MULTINATIONAL WAR IS HARD

JONATHAN WEED

Abstract. War is a simple children's game with no apparent strategy. However, players do have the ability to influence the game's outcome by deciding how to return cards to the bottom of their decks. This small amount of choice is enough to give the game strategic complexity. In this paper, we show that the problem of determining whether one player can force a win in a multiplayer version of War is pspace-hard.

1. Introduction

War is a card game so simple that only a child could love it. Here are the basics: the deck is divided into piles, one for each player. At the beginning of each round, each player reveals the top card of her deck. Then, whoever has the best card wins all the revealed cards and places them at the bottom of her deck.

The fun, such as it is, comes when there is a tie. What a tie occurs, the players do "battle": each puts down additional cards before turning over a final card, which decides who wins all the cards on the table. Players are eliminated when they lose all their cards; the winner is the last player standing.[1]

War is usually played between two players, but in this paper we consider a multiplayer version, which we call Multinational War. The full rules of Multinational War are given in Section 3, but for now it suffices to say that the game works exactly as it does with two players, except it tends to take longer.

One might try to ask various mathematical questions about War or its Multinational variant—about distributions of cards, length of play, and so forth[2]—but it's hard to imagine analyzing it as a *game*. After all, it doesn't even seem that players have any control over the outcome! That impression isn't quite true: the players have one way to exert control, in that the rules of the game do not specify the *order* in which captured cards are returned to the bottom of the deck.

Does this small choice give War any interesting strategic properties? Is there genuinely a game to play?

In this paper, we use the tools of computational complexity to analyze this question and establish that finding the best strategy in Multinational War is a hard problem. We show that, in this sense, playing Multinational War strategically is at least as hard as playing other classic games, such as Reversi [9], Rush Hour [8], Mahjong Solitaire [4], and Hex [13].

Date: June 29, 2016.

This work was partially supported by NSF Graduate Research Fellowship DGE-1122374.

[1]For more details, consult the Internet or the nearest four-year-old.

[2]We are aware of several such studies [2,3,11,15]. Strikingly, all of them ignore battles, which to many players are the most salient part of the game.

We prove:

Main Theorem (Informal version)**.** *Playing Multinational War strategically is computationally hard.*

Or, more precisely:

Main Theorem (Formal version)**.** *It is* PSPACE*-hard to decide whether an arrangement of cards is a win for Player 1 in Multinational War.*

The remainder of the paper shows how to move from the informal version to the formal version of the Main Theorem before giving a proof. Section 2 contains a gentle introduction to computational complexity for readers new to the subject. A formal definition of the problem appears in Section 3. Section 4 contains a description of the reduction that proves the Main Theorem. A detailed example of this reduction at work can be found in Section 5. We end with some open questions for future work.

2. An introduction to computational complexity

2.1. **When is a game hard?** Before describing the game of Multinational War in detail, it's worth pausing to discuss a more fundamental question: what does it mean to say that a game is hard?

One possible meaning is that the game is complicated. Perhaps the game has many rules that are difficult to understand, or it requires specialized equipment to play.

Of course, this is not a very satisfying definition. A game might have complicated dynamics but still be conceptually simple—children make up such games all the time. On the other hand, many famously difficult games, like Go, do not at first blush seem very complicated at all.

A more fruitful approach considers instead a game's strategic complexity. Computer scientists have developed sophisticated methods of measuring the complexity of problems in terms of the amount of computational resources (like computing time or disk space) it would take an ideal computer to solve them. By using their techniques on the problem of deciding how to play a game strategically, we arrive at a precise notion of what it means for a game to be hard. This approach has an appealing intuitive interpretation: a game is hard if it is difficult for a computer to find a winning strategy.

2.2. **PSPACE-hardness.** Defining the term "PSPACE-hard" in the formal statement of the Main Theorem requires some background material on computational complexity. To avoid unimportant technicalities, the following discussion will be somewhat vague. Experts should have no problem supplying details, and novices are encouraged to consult one of the many excellent textbooks on the subject [1,12,14].

Computer scientists often judge the difficulty of a problem by its *asymptotic worst-case complexity*. Consider for concreteness the problem PRIME(n) of deciding whether the positive integer n is a prime number. If we fix the integer n from the beginning—say $n = 52$—it is of course easy to program a computer to answer the question PRIME(52). (The solution is a one-line program that prints the word "No.") But this does not give us any information about the question of deciding whether a general integer n is prime. So instead we consider an arbitrary, large n, and ask how long it takes a computer to decide PRIME(n) as a function of n. If

there is an algorithm that can always do this correctly in a short amount of time even for very large values of n, then the problem seems easy. If, on the other hand, the best algorithms take an extremely long time as n gets large, then the problem seems hard. Making this intuition rigorous supplies a precise notion of hardness for computational problems.

It is important in the above definition that the algorithm be evaluated on worst-case inputs. For instance, when n is even, then PRIME(n) is easy no matter how large n is. To avoid such trivial cases, we judge an algorithm based on its performance on the hardest possible inputs. In other words, a problem is easy only if it can be solved quickly even in the worst case.

Once we agree to focus on asymptotic worst-case complexity, we can group together problems of similar difficulty. Two famous classes of problems are P, the class of problems for which a solution can be *found* in time polynomial in the size of the input, and NP, the class of problems for which a solution can be *checked* in time polynomial in the size of the input. It is clear that P $\subseteq$ NP, since checking a solution is no harder than coming up with a solution in the first place. (Deciding whether NP $\subseteq$ P is, to put it mildly, "still open" [6].)

The definitions of both P and NP refer to the amount of *time* a problem requires. By contrast, PSPACE is the class of problems solvable in polynomial *space*. That is, a problem is in PSPACE if there is a computer algorithm that can produce a solution using an amount of computer memory polynomial in the size of the input. It is not hard to verify that NP $\subseteq$ PSPACE, essentially because any program that runs in a polynomial amount of time cannot use more than a polynomial amount of space. However, it is believed that PSPACE is a strict superspace of NP and hence P, the intuition being that there are problems for which an optimal algorithm requires only a small memory footprint but still takes a long time to run.[3]

Informally, computer scientists often regard the problems in P as being "easy." One reason most computer scientists believe P $\neq$ NP is that NP seems to contain some problems that are very hard indeed. While we cannot yet prove that there are problems in NP (or PSPACE) that do not have polynomial-time solutions, there is still a way of making precise the idea that some problems are harder than others.

