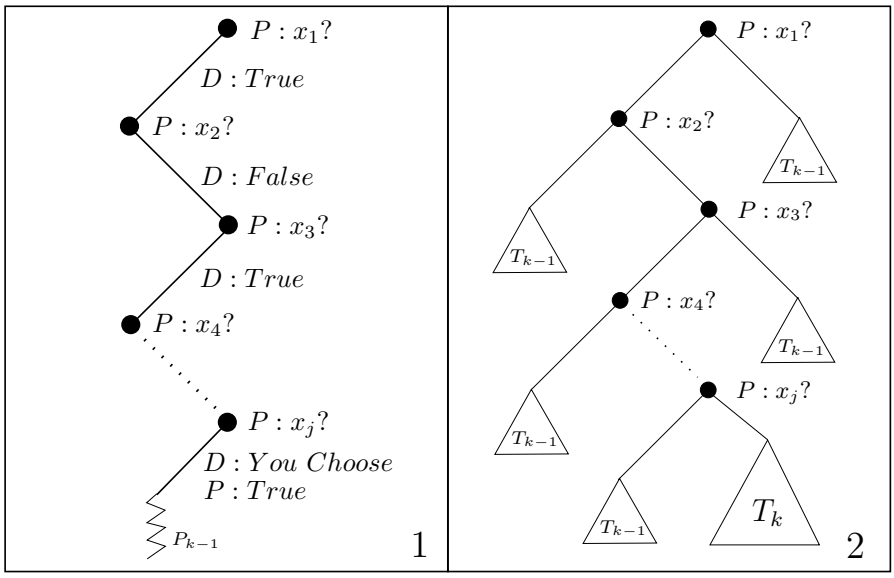

Figure 2: A path representing a game in which k points are scored together with its corresponding DPLL tree.

We will now fill in the rest of the tree. The first "YC" is preceeded by 0 or more non-"YC" responses. Consider any one of the variables for which a non-"YC" answer is given. Without loss of generality, let us examine variable x_2. The Delayer could have replied "YC" to the query on x_2, and this change in

strategy could not have increased the number of points won. But we have already seen that the false path leads to a situation where the Delayer can win k *more* points for a total of $k+1$ points, so in response, the Prover would have to have said "True".

Let F_2 be the formula resulting from setting x_2 to true. The Delayer couldn't win k points here, because that would mean that saying "YC" on x_2 is a better strategy than saying "False". Therefore the Prover has a strategy limiting the Delayer to at most $k-1$ points on F_2, and our Induction Hypothesis applies, so F_2 has a DPLL tree T_{k-1} of size $O(n^{k-1})$. A similar argument applies to each of $x_1, ..., x_{j-1}$, as shown in Figure 2.2.

Now consider x_j, the first variable for which the Delayer says "YC". Without loss of generality, assume that the Prover decides to set x_j to true, resulting in the restricted formula $F_{j=true}$. Since one point has been scored, the Delayer can score a maximum of $k-1$ points on $F_{j=true}$, so the Induction Hypothesis applies and it has a DPLL tree of size $O(n^{k-1})$, as indicated.

The right subtree, however, must be treated similarly to its analogue in the basis. If $< k$ points can be won in it by the Delayer, then the Ind. Hyp. applies and we are done. We know that the Delayer cannot win $> k$ points in it or else the response on x_j would have been "True" rather than "YC", allowing for $> k$ points to be won overall, a contradiction.

Therefore assume that the right subtree is rooted at a restricted formula $F_{j=false}$ on which the Delayer can win exactly k points. In other words, it is as if we are starting our tree-building process all over again. In order to build T_k, we therefore play a new game on $F_{j=false}$, and use exactly the same technique as above to turn this game into a tree. Again, it is important to note that $F_{j=false}$ contains strictly fewer distinct variables than F, and the height of our entire DPLL tree can be at most n, so in the worst case this process can continue until the remaining height is $\leq k$, at which point the Delayer must say "YC" continuously to get up to k points, and the Ind. Hyp. applies to both subtrees since each has height $\leq k-1$ and $\geq k$ points cannot be won in only $\leq k-1$ queries. When this process finishes, every subtree that we've filled in has size $O(n^{k-1})$, so we will end up with a tree consisting of a path of length $\leq n$ with each child tree hanging off of it having size $O(n^{k-1})$. Therefore our tree has size $O(n^k)$.

