On the Complexity of Two-Player Attrition Games Played on Graphs

Abstract

The attrition game considered in this study is a graph based strategic game which is a movement-prohibited analogue of small-scale combat situations that arise frequently in popular real-time strategy video games. We present proofs that the attrition game, under a variety of assumptions, is a computationally hard problem in general. We also analyze the 1 vs. n unit case, for which we derive optimal target-orderings that can be computed in polynomial time and used as a core for heuristics for the general case. Finally, we present small problem instances that require randomizing moves — a fact that at first glance seems counter-intuitive.

Introduction

The work on the attrition game we present in this paper was motivated by creating entries for a real-time strategy (RTS) game programming competition. RTS games are fast-paced video games in which players create armies that fight over resources scattered on the terrain with the ultimate goal of destroying all enemy units and structures. In these popular games, units regularly combat others. An example is shown in Fig. 1. Good RTS game players are able to move dozens of units into advantageous positions quickly, and coordinate unit attacks effectively. Handling this aspect of RTS games well is crucial to winning, but also tiresome. It is therefore natural to ask how AI modules can be constructed to which time critical tasks such as unit targeting can be delegated, so that human players have more time to focus on more strategic decisions.

In this paper we establish the theoretical foundation of this line of research by defining simplified versions of the RTS game combat subgame and establishing the computational complexity of solving such games. Our results show that likely no polynomial time algorithms for the general problem exist and that in general one has to consider mixed strategies.

We start by discussing related work and defining the class of attrition games considered here. We then investigate the basic "1 vs. n" case in some detail before going on to prove that solving attrition games is hard in general. After this we determine the computational complexity of some game variants and present small game scenarios that show that mixing strategies is sometimes required. We conclude the paper by discussing future research directions in this area.

Related Work

Games of attrition have been studied in military research in which the main focus has been on modelling warfare globally by means of differential equations (Gozel 2000; Taylor 1983). Applications of such models include predicting winners and estimating inflicted damage in battle simulations. In commercial RTS video games and recent RTS game AI competitions (Buro *et al.* 2006) small-scale combat is usually be addressed by scripting simple behaviors such as attacking the closest or the weakest target in range. The advantage of this approach is fast execution speed and focusing fire implicitely. As we will see later, optimal target ordering depends on the attack value–hit point ratio. In artificial intelligence research, attrition games have been studied in the setting of popular RTS video games. For instance, (Kovarsky & Buro 2005) and (Balla & Fern 2009) apply heuristic search methods such as alpha-beta, Monte Carlo, and UCT search to small RTS game combat scenarios. These methods attempt to produce approximate solutions, given that the state and action spaces, even for small problems, can be huge, and the available time for making tactical decisions in RTS games is usually short. As far as we know, our paper is the first to address theoretical aspects of discrete attrition games.

Figure 1: A typical RTS game combat scene (StarCraft™ 2).

Analysing the computational complexity of games has a long history (Eppstein 2010). For our proofs we utilize known hardness results for the 0-1 knapsack and the quantified Boolean formula problems whose proofs can be found in computational complexity text books (e.g. (Arora & Barak 2009)).

Attrition Games Played on Graphs

The attrition game (AG) we consider in this paper is a graph-based simultaneous move game in which two players, black and white, attempt to destroy each other's nodes. A player is said to win if he destroys all opposing nodes while preserving at least one of his nodes. All nodes have two integer attributes, health and attack power, denoted by a pair $\langle h, a\rangle$. Each node may have distinct health and attack values. The nodes are arranged in a directed graph in which an edge exists from node x to node y if and only if node x may attack node y.

In the discrete case the game proceeds in a series of rounds, such that in every round each node may select at most one opposing node to attack. Attacks are then made simultaneously, with the health of a node being decreased by the sum of all the attacks made against it that round. After all attacks have been computed, nodes which have a health value less than or equal to 0 are removed.

In the continuous case units attack constantly and are immediately removed when their health reaches 0. Also, units are effectively permitted to divide their attack power (e.g. total damage dealt per second) amongst their legal targets, using any non-negative weights whose sum totals 1.