The primary tool for establishing relationships of this kind is the reduction. A reduction is a procedure that converts a problem of one type to a problem of another type. If problem A is easy (say, in P) and problem B can be converted into A easily (say, by a procedure that itself takes at most polynomial time), then B is easy (in P) too: just convert the instance of B into an instance of A and solve that. In the other direction, if B is hard and can be converted into A easily, then A is hard too, since otherwise we could solve any instance of B by first converting it into an instance of A. In short, if B reduces to A, then A is at least as hard as B. In this way, a hierarchy of difficulty can be established between problems of different types.

The hardest problems in any complexity class are called *complete* problems. For instance, an NP-complete problem is a problem in NP that *any* problem in NP can be reduced to (i.e., converted into). A solution procedure for the complete problem immediately implies a solution procedure for any other problem in the class.

[3]Like the P versus NP problem, there has been essentially no progress on the question of deciding whether the inclusion NP $\subseteq$ PSPACE is strict. It is even possible that P = NP = PSPACE, though that is not considered likely.

Many famous problems are NP-complete, such as the graph coloring problem, the knapsack problem, and the boolean satisfiability problem (SAT). (These examples appear in the seminal paper by Karp [10]; many more examples can be found in the monograph of Garey and Johnson [7].)

A PSPACE-complete problem is a problem in PSPACE that is at least as hard as any other problem in PSPACE. Since it is widely believed that NP is a strict subset of PSPACE, people believe PSPACE-complete problems to be significantly harder than NP-complete problems. A PSPACE-hard problem is a problem that is at least as hard as any PSPACE-complete problem. To show a problem is PSPACE-hard, as we will shortly do, it suffices to provide a reduction to the problem from any problem known to be PSPACE-complete.

With these definitions in place, we restate the formal version of the Main Theorem.

Main Theorem (Formal version)**.** *It is* PSPACE*-hard to decide whether an arrangement of cards is a win for Player 1 in Multinational War.*

Therefore, if P $\neq$ NP (or, to make a weaker assumption, if P $\neq$ PSPACE), then there is no polynomial-time algorithm for deciding how to act in Multinational War.

3. Multinational War

War is traditionally played by two players with a standard 52-card deck. In this paper, we consider Multinational War—a multiplayer generalization of this game to many players. In order to leverage the tools of asymptotic analysis, it is necessary to generalize the problem size, which we do by playing the game with a deck of n cards.

Making the reduction rigorous requires specifying the rules of Multinational War very precisely. We do so below. Many of the rules could be changed slightly without changing the substance of the reduction.

Since the suits of cards play no role in War, we ignore them here as well and stipulate only that each card is labeled with a nonnegative number called its *rank*. (Of course, the ranks on the cards are not unique.) To avoid having to talk about cards with very high rank, we *reverse* the standard order on the cards and say that a card of rank i beats a card of rank j if $i < j$. So, for instance, a card of rank 2 beats a card of rank 3, and no cards beat a card of rank 0.

Play proceeds in rounds. At the beginning of each round, each player reveals the top card of her deck. If one player's card beats all the other players', then she wins all the revealed cards. If, on the other hand, two or more players tie for the best card, then those players have a *battle*. Each player in a battle puts three more cards on the table and then reveals the fourth card.[4] If one player's fourth card beats all the other players' fourth cards, she wins all the cards played during the round, including those from the battle. If two or more players once again tie, then the tied players have another battle, and so on. (This "repeated battle" dynamic will be used extensively in our reduction.)

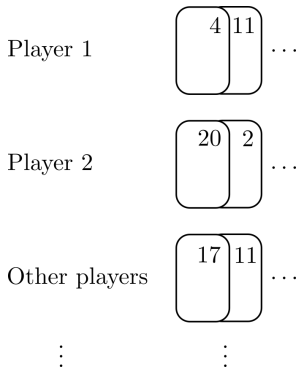
[4]Other variants of War requiring battling players to put down only one or two additional cards. Our reduction is tailored to the three-card case, but a modified version will work for any number of additional cards, as long as it remains constant across rounds.

Players are eliminated when they no longer have any cards in their deck. If players run out of cards in the middle of a battle, they are immediately eliminated.[5]

Finally, we imagine Multinational War in this context as a game of perfect information: every player knows the arrangement of every other player's deck at all times. (This version of the game is only easier than the one where Player 1 has to learn the position of other players' cards over the course of the game.) For simplicity, we depict all cards face up.

Given an initial arrangement of cards in the players' decks, deciding whether a position is a win for Player 1—that is, whether Player 1 can win no matter how the other players act—is a well defined problem with a fixed answer. It is this decision problem that we will show to be PSPACE-hard by providing a reduction from a PSPACE-complete problem. This reduction appears in Section 4.

In describing arrangements of cards in Multinational War, we will employ the compact notation described in Figure 1. The precise value of most cards in the game does not matter—all that matters is that these cards are not strong enough to affect the winner of the round. So we use the letter "x" to denote an unspecified weak card, with the promise that no x card will ever win a round. This allows us to better focus on the role of important cards in the reduction. Moreover, if players' decks contain only x cards during a particular set of rounds, we omit them from the listing.



P1	P2
4	x
x	2
⋮	⋮

FIGURE 1. Example of compact notation. The arrangement of cards at the top of Player 1 and Player 2's decks on the left-hand side will be written as in the table on the right-hand side. The top of the table stands for the top of each player's deck. The letter "x" will be used to denote an unspecified weak card whose precise value does not matter to the reduction. Players whose hands contain only x cards during certain rounds are omitted from tabular listings.

4. REDUCTION

The goal of this section is to give a proof of the Main Theorem by showing how to reduce a PSPACE-complete problem called QSAT to Multinational War. Section 4.1 introduces QSAT. Section 4.2 describes the overall structure of the reduction from

[5]This situation never arises in the instances of Multinational War produced in our reduction, so this choice of rule is arbitrary and does not affect the result.

QSAT to Multinational War. Section 4.3 gives more detail about the two most important pieces of the reduction: the *choice* and *checker gadgets*.