Therefore, by induction, if the Prover has a strategy limiting the Delayer to at most k points playing on formula F which contains $\leq n$ distinct variables, then F has a DPLL tree of size $O(n^k)$, as required. □

When we combine this result with Theorem 4.2, we get the result that any DPLL tree must have size at least $O(2^k)$ and at most $O(n^k)$, where $k = PD(F)$.

5.2 Non-Constructive Proof

Although Theorem 5.1 above yields very good intuition into how the Prover / Delayer Game can be turned into a DPLL tree, it is possible to get tighter bounds, albeit with a non-constructive proof. The proof requires the following Lemma:

Lemma 5.2 ([ET01]). *Let F be an unsatisfiable formula on n distinct variables. If F has a* T-RES *refutation requiring clause space $CS(F)$, then it has a* T-RES *refutation of size $\binom{n+CS(F)}{CS(F)}$.*

Theorem 5.3. *If the Prover has a strategy limiting the Delayer to at most k points playing on formula F which contains $\leq n$ distinct variables, then F has a* T-RES *proof of size $\leq e^{k+1}(\frac{n}{k+1}+1)^{k+1}$.*

Proof: Suppose that the Prover has a strategy limiting the Delayer to at most k points playing on formula F. By Theorem 3.2, we know that F has a T-RES proof with $CS(F) = k+1$. By Lemma 5.2, F has a T-RES refutation of size $\binom{n+k+1}{k+1}$, which is bounded above by $e^{k+1}(\frac{n}{k+1}+1)^{k+1}$. Therefore F has a T-RES proof of size $\leq e^{k+1}(\frac{n}{k+1}+1)^{k+1}$, as required.

□

5.3 Examples

Having T-RES size upper bounds based on the Prover / Delayer game is particularly useful because it allows us to drastically simplify proofs of T-RES upper bounds.

5.3.1 Example 1: $H(K_n^*)$ Formulas

In [HH06], a reduction from the Hamiltonian Cycle problem to SAT is given. Intuitively, the reduction takes an input graph G and produces a formula that enforces a mapping from the vertices in G to the positions of a Hamiltonian Cycle. The mapping must be a bijection, and can include clauses that enforce the mapping to be Total, 1-1, a Function, and Onto. In addition, there are clauses which enforce the edge structure of G. The output formula has variables of the form $m_{i,j}$ which are interpreted as meaning that vertex i in G is mapped to position j in the Hamiltonian Cycle. Let the resulting formulas of this reduction be called $H(G)$. Additional subscripts are added to this notation to indicate which clauses were used to enforce the bijection. For example, if the reduction used clauses from the total and 1-1 groups, then the resulting formula is labelled as $H(G)_{T,1}$.

This reduction is applied to a family of graphs called K_n^*. Each K_n^* graph consists of the complete graph K_n with the addition of a single degree-0 vertex called x. Since each K_n^* is disconnected, it is clearly non-Hamiltonian, which in turn means that every formula $H(K_n^*)$ is unsatisfiable.

When the Total, Onto, and Function clauses are used, then the resulting formula has polynomial upper bounds:

Theorem 5.4 ([HH06]). T-RES *proofs for the unsatisfiability of $H(K_n^*)_{T,O,F}$ formulas have $O(n^2)$ size upper bounds, where n is the number of distinct variables contained in the formulas.*

The proof is a fairly standard argument which uses a DPLL tree template. Although clear enough, these types of proofs are difficult to write-up because drawing proof templates requires a lot of time and effort. The Prover / Delayer Game can be used to significantly simplify the proof:

Theorem 5.5. $PD(H(K_n^*)_{T,O,F}) \leq 2$

Proof: We describe a strategy for the Prover that limits the Delayer to at most 2 points:

First the Prover tries to map vertex x to some position in the cycle. This is done by first querying $m_{x,1}$; if the Delayer says "False", then the Prover queries $m_{x,2}$, and so on until the Delayer finally says "True" or "You Choose". If the Delayer says "False" to every query, then a Total clause is falsified, and the game is over. Therefore the Delayer has no choice but to concede that x gets mapped somewhere, and can win at most one point during this phase by saying "You Choose", at which point the Prover says "True".

Next the Prover tries to map every single vertex (including x) to the position adjacent to the one which x was mapped to. Since x has degree zero, mapping any vertex other than x into an adjacent position would falsify an Edge clause. Mapping x to two places would falsify a Function clause. However, if no vertex is mapped to that position, then an Onto clause is falsified. The Delayer's best strategy is therefore to simply say "You Choose", and accept the inevitable defeat while winning another point for a total of 2, the most possible given this Prover strategy, as required.