The payoff structure ranges from assigning $-1, 0, +1$ to terminal nodes depending on whose units have been completely eliminated, over accumulating rewards for killing individual units, to assigning payoffs non-linearly depending on the composition of the standing units.

Action sets seen in popular RTS video games are often much more complex. For instance, weapons may have cooldown periods, units may be repaired, or action effects may be nullified by opponents' actions. In addition, in RTS games units are free to move into or out of attack range, whereas our model assumes attack graphs to be static. However, as we will see shortly, the basic setting we consider here already leads to complex decision problems.

1 vs. n Units

Consider the case of one white unit versus n black units, with all units able to attack all opponent units. The strategy for the black player is obvious — direct all attack power towards the lone white target. Depending on the specific winning condition or scoring function we choose to impose on the scenario, there may be several (or an infinite number) of equivalent strategies for the white player. Most reasonable objectives are satisfied by having the white player minimize the amount of damage taken by its unit. As the following results show, this can be accomplished by an easy-to-compute target ordering:

Theorem 1. *Let h_i and a_i be the health and attack power of the black units, for $i = 1..n$, and a_0 the attack power of the lone white unit. Then*

a) in the discrete case, in which units fire in rounds and only one target can be selected at any given time, to minimize white's total sustained damage it is sufficient to order its targets by decreasing value of $a_i/\lceil h_i/a_0\rceil$ and never change targets until they have been destroyed.

b) Similarly, in the continuous case, in which units fire continuously and are able to distribute their attacks over multiple targets, to minimize white's total sustained damage it is sufficient to order its targets by decreasing value of a_i/h_i and focus exclusively on single targets until they are destroyed.

Proof. a) Fix an arbitrary ordering in which the black targets are being destroyed. Without loss of generality our following use of subscripts will refer to a unit's position within this ordering, rather than the initial labelling. We first note that it is sufficient for white to stick to targets, i.e. to attack single units without focus change until they are destroyed. To see this, consider time k_i at which unit i is killed in a given attack sequence for $i = 1..n$ and $k_0 = 0$. As unit i is destroyed at time step k_i, there is no need to target it any longer. Also, if between time steps k_{i-1} and k_i a target other than unit i is attacked, it is beneficial for white to attack target i instead. This swap does not influence k_j for $j > i$ and k_i is decreased, which lowers the total sustained damage. Iterating this swapping procedure for all i results in an attack sequence in which units are targeted in turn until they are destroyed. Therefore, we can assume $k_i = \sum_{j=1}^{i} \lceil h_j/a_0\rceil$, where $\lceil h_j/a_0\rceil$ is the time it takes white to destroy unit j, and concentrate on optimizing target orderings.

Let $d_i = a_i \cdot k_i$ be the damage dealt by unit i over its lifetime, given the target ordering. Now consider swapping two adjacent units within the target ordering, say, units j and $j+1$. Let k'_i and d'_i be the values resulting from that swap.

Note that $d_i = d'_i$ for units $i \notin \{j, j+1\}$. For $k = k_{j-1}$ (with $k_0 := 0$) and $t_j = \lceil h_j/a_0\rceil$ we have:

$$d_j = a_j(k + t_j) \qquad d'_j = a_j(k + t_{j+1} + t_j)$$
$$d_{j+1} = a_{j+1}(k + t_j + t_{j+1}) \qquad d'_{j+1} = a_{j+1}(k + t_{j+1})$$

and elementary arithmetic yields:

$$\begin{aligned} d_j + d_{j+1} \le d'_j + d'_{j+1} \quad &\text{iff} \\ a_{j+1}t_j \le a_j t_{j+1} \quad &\text{iff} \\ \frac{a_{j+1}}{t_{j+1}} \le \frac{a_j}{t_j}&, \end{aligned}$$

which gives the desired result.