Throughout this section, some less important details of the reduction are omitted for clarity. A full example showing how the reduction functions on a particular QSAT instance appears in Section 5. The interested reader is encouraged to consult that section for full details on all parts of the construction.

4.1. **Quantified boolean satisfiability.** Giving a reduction requires specifying a starting point: a PSPACE-complete problem to reduce from. The problem we will use is called quantified boolean satisfiability (QSAT). As noted above, boolean satisfiability (SAT) is a complete problem for the class NP. The problem QSAT is an analogue of SAT, which is complete for the complexity class PSPACE.

Here is an example of a QSAT instance:

$$\exists x_1 \forall x_2 \exists x_3 : (x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)\,.$$

We say such an instance is *satisfiable* if it is logically true.

A QSAT instance is a boolean formula preceded by alternating quantifiers, such that every odd-numbered variable appears with an existential quantifier and every even-numbered variable appears with a universal quantifier. The boolean formula itself is given in "conjunctive normal form" (CNF). A CNF formula is composed of *clauses* joined by logical ANDs. Each clause consists of the logical OR of one or more variables or their negations. (These are called *literals*.)

To check whether the above formula is satisfiable, we need to check that there exists an assignment of x_1 such that for all assignments of x_2 there exists an assignment of x_3 such that the expression $(x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$ evaluates to true. (In fact, it is satisfiable: set x_1 to false. No matter what choice is made for the value of x_2, setting $x_3 = x_2$ satisfies the formula.)

There is another way to view the alternating quantifiers that appear at the beginning of a QSAT instance. These quantifiers act like a two-player game, in which the universal quantifiers are seen as choices made by an adversary. Players take turns choosing variables. Player 1 is trying to make the boolean formula true, and Player 2 is trying to make it false. (This is called a *formula game*.) The QSAT instance is satisfiable if Player 1 can force the formula to be true no matter what Player 2 does. (That's what it means for there to *exist* moves that work *for all* moves that the adversary makes.) In other words, the instance is satisfiable if and only if Player 1 has a winning strategy for the "game" of setting the formula to true.

4.2. **Overview of the reduction from QSAT.** Our reduction takes any QSAT instance and produces a corresponding configuration of cards in a game of Multinational War. Suppose we are given a QSAT instance with m clauses and $2n$ variables. We will produce a configuration of cards in a game of Multinational War with $m+6$ players. Two players, Player 1 and Player 2, will play the roles of the two players described in the formula game above. The remaining players correspond to the clauses of the QSAT instance and perform other infrastructure functions.

The configuration we produce will correspond to a game already in progress. Player 1 will have comparatively few cards, and will need to play very strategically in order to win the game. The goal of the reduction is to ensure that Player 1 can

eventually win the game if and only if she can force the QSAT instance's boolean formula to true—in other words, if and only if the QSAT instance is satisfiable.[6]

In this section, we give a general overview of the structure of the reduction, omitting some less interesting aspects in the service of showing the overall plan of attack. Section 4.2.1 analyzes the game mechanics of QSAT itself. Section 4.2.2 shows how these mechanics can be replicated inside a game of Multinational War. The Multinational War instance we produce naturally divides into several phases; these are described in more detail in Section 4.2.3.

4.2.1. *How can a* QSAT *instance become a game?* The formula game described in Section 4.1 is a game only in a very mild sense. To show a reduction, we will have to translate such a formula game into an instance of Multinational War.

Consider again the QSAT instance from above:

$$\exists x_1 \forall x_2 \exists x_3 : (x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)\,.$$

Let's try to take more seriously the idea that this instance is a game. Perhaps we can pretend for a moment that we're trying to sell QSAT to an executive in charge of developing new two-player board games. (If she's willing to consider QSAT, she must be pretty desperate for ideas.)

We've already shown how the first part of the instance (the alternating quantifiers $\exists x_1 \forall x_2 \exists x_3$) acts like a back-and-forth between two players, one who is trying to force the formula to evaluate to true, and the other who is trying to force the formula to evaluate to false. When pitching QSAT to the board-game executive, we describe this part of the game as the *assignment phase*, where players take turns setting variables to true or false.

Next, we must check whether the resulting boolean expression evaluates to true. To please the executive, we should also make this part as game-like as we can. One observation is that since the clauses in a CNF formula are joined by logical ANDs, the entire formula evaluates to true if and only if each clause does. So maybe this part can be interpreted as part of a game, too: the players can consider clauses one at a time, and check whether each one is satisfied. Since clauses consist of literals joined by logical ORs, a clause is satisfied if and only if it contains at least one true literal.

We can make this part of the game sound even more like a game by having Player 1 collect tokens as they check the clauses. Each clause has a unique token, so that Player 1 can't cheat by substituting one clause for another. For each clause, we award Player 1 a token whenever the clause contains a true literal. The token acts as proof that the clause is satisfied. Let's call this the *collection phase*.

There's one more step: we need to decide who wins and who loses. Since Player 1 has been collecting tokens during the collection phase, this part is simple. We should just have Player 2 check whether Player 1 has one token from each clause. If Player 1 can produce one token for each clause, that's proof that the formula is satisfied and Player 1 wins the game. Otherwise, Player 2 wins. We'll call this the *verification phase*.

[6]There is a further technical point: we must ensure that the reduction itself can be constructed in polynomial time. It is not hard to verify that the reduction given here satisfies these requirements. Given a QSAT instance with m clauses and $2n$ variables, the Multiplayer War instance we construct has $m + 6$ players and $\Theta(nm^2)$ cards.

To summarize, we can interpret a QSAT instance as a game with three phases: assignment, collection, and verification. To reduce a QSAT instance to an instance of Multinational War, we should find a way to simulate this game inside a game of War.

4.2.2. *Simulating* QSAT *inside Multinational War.* Simulating the QSAT instance inside a game of Multinational War involves adding $m+4$ players to the two players already described. These extra players encode the structure of the assignment, collection, and verification phases. Players 1 and 2 will start with very short decks, and the remaining players will start with much longer ones. (See Figure 2.) Player 2 is eliminated after the collection phase, and Player 1 cycles through her deck many times before the other players reach the end of their starting decks.

Fixing these starting decks allows us to control the behavior of every player other than Players 1 and 2 for most of the game. For example, if one player has a starting deck that is 300 cards long, then describing that player's starting deck fixes the player's behavior for the first 300 rounds. Moreover, if we arrange such that Player 1 has already either won or lost by the end of the first 300 rounds, then any other player whose starting deck is at least 300 cards long cannot influence the game in any way.