□

Together with Theorem 5.1, this result gives us Theorem 5.4. It is easy to see that the proof using the Prover / Delayer Game is considerably simpler and easier to write than the diagrammatic T-RES template given in [HH06].

5.3.2 Example 2: $H(G_{\frac{n}{2},\frac{n}{2}})$ Formulas

Let $G_{\frac{n}{2},\frac{n}{2}}$ be the graph consisting of two disjoint cliques of size $\frac{n}{2}$. If we take these graphs and apply the reduction from [HH06] which uses the T, O, 1, and F clauses, the resulting formulas have polynomial T-RES upper bounds:

Theorem 5.6. $PD(H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}) \leq 3$

Proof: We describe a strategy for the Prover that limits the Delayer to at most 3 points. Assume that the vertices in one clique are red, numbered $r_1, ..., r_{fracn2}$, and those in the other are blue, numbered $b_1, ..., b_{fracn2}$.

First the Prover tries to map vertex r_1 to some position in the cycle. To avoid the Total clause for r_1 being falsified, the Delayer must let it be mapped somewhere, and can win at most 1 point in doing so by saying "You Choose". Without loss of generality, let us assume that r_1 was mapped to position 1.

Next, the Prover tries to map vertex b_1 to position 2. If the Delayer says "False", then the Prover tries to map every remaining blue vertex to position 2. If the Delayer says "True", or "You Choose" (followed by the Prover saying "True"), then a blue vertex and a red vertex are adjacent, which falsifies an Edge clause. In this case, the Delayer wins at most 2 points.

On the other hand, if the Delayer doesn't allow any blue vertices to be mapped to position 2, then the Prover attempts to map every blue vertex to position $i = 3, 4, ..., n$. This leads to two cases:

Case 1: The Delayer continuously says "False". In this case, the Prover attempts to map all possible blue vertices to position 1 (the position occupied by r_1). If the Delayer says "True" or "You Choose" for one of them (followed by the Prover saying "True"), then two vertices have been mapped to the same slot, causing the falsification of a 1-1 clause. If the Delayer again says "False" and doesn't allow a blue vertex to be mapped to position 1, then there is a blue vertex which was mapped to nowhere, causing the falsification of an Onto clause. Within Case 1, the greatest possible number of points scored is 1, for a total of 2.

Case 2: The Delayer allows a blue vertex to be mapped to position i. We may assume that this came about by the Delayer saying "You Choose", at which point the Prover would say "True", thereby winning another 1 point for the Delayer. We now have a red vertex in position 1, a blue vertex in position i, and we know that positions $2, ..., i-1$ do not have blue vertices in them. At this point the Prover attempts to map all possible red vertices to position $i-1$. If the Delayer says "False" to all of them, then an Onto clause is violated, and the game ends with 2 points. The alternative is that the Delayer allows a red vertex to be mapped to position $i-1$ (by saying "You Choose" for a total of 3 points). Since position i already has a blue vertex in it, this falsifies an Edge clause. In this case, the Delayer wins 3 points, which is the maximum of all cases, as required.

□

Together with Theorem 5.1, this result gives us the following Corollary:

Corollary 5.7. T-RES *proofs for the unsatisfiability of* $H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}$ *formulas have* $O(n^3)$ *size upper bounds, where* n *is the number of distinct variables contained in the formulas.*

Proof is much simpler than the alternative of drawing a size $O(n^3)$ T-RES proof template.

References

[BSIW04] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near Optimal Separation of Tree-like and General Resolution. *Combinatorica*, Vol. 24, Issue 4:585 – 604, 2004.

[CK01] P. Clote and E. Kranakis. *Boolean Functions and Computation Models.* Springer-Verlag, Berlin, 2001.

[ET01] J. Esteban and J. Torán. Space Bounds for Resolution. *Information and Computation*, 171:84 – 97, 2001.

[ET03] J. Esteban and J. Torán. A Combinatorial Characterization of Treelike Resolution Space. *Electronic Colloquium on Computational Complexity*, 44, 2003.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability -A Guide to the Theory of NP-Completeness*. Freeman & Company, New York, 1979.

[HH06] A. Hertel and P. Hertel. Formalizing Dangerous Reductions. 2006.