b) In the continuous case white selects a series of time points when new sets of targets together with the corresponding attack value distributions are chosen. As before, we fix a target ordering and let k_i denote the time unit i is destroyed. By a swapping argument analogous to a) it is easy to see that the sustained total damage is not increased by sticking to single targets until they are destroyed. The final result of applying this procedure is a sequence of time points $k_i = \sum_{j=1}^{i} h_j/a_0$, $i = 0..n$ at which unit i has been destroyed (if $i > 0$) and *the* next target will be chosen (if $i < n$). Setting $t_i = h_j/a_0$ and following the same steps shown above proves the claim. □

There is room to generalize these results so that they directly apply to real-time strategy video games in which attacks usually proceed in rounds and attack values are independent and uniformly distributed:

Theorem 2. *Let h_i be the health and a_i the expected attack power of the black units, for $i = 1..n$, and $p(x)$ the probability of the lone white unit inflicting damage x. Then to minimize white's expected damage it is sufficient to order targets by decreasing value of a_i/t_i and never change targets until they have been destroyed. Here, t_i is the expected lifetime of unit i which only depends on p and h_i.*

Proof sketch due to space limitation. We follow the same proof steps as before, now using $t_j = E(t : \text{unit } j \text{ dies after exactly}$

t steps when attacked) and the fact that the expected lifetime damage unit j deals is the product of its expected attack power and its expected life time. t_j only depends on p and h_j and can either be determined analytically if p has a simple form, or estimated using simulation. □

The above results can be used to decide whether the white unit can survive by sorting the n targets according to their health over (expected) survival time ratios and then computing the damage inflicted on the white unit. If this value meets or exceeds white's health, the unit dies. Otherwise, it survives.

If the white unit is unable to survive we shall concentrate on minimizing the long-term ability of the black units to inflict damage (say in the case in which another possibly identical singleton white unit will arrive after the current battle). This can be modelled by a non-negative reward for white for destroying each of n black units. Clearly if the singleton unit can destroy all opposition the maximum profit may be obtained. If the white unit cannot do this then it must select a subset that it can destroy, so as to maximize the reward. Once a subset is selected the order in which to destroy those units (assuming they can all be destroyed) is well-defined by our previous argument. We can show that this problem is hard in general:

Theorem 3. *Given a discrete AG scenario with n black units with health h_i, attack a_i, and kill reward $r_i \geq 0$ for white, and a single white unit with health h_0 and attack a_0, it is NP-hard for white to decide what the reward-maximal target ordering is, in case white does not survive.*

Proof. We show the result by the following reduction to a 0-1 knapsack problem. Intuitively, we will create a collection of black units with minimal attack (such that the order in which they are destroyed does not affect the white unit's lifetime) each representing a good to be placed in the knapsack, plus one indestructible black unit to enforce the budget. Specifically, given a 0-1 Knapsack instance:

$$\text{maximize} \sum_{j=1}^{n} p_j x_j \quad \text{subject to} \sum_{j=1}^{n} w_j x_j \leq c$$
$$x_j \in \{0,1\}, \quad j = 1, \ldots, n,$$

we construct black units $\langle w_j, 0, p_j \rangle$ for $j = 1, \ldots, n$ and $\langle \infty, 1, 0 \rangle$, where the elements in the triple are health, attack power, and the opposing player's reward for destroying the unit respectively. The white unit is $\langle c, 1 \rangle$. The equivalence between the two problems is straightforward. □

The construction of units with zero attack power in the proof may be somewhat unsatisfying. Note, however, that these zeros may be replaced with an appropriately small epsilon and then rescaled to integer values.

Solving Basic Attrition Games

In this section we consider discrete time and attack attrition games whose payoffs are limited to $-1, 0$, and $+1$ (loss, draw, win) depending on which player is still alive at the end. We call such games "basic attrition games" and denote their set by BAG. Computing the minimax value of attrition games is straight-forward, but slow (Kovarsky & Buro 2005): as 2-player zero-sum games with simultaneous moves but no state component hidden from either player, each state constitutes a matrix game whose payoff matrix is populated recursively with the minimax values of the successor states. Given a payoff matrix the standard linear programming formulation can be used to determine the minimax value in each state. Thus, the total runtime of this ad-hoc method is polynomial in the game tree size, which by itself can potentially be super-exponential in the input size.