By carefully designing the contents of the other players' decks, we can ensure that Players 1 and 2 actually perform the tasks (such as setting variables and checking clauses) we would like them to perform. We can arrange the other players' decks such that if Player 1 tries to deviate from the intended course of the game, Player 1 will immediately be eliminated.

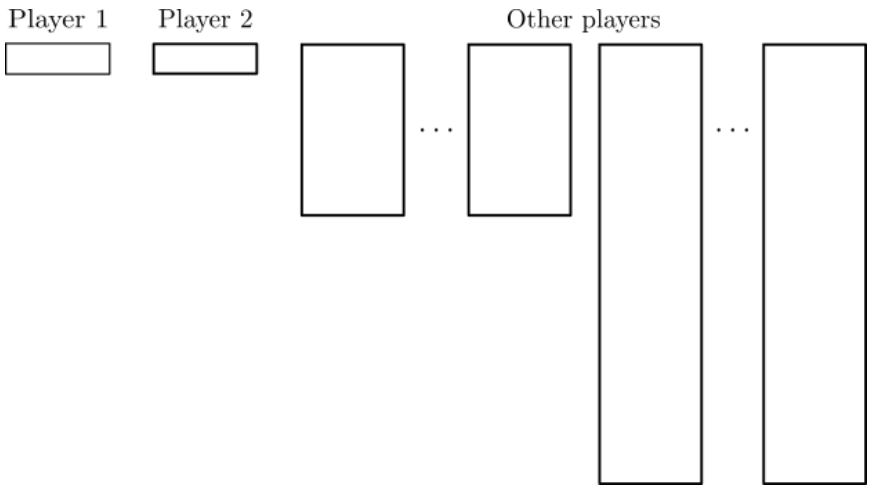


FIGURE 2. Relative starting deck lengths. Player 1 will cycle through her deck many times before the other players' decks run out.

In total, the reduction will require $m + 6$ players. A list of players appears in Table 1. Players 1 and 2 are the players who set the value of the variables—our goal is to determine whether Player 1 has a winning strategy. Each of the Clause players corresponds to a particular clause in the QSAT instance. Player 1 will prove

Name	Abbreviation	Role
Players 1 & 2	P1 & P2	Choose variable assignments
Clauses $1 \dots m$	$C1 \dots Cm$	Give tokens to P1 when clause is satisfied
Verifier	V	Verify that P1 holds tokens for all clauses
Filter	F	Verify that P1 holds tokens for all clauses
Parity [2]		Infrastructure

TABLE 1. Players

that she has satisfied a particular clause by winning a particular card from the corresponding Clause player. The Filter and Verifier players' decks contain the cards necessary to implement the verification phase. The Parity players have no role in the game except to ensure that there is the right number of players left during the verification phase.

All players other than Players 1 and 2 will have no choices that affect the outcome of the game. This is either because their decks are so long that none of their choices has any influence (the Filter, Verifier, and Parity players), or because they run out of cards without winning a single hand (the Clause players).

4.2.3. *Description of the phases.* The phases correspond to parts of the players' starting decks. Figure 3 gives a schematic representation of way the phases are encoded. The figure is highlighted to indicate the presence of "gadgets"—particular arrangements of cards that have a desired effect on the course of the game.

During the assignment phase, which lasts exactly the length P1 and P2's starting decks, either P1 or P2 will win every single round. By returning the cards they win in these rounds to the bottoms of their decks in a specific order, P1 and P2 will indicate whether they want to set particular variables to true or false. For instance, P1 will win the fist round of the game because the top card of her deck is more powerful than the top card of any other players' deck. When she collects the cards from that first round and returns them to the bottom of the deck, she can choose where to put that powerful card among the other cards won during the round. Putting it in a particular location will line it up with token cards held by the Clause players.

The collection phase begins directly after the assignment phase. The choices P1 and P2 made about how to order their cards during the assignment phase will allow them to capture tokens corresponding to satisfied clauses. This is because the Clause players' decks are arranged in such a way that they contain cards of particular ranks at particular locations corresponding to literals. Cards of these special ranks act as the tokens P1 is trying to collect. Each Clause player's deck contains tokens at locations that correspond to the literals that appear in the corresponding clause.

We will represent token cards with the notation $T1, T2, \dots$. It will be important for our reduction that token cards can only be captured by cards of rank 0, 1, 2,

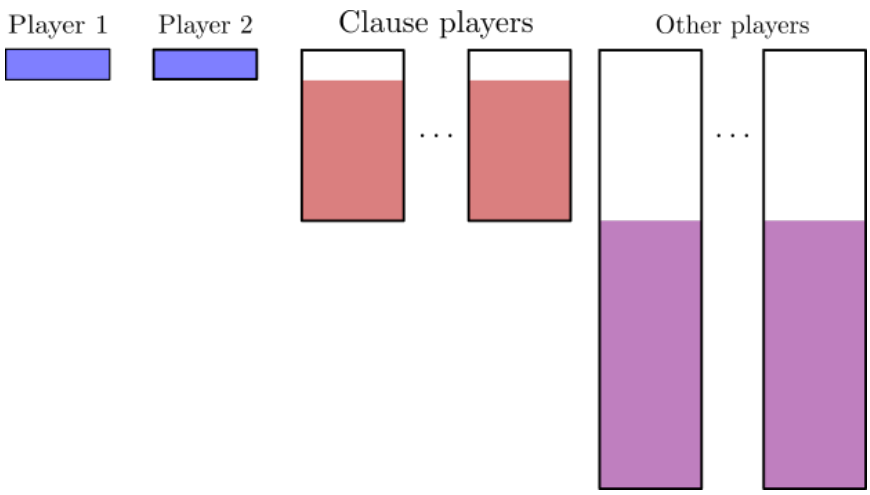


FIGURE 3. A schematic representation of the phases of Multinational War, with parts of the players' starting decks highlighted in places where they correspond to important gadgets. Areas marked in white contain unimportant x cards. The blue portions correspond to the assignment phase and contain *choice gadgets* that allow Players 1 and 2 to set the value of variables. The red portions correspond to the collection phase and indicate the region of the Clause players' decks where tokens are located. The purple portions correspond to the verification phase. The other players' decks contain *checker gadgets*, which check whether Player 1 possesses at least one token for each clause.

or 3.[7] If there are m clauses, then there are m token types $\mathrm{T}1, \ldots, \mathrm{T}m$, each one appearing in the deck of a single Clause player.