The following results establish that minimax value computation for basic attrition games in general is computationally hard, and the proof of part b) describes a faster way of computing the minimax value:

Theorem 4.

a) The decision problem of determining the existence of pure winning (i.e. minimax value 1) strategies for white (BAG-WIN) is PSPACE-hard, and

b) BAG-WIN $\in$ EXPTIME

As an immediate consequence we obtain the following complexity result for general minimax value computations:

Corollary 5. *The problem of deciding whether the minimax value for white in a given basic attrition game instance is $\geq v$ is PSPACE-hard.*

Proof. Set $v = 1$ and apply Theorem 4 a). □

Proof of Theorem. a) Our plan is to reduce the Quantified Boolean Formula (QBF) problem, which is known to be PSPACE-complete (Arora & Barak 2009), to BAG-WIN. First we define a small delaying widget shown in Fig. 2. This widget forces some white node or a set of nodes (not shown), to deal at least t points of damage to node 1 within the first t rounds. If this does not happen then an isolated black node (3) will remain, preventing white from winning. For our purposes the white node being forced to attack will have an attack power of one. Thus, the white node will not be able to attack any other nodes for the first t rounds without white losing the game.

Next we present two widgets which we will use in our QBF reduction. These widgets allow the "choosing" player to determine, out of three nodes, which one will survive. Moreover, this choice may be made at any time up to a given round i, with no penalty for delaying the choice until that round (i.e., no information about the choosing player's strategy is revealed before the choice is made). In the figures depicting both widgets it may be the case that the white nodes have outgoing edges which are not shown (specifically from those labelled x and/or $\neg x$, but there are no omitted incoming edges to any of the nodes.

Existential Quantifier Widget. To emulate an existential quantifier we create the widget shown in Fig. 3. The previously defined delaying widget is depicted as a box, with the delaying time shown in brackets. A more compact existential quantifier widget exists, but this version is perhaps more interesting in that it lets us make the entire QBF reduction attack graph acyclic.

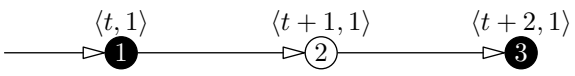


Figure 2: A delaying widget. A white node (not shown) must deal t points of damage within the first t rounds. If the incoming node has attack power 1, then this node cannot attack elsewhere until round $t + 1$, lest node 3 survive.

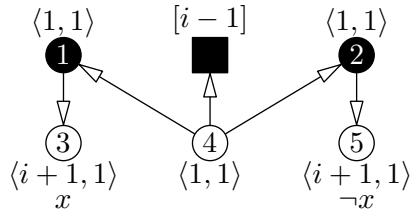


Figure 3: ∃ widget. The white player may save either the x or the $\neg x$ node by attacking node 1 or 2 during round i. The node not saved will be killed the next round. The black box represents a delay widget with $t = i - 1$.

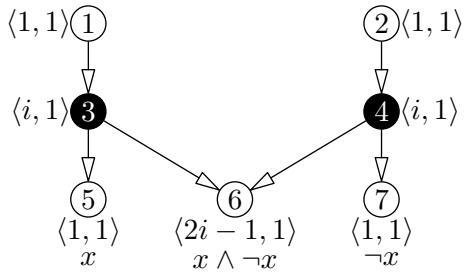


Figure 4: ∀ widget. Black may determine which of x or $\neg x$ will die by attacking node 5 or 7 at any time up to step i. Making more than one such attack results in node 6 surviving, which is a dominated action in the constructed game.

This widget allows the white player to save either the node corresponding to x or the node corresponding to $\neg x$. Attempting to save both will result in a surviving black node in the delaying widget, and thus in an inability for white to win. Before the unchosen white node is destroyed, each of nodes 3 and 5 will be able to deal at most $i + 1$ points of damage (via outgoing edges which are not shown). The remainder of the graph shall be constructed so as to make this damage unimportant. All nodes in this widget (except perhaps 3 and 5) either have only one target or have their actions forced — the corresponding strategy for these nodes is obvious.