The data of the clauses are encoded by the location of token cards. For example, suppose the first clause in the QSAT instance is $(x_3 \vee \neg x_{12} \vee x_{13})$. Then C1's starting deck would contain T1 tokens in the positions corresponding to x_3, $\neg x_{12}$, and x_{13}. Other clauses containing these literals would have their corresponding tokens in these positions as well. If P1 sets x_3 to true, then she will line up a powerful card with the x_3 position in the Clause players' decks. When the corresponding round occurs, P1's powerful card will capture tokens from all the clauses containing x_3 literals. This corresponds to the fact that by setting x_3 to true, P1 has satisfied all the clauses in which x_3 appears.

The verification phase occurs once the Clause players' decks end. The design of the verification phase is the most difficult part of the reduction: the players' decks must be arranged in such a way that Player 1 will win during the verification phase if and only if she has collected at least one token from each clause. This is where battles are used: Player 1 will "prove" that she holds a particular token by ordering her cards in such a way that she causes a battle to occur between herself and the

[7]Precisely, we assume that $3 < \mathrm{T}1 < \mathrm{T}2 < \cdots < 4$. Even though we use special notation for token cards, we stress that these are still regular cards in the deck. They are special by virtue of their position in the Clause players' decks and their role in the verification phase.

P1	P2
2	x
x	5

(A) P1 choice gadget

P1	P2
x	1
2	x

(B) P2 choice gadget

TABLE 2. Choice gadgets. Players other than P1 and P2 omitted. See Figure 1 for a description of compact notation.

Verifier, who also holds a copy of the token. If she cannot provoke a battle in this way, we will arrange the cards in the other players' decks such that she immediately loses the game.

If P1 manages to survive the verification phase, we will show that P1 is then able to win the game.

4.3. **Gadgets.** A *gadget* is a set of rounds in the game of Multiplayer War designed for a particular end. The gadgets in this reduction involve multiple players, and a gadget is specified by saying which cards the players have in their decks at a particular location. In this section, we will describe the two most important gadgets. The first, the *choice gadget* is the main building block of the assignment phase. The choice gadget allows Players 1 and 2 to set the value of variables in the manner described above. Combining $2n$ copies of the choice gadget allows the players to set the value of all $2n$ variables.

The *checker gadget* is the main building block of the verification phase. It checks for the presence of a particular token in Player 1's deck. The verification phase contains m checker gadgets, one to check each token. (Recall that Player 1 should possess at least one token of each type if she has satisfied the QSAT instance.)

4.3.1. *Choice gadgets.* The QSAT instance specifies that Player 1 sets the odd-numbered variables $(x_1, x_3, \dots)$ in alternation with Player 2, who sets the even-numbered variables $(x_2, x_4, \dots)$. These choices correspond to two slightly different gadgets—a Player 1 choice gadget and a Player 2 choice gadget. P1 and P2's starting decks contain these gadgets in alternation, one for each variable. Each choice gadget is two rounds long.

The Player 1 and Player 2 choice gadgets appear in Table 2.

In the P1 choice gadget, the most powerful card of the first round, which is of rank 2, belongs to Player 1. (All players other than Player 1 have x cards at the tops of their decks.) So Player 1 wins the first round and places the card of rank 2 (as well as less powerful cards from the other players) at the bottom of her deck.

Player 1's choice is encoded by where she decides to put the card of rank 2. If she puts it on top of the set of cards that she returns to the bottom of her deck, then it will line up with tokens in the decks of Clause players whose corresponding clauses contain the variable in positive (non-negated) form. Putting the card of rank 2 in this position will lead to P1's being able to capture tokens from all clauses in which the variable appears in positive form in a later round. In other words, putting the card of rank 2 on top corresponds to setting the variable to true.

If P1 places the card of rank 2 second, it will line up with tokens of Clause players for clauses where the variable appears in negated form, which P1 will then

be able to capture in a later round. Putting the card in the second position therefore corresponds to setting the variable to false.

If the card of rank 2 is placed elsewhere, it will not line up with a token and will be captured by another player, so placing the card of rank 2 elsewhere does not help Player 1.

Finally, Player 2 wins the second round of the P1 choice gadget, and the order in which those cards are returned to the bottom of his deck has no bearing on the game. (The second round exists only so that Player 1 and Player 2 win the same number of rounds during each choice gadget, which ensures that their decks stay in sync.)

The Player 2 choice gadget is very similar to the Player 1 choice gadget except that P2 wins the first round and makes the first choice about how to return cards to the bottom of the deck. By doing so, P2 forces P1 to set the variable to a particular value.

In the Player 2 choice gadget, the first round is won by Player 2. He should put the powerful card of rank 1 in a position *opposite* the value he seeks to set: putting it first corresponds to "blocking" true (thereby setting the variable to false), and putting it second blocks false (thereby setting the variable to true). When Player 1 wins the second hand in the choice gadget, she has a chance to place a card of rank 2 at some position at the bottom of her deck. If she lines the 2 card up with the 1 card, then she loses it to Player 2 without winning any tokens. So she should place it opposite Player 2's choice. (As before, placing it somewhere other than the first two positions is tantamount to throwing it away.) In this way, Player 2 can force Player 1 to set even-numbered variables to particular values.

4.3.2. *Checker gadget.* By the beginning of the verification phase, P2 has already run out of cards, and the only players remaining are P1, F, V and the two Parity players who hold only unimportant x cards. The verification phase verifies that Player 1 has captured at least one token of each type. This is accomplished by the checker gadget, which checks that Player 1's deck contains a card of a certain rank.

The checker gadget is based on repeated battles. If P1 has a token of the correct type, she can force a battle with the Verifier player by placing the token in a particular location in her deck. The battle between P1 and V causes a later string of battles between V and F. This string of battles between V and F will not involve P1 and will consume a long run of powerful cards in V and F's decks, which P1 has no chance of beating. Only if P1 has the right token can she set off this string of battles between V and F. If she does not place a token from each clause in the correct position, the battles between V and F will not occur and the long run of high cards will ensure that she loses the game.

The run of high cards in V and F's decks is called *the gauntlet*. These powerful cards are the same in the Verifier and Filter players' decks, but they are shifted relative to each other by four cards. An example checker gadget and gauntlet appear in Table 3A.

If Player 1 causes a battle with the Verifier by placing a token in the correct position, then P1 and V alone play four additional cards. As a result, F and V's decks shift relative to each other by four cards, and the alternating sets of the gauntlet will align. The state of the decks after the battle for the T1 token appears in Table 3B.

P1	V	F	Notes
T1	T1	x	T1 checker
x	1	x	
x	1	x	
x	1	x	
x	1	x	
3	4	2	Gauntlet
x	1	2	
x	1	2	
x	1	2	
x	2	1	
	2	1	
	2	1	
	2	1	
	...		

(A) Before battle for T1 token.

P1	V	F	Notes
3	4	x	Gauntlet
x	1	x	
x	1	x	
x	1	x	
x	2	2	
	2	2	
	2	2	
	2	2	
	...		

(B) After battle for T1 token.

TABLE 3. The checker gadget before and after the battle for the T1 token. In Table A, the first card in the battle between P1 and V is marked in red. P1 and V then play the four additional cards marked in blue. This lines up the purple portions in V and F's decks. In Table B, the purple portion of V and F's decks are aligned. The cards highlighted in red will cause a battle.

Now, when the gauntlet is reached, the Verifier and Filter players do battle among themselves repeatedly before F eventually wins all the cards in the gauntlet. P1 has not lost any cards except for those lost in the initial battle with V. The high cards of the gauntlet are now at the bottom of F's deck. After this run of high cards, V and F's decks contain the checker gadget corresponding to the next token, and this pattern continues until all of P1's tokens have been checked.

5. A DETAILED EXAMPLE

Because of the large number of cards required to represent even a simple instance of QSAT, it is impractical to give a full example of this reduction except in trivial cases. In this section, we will do just that: though our QSAT instance is small, the Multinational War instance we produce will contain all gadgets necessary to produce an instance in the general case. Where necessary, we will indicate how to generalize the constructions presented here to the case where the QSAT instance has m clauses and $2n$ variables.

Consider the following QSAT instance:

$$\exists x_1 \forall x_2 (x_1) \wedge (x_1 \vee \neg x_2).$$

We note that this QSAT instance is satisfiable, and so our reduction produces a winning game for P1.

Phase	Purpose
1: Assignment	P1 and P2 use choice gadgets to set each variable
2: Collection	P1 wins tokens from C players whenever clause is satisfied
3: Sanitization	P1 discards excess cards
4: Verification	Checker gadgets verify that P1 has satisfied all clauses
5: Destruction	P1 wins enough powerful cards to win all remaining rounds

TABLE 4. Phases

The phases of the game appear in Table 4. In addition to the assignment, collection, and verification phases described in Section 4, the full reduction requires two additional phases for technical reasons.

The Sanitization phase allows Player 1 to prepare for the Verification phase. The rules of Multinational War do not allow players to arbitrarily reorder their deck or discard cards, but Player 1's deck must be in a particular arrangement if she is to successfully provoke the required battles during the Verification phase. This is the purpose of the Sanitization phase, which gives P1 opportunities to discard excess cards and reorder her deck as necessary. More details about the construction of the Sanitization phase appear in Section 5.3.

The Destruction phase allows Player 1 to win the game if she survives the Verification phase. If P1 does not have the required tokens during the Verification phase, she will be eliminated. But we wish to guarantee that if she does have the required tokens, she will be able to win the game. The Destruction phase gives P1 enough powerful cards to win the game. It is described in Section 5.5.

The starting decks of Players 1 and 2 contain choice gadgets allowing them to set the value of variables. The starting decks of the other 6 players contain information about the clauses as well as the infrastructure necessary to implement the sanitization, verification, and destruction phases. Schematics showing the composition of each player's starting deck appear in Figures 4, 5, and 6.

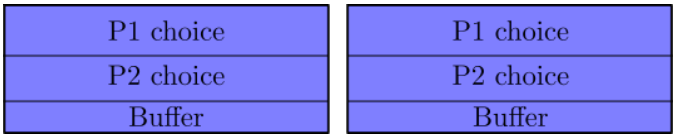


FIGURE 4. The composition of P1 and P2's starting decks. The top of the figure corresponds to the top of the deck. The listing of cards in each players deck may be found in Table 5.

5.1. **Assignment.** Table 5 shows the starting decks for P1 and P2.

For concreteness, we will suppose that the players set both x_1 and x_2 to true.

P1 wins the first round in the P1 choice gadget, thereby winning a set of cards $\{2, \mathrm{x}, \ldots, \mathrm{x}\}$. To set x_1 to true, she places the card of rank 2 in the first position at the bottom of her deck. P2 then wins a round, so that both players have won the same number of cards and therefore stay in sync.

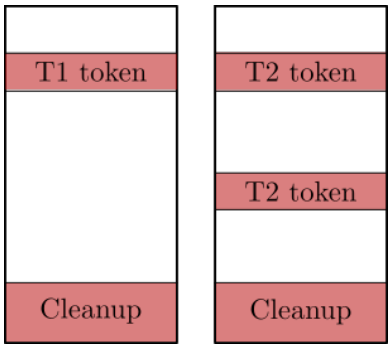


FIGURE 5. The composition of C1 and C2's starting deck. The top of the figure corresponds to the top of the deck. White areas indicate the location of unimportant x cards. The location of the T1 token indicates that the first clause contains the literal x_1. The location of the T2 tokens indicates that the second clause contains the literals x_1 and $\neg x_2$. More detail about the clause gadgets appears in Table 6.

P1	P2	Notes
2	x	P1 choice gadget
x	5	
x	1	P2 choice gadget
2	x	
2	x	Buffer

TABLE 5. Starting decks

P2 wins the first round in the P2 choice gadget and wins the set $\{1, \mathrm{x}, \ldots, \mathrm{x}\}$. To set x_2 to true, he places the card of rank 1 in the *second* position, which has the effect of blocking the false assignment. When P1 wins the second hand in the choice gadget, she wins the set $\{2, \mathrm{x}, \ldots, \mathrm{x}\}$. If she lines the 2 card up with the 1 card, she does not win any rounds corresponding to x_2, so she puts it in the second position.