Universal Quantifier Widget. To emulate a universal quantifier we use the widget shown in Fig. 4. This widget allows black to kill at most two of nodes 5, 6, and 7 (corresponding to x, $\neg x$, and $x \wedge \neg x$). This can be seen by noting that nodes 5, 6, and 7 have a combined $2i + 1$ hit-points, but black will be able to deal no more than $2i$ points of damage, since nodes 1 and 2 will kill the black nodes on step i. As in the other widget, the white nodes corresponding to truth assignments will each be able to deal at most i points of damage before the black nodes are killed.

QBF Reduction. We will proceed to show that a QBF problem instance may be transformed in polynomial time into a BAG instance such that the white player has a pure winning strategy if and only if the QBF existence player has a winning strategy. We assume that the QBF problem is written in conjunctive normal form, with all variables being governed by a quantifier, and all quantifiers being at the beginning of the formula.

At a high level, we will create black nodes corresponding to clauses and white nodes corresponding to variable assignments (true or false), as in the widget node labels. The white variable nodes will be able to attack only those black clauses in which they appear.

The reduction proceeds as follows. For each QBF existential/universal quantifier, create a corresponding existential/universal attrition widget, shown in Figs. 3 and 4 respectively, where "x" in the widget corresponds to the particular variable governed by that quantifier. The value of i in each widget should be equal to the position (from left to right) of the respective quantifier, starting at 1. In this way a player may delay choosing within a particular widget until the choice of the previous widget/quantifier is known. Making a choice early is never more beneficial than waiting until step i.

In this manner, white nodes now exist corresponding to each QBF variable and its negation. Now create one black node for each QBF clause, with incoming edges from all white nodes whose label occurs in the clause. In the case of "$x \wedge \neg x$" nodes, treat the node as having both x and $\neg x$ labels. The clause nodes have no outgoing edges, so their attack power is arbitrary (say 1), but set their health to some large number such that it is impossible to kill them before the quantifier widgets are stable. That is, until any potential sacrifices within the widgets have been played out and the sacrificed nodes are dead (assuming optimal play). To accomplish this it is sufficient to use $(n + 2)$ times the number of incoming edges on the clause node, where n is the number of QBF quantifiers. It is thus apparent that (supporting our previous assertion) sacrificing nodes in the existential widgets is not beneficial for white.

If the existence player has a winning QBF strategy, white may simply choose to save the correspondingly labelled nodes, delaying each choice as long as possible. It is clear that white may kill all black nodes in the quantifier widgets while making this choice. This leaves only the clause nodes, which by construction must all have a live white node able to attack them. Thus, all black nodes will eventually be killed and white will win. Conversely, if the forall player has a winning QBF strategy, black may choose to save only those white nodes in the universal widgets which correspond to its QBF strategy. By construction there must thus be some clause node which is unable to be killed, corresponding to an unsatisfied clause in the QBF variable assignment.

BAG-WIN $\in$ EXPTIME. To see that a pure winning strategy for white can be computed in exponential time, let L be the input length and H be the maximum number of health values any unit can take (i.e., the maximum unit health plus one), and let $h := \lceil \log_2 H \rceil$ be the size of its encoding. For convenience assume that $H = 2^h$. Then the number of possible worlds is bounded by H^{n+m}, since (given a fixed targeting graph) the world is uniquely described by the health of each of the $n + m$ units. Note that n and m are effectively encoded in unary form, since each unit requires a separate integer describing its health (and another for its attack).

We will compute the value of each world recursively, using a memoizing lookup table (one entry for each possible world) to avoid repeated work. To prove the (non-)existence of a (first player) pure winning strategy in a given world we need only find a row in the strategy matrix where all the entries are 1, or show that such a row does not exist. The value of each matrix entry is determined in the obvious manner — by computing the successor world corresponding to each player's row/column action, and then computing whether this successor has a pure winning strategy. The base case of this recursion are those states where no unit may attack an opponent, in

which case the winner is determined by which player, if any, still has standing units.