This phase ends with one buffer round (n rounds in general). These rounds are always won by P1, and her choice of how to arrange these cards has no bearing on the rest of the game. The purpose of these cards is to ensure that P1 does not give up meaningful cards when P2 is being eliminated at the end of the Collection phase.

5.2. **Collection.** The Clause gadgets appear in Table 6, which depicts the entirety of the Collection phase. Note that most rounds in this phase do not carry any particular meaning and are won automatically by F. P2's deck at the beginning of this phase is shorter than P1's deck, but P2 wins the bolded round and these cards

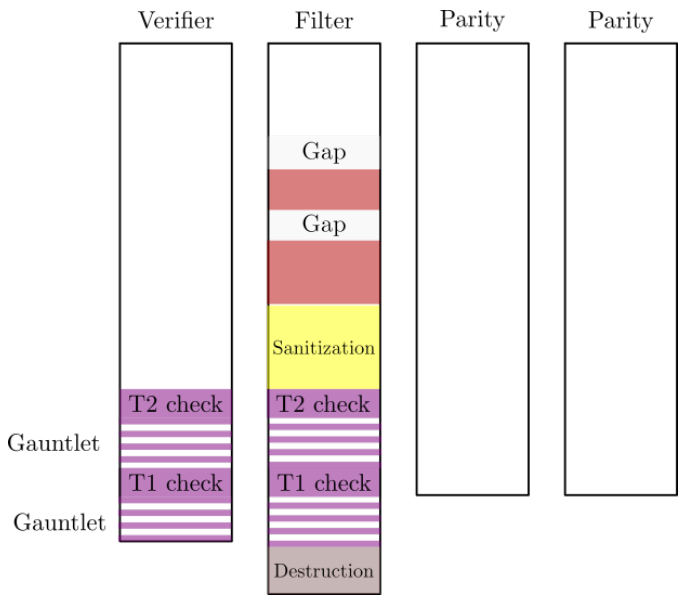


FIGURE 6. The composition of V, F, and the two Parity players' starting decks. The top of the figure corresponds to the top of the deck. White areas indicate the location of unimportant x cards. The red areas indicate the location of powerful cards held by F during the Collection phase, with gaps to allow P1 to win tokens. The yellow, purple, and gray sections correspond to the Sanitization, Verification, and Destruction phases, respectively. The Parity players have no role except to ensure that there are five players remaining during the Verification phase.

(along with P1's buffer cards) are captured by F during the period marked cleanup. P2, C1, and C2 run out of cards at the end of this phase.

The placement of the tokens in the decks of C1 and C2 reflects the presence of the corresponding literals in clause 1 and clause 2, respectively. For instance, since the second clause of the boolean formula is $(x_1 \vee \neg x_2)$, C2 has a token at the x_1 and $\neg x_2$ positions in the deck. (In general, if a clause contains a positive or negative literal corresponding to variable x_i, the corresponding Clause player will have a token at the $(5n + (i-1)(m+6) + 1)$th or $(5n + (i-1)(m+6) + 2)$th position, respectively.)

P1 wins tokens from both C1 and C2, corresponding to the fact that setting x_1 to true satisfies both clauses in the initial instance of QSAT. At the end of this phase, her deck contains T1 and T2 tokens as well as other cards, which will not be necessary for the remainder of the reduction. In order to prepare herself for the Sanitization phase, P1 should make sure the last card in her deck is a 2 card; the arrangement of all other cards is arbitrary.

5.3. **Sanitization.** P1's deck now contains $2n(m+6)$ cards, of which only some are valuable. The purpose of the Sanitization phase is to give her the chance to discard unwanted cards. The gadgets for the Sanitization phase appear in Table 7,

P1	P2	C1	C2	F	Notes
2	x	T1	T2	3	x_1
x	x	x	x	3	$\neg x_1$
x	x	x	x	1	
x	x	x	x	1	
x	x	x	x	1	
x	x	x	x	1	
x	x	x	x	1	
x	x	x	x	1	
2	x	x	x	3	x_2
x	**1**	**x**	**T**2	**3**	$\neg x_2$
x	x	x	x	1	
x	x	x	x	1	
x	x	x	x	1	
x	x	x	x	1	
x	x	x	x	1	
x	x	x	x	1	
2		x	x	0	Cleanup
x		x	x	0	
x		x	x	0	
x		x	x	0	
x		x	x	0	
x		x	x	0	
x		x	x	0	
x		x	x	0	

TABLE 6. Collection phase

where we use a question mark symbol to indicate that the order of P1's cards does not matter. The cards marked x + 1 in F's deck indicate that we want P1 to win each of the first $2n(m+6)$ rounds, so during this period no player should hold cards more powerful than anything P1 has. We required that P1's last card at the end of the Collection phase be a 2 card so that she can win a 3 card, which Player 1 will use at the beginning of the Destruction phrase.

In order to survive the Verification phase, P1 will need to hold $5m + 5$ cards: m unique tokens, 1 card of rank 3, and $4m + 4$ dummy cards spaced evenly between them. If P1 has managed to collect tokens of all types during the Collection phase, she will be able to use the Sanitization phase to set her deck appropriately before Verification.

She can accomplish this during the second half of the Sanitization phase by shifting cards she wishes to preserve into the "windows" that appear in F's deck every five rounds during the second half of the phase. Any token that she places in a window is preserved in the next phase. Cards preserved in this way should always be placed in the first position as they are returned to the bottom of the deck.

P1	F	Notes
?	x + 1	$2n(m+6)-1$ rounds
2	3	1 round
?	4	$10n(m+6)$ rounds
?	1	
?	1	
?	1	
?	1	

TABLE 7. Sanitization phase

5.4. **Verification.** One sub-phase of the Verification phase is presented in Table 8. Note that P1's deck is of the desired form. The first sub-phase checks for the presence of a T2 token, and the second checks for the presence of a T1 token. (In general, there will be m sub-phases, one for each token.) Recall from Section 4.3 that the gauntlets between checkers are designed to ensure that P1 loses immediately if she does not hold tokens for each clause.