Because the number of worlds is bounded by an exponential function of the input length, $2^{h(n+m)} \leq 2^{L^2}$, we need only show that the amount of work in each of those worlds (excluding any recursion, since we are memoizing) is also at most exponential. The number of joint actions in a given world is bounded by $n^m \cdot m^n = 2^{m \log n + n \log m} \leq 2^{2L^2}$, which is obtained when each unit may attack every opponent unit. As such, the time required to loop over each row in the strategy matrix is clearly at most exponential. Computing each successor state involves only a polynomial number of poly-time subtractions and comparisons (to prevent negative values). This procedure is repeated for all table entries until there are no more changes. Thus, the entire computation (including initializing the lookup table) is performed in time $O(2^{p(L)})$ for a low-degree polynomial p, and therefore BAG-WIN $\in$ EXPTIME. □

Corollary 6. *Determining the existence of pure winning strategies for white in basic AG instances in which attack graphs are acyclic and the maximum health value is polynomial in the size of the encoding is PSPACE-complete.*

Proof. The attack graphs in the QBF reduction we presented in the proof of Theorem 4 a) are acyclic. Moreover, the maximum health value across all nodes is polynomial in the encoding size of the graph. Therefore, determining the existence of pure winning strategies for white in such graphs is PSPACE-hard. Revisiting the algorithm presented in part b) with a polynomial health bound shows that above decision problem lies in PSPACE, because at each recursion level only row and column indexes have to be maintained whose length is bounded by $\log \max\{n^m, m^n\} \leq \log I^I = I \log I$. The decision problem is therefore PSPACE-complete. □

Attrition Games with Attack Partitioning

If we discretize kill times but still allow each unit to partition its attack power amongst all of its potential targets, we are left with a problem formulation somewhere between the continuous case and the fully-discrete description (in which attacks are all-or-nothing). Specifically we do not have to consider overkill in the sense that a unit is never forced to commit excess attack power to any target. Let APAG denote the set of all encodings of basic game scenarios in which attack partitioning is allowed, and APAG-WIN the subset of APAG in which white has a pure winning strategy. Then the following statement is true:

Theorem 7. *APAG-WIN is NP-hard.*

Proof. We reduce the subset sum problem, which is a standard NP-complete problem (Arora & Barak 2009), to APAG in polynomial time. Let S be a set of positive integer values and n a non-negative target value, such that the subset sum decision problem is true if and only if there exists a subset $S' \subseteq S$ such that $\sum_{x \in S'} x = n$. Given such a subset sum instance, we construct a corresponding APAG instance as follows: let $\sigma := \sum_{x \in S} x$, $N := 3\sigma + 1$, and consider white unit $\langle N + \sigma + (\sigma - n) + 1, \sigma \rangle$ and black units $\langle s, s \rangle$ for $s \in S$ and $\langle \sigma - n, N \rangle$. Without loss of generality, $\sigma > n$. Otherwise, the subset sum problem has the trivial solution of selecting all elements of S.

Note that the "large" black unit $\langle \sigma - n, N \rangle$ has sufficient attack power to kill the single white unit in two steps, because $2N = N + 3\sigma + 1 > N + \sigma + (\sigma - n) + 1$. Thus, any winning strategy for white must involve killing that large unit on the first step. That leaves $\sigma - (\sigma - n) = n$ damage to allocate amongst the other black units.