At the end of the Sanitization phase, if P1 has satisfied the instance, then she holds m unique tokens, one for each clause. Each token is separated from the next by 4 dummy cards whose ranks do not matter. We do not assume that the tokens are in any particular order.

Tokens are checked in order from least to most powerful. As a token is checked, it is captured by the Verifier player. To check for the presence of token Ti, the Verifier player contains i copies of the Ti checker gadget in a row, followed by the gauntlet. When token Ti is being checked, P1 has exactly i tokens left in her deck: T1, ..., Ti. When token Tj for $j < i$ meets token Ti in the Verifier's deck, P1 wins the round and places the copy of Tj back at the bottom of the deck along with the four other cards won during the round (one from V, one from F, and two from the Parity players), with the token card Tj placed first. When P1's token Ti meets token Ti in the Verifier's deck, a battle occurs. As long as exactly one battle occurs over the course of the i copies of the Ti checker gadget, P1 will be able to pass through the gauntlet. If P1 causes no battles or more than one battle, she will lose the game.

The design of the checker gadget makes necessary the presence of the two Parity players in Table 1. The length of each checker gadget is a multiple of 5, so the number of cards in Player 1's deck should also be a multiple of 5. Without the two Parity players, Player 1 would only win 3 cards on any round during this stage of the game.

5.5. **Destruction.** At the end of the Verification phase, P1 holds five cards, one of which is a 3 card and one of which is a 4 card. If P1 has managed to survive this long, then she has demonstrated that the boolean formula evaluates to true. The purpose of the Destruction phase is to allow her to win the game.

During the Destruction phase, F loses a series of battles with P1, during which P1 wins a great number of unbeatable cards. A battle allows a player with weaker cards to win stronger cards if those stronger cards appear as part of the set of three

P1	V	F	Notes
T1	T2	x	T2 checker
x	1	x	
x	1	x	
x	1	x	
x	1	x	
T2	T2	x	T2 checker
x	1	x	
x	1	x	
x	1	x	
x	1	x	
3	4	2	Gauntlet
x	1	2	
x	1	2	
x	1	2	
x	2	1	
	2	1	
	2	1	
	2	1	
	...		

TABLE 8. Verification sub-phase, which checks for the presence of a T2 token.

cards each player plays during a battle before turing up the fourth card. Since the contents of P1's deck are entirely knowable, we can arrange for F's cards to match P1's repeatedly as P1 wins more and more battles. Each time she does so, her deck gets larger and eventually contains a run of cards long enough to eliminate F and V entirely. If P1 enters this round, therefore, she can successfully win the game. Table 9 shows the first two stage of Destruction, in which F gives P1 a large number of 0 cards—the most powerful cards in the game. We require at the end of the first stage that P1 place captured 0 cards at the first and fifth positions in her stack, so that she continues to cause battles with F in the second stage. Continuing in this way, F can feed enough 0 cards to P1 so she can win the game.

6. Conclusion and open questions

We have shown via a reduction from QSAT that determining whether a given arrangement of cards is a win for Player 1 in Multinational War is PSPACE-hard. The most tantalizing question left open is whether the two-player version of War is also hard. The present reduction relies heavily on the presence of multiple players, so it is unlikely that a similar approach would work in a two-player setting.

Another natural question is whether the problem we consider is itself in PSPACE and therefore PSPACE-complete. There are reasons to believe that this is not the case. Even in the two-player setting, it is known that there are finite War instances

P1	F	Notes
3	3	Stage 1
x	0	
x	0	
x	0	
4	x	
	0	Stage 2
	0	
	0	
	0	
	0	
	0	
	0	
	0	
	x	
	...	

TABLE 9. Destruction phase

that nevertheless take an exponential amount of time to play [5]. If solving Multinational War is not in PSPACE, it may be possible to show that Multinational War is hard with respect to some larger complexity class.

The story of War's computational complexity is far from over, which is good news: it gives you something to think about if someone ever makes you actually play.

ACKNOWLEDGEMENTS

Thanks to Erik Demaine, Sarah Eisenstat, and Robert Niles for fruitful discussions.

REFERENCES

1. Sanjeev Arora and Boaz Barak, *Computational complexity: a modern approach*, Cambridge U. P., 2009.
2. E. Ben-Naim and P. L. Krapivsky, *Parity and ruin in a stochastic game*, Eur. Phys. J. B Condens. Matter Phys. **25** (2002), no. 2, 239–243. MR 1892420 (2003b:60130)
3. Marc A. Brodie, *Avoiding Your Spouse at a Party Leads to War*, Math. Mag. **75** (2002), no. 3, 203–208. MR 1573610
4. Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter Shor, *Random debaters and the hardness of approximating stochastic functions*, SIAM J. Comput. **26** (1997), no. 2, 369–400. MR 1438521 (98a:68067)
5. Erik Demaine, personal communication.
6. Lance Fortnow, *The status of the P versus NP problem*, Commun. ACM **52** (2009), no. 9, 78–86.
7. Michael R. Garey and David S. Johnson, *Computers and intractability*, Freeman, 1979.
8. Robert A. Hearn and Erik D. Demaine, *PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation*, Theoret. Comput. Sci. **343** (2005), no. 1-2, 72–96. MR 2168845 (2006d:68070)

9. Shigeki Iwata and Takumi Kasai, *The Othello game on an $n \times n$ board is PSPACE-complete*, Theoret. Comput. Sci. **123** (1994), no. 2, 329–340. MR 1256205 (95a:68043)
10. Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972), Plenum, New York, 1972, pp. 85–103. MR 0378476 (51 #14644)
11. Evgeny Lakshtanov and Vera Roshchina, *On finiteness in the card game of war*, Amer. Math. Monthly **119** (2012), no. 4, 318–323. MR 2900977
12. C. Papadimitriou, *Computational complexity*, Wiley, 2003.
13. Stefan Reisch, *Hex ist PSPACE-vollständig*, Acta Inform. **15** (1981), no. 2, 167–191. MR 599616 (83d:68043)
14. Michael Sipser, *Introduction to the theory of computation*, Cengage, 2012.
15. Michael Z. Spivey, *Cycles in War*, Integers **10** (2010), G02, 747–764. MR 2797779 (2012c:05021)

MIT Department of Mathematics, 77 Massachusetts Avenue, Cambridge, MA 02139
E-mail address: jweed@mit.edu