On the first step black will deal $N + \sigma$ damage, leaving the white unit with $\sigma - n + 1$ health at the start of the second round. Therefore, if the remaining black units have more than $\sigma - n$ firepower then the white unit will die on the second step. This can only be the case if, on the first step, white is unable to kill a subset of the non-large black units having total health/attack power n, i.e., if a subset of S totalling n does not exist. Thus, in summary, the given subset sum problem has a solution if and only if in the corresponding APAG instance white has a pure winning strategy. Because the described transformation is clearly computable in polynomial time, APAG-WIN is NP-hard. □

Other Objective Functions

We may want to choose winning criteria based on realistic assumptions about game mechanics (if any) which are abstracted away by the graph representation. If we consider the case of real-time strategy games, replacement/repair cost of the units that are destroyed/damaged may be important, or we may seek to only delay the enemy units until reinforcements can arrive. We do not even consider the complexity introduced by allowing unit motion, i.e. changing attack graphs. In what follows we will consider two basic additional objectives which both lead to hard decision problems:

Theorem 8. *If the AG decision problem is modified to include either a maximum number of rounds in which to destroy the opposing units, or the number of survivors, then deciding whether white can accomplish either of these goals is NP-hard.*

Proof. This can be seen by reducing from subset sum. Similar to the attack partitioning case, consider a set S of positive integers with target sum n. Let $\sigma := \sum_{x \in S} x$, $X = \{\langle n, 1 \rangle, \langle \sigma - n, 1 \rangle\}$, and $Y = \{\langle 3, s \rangle | s \in S\}$.

A subset sum exists iff the AG instance can be won by Y in one round, or the AG instance can be won by Y without losing any units. In both cases Y clearly attempts to find a partition of its units such that each partition's attack power exactly matches the health of one of the two X units, killing them both in the first round. □

To see that minimizing the total number of rounds does not necessarily maximize the number of survivors, consider the example in Fig. 5. In this scenario the labelled white node

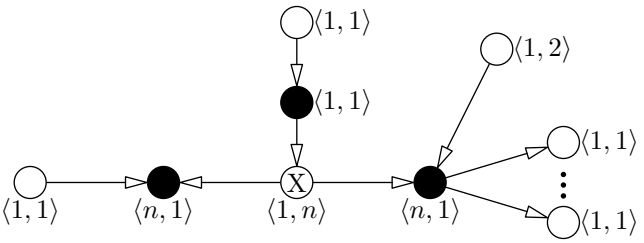


Figure 5: Example showing the non-equivalence of minimizing total time and maximizing the number of survivors.

must decide in the first round whether to destroy the left or the right black node. If the left, then the right black node will destroy one white node each round, for $\lceil n/2 \rceil$ rounds. If the right node is destroyed, then the left black node will not cause any damage, but cannot be killed until n rounds have passed.

Mixed Strategies

In this section we provide small examples to show that in general in discrete attrition games and those with attack partitioning mixed strategies may be necessary for optimal play. At first this seems counter-intuitive, because here, unlike say the game of rock-paper-scissors, chosen actions always succeed in that they inflict damage irrespective of the opponent's choice. However, one can imagine cases in which there exist multi-step counter strategies with distinct first moves for each of the opponent's choices.

As an example, consider the 2 versus 2, fully-connected AG graph shown in Fig. 6. For each player we consider all possible first moves and show the resulting final game values in view of row player black who is attempting to maximize. An entry of $+1$ indicates a win for black, -1 a win for white, and 0 a draw — namely both players destroying each other. In this scenario mixing is only required for the first move. The expected score for black increases from 0 (playing a known pure strategy) to 0.5 if one or both players choose from their two non-dominated actions uniformly.

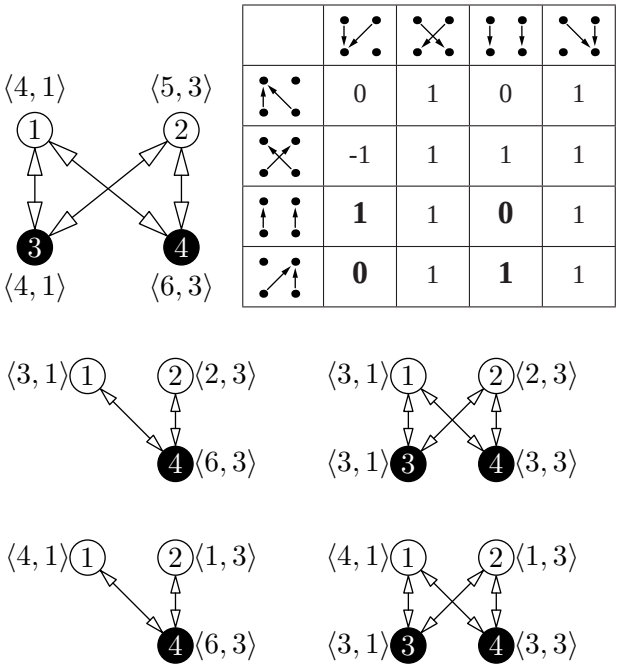
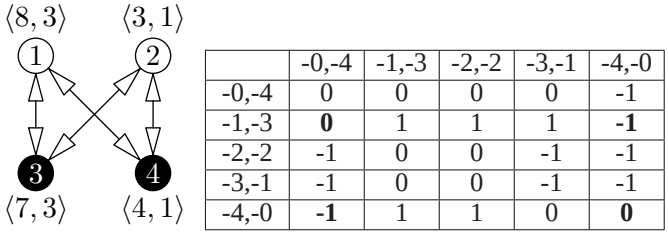


Figure 6: 2 versus 2 example showing the need for mixed strategies: the start state at the top left, followed by the payoff matrix in view of the row player (black), and the successor states for all non-dominated action pair choices, which are highlighted in the payoff matrix.



	-0,-4	-1,-3	-2,-2	-3,-1	-4,-0
-0,-4	0	0	0	0	-1
-1,-3	**0**	1	1	1	**-1**
-2,-2	-1	0	0	-1	-1
-3,-1	-1	0	0	-1	-1
-4,-0	**-1**	1	1	0	**0**

Figure 7: 2 versus 2 example with attack partitioning showing the need for randomized strategies. The column player (white) is attempting to minimize the game value, while the row player (black) is attempting to maximize.

It is also the case that mixed strategies may be necessary for optimal attack partition play. To see this, consider the 2 versus 2 case shown in Fig. 7.

Conclusion and Future Work

In this paper we have established computational complexity results for playing attrition games on graphs which model combat mechanics seen in popular real-time strategy games. Our main results indicate that computing winning strategies for this class of games is computational hard in general. Moreover, playing optimally sometimes requires using mixed strategies. In practice, we therefore have to resort to approximations. As a starting point for the development of heuristics we considered the basic 1 vs. n case for which we identified optimal target orderings that are based on attack value over hit point ratios. We propose replacing simpler measures – such as focusing on the weakest or most powerful unit — in existing RTS game AI systems by above ratio to improve combat performance.

Although we have shown that computing the existence of deterministic winning strategies for the basic attrition game in general is PSPACE-hard and in EXPTIME, it remains unclear for which class the problem is complete. Because attrition games, as we defined them, can last an exponential number of rounds in case of large health and small attack values, showing PSPACE-completeness could perhaps be achieved by establishing optimality of macro operators that collapse long action chains. Another interesting theoretical question is where exactly the transition from P to NP-hard occurs, i.e. what is the smallest k for which the k vs. n problem NP-hard?

References

Arora, S., and Barak, B. 2009. *Computational Complexity: A Modern Approach*. New York, NY, USA: Cambridge Univ. Press.

Balla, R.-K., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2009)*.

Buro, M.; Bergsma, J.; Deutscher, D.; Furtak, T.; Sailer, F.; Tom, D.; and Wiebe, N. 2006. Ai system designs for the first rts-game ai competition. In *Proceedings of the GameOn Conference, pp.13-17*.

Eppstein, D. 2010. Game complexity overview. `http://www.ics.uci.edu/~eppstein/cgt/hard.html`.

Gozel, R. 2000. Firepower score attrition algorithms in highly aggregated combat models. *RAND* 47–60.

Kovarsky, A., and Buro, M. 2005. Heuristic search applied to abstract combat games. In *Proceedings of the The Eighteenth Canadian Conference on Artificial Intelligence*.

Taylor, J. 1983. Lanchester models of warfare. In *Operations Res. Soc. Vol 1+2